

CoRE
Internet-Draft
Intended status: Experimental
Expires: 6 May 2021

C. Amsüss
2 November 2020

CoAP over GATT (Bluetooth Low Energy Generic Attributes)
draft-amsuess-core-coap-over-gatt-01

Abstract

Interaction from computers and cell phones to constrained devices is limited by the different network technologies used, and by the available APIs. This document describes a transport for the Constrained Application Protocol (CoAP) that uses Bluetooth GATT (Generic Attribute Profile) and its use cases.

Note to Readers

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

Source for this draft and an issue tracker can be found at <https://gitlab.com/chrysn/coap-over-gatt/> (<https://gitlab.com/chrysn/coap-over-gatt/-/tree/master>).

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 6 May 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Procedural status	3
1.2. Application example	3
2. Terminology	4
3. Protocol description	4
3.1. Requests and responses	4
3.2. Addresses	5
3.2.1. Scheme-free alternative	5
3.3. Compression and reinterpretation of non-CoAP characteristics	6
4. IANA considerations	6
4.1. Uniform Resource Identifier (URI) Schemes	6
5. Security considerations	6
6. References	6
6.1. Normative References	6
6.2. Informative References	7
Appendix A. Change log	8
Author's Address	8

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] can be used with different network and transport technologies, for example UDP on 6LoWPAN networks.

Not all those network technologies are available at end user devices in the vicinity of the constrained devices, which inhibits direct communication and necessitates the use of gateway devices or cloud services. In particular, 6LoWPAN is not available at all in typical end user devices, and while 6LoWPAN-over-BLE (IPSP, the Internet Protocol Support Profile of Bluetooth Low Energy (BLE), [RFC7668]) might be compatible from a radio point of view, many operating systems or platforms lack support for it, especially in a user-accessible way.

As a workaround to access constrained CoAP devices from end user devices, this document describes a way encapsulate generic CoAP exchanges in Bluetooth GATT (Generic Attribute Profile). This is explicitly not designed as means of communication between two devices in full control of themselves - those should rather build an IP based network and transport CoAP as originally specified. It is intended as a means for an application to escape the limitations of its environment, with a special focus on web applications that use the Web Bluetooth [webbluetooth]. In that, it is similar to CoAP-over-WebSockets [RFC8323].

1.1. Procedural status

[This section will be removed before publication.]

The path of this document is currently not clear. It might attract interest in the CoRE working group, but might be easier to process as an independent submission.

1.2. Application example

Consider a network of home automation light bulbs and switches, which internally uses CoAP on a 6LoWPAN network and whose basic pairing configuration can be done without additional electronic devices.

Without CoAP-over-GATT, an application that offers advanced configuration requires the use of a dedicated gateway device or a router that is equipped and configured to forward between the 6LoWPAN and the local network. In practice, this is often delivered as a wired gateway device and a custom app.

With CoAP-over-GATT, the light bulbs can advertise themselves via BLE, and the configuration application can run as a web site. The user navigates to that web site, and it asks permission to contact the light bulbs using Web Bluetooth. The web application can then exchange CoAP messages directly with the light bulb, and have it proxy requests to other devices connected in the 6LoWPAN network.

For browsers that do not support Web Bluetooth, the same web application can be packaged into a native application consisting of a proxy process that forwards requests received via CoAP-over-WebSockets on the loopback interface to CoAP-over-GATT, and a browser view that runs the original web application in a configuration to use WebSockets rather than CoAP-over-GATT.

That connection is no replacement when remote control of the system is desired (in which case, again, a router is required that translates 6LoWPAN to the rest of the network), but suffices for many commissioning tasks.

2. Terminology

3. Protocol description

3.1. Requests and responses

[This section is not thought through or implemented yet, and could probably end up very different.]

CoAP-over-GATT uses individual GATT Characteristics to model a reliable request-response mechanism. Therefore, it has no message types or message IDs (in which it resembles CoAP-over-TCP [RFC8323]), and no tokens. In the place of tokens, different Bluetooth characteristics (comparable to open ports in IP based networks) can be used. All messages use GATT to ensure reliable transmission.

A GATT server announces service of UUID 8df804b7-3300-496d-9dfa-f8fb40a236bc (abbreviated US in this document), with one or more characteristics of UUID 2a58fc3f-3c62-4ecc-8167-d66d4d9410c2 (abbreviated UC).

[Right now, this only supports requests from the GATT client to the GATT server; role reversal might be added later.]

A client can start a CoAP request by writing to the UC characteristic a sequence composed of a single code byte, any options encoded in the option format of [RFC7252] Section 3.1, optionally followed by a payload marker and the request payload.

After the successful write, the client can read the response back from the server on the same characteristic. The client may need to attempt reading the characteristic several times until the response is ready, and may subscribe to indications to get notified when the response is ready.

The server does not need to keep the response readable after it has been read successfully.

If the request and initial response establish an observation, the client may keep reading; the server may keep the latest notification available indefinitely (especially if it turns out that "has been read successfully" is hard to determine) or make it readable only once for each new state.

Once the client writes a new request to a UC characteristic, any later reads pertain to that request, and any observation previously established is cancelled implicitly.

Attribute values are limited to 512 Bytes ([bluetooth52] Part F Section 3.2.9), practically limiting blockwise operation ([RFC7959]) to size exponents to 4 (resulting in a block size of 256 byte). Even smaller messages might enhance the transfer efficiency when they avoid fragmentation at the L2CAP level.

If a server provides multiple OC typed characteristics, parallel requests or observations are possible; otherwise, this transport is limited to a single pending request.

3.2. Addresses

```
[ ... coap+bluetooth://00-11-22-33-44-55-66-77-88-99/.well-known/core
... ]
```

Note that when using Web Bluetooth [webbluetooth], neither the own nor the peer's address are known to the application. They may come up with an application-internal authority component (e. g. "coap+bluetooth://id-SomeInternalIdentifier/.well-known/core"), but must be aware that those can not be expressed towards anything outside the local stack.

3.2.1. Scheme-free alternative

As an alternative to the abovementioned scheme, a zone in .arpa could be registered to use addresses like

```
coap://001122334455.ble.arpa/.well-known/core
```

where the .ble.arpa address do not resolve to any IP addresses.

```
[ Accepting this will require a .arpa registering IANA consideration
to replace the URI one. ]
```

3.3. Compression and reinterpretation of non-CoAP characteristics

The use of SCHC is being evaluated in combination with CoAP-over-GATT; the device can use the characteristic UUID to announce the static context used.

Together with non-traditional response forms ([I-D.bormann-core-responses] and contexts that expand, say, a numeric value 0x1234 to a message like

```
"2.05 Content Response-For: GET /temperature Content-Format:
application/senml+cbor Payload (in JSON-ish equivalent): [ {1 /* unit
*/: "K", 2 /* value */: 0x1234} ]"
```

This enables a different use case than dealing with limited environments: Accessing BLE devices via CoAP without application specific gateways. Any required information about the application can be expressed in the SCHC context.

4. IANA considerations

4.1. Uniform Resource Identifier (URI) Schemes

IANA is asked to enter a new scheme into the "Uniform Resource Identifier (URI) Schemes" registry set up in [RFC7595]:

- * URI Scheme: "coap+gatt"
- * Description: CoAP over Bluetooth GATT (sharing the footnote of coap+tcp)
- * Well-Known URI Support: yes, analogous to [RFC7252]

5. Security considerations

All data received over GATT is considered untrusted; secure communication can be achieved using OSCORE [RFC8613].

Physical proximity can not be inferred from this means of communication.

6. References

6.1. Normative References

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

6.2. Informative References

- [RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015, <<https://www.rfc-editor.org/info/rfc7668>>.
- [webbluetooth] Grant, R. and O. Ruiz-Henríquez, "Web Bluetooth", 24 February 2020, <<https://webbluetoothcg.github.io/web-bluetooth/>>.
- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [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>>.
- [bluetooth52] "Bluetooth Core Specification v5.2", 31 December 2019, <https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=478726>.
- [I-D.bormann-core-responses] Bormann, C., "CoAP: Non-traditional response forms", Work in Progress, Internet-Draft, draft-bormann-core-responses-00, 12 November 2017, <<http://www.ietf.org/internet-drafts/draft-bormann-core-responses-00.txt>>.

Appendix A. Change log

Since -00:

- * Add note on SCHC possibilities.

Author's Address

Christian Amsüss
Austria

Email: christian@amsuess.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 14, 2021

M. Koster
SmartThings
B. Silverajan, Ed.
Tampere University
July 13, 2020

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-11

Abstract

This specification defines Link Bindings, which provide dynamic linking of state updates between resources, either on an endpoint or between endpoints, for systems using CoAP (RFC7252). This specification also defines Conditional Notification Attributes that work with Link Bindings or with CoAP Observe (RFC7641).

Editor note

The git repository for the draft is found at <https://github.com/core-wg/dynlink>

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 January 14, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Conditional Notification Attributes	4
3.1. Attribute Definitions	4
3.1.1. Minimum Period (pmin)	5
3.1.2. Maximum Period (pmax)	5
3.1.3. Change Step (st)	5
3.1.4. Greater Than (gt)	6
3.1.5. Less Than (lt)	6
3.1.6. Notification Band (band)	6
3.2. Server processing of Conditional Notification Attributes	8
4. Link Bindings	8
4.1. The "bind" attribute and Binding Methods	9
4.1.1. Polling	10
4.1.2. Observe	10
4.1.3. Push	11
4.1.4. Execute	11
4.2. Link Relation	11
5. Binding Table	12
6. Implementation Considerations	13
7. Security Considerations	14
8. IANA Considerations	14
8.1. Resource Type value 'core.bnd'	14
8.2. Link Relation Type	14
9. Acknowledgements	15
10. Contributors	15
11. Changelog	16
12. References	18
12.1. Normative References	18
12.2. Informative References	18
Appendix A. Examples	19
A.1. Minimum Period (pmin) example	19
A.2. Maximum Period (pmax) example	20
A.3. Greater Than (gt) example	21
A.4. Greater Than (gt) and Period Max (pmax) example	22
Authors' Addresses	23

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol [RFC7252] and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format [RFC6690] is a standard for doing Web Linking [RFC8288] 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 state updates are conveyed. Specifically, a Link Binding is a unidirectional link for binding the states of source and destination 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 Conditional Notification Attributes for use with Link Bindings and with CoRE Observe [RFC7641].

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.

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC8288] 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.

Notification Band: A resource value range that results in state synchronization. The value range may be bounded by a minimum and maximum value or may be unbounded having either a minimum or maximum value.

3. Conditional Notification Attributes

3.1. Attribute Definitions

This specification defines Conditional Notification Attributes, which provide for fine-grained control of notification and state synchronization when using CoRE Observe [RFC7641] or Link Bindings (see Section 4). Conditional Notification Attributes define the conditions that trigger a notification.

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [RFC7641] MAY also be used to observe any changes in a resource, and receive asynchronous notifications as a result. A resource marked as Observable in its link description SHOULD support these Conditional Notification Attributes.

The set of parameters defined here 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.

One or more Notification Attributes MAY be included as query parameters in an Observe request.

These attributes are defined below:

Attribute	Parameter	Value
Minimum Period (s)	pmin	xs:decimal (>0)
Maximum Period (s)	pmax	xs:decimal (>0)
Change Step	st	xs:decimal (>0)
Greater Than	gt	xs:decimal
Less Than	lt	xs:decimal
Notification Band	band	xs:boolean

Table 1: Conditional Notification Attributes

Conditional Notification Attributes SHOULD be evaluated on all potential notifications from a resource, whether resulting from an

internal server-driven sampling process or from external update requests to the server.

Note: In this draft, we assume that there are finite quantization effects in the internal or external updates to the value of a resource; specifically, that a resource may be updated at any time with any valid value. We therefore avoid any continuous-time assumptions in the description of the Conditional Notification Attributes and instead use the phrase "sampled value" to refer to a member of a sequence of values that may be internally observed from the resource state over time.

3.1.1. Minimum Period (pmin)

When present, the minimum period indicates the minimum time, in seconds, between two consecutive notifications (whether or not the resource value has changed). In the absence of this parameter, the minimum period is up to the server. The minimum period **MUST** be greater than zero otherwise the receiver **MUST** return a CoAP error code 4.00 "Bad Request" (or equivalent).

A server **MAY** report the last sampled value that occurred during the pmin interval, after the pmin interval expires.

Note: Due to finite quantization effects, the time between notifications may be greater than pmin even when the sampled value changes within the pmin interval. Pmin may or may not be used to drive the internal sampling process.

3.1.2. Maximum Period (pmax)

When present, the maximum period indicates the maximum time, in seconds, between two consecutive notifications (whether or not the resource value has changed). In the absence of this parameter, the maximum period is up to the server. 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).

3.1.3. Change Step (st)

When present, the change step indicates how much the value of a resource **SHOULD** change before triggering a notification, compared to the value of the previous notification. Upon reception of a query including the st attribute, the most recently sampled value of the resource is reported, and then set as the last reported value (last_rep_v). When a subsequent sample or update of the resource value differs from the last reported value by an amount, positive or

negative, greater than or equal to st, and the time for pmin has elapsed since the last notification, a notification is sent and the last reported value is updated to the value sent in the notification. The change step MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

The Change Step parameter can only be supported on resources with a scalar numeric value.

Note: Due to sampling and other constraints, e.g. pmin, the resource value received in two sequential notifications may differ by more than st.

3.1.4. Greater Than (gt)

When present, Greater Than indicates the upper limit value the sampled value SHOULD cross before triggering a notification. A notification is sent whenever the sampled value crosses the specified upper limit value, relative to the last reported value, and the time for pmin has elapsed since the last notification. The sampled value is sent in the notification. If the value continues to rise, no notifications are generated as a result of gt. If the value drops below the upper limit value then a notification is sent, subject again to the pmin time.

The Greater Than parameter can only be supported on resources with a scalar numeric value.

3.1.5. Less Than (lt)

When present, Less Than indicates the lower limit value the resource value SHOULD cross before triggering a notification. A notification is sent when the sampled value crosses the specified lower limit value, relative to the last reported value, and the time for pmin has elapsed since the last notification. The sampled value is sent in the notification. If the value continues to fall no notifications are generated as a result of lt. If the value rises above the lower limit value then a new notification is sent, subject to the pmin time..

The Less Than parameter can only be supported on resources with a scalar numeric value.

3.1.6. Notification Band (band)

The notification band attribute allows a bounded or unbounded (based on a minimum or maximum) value range that may trigger multiple notifications. This enables use cases where different ranges results

in differing behaviour. For example: monitoring the temperature of machinery. Whilst the temperature is in the normal operating range only periodic observations are needed. However as the temperature moves to more abnormal ranges more frequent synchronization/reporting may be needed.

Without a notification band, a transition across a less than (lt), or greater than (gt) limit only generates one notification. This means that it is not possible to describe a case where multiple notifications are sent so long as the limit is exceeded.

The band attribute works as a modifier to the behaviour of gt and lt. Therefore, if band is present in a query, gt, lt or both, MUST be included.

When band is present with the lt attribute, it defines the lower bound for the notification band (notification band minimum). Notifications occur when the resource value is equal to or above the notification band minimum. If lt is not present there is no minimum value for the band.

When band is present with the gt attribute, it defines the upper bound for the notification band (notification band maximum). Notifications occur when the resource value is equal to or below the notification band maximum. If gt is not present there is no maximum value for the band.

If band is present with both the gt and lt attributes, notification occurs when the resource value is greater than or equal to gt or when the resource value is less than or equal to lt.

If a band is specified in which the value of gt is less than that of lt, in-band notification occurs. That is, notification occurs whenever the resource value is between the gt and lt values, including equal to gt or lt.

If the band is specified in which the value of gt is greater than that of lt, out-of-band notification occurs. That is, notification occurs when the resource value not between the gt and lt values, excluding equal to gt and lt.

The Notification Band parameter can only be supported on resources with a scalar numeric value.

3.2. Server processing of Conditional Notification Attributes

Pmin, pmax, st, gt, lt and band may be present in the same query. However, they are not defined at multiple prioritization levels. The server sends a notification whenever any of the parameter conditions are met, upon which it updates its last notification value and time to prepare for the next notification. Only one notification occurs when there are multiple conditions being met at the same time. The reference code below illustrates the logic to determine when a notification is to be sent.

```
bool notifiable( Resource * r ) {

#define BAND r->band
#define SCALAR_TYPE ( num_type == r->type )
#define STRING_TYPE ( str_type == r->type )
#define BOOLEAN_TYPE ( bool_type == r->type )
#define PMIN_EX ( r->last_sample_time - r->last_rep_time >= r->pmin )
#define PMAX_EX ( r->last_sample_time - r->last_rep_time > r->pmax )
#define LT_EX ( r->v < r->lt ^ r->last_rep_v < r->lt )
#define GT_EX ( r->v > r->gt ^ r->last_rep_v > r->gt )
#define ST_EX ( abs( r->v - r->last_rep_v ) >= r->st )
#define IN_BAND ( ( r->gt <= r->v && r->v <= r->lt ) || ( r->lt <= r->gt && r->gt
<= r->v ) || ( r->v <= r->lt && r->lt <= r->gt ) )
#define VB_CHANGE ( r->vb != r->last_rep_vb )
#define VS_CHANGE ( r->vs != r->last_rep_vs )

    return (
        PMIN_EX &&
        ( SCALAR_TYPE ?
            ( ( !BAND && ( GT_EX || LT_EX || ST_EX || PMAX_EX ) ) ||
              ( BAND && IN_BAND && ( ST_EX || PMAX_EX ) ) )
        : STRING_TYPE ?
            ( VS_CHANGE || PMAX_EX )
        : BOOLEAN_TYPE ?
            ( VB_CHANGE || PMAX_EX )
        : false )
    );
}
```

Figure 1: Code logic for conditional notification attribute interactions

4. 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 such an abstract relationship between two resources is defined, 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 such a binding is to synchronize content updates 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.

Conditional Notification Attributes defined in Section 3 can be used with Link Bindings in order to customize the notification behavior and timing.

4.1. The "bind" attribute and Binding Methods

A binding method defines the rules to generate the network-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.

This specification defines a new CoRE link attribute "bind". This is the identifier for a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

Attribute	Parameter	Value
Binding method	bind	xs:string

Table 2: The bind attribute

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
Execute	exec	Source	POST

Table 3: Binding Method Summary

The description of a binding method defines 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.

Conditional Notification: How Conditional Notification Attributes are used in the binding.

The binding methods are described in more detail below.

4.1.1. 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).

4.1.2. 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 parameters in the Observe request (see Section 3).

4.1.3. Push

The Push method can be used to allow a source endpoint to replace an outdated resource state at the destination with a newer representation. When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource when the Conditional Notification Attributes are satisfied for the source resource. The source endpoint **SHOULD** only send a notification request if any included Conditional Notification Attributes are met. The binding entry for this method **MUST** be stored on the source endpoint.

4.1.4. Execute

An alternative means for a source endpoint to deliver change-of-state notifications to a destination resource is to use the Execute Method. While the Push method simply updates the state of the destination resource with the representation of the source resource, Execute can be used when the destination endpoint wishes to receive all state changes from a source. This allows, for example, the existence of a resource collection consisting of all the state changes at the destination endpoint. When the Execute method is assigned to a binding, the source endpoint sends POST requests to the destination resource when the Conditional Notification Attributes are satisfied for the source resource. The source endpoint **SHOULD** only send a notification request if any included Conditional Notification Attributes are met. The binding entry for this method **MUST** be stored on the source endpoint.

Note: Both the Push and the Execute methods are examples of Server Push mechanisms that are being researched in the Thing-to-Thing Research Group (T2TRG) [I-D.irtf-t2trg-rest-iot].

4.2. Link Relation

Since Binding involves the creation of a link between two resources, Web Linking and the CoRE Link-Format used to represent binding information. This involves the creation of a new relation type, "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.

5. Binding Table

The Binding Table is a special resource that describes the bindings on an endpoint. An endpoint offering a representation of the Binding Table resource SHOULD indicate its presence and enable its discovery by advertising a link at `"/.well-known/core"` [RFC6690]. If so, the Binding Table resource MUST be discoverable by using the Resource Type (rt) `'core.bnd'`.

The Methods column defines the REST methods supported by the Binding Table, which are described in more detail below.

Resource	rt=	Methods	Content-Format
Binding Table	core.bnd	GET, PUT	link-format

Table 4: Binding Table Description

The REST methods GET and PUT are used to manipulate a Binding Table. A GET request simply returns the current state of a Binding Table. A request with a PUT method and a content format of `application/link-format` is used to clear the bindings to the table or replaces its entire contents. All links in the payload of a PUT request MUST have a relation type `"boundto"`.

The following example shows requests for discovering, retrieving and replacing bindings in a binding table.

```
Req: GET /.well-known/core?rt=core.bnd (application/link-format)
Res: 2.05 Content (application/link-format)
</bnd/>;rt=core.bnd;ct=40

Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/a/switch1/>;
    rel=boundto;anchor=/a/fan;;bind="obs",
<coap://sensor.example.com/a/switch2/>;
    rel=boundto;anchor=/a/light;bind="obs"

Req: PUT /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
```

Figure 2: Binding Table Example

Additional operations on the Binding Table can be specified in future documents. Such operations can include, for example, the usage of the iPATCH or PATCH methods [RFC8132] for fine-grained addition and removal of individual bindings or binding subsets.

6. Implementation Considerations

When using multiple resource bindings (e.g. multiple Observations of resource) with different bands, consideration should be given to the resolution of the resource value when setting sequential bands. For example: Given BandA (Abmn=10, BbmX=20) and BandB (Bbmn=21, BbmX=30). If the resource value returns an integer then notifications for values between and inclusive of 10 and 30 will be triggered. Whereas if the resolution is to one decimal point (0.1) then notifications for values 20.1 to 20.9 will not be triggered.

The use of the notification band minimum and maximum allow for a synchronization whenever a change in the resource value occurs. Theoretically this could occur in-line with the server internal sample period for the determining the resource value. Implementors SHOULD consider the resolution needed before updating the resource, e.g. updating the resource when a temperature sensor value changes by 0.001 degree versus 1 degree.

The initiation of a Link Binding can be delegated from a client to a link state machine implementation, which can be an embedded client or a configuration tool. Implementation considerations have to be given to how to monitor transactions made by the configuration tool with regards to Link Bindings, as well as any errors that may arise with establishing Link Bindings in addition to established Link Bindings.

7. Security Considerations

Consideration has to be given to what kinds of security credentials the state machine of a configuration tool or an embedded client needs to be configured with, and what kinds of access control lists client implementations should possess, so that transactions on creating Link Bindings and handling error conditions can be processed by the state machine.

8. IANA Considerations

8.1. Resource Type value 'core.bnd'

This specification registers a new Resource Type Link Target Attribute 'core.bnd' in the Resource Type (rt=) registry established as per [RFC6690].

Attribute Value: core.bnd

Description: See Section 5. This attribute value is used to discover the resource representing 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

8.2. Link Relation Type

This specification registers the new "boundto" link relation type as per [RFC8288].

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

9. 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. Christian Amsuss supplied a comprehensive review of draft -06. Hannes Tschofenig and Mert Ocaak highlighted syntactical corrections in the usage of pmax and pmin in a query. Discussions with Ari Keraenen led to the addition of an extra binding method supporting POST operations.

10. Contributors

Christian Groves
Australia
email: cngroves.std@gmail.com

Zach Shelby
ARM
Vuokatti
FINLAND
phone: +358 40 7796297
email: zach.shelby@arm.com

Matthieu Vial
Schneider-Electric
Grenoble
France
phone: +33 (0)47657 6522
eMail: matthieu.vial@schneider-electric.com

Jintao Zhu
Huawei
Xi'an, Shaanxi Province
China
email: jintao.zhu@huawei.com

11. Changelog

draft-ietf-core-dynlink-11

- o Updates to author list

draft-ietf-core-dynlink-10

- o Binding methods now support both POST and PUT operations for server push.

draft-ietf-core-dynlink-09

- o Corrections in Table 1, Table 2, Figure 2.
- o Clarifications for additional operations to binding table added in section 5
- o Additional examples in Appendix A

draft-ietf-core-dynlink-08

- o Reorganize the draft to introduce Conditional Notification Attributes at the beginning
- o Made pmin and pmax type xs:decimal to accommodate fractional second timing
- o updated the attribute descriptions. lt and gt notify on all crossings, both directions
- o updated Binding Table description, removed interface description but introduced core.bnd rt attribute value

draft-ietf-core-dynlink-07

- o Added reference code to illustrate attribute interactions for observations

draft-ietf-core-dynlink-06

- o Document restructure and refactoring into three main sections
- o Clarifications on band usage
- o Implementation considerations introduced
- o Additional text on security considerations

draft-ietf-core-dynlink-05

- o Addition of a band modifier for gt and lt, adapted from draft-groves-core-obsattr
- o Removed statement prescribing gt MUST be greater than lt

draft-ietf-core-dynlink-03

- o 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".
- o New author and editor added.

draft-ietf-core-dynlink-02

- o 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 conflict with the core resource directory draft lifetime "lt" attribute.
- o Clause 6.1: Addressed the editor's note by changing the link target attribute to "core.binding".
- o Added Appendix A for examples.

draft-ietf-core-dynlink-01

- o General: The term state synchronization has been introduced to describe the process of synchronization between destination and source resources.
- o General: The document has been restructured to make the information flow better.
- o Clause 3.1: The descriptions of the binding attributes have been updated to clarify their usage.
- o Clause 3.1: A new clause has been added to discuss the interactions between the resources.
- o Clause 3.4: Has been simplified to refer to the descriptions in 3.1. As the text was largely duplicated.
- o Clause 4.1: Added a clarification that individual resources may be removed from the binding table.

- o Clause 6: Formailised the IANA considerations.

draft-ietf-core-dynlink Initial Version 00:

- o This is a copy of draft-groves-core-dynlink-00

draft-groves-core-dynlink Draft Initial Version 00:

- o This initial version is based on the text regarding the dynamic linking functionality in I.D.ietf-core-interfaces-05.
- o The WADL description has been dropped in favour of a thorough textual description of the REST API.

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

12.2. Informative References

- [I-D.irtf-t2trg-rest-iot] Keranen, A., Kovatsch, M., and K. Hartke, "RESTful Design for Internet of Things Systems", draft-irtf-t2trg-rest-iot-06 (work in progress), May 2020.
- [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>>.
- [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>>.

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. Minimum Period (pmin) example

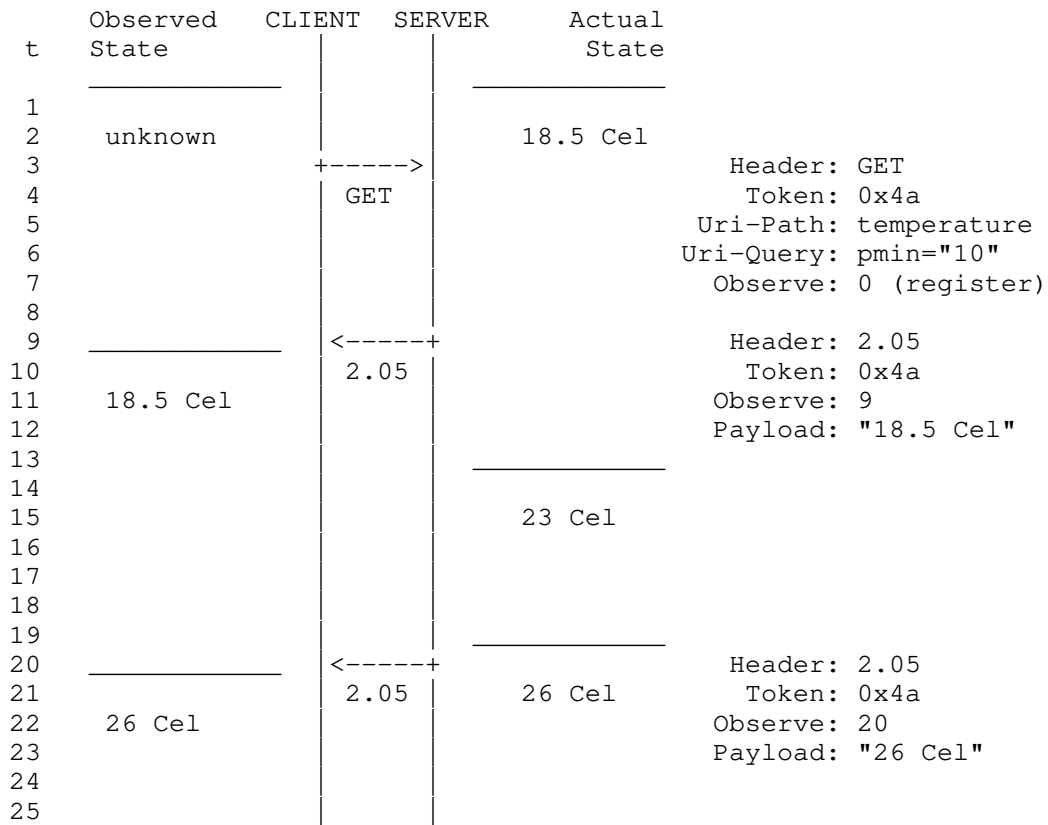
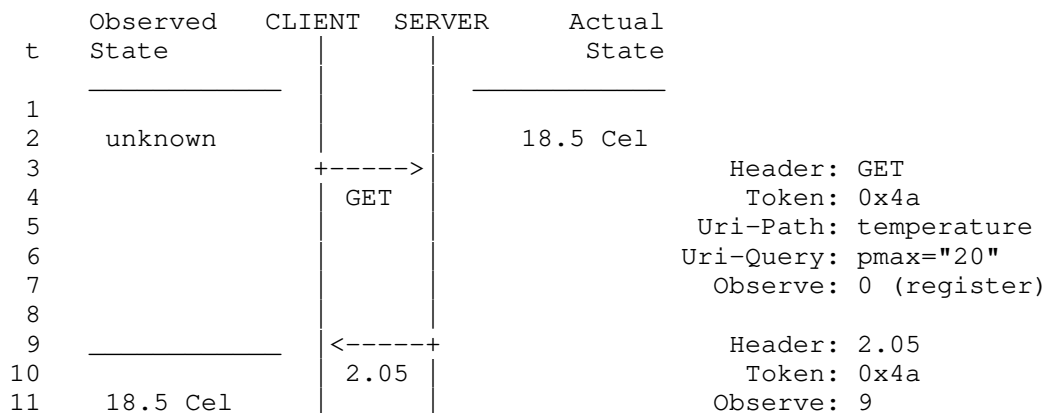


Figure 3: Client registers and receives one notification of the current state and one of a new state state when pmin time expires.

A.2. Maximum Period (pmax) example



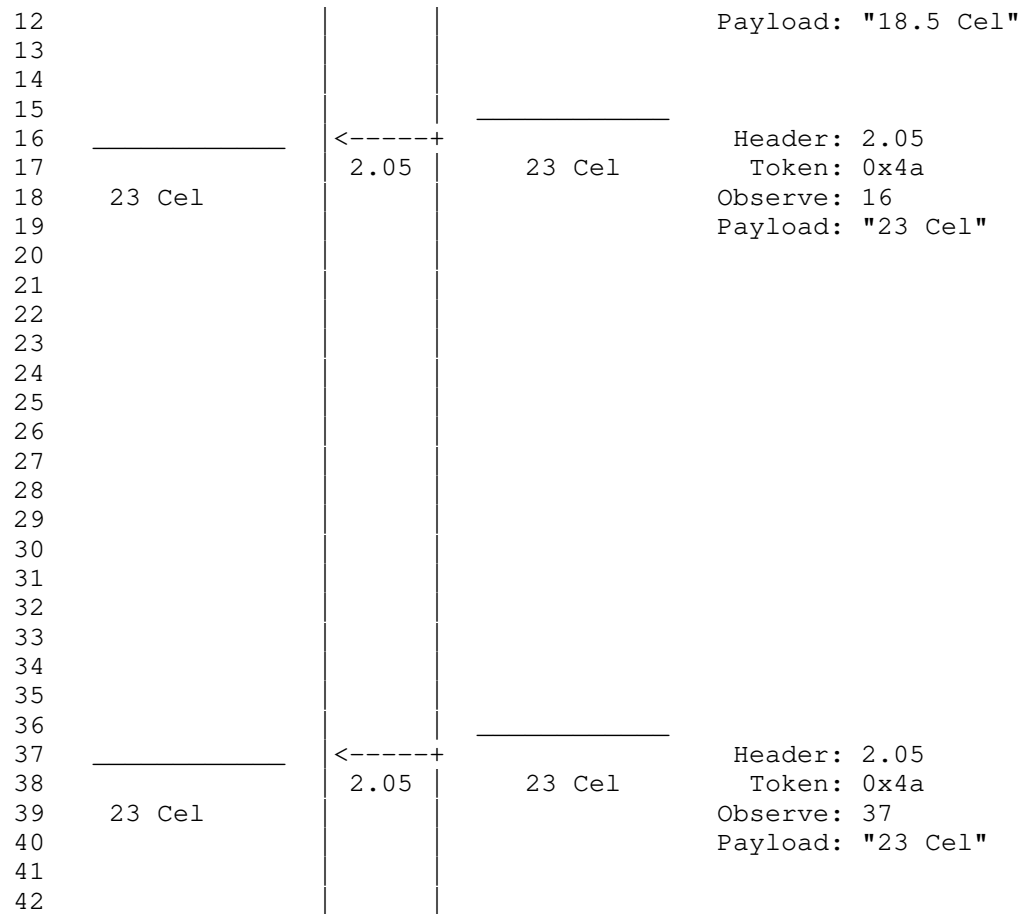


Figure 4: Client registers and receives one notification of the current state, one of a new state and one of an unchanged state when pmax time expires.

A.3. Greater Than (gt) example

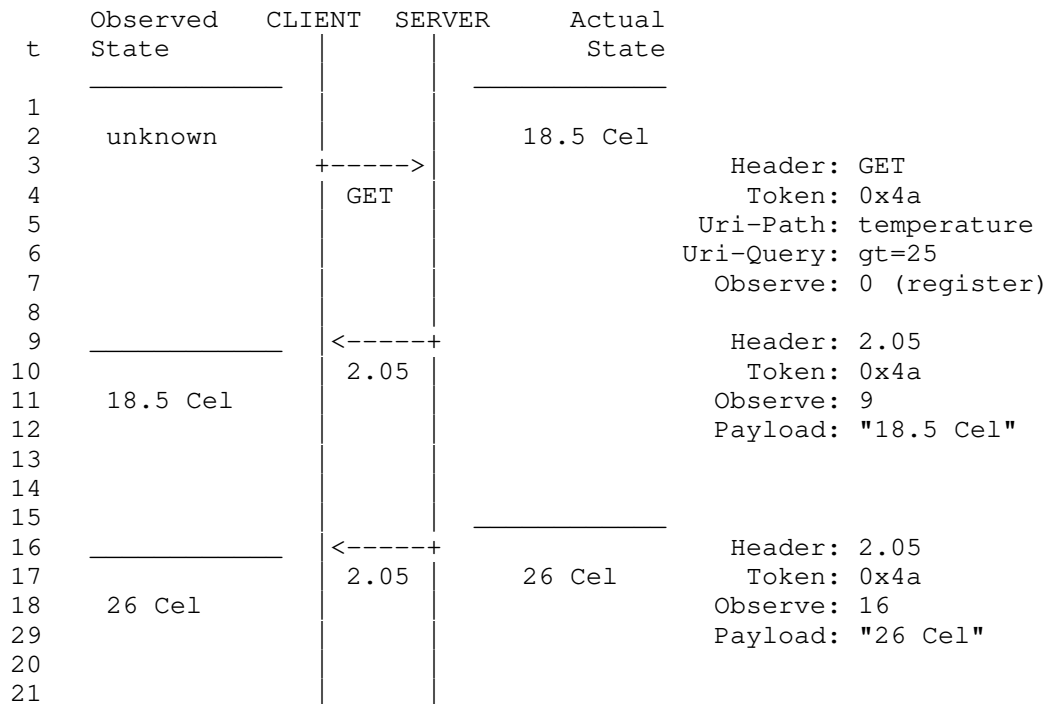
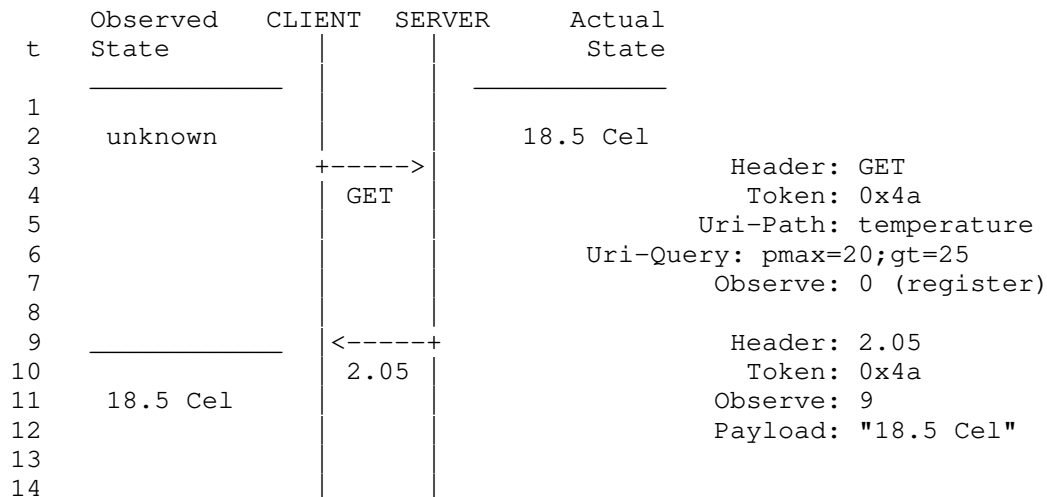


Figure 5: Client registers and receives one notification of the current state and one of a new state when it passes through the greater than threshold of 25.

A.4. Greater Than (gt) and Period Max (pmax) example



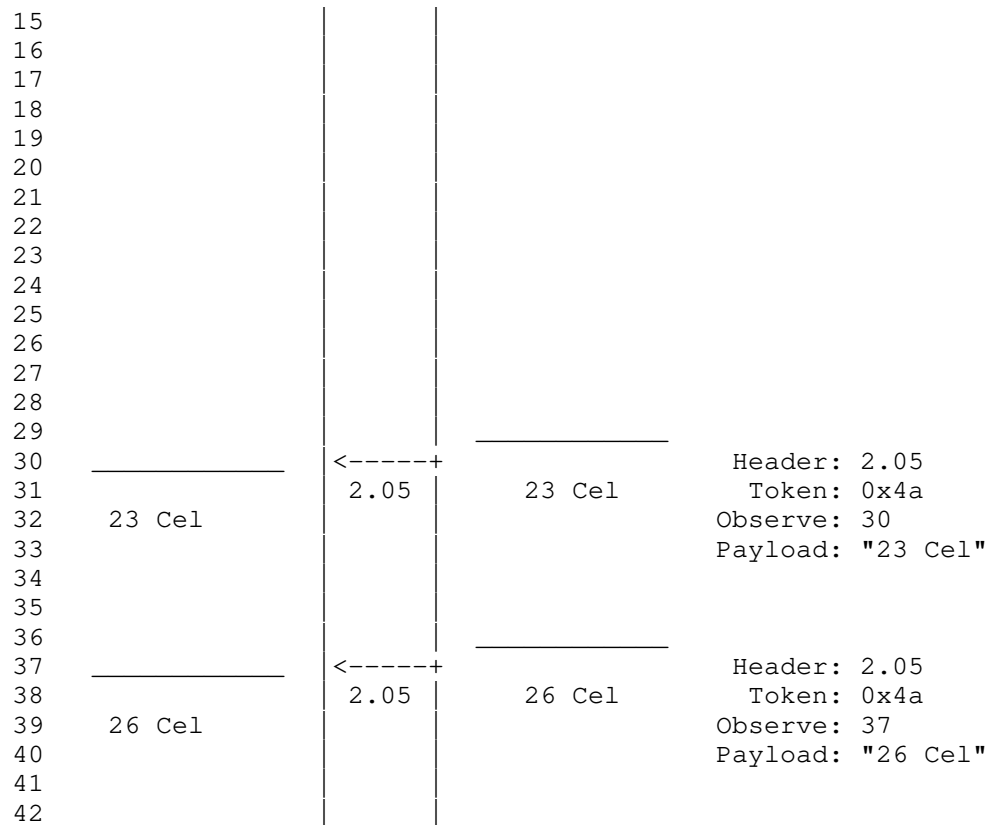


Figure 6: Client registers and receives one notification of the current state, one when pmax time expires and one of a new state when it passes through the greater than threshold of 25.

Authors' Addresses

Michael Koster
 SmartThings
 665 Clyde Avenue
 Mountain View 94043
 USA

Email: michael.koster@smarththings.com

Bilhanan Silverajan (editor)
Tampere University
Kalevantie 4
Tampere FI-33100
Finland

Email: bilhanan.silverajan@tuni.fi

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 13, 2022

M. Koster
SmartThings
B. Silverajan, Ed.
Tampere University
July 12, 2021

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-14

Abstract

This specification defines Link Bindings, which provide dynamic linking of state updates between resources, either on an endpoint or between endpoints, for systems using CoAP (RFC7252).

Editor note

The git repository for the draft is found at <https://github.com/core-wg/dynlink>

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 January 13, 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Link Bindings	3
3.1. The "bind" attribute and Binding Methods	4
3.1.1. Polling	5
3.1.2. Observe	5
3.1.3. Push	5
3.1.4. Execute	6
3.2. Link Relation	6
4. Binding Table	6
5. Implementation Considerations	8
6. Security Considerations	8
7. IANA Considerations	8
7.1. Resource Type value 'core.bnd'	8
7.2. Link Relation Type	8
8. Acknowledgements	9
9. Contributors	9
10. Changelog	10
11. References	12
11.1. Normative References	12
11.2. Informative References	13
Authors' Addresses	13

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol [RFC7252] and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format [RFC6690] is a standard for doing Web Linking [RFC8288] 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 state updates are conveyed. Specifically, a Link Binding is a unidirectional link for binding the states of source and destination 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.

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.

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC8288], [RFC6690] and [RFC7641]. 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 such an abstract relationship between two resources is defined, 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 such a binding is to synchronize content updates 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.

Conditional Notification Attributes defined in [I-D.ietf-core-conditional-attributes] can be used with Link Bindings in order to customize the notification behavior and timing.

3.1. The "bind" attribute and Binding Methods

A binding method defines the rules to generate the network-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.

This specification defines a new CoRE link attribute "bind". This is the identifier for a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

Attribute	Parameter	Value
Binding method	bind	xs:string

Table 1: The bind attribute

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
Execute	exec	Source	POST

Table 2: Binding Method Summary

The description of a binding method defines 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.

Conditional Notification: How Conditional Notification Attributes defined in [I-D.ietf-core-conditional-attributes] are used in the 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 defined in [I-D.ietf-core-conditional-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 parameters in the Observe request (see [I-D.ietf-core-conditional-attributes]).

3.1.3. Push

The Push method can be used to allow a source endpoint to replace an outdated resource state at the destination with a newer representation. When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource when the Conditional Notification Attributes are satisfied for the source resource. The source endpoint SHOULD only send a notification request if any included Conditional Notification Attributes are met.

The binding entry for this method **MUST** be stored on the source endpoint.

3.1.4. Execute

An alternative means for a source endpoint to deliver change-of-state notifications to a destination resource is to use the Execute Method. While the Push method simply updates the state of the destination resource with the representation of the source resource, Execute can be used when the destination endpoint wishes to receive all state changes from a source. This allows, for example, the existence of a resource collection consisting of all the state changes at the destination endpoint. When the Execute method is assigned to a binding, the source endpoint sends POST requests to the destination resource when the Conditional Notification Attributes are satisfied for the source resource. The source endpoint **SHOULD** only send a notification request if any included Conditional Notification Attributes are met. The binding entry for this method **MUST** be stored on the source endpoint.

Note: Both the Push and the Execute methods are examples of Server Push mechanisms that are being researched in the Thing-to-Thing Research Group (T2TRG) [I-D.irtf-t2trg-rest-iot].

3.2. Link Relation

Since Binding involves the creation of a link between two resources, Web Linking and the CoRE Link-Format used to represent binding information. This involves the creation of a new relation type, "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.

4. Binding Table

The Binding Table is a special resource that describes the bindings on an endpoint. An endpoint offering a representation of the Binding Table resource **SHOULD** indicate its presence and enable its discovery by advertising a link at `"/.well-known/core"` [RFC6690]. If so, the Binding Table resource **MUST** be discoverable by using the Resource Type (rt) `'core.bnd'`.

The Methods column defines the REST methods supported by the Binding Table, which are described in more detail below.

Resource	rt=	Methods	Content-Format
Binding Table	core.bnd	GET, PUT	link-format

Table 3: Binding Table Description

The REST methods GET and PUT are used to manipulate a Binding Table. A GET request simply returns the current state of a Binding Table. A request with a PUT method and a content format of application/link-format is used to clear the bindings to the table or replaces its entire contents. All links in the payload of a PUT request MUST have a relation type "boundto".

The following example shows requests for discovering, retrieving and replacing bindings in a binding table.

```
Req: GET /.well-known/core?rt=core.bnd (application/link-format)
Res: 2.05 Content (application/link-format)
</bnd/>;rt=core.bnd;ct=40
```

```
Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/a/switch1/>;
    rel=boundto;anchor=/a/fan;bind="obs",
<coap://sensor.example.com/a/switch2/>;
    rel=boundto;anchor=/a/light;bind="obs"
```

```
Req: PUT /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
```

Figure 1: Binding Table Example

Additional operations on the Binding Table can be specified in future documents. Such operations can include, for example, the usage of the iPATCH or PATCH methods [RFC8132] for fine-grained addition and removal of individual bindings or binding subsets.

5. Implementation Considerations

The initiation of a Link Binding can be delegated from a client to a link state machine implementation, which can be an embedded client or a configuration tool. Implementation considerations have to be given to how to monitor transactions made by the configuration tool with regards to Link Bindings, as well as any errors that may arise with establishing Link Bindings in addition to established Link Bindings.

6. Security Considerations

Consideration has to be given to what kinds of security credentials the state machine of a configuration tool or an embedded client needs to be configured with, and what kinds of access control lists client implementations should possess, so that transactions on creating Link Bindings and handling error conditions can be processed by the state machine.

7. IANA Considerations

7.1. Resource Type value 'core.bnd'

This specification registers a new Resource Type Link Target Attribute 'core.bnd' in the Resource Type (rt=) registry established as per [RFC6690].

Attribute Value: core.bnd

Description: See Section 4. This attribute value is used to discover the resource representing 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

7.2. Link Relation Type

This specification registers the new "boundto" link relation type as per [RFC8288].

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

8. 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. Christian Amsuss supplied a comprehensive review of draft -06. Discussions with Ari Keraenen led to the addition of an extra binding method supporting POST operations.

9. Contributors

Christian Groves
Australia
email: cngroves.std@gmail.com

Zach Shelby
ARM
Vuokatti
FINLAND
phone: +358 40 7796297
email: zach.shelby@arm.com

Matthieu Vial
Schneider-Electric
Grenoble
France
phone: +33 (0)47657 6522
eMail: matthieu.vial@schneider-electric.com

Jintao Zhu
Huawei
Xi'an, Shaanxi Province
China
email: jintao.zhu@huawei.com

10. Changelog

draft-ietf-core-dynlink-14

- o Conditional Attributes section removed and submitted as draft-ietf-core-conditional-attributes-00

draft-ietf-core-dynlink-13

- o Conditional Attributes section restructured
- o "edge" and "con" attributes added
- o Implementation considerations, clarifications added when pmax == pmin
- o rewritten to remove talk of server reporting values to clients

draft-ietf-core-dynlink-12

- o Attributes epmin and epmax included
- o pmax now can be equal to pmin

draft-ietf-core-dynlink-11

- o Updates to author list

draft-ietf-core-dynlink-10

- o Binding methods now support both POST and PUT operations for server push.

draft-ietf-core-dynlink-09

- o Corrections in Table 1, Table 2, Figure 2.
- o Clarifications for additional operations to binding table added in section 5
- o Additional examples in Appendix A

draft-ietf-core-dynlink-08

- o Reorganize the draft to introduce Conditional Notification Attributes at the beginning

- o Made pmin and pmax type xs:decimal to accommodate fractional second timing
- o updated the attribute descriptions. lt and gt notify on all crossings, both directions
- o updated Binding Table description, removed interface description but introduced core.bnd rt attribute value

draft-ietf-core-dynlink-07

- o Added reference code to illustrate attribute interactions for observations

draft-ietf-core-dynlink-06

- o Document restructure and refactoring into three main sections
- o Clarifications on band usage
- o Implementation considerations introduced
- o Additional text on security considerations

draft-ietf-core-dynlink-05

- o Addition of a band modifier for gt and lt, adapted from draft-groves-core-obsattr
- o Removed statement prescribing gt MUST be greater than lt

draft-ietf-core-dynlink-03

- o 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".
- o New author and editor added.

draft-ietf-core-dynlink-02

- o 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 conflict with the core resource directory draft lifetime "lt" attribute.
- o Clause 6.1: Addressed the editor's note by changing the link target attribute to "core.binding".

- o Added Appendix A for examples.

draft-ietf-core-dynlink-01

- o General: The term state synchronization has been introduced to describe the process of synchronization between destination and source resources.
- o General: The document has been restructured to make the information flow better.
- o Clause 3.1: The descriptions of the binding attributes have been updated to clarify their usage.
- o Clause 3.1: A new clause has been added to discuss the interactions between the resources.
- o Clause 3.4: Has been simplified to refer to the descriptions in 3.1. As the text was largely duplicated.
- o Clause 4.1: Added a clarification that individual resources may be removed from the binding table.
- o Clause 6: Formalised the IANA considerations.

draft-ietf-core-dynlink Initial Version 00:

- o This is a copy of draft-groves-core-dynlink-00

draft-groves-core-dynlink Draft Initial Version 00:

- o This initial version is based on the text regarding the dynamic linking functionality in I.D.ietf-core-interfaces-05.
- o The WADL description has been dropped in favour of a thorough textual description of the REST API.

11. References

11.1. 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>>.
- [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>>.

11.2. Informative References

- [I-D.ietf-core-conditional-attributes]
Koster, M. and B. Silverajan, "Conditional Attributes for Constrained RESTful Environments", draft-ietf-core-conditional-attributes-00 (work in progress), July 2021.
- [I-D.irtf-t2trg-rest-iot]
Keranen, A., Kovatsch, M., and K. Hartke, "RESTful Design for Internet of Things Systems", draft-irtf-t2trg-rest-iot-07 (work in progress), February 2021.
- [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>>.
- [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>>.

Authors' Addresses

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smarththings.com

Bilhanan Silverajan (editor)
Tampere University
Kalevantie 4
Tampere FI-33100
Finland

Email: bilhanan.silverajan@tuni.fi

CoRE Working Group
Internet-Draft
Intended status: Experimental
Expires: April 22, 2021

I. Jarvinen
M. Kojo
I. Raitahila
University of Helsinki
Z. Cao
Huawei
October 19, 2020

Fast-Slow Retransmission Timeout and Congestion Control Algorithm for
CoAP
draft-ietf-core-fasor-01

Abstract

This document specifies an alternative retransmission timeout and congestion control back off algorithm for the CoAP protocol, called Fast-Slow RTO (FASOR).

The algorithm specified in this document employs an appropriate and large enough back off of Retransmission Timeout (RTO) as the major congestion control mechanism to allow acquiring unambiguous RTT samples with high probability and to prevent building a persistent queue when retransmitting. The algorithm also aims to retransmit quickly using an accurately managed retransmission timeout when link-errors are occurring, basing RTO calculation on unambiguous round-trip time (RTT) samples.

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 April 22, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Conventions	3
3. Problems with Existing CoAP Congestion Control Algorithms . .	3
4. FASOR Algorithm	4
4.1. Computing Normal RTO (FastRTO)	4
4.2. Slow RTO	5
4.3. Retransmission Timeout Back Off Logic	6
4.3.1. Overview	6
4.3.2. Retransmission State Machine	7
4.4. Retransmission Count Option	9
4.5. Alternatives for Exchanging Retransmission Count Information	11
5. Security Considerations	11
6. IANA Considerations	11
7. References	11
7.1. Normative References	11
7.2. Informative References	12
Appendix A. Pseudocode for Basic FASOR without Dithering	13
Authors' Addresses	15

1. Introduction

CoAP senders use retransmission timeout (RTO) to infer losses that have occurred in the network. For such a heuristic to be correct, the RTT estimate used for calculating the retransmission timeout must match to the real end-to-end path characteristics. Otherwise, unnecessary retransmission may occur. Both default RTO mechanism for CoAP [RFC7252] and CoCoA [I-D.ietf-core-cocoa] have issues in dealing with unnecessary retransmissions and in the worst-case the situation can persist causing congestion collapse [JRCK18a].

This document specifies FASOR retransmission timeout and congestion control algorithm [JRCK18b]. FASOR algorithm ensures unnecessary retransmissions that a sender may have sent due to an inaccurate RTT estimate will not persist avoiding the threat of congestion collapse. FASOR also aims to quickly restore the accuracy of the RTT estimate. Armed with an accurate RTT estimate, FASOR not only handles congestion robustly but also can quickly infer losses due to link errors.

2. Conventions

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

3. Problems with Existing CoAP Congestion Control Algorithms

Correctly inferring losses requires the retransmission timeout (RTO) to be longer than the real RTT in the network. Under certain circumstances the RTO may be incorrectly small. If the real end-to-end RTT is larger than the retransmission timeout, it is impossible for the sender to avoid making unnecessary retransmissions that duplicate data still existing in the network because the sender cannot receive any feedback in time. Unnecessary retransmissions cause two basic problems. First, they increase the perceived end-to-end RTT if the bottleneck has buffering capacity, and second, they prevent getting unambiguous RTT samples. Making unnecessary retransmissions is also a pre-condition for the congestion collapse [RFC0896], which may occur in the worst case if retransmissions are not well controlled [JRCK18a]. Therefore, the sender retransmission timeout algorithm should actively attempt to prevent unnecessary retransmissions from persisting under any circumstance.

Karn's algorithm [KP87] has prevented unnecessary retransmission from turning into congestion collapse for decades due to robust RTT estimation and retransmission timeout backoff handling. The recent CoAP congestion control algorithms, however, diverge from the principles of Karn's algorithm in significant ways and may pose a threat to the stability of the Internet due to those differences.

The default RTO mechanism for CoAP [RFC7252] uses only an initial RTO dithered between 2 and 3 seconds, while CoCoA [I-D.ietf-core-cocoa] measures RTT both from unambiguous and ambiguous RTT samples and applies a modified version of the TCP RTO algorithm [RFC6298]. The algorithm in RFC 7252 lacks solution to persistent congestion. The binary exponential back off used for the retransmission timeout does not properly address unnecessary retransmissions when RTT is larger

than the default RTO (ACK_TIMEOUT). If the CoAP sender performs exchanges over an end-to-end path with such a high RTT, it persistently keeps making unnecessary retransmissions for every exchange wasting some fraction of the used resources (network capacity, battery power).

CoCoA [I-D.ietf-core-cocoa] attempts to improve scenarios with link-error related losses and solve persistent congestion by basing its RTO value on an estimated RTT. However, there are couple of exceptions when the RTT estimation is not available:

- At the beginning of a flow where initial RTO of 2 seconds is used.
- When RTT suddenly jumps high enough to trigger the rule in CoCoA that prevents taking RTT samples when more than two retransmissions are needed. This may also occur when the packet drop rate on the path is high enough.

When RTT estimate is too small, unnecessary retransmission will occur also with CoCoA. CoCoA being unable to take RTT samples at all is a particularly problematic phenomenon as it is similarly persisting state as with the algorithm outlined in RFC 7252 and the network remains in a congestion collapsed state due to persisting unnecessary retransmissions.

4. FASOR Algorithm

FASOR [JRCK18b] is composed of three key components: RTO computation, Slow RTO, and novel retransmission timeout back off logic.

4.1. Computing Normal RTO (FastRTO)

The FASOR algorithm measures the RTT for an CoAP message exchange over an end-to-end path and computes the RTO value using the TCP RTO algorithm specified in [RFC6298]. We call this normal RTO or FastRTO. In contrast to the TCP RTO mechanism, FASOR SHOULD NOT use 1 second lower-bound when setting the RTO because RTO is only a backup mechanisms for loss detection with TCP, whereas with CoAP RTO is the primary and only loss detection mechanism. A lower-bound of 1 second would impact timeliness of the loss detection in low RTT environments. The RTO value MAY be upper-bounded by at least 60 seconds. A CoAP sender using the FASOR algorithm SHOULD set initial RTO to 2 seconds. The computed RTO value as well as the initial RTO value is subject to dithering; they are dithered between $RTO + 1/4 \times SRTT$ and $RTO + SRTT$. For dithering initial RTO, SRTT is unset; therefore, SRTT is replaced with initial RTO / 3 which is derived from the RTO formula and equals to a hypothetical initial RTT that

would yield the initial RTO using the SRTT and RTTVAR initialization rule of RFC 6298. That is, for initial RTO of 2 seconds we use SRTT value of 2/3 seconds.

FastRTO is updated only with unambiguous RTT samples. Therefore, it closely tracks the actual RTT of the network and can quickly trigger a retransmission when the network state is not dubious. Retransmitting without extra delay is very useful when the end-to-end path is subject to losses that are unrelated to congestion. When the first unambiguous RTT sample is received, the RTT estimator is initialized with that sample as specified in [RFC6298] except RTTVAR that is set to $R/2K$.

4.2. Slow RTO

We introduce Slow RTO as a safe way to ensure that only a unique copy of message is sent before at least one RTT has elapsed. To achieve this the sender must ensure that its retransmission timeout is set to a value that is larger than the path end-to-end RTT that may be inflated by the unnecessary retransmission themselves. Therefore, whenever a message needs to be retransmitted, we measure Slow RTO as the elapsed time required for getting an acknowledgement. That is, Slow RTO is measured starting from the original transmission of the request message until the receipt of the acknowledgement, regardless of the number of retransmissions. In this way, Slow RTO always covers the worst-case RTT during which a number of unnecessary retransmissions were made but the acknowledgement is received for the original transmission. In contrast to computing normal RTO, Slow RTO is not smoothed because it is derived from the sending pattern of the retransmissions (that may turn out unnecessary). In order to drain the potential unnecessary retransmissions successfully from the network, it makes sense to wait for the time used for sending them rather than some smoothed value. However, Slow RTO is multiplied by a factor to allow some growth in load without making Slow RTO too aggressive (by default the factor of 1.5 is used). FASOR then applies Slow RTO as one of the backed off timer values used with the next request message.

Slow RTO allows rapidly converging towards stable operating point because 1) it lets the duplicate copies sent earlier to drain from the network reducing the perceived end-to-end RTT, and 2) allows enough time to acquire an unambiguous RTT sample for the RTO computation. Robustly acquiring the RTT sample ensures that the next RTO is set according to the recent measurement and further unnecessary retransmissions are avoided. Slow RTO itself is a form of back off because it includes the accumulated time from the retransmission timeout back off of the previous exchange. FASOR uses this for its advantage as the time included into Slow RTO is what is

needed to drain all unnecessary retransmissions possibly made during the previous exchange. Assuming a stable RTT and that all of the retransmissions were unnecessary, the time to drain them is the time elapsed from the original transmission to the sending time of the last retransmission plus one RTT. When the acknowledgement for the original transmission arrives, one RTT has already elapsed, leaving only the sending time difference still unaccounted for which is at minimum the value for Slow RTO (when an RTT sample arrives immediately after the last retransmission). Even if RTT would be increasing, the draining still occurs rapidly due to exponentially backed off frequency in sending the unnecessary retransmissions.

4.3. Retransmission Timeout Back Off Logic

4.3.1. Overview

FASOR uses normal RTO as the base for binary exponential back off when no retransmission were needed for the previous CoAP message exchange. When retransmission were needed for the previous CoAP message exchange, the algorithm rules, however, are more complicated than with the traditional RTO back off because Slow RTO is injected into the back off series to reduce high impact of using Slow RTO. FASOR logic chooses from three possible back off series alternatives:

FAST back off: Perform traditional RTO back off with the normal RTO as the base. Applied when the previous message was not retransmitted.

FAST_SLOW_FAST back off: First perform a probe using the normal RTO for the original transmission of the request message to improve cases with losses unrelated to congestion. If the probe for the original transmission of the request message is successful without retransmissions, continue with FAST back off for the next message exchange. If the request message needs to be retransmitted, continue by using Slow RTO for the first retransmission in order to respond to congestion and drain the network from the unnecessary retransmissions that were potentially sent for the previous exchange. If still further RTOs are needed, continue by backing off the normal RTO further on each timeout. FAST_SLOW_FAST back off is applied just once when the previous request message using FAST back off required one or more retransmissions.

SLOW_FAST back off: Perform Slow RTO first for the original transmisssion to respond to congestion and to acquire an unambiguous RTT sample with high probability. Then, if the original request needs to be retransmitted, continue with the normal RTO-based RTO back off serie by backing off the normal RTO

on each timeout. SLOW_FAST back off is applied when the previous request message using FAST_SLOW_FAST or SLOW_FAST back off required one or more retransmissions. Once an acknowledgement for the original transmission with unambiguous RTT sample is received, continue with FAST back off for the next message exchange.

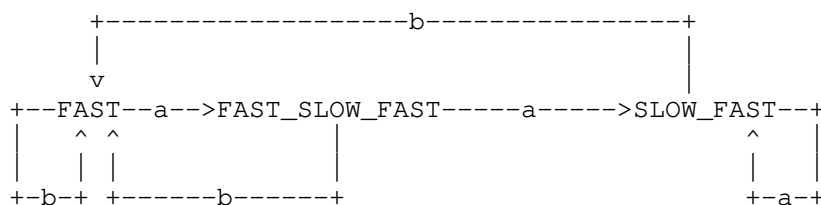
For the initial message, FAST is used with INITIAL_RTO as the FastRTO value. From there on, state is updated when an acknowledgement arrives. Following unambiguous RTT samples, FASOR always uses FAST. Whenever retransmissions are needed, the back off series selection is first downgraded to FAST_SLOW_FAST back off and then to SLOW_FAST back off if further retransmission are needed in FAST_SLOW_FAST.

When Slow RTO is used as the first RTO value, the sender is likely to acquire unambiguous RTT sample even when the network has high delay due to congestion because Slow RTO is based on a very recent measurement of the worst-case RTT. However, using Slow RTO may negatively impact the performance when losses unrelated to congestion are occurring. Due to its potential high cost, FASOR algorithm attempts to avoid using Slow RTO unnecessarily.

The CoAP protocol is often used by devices that are connected through a wireless network where non-congestion related losses are much more frequent than in their wired counterparts. This has implications for the retransmission timeout algorithm. While it would be possible to implement FASOR such that it immediately uses Slow RTO when a dubious network state is detected, which would handle congestion very well, it would do significant harm for performance when RTOs occur due to non-congestion related losses. Instead, FASOR uses first normal RTO for one transmission and only responds using Slow RTO if RTO expires also for that request message. Such a pattern quickly probes if the losses were unrelated to congestion and only slightly delays response if real congestion event is taking place. To ensure that an unambiguous RTT sample is also acquired on a congested network path, FASOR then needs to use Slow RTO for the original transmission of the subsequent packet if the probe was not successful.

4.3.2. Retransmission State Machine

FASOR consists of the three states discussed above while making retransmission decisions, FAST, FAST_SLOW_FAST and SLOW_FAST. The state machine of the FASOR algorithm is depicted in Figure 1.



a: retransmission acknowledged, ambiguous RTT sample acquired;
b: no retransmission, unambiguous RTT sample acquired;

Figure 1: State Machine of FASOR

In the FAST state, if the original transmission of the message has not been acknowledged by the receiver within the time defined by FastRTO, the sender will retransmit it. If there is still no acknowledgement of the retransmitted packet within $2 \times \text{FastRTO}$, the sender performs the second retransmission and if necessary, each further retransmission applying binary exponential back off of FastRTO. The retransmission interval in this state is defined as FastRTO, $2^1 \times \text{FastRTO}$, ..., $2^i \times \text{FastRTO}$.

When there is an acknowledgement after any retransmission, the sender will calculate SlowRTO value based on the algorithm defined in Section 4.2.

When there is an acknowledgement after any retransmission, the sender will also switch to the second state, FAST_FLOW_FAST. In this state, the retransmission interval is defined as FastRTO, $\text{Max}(\text{SlowRTO}, 2 \times \text{FastRTO})$, $\text{FastRTO} \times 2^1$, ..., $2^i \times \text{FastRTO}$. The state will be switched back to the FAST state once an acknowledgement is returned within FastRTO, i.e., no retransmission happens for a message. This is reasonable because it shows the network has recovered from congestion or bloated queue.

If some retransmission has been made before the acknowledged arrives in the FAST_SLOW_FAST state, the sender updates the SlowRTO value, and moves to the third state, SLOW_FAST. The retransmission interval in the SLOW_FAST state is defined as SlowRTO, FastRTO, $\text{FastRTO} \times 2^1$, ..., $2^i \times \text{FastRTO}$.

In SLOW_FAST state, the sender switches back to the FAST state if an unambiguous acknowledgement arrives. Otherwise, the sender stays in the SLOW_FAST state if retransmission happens again.

4.4. Retransmission Count Option

When retransmissions are needed to deliver a CoAP message, it is not possible to measure RTT for the RTO computation as the RTT sample becomes ambiguous. Therefore, it would be beneficial to be able to distinguish whether an acknowledgement arrives for the original transmission of the message or for a retransmission of it. This would allow reliably acquiring an RTT sample for every CoAP message exchange and thereby compute a more accurate RTO even during periods of congestion and loss.

The Retransmission Count Option is used to distinguish whether an Acknowledgement message arrives for the original transmission or one of the retransmissions of a Confirmable message. However, the Retransmission Count Option cannot be used with an Empty Acknowledgement (or Reset) message because the CoAP protocol specification [RFC7252] does not allow adding options to an Empty message. Therefore, Retransmission Count Option is useful only for the common case of Piggybacked Response. In case of Empty Acknowledgements the operation of FASOR is the same as without the option. This restriction with Empty Acknowledgements may limit the usefulness of the Retransmission Count Option in deployment scenarios where the receiver is a proxy that will typically respond with an Empty Acknowledgement when it receives a request message.

No.	C	U	N	R	Name	Format	Length	Default
TBD			X		Rexmit-Cnt	uint	0-1	0

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

Table 1: Retransmission Count Option

Implementation of the Retransmission Count option is optional and it is identified as elective. However, when it is present in a CoAP message and a CoAP endpoint processes it, it MUST be processed as described in this document. The Retransmission Count option MUST NOT occur more than once in a single message.

The value of the Retransmission Count option is a variable-size (0 to 1 byte) unsigned integer. The default value for the option is the number 0 and it is represented with an empty option value (a zero-length sequence of bytes). However, when a client intends to use Retransmit Count option, it MUST reserve space for it by limiting the request message size also when the value is empty in order to fit the full-sized option into retransmissions.

The Retransmission Count option can be present in both the request and response message. When the option is present in a request it indicates the ordinal number of the transmission for the request message.

If the server supports (implements) the Retransmission Count option and the option is present in a request, the server **MUST** echo the option value in its Piggybacked Response unmodified. If the server replies with an Empty Acknowledgement the server **MUST** silently ignore the option and **MUST NOT** include it in a later separate response to that request.

When Piggybacked Response carrying the Retransmission Count option arrives, the client uses the option to match the response message to the corresponding transmission of the request. In order to measure a correct RTT, the client must store the timestamp for the original transmission of the request as well as the timestamp for each retransmission, if any, of the request. The resulting RTT sample is used for the RTO computation. If the client retransmitted the request without the option but the response includes the option, the client **MUST** silently ignore the option.

The original transmission of a request is indicated with the number 0, except when sending the first request to a new destination endpoint (i.e., an endpoint not already in the memory). The first original transmission of the request to a new endpoint carries the number 255 (0xFF) and is interpreted the same as an original transmission carrying the number 0. Once the first Piggybacked Response from the new endpoint arrives the client learns whether or not the other endpoint implements the option. If the first response includes the echoed option, the client learns that the other endpoint supports the option and may continue including the option to each retransmitted request. From this point on the original transmissions of requests implicitly include the option number 0 and a zero-byte integer will be sent according to the CoAP uint-encoding rules. If the first Piggybacked Response does not include the option, the client **SHOULD** stop including the option into the requests to that endpoint. Retransmissions, if any, carry the ordinal number of the retransmission. That is, the client increments the retransmission count by one for each retransmission of the message.

When the Retransmission Count option is in use, the client bases the retransmission timeout for the normal RTO in the back off series as follows:

$$\max(\text{RTO}, \text{Previous-RTT-Sample})$$

Previous-RTT-Sample is the RTT sample acquired from the previous message exchange. If no RTT sample was available with the previous message exchange (e.g., the server replied with an Empty Acknowledgement), RTO computed earlier is used like in case the Retransmission Count option is not in use.

4.5. Alternatives for Exchanging Retransmission Count Information

An alternative way of exchanging the retransmission count information between a client and server is to encode it in the Token. The Token is a client-local identifier and a client solely decides how it generates the Token. Therefore, including a varying Token value to retransmissions of the same request is all possible as long as the client can use the Token to differentiate between requests and match a response to the corresponding request. The server is required to make no assumptions about the content or structure of a Token and always echo the Token unmodified in its response.

How exactly a client encodes the retransmission count into a Token is an implementation issue. Note that the original transmission of a request may carry a zero-length Token given that the rules for generating a Token as specified in RFC 7252 [RFC7252] are followed. This allows reducing the overhead of including the Token into the requests in such cases where Token could otherwise be omitted. However, similar to Retransmit Count option the maximum request message size MUST be limited to accommodate the Token with retransmit count into the retransmissions of the request.

5. Security Considerations

6. IANA Considerations

This memo includes no request to IANA.

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>>.
- [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>>.

7.2. Informative References

- [I-D.ietf-core-cocoa] Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-ietf-core-cocoa-03 (work in progress), February 2018.
- [JRCK18a] Jarvinen, I., Raitahila, I., Cao, Z., and M. Kojo, "Is CoAP Congestion Safe?", Applied Networking Research Workshop (ANRW'18), July 2018.
- [JRCK18b] Jarvinen, I., Raitahila, I., Cao, Z., and M. Kojo, "FASOR Retransmission Timeout and Congestion Control Mechanism for CoAP", Proceedings of IEEE Global Communications Conference (Globecom 2018), December 2018.
- [KP87] Karn, P. and C. Partridge, "Improving Round-trip Time Estimates in Reliable Transport Protocols", SIGCOMM'87 Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology, August 1987.
- [RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <<https://www.rfc-editor.org/info/rfc896>>.

Appendix A. Pseudocode for Basic FASOR without Dithering

```
var state = NORMAL_RTO

rfc6298_init(var fastrto, 2 secs)

var slowrto
SLOWRTO_FACTOR = 1.5

var original_sendtime
var retransmit_count

/*
 * Sending Original Copy and Retransmitting 'req'
 */
send_request(req) {
    original_sendtime = time.now
    retransmit_count = 0

    arm_rto(calculate_rto())
    send(req)
}

rto_for(req) {
    retransmit_count += 1

    arm_rto(calculate_rto())
    send(req)
}

/*
 * ACK Processings
 */
ack() {
    sample = time.now - original_sendtime
    if (retransmit_count == 0)
        unambiguous_ack(sample)
    else
        ambiguous_ack(sample)
}

unambiguous_ack(sample) {
    k = 4 // RFC6298 default K = 4
    if (rfc6298_is_first_sample(fastrto))
        k = 1
    rfc6298_update(fastrto, k, sample) // Normal RFC6298 processing
    state = NORMAL_RTO
}
```

```
ambiguous_nextstate = {
    [NORMAL_RTO] = FAST_SLOW_FAST_RTO,
    [FAST_SLOW_FAST_RTO] = SLOW_FAST_RTO,
    [SLOW_FAST_RTO] = SLOW_FAST_RTO
}

ambiguous_ack(sample) {
    slowrto = sample * SLOWRTO_FACTOR
    state = ambiguous_nextstate[state]
}

/*
 * RTO Calculations
 */
calculate_rto() {
    return <state>_rtoseries()
}

normal_rtoseries() {
    switch (retransmit_count) {
        case 0: return fastrto_series_init()
        default: return fastrto_series_backoff()
    }
}

fastslowfast_rtoseries() {
    switch (retransmit_count) {
        case 0: return fastrto_series_init()
        case 1: return MAX(slowrto, 2*fastrto)
        default: return fastrto_series_backoff()
    }
}

slowfast_rtoseries() {
    switch (retransmit_count) {
        case 0: return slowrto
        case 1: return fastrto_series_init()
        default: return fastrto_series_backoff()
    }
}

var backoff_series_timer

fastrto_series_init() {
    backoff_series_timer = fastrto
    return backoff_series_timer
}
```

```
fastrto_series_backoff() {  
    backoff_series_timer *= 2  
    return backoff_series_timer  
}
```

Figure 2

Authors' Addresses

Ilpo Jarvinen
University of Helsinki
P.O. Box 68
FI-00014 UNIVERSITY OF HELSINKI
Finland

EMail: ilpo.jarvinen@cs.helsinki.fi

Markku Kojo
University of Helsinki
P.O. Box 68
FI-00014 UNIVERSITY OF HELSINKI
Finland

EMail: markku.kojo@cs.helsinki.fi

Iivo Raitahila
University of Helsinki
Helsinki
Finland

EMail: iivo.raitahila@alumni.helsinki.fi

Zhen Cao
Huawei
Beijing
China

EMail: zhencao.ietf@gmail.com

CoRE Working Group
Internet-Draft
Obsoletes: 7390 (if approved)
Updates: 7252, 7641 (if approved)
Intended status: Standards Track
Expires: May 6, 2021

E. Dijk
IoTconsultancy.nl
C. Wang
InterDigital
M. Tiloca
RISE AB
November 02, 2020

Group Communication for the Constrained Application Protocol (CoAP)
draft-ietf-core-groupcomm-bis-02

Abstract

This document specifies the use of the Constrained Application Protocol (CoAP) for group communication, using UDP/IP multicast as the underlying data transport. Both unsecured and secured CoAP group communication are specified. Security is achieved by use of the Group Object Security for Constrained RESTful Environments (Group OSCORE) protocol. The target application area of this specification is any group communication use cases that involve resource-constrained networks. The most common of such use cases are also discussed. This document replaces [RFC7390] and updates [RFC7252] and [RFC7641].

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <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 May 6, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Scope	4
1.2. Terminology	4
2. General Group Communication Operation	5
2.1. Group Definition	5
2.1.1. CoAP Group	5
2.1.2. Application Group	6
2.1.3. Security Group	6
2.1.4. Relations Between Group Types	6
2.2. Group Configuration	9
2.2.1. Group Naming	9
2.2.2. Group Creation and Membership	10
2.2.3. Group Discovery	11
2.2.4. Group Maintenance	12
2.3. CoAP Usage	12
2.3.1. Request/Response Model	13
2.3.2. Port and URI Path Selection	16
2.3.3. Proxy Operation	17
2.3.4. Congestion Control	18
2.3.5. Observing Resources	19
2.3.6. Block-Wise Transfer	22
2.4. Transport	22
2.4.1. UDP/IPv6 Multicast Transport	22
2.4.2. UDP/IPv4 Multicast Transport	23
2.4.3. 6LoWPAN	23
2.5. Interworking with Other Protocols	23
2.5.1. MLD/MLDv2/IGMP/IGMPv3	23
2.5.2. RPL	24
2.5.3. MPL	25
3. Unsecured Group Communication	25
4. Secured Group Communication using Group OSCORE	26
4.1. Secure Group Maintenance	27
5. Security Considerations	28
5.1. CoAP NoSec Mode	28
5.2. Group OSCORE	29
5.2.1. Group Key Management	29

5.2.2.	Source Authentication	30
5.2.3.	Countering Attacks	30
5.3.	Replay of Non Confirmable Messages	32
5.4.	Use of CoAP No-Response Option	32
5.5.	6LoWPAN	33
5.6.	Wi-Fi	33
5.7.	Monitoring	34
5.7.1.	General Monitoring	34
5.7.2.	Pervasive Monitoring	34
6.	IANA Considerations	35
7.	References	35
7.1.	Normative References	35
7.2.	Informative References	37
Appendix A.	Use Cases	39
A.1.	Discovery	39
A.1.1.	Distributed Device Discovery	40
A.1.2.	Distributed Service Discovery	40
A.1.3.	Directory Discovery	40
A.2.	Operational Phase	41
A.2.1.	Actuator Group Control	41
A.2.2.	Device Group Status Request	41
A.2.3.	Network-wide Query	41
A.2.4.	Network-wide / Group Notification	42
A.3.	Software Update	42
Appendix B.	Document Updates	42
B.1.	Version -01 to -02	42
B.2.	Version -00 to -01	43
Acknowledgments	43
Authors' Addresses	43

1. Introduction

This document specifies group communication using the Constrained Application Protocol (CoAP) [RFC7252] together with UDP/IP multicast. CoAP is a RESTful communication protocol that is used in resource-constrained nodes, and in resource-constrained networks where packet sizes should be small. This area of use is summarized as Constrained RESTful Environments (CoRE).

One-to-many group communication can be achieved in CoAP, by a client using UDP/IP multicast data transport to send multicast CoAP request messages. In response, each server in the addressed group sends a response message back to the client over UDP/IP unicast. Notable CoAP implementations supporting group communication include the framework "Eclipse Californium" 2.0.x [Californium] from the Eclipse Foundation and the "Implementation of CoAP Server & Client in Go" [Go-OCF] from the Open Connectivity Foundation (OCF).

Both unsecured and secured CoAP group communication over UDP/IP multicast are specified in this document. Security is achieved by using Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm], which in turn builds on Object Security for Constrained Restful Environments (OSCORE) [RFC8613]. This method provides end-to-end application-layer security protection of CoAP messages, by using CBOR Object Signing and Encryption (COSE) [I-D.ietf-cbor-7049bis][I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs].

All guidelines in [RFC7390] are updated by this document, which replaces and obsoletes [RFC7390]. Furthermore, this document updates [RFC7252], by specifying a group request/response model and by adding security for CoAP group communication. Finally, this document also updates [RFC7641], by adding the multicast usage of CoAP Observe for both the GET and FETCH methods.

All sections in the body of this document are normative, while appendices are informative. For additional background about use cases for CoAP group communication in resource-constrained devices and networks, see Appendix A.

1.1. Scope

For group communication, only solutions that use CoAP over UDP/IP multicast are in the scope of this document. There are alternative methods to achieve group communication using CoAP, for example Publish-Subscribe [I-D.ietf-core-coap-pubsub] which uses a central broker server that CoAP clients access via unicast communication. These methods may be usable for the same or similar use cases as are targeted in this document.

Furthermore, this document defines Group OSCORE [I-D.ietf-core-oscore-groupcomm] as the default group communication security solution for CoAP. Security solutions for group communication and configuration other than Group OSCORE are not in scope. General principles for secure group configuration are in scope.

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.

This specification requires readers to be familiar with CoAP terminology [RFC7252]. Terminology related to group communication is defined in Section 2.1.

Furthermore, "Security material" refers to any security keys, counters or parameters stored in a device that are required to participate in secure group communication with other devices.

2. General Group Communication Operation

The general operation of group communication, both unsecured and secured, is specified in this section. First, different group types are defined in Section 2.1. Group configuration, including group creation and maintenance by an application, user or commissioning entity is considered next in Section 2.2. Then the use of CoAP for group communication including support for protocol extensions (block-wise transfer, Observe) follows in Section 2.3. How CoAP group messages are carried over various transport layers is the subject of Section 2.4. Finally, Section 2.5 covers the interworking of CoAP group communication with other protocols that may operate in the same network.

2.1. Group Definition

Three types of groups and their mutual relations are defined in this section: CoAP group, application group, and security group.

2.1.1. CoAP Group

A CoAP group is defined as a set of CoAP endpoints, where each endpoint is configured to receive CoAP multicast messages that are sent to the group's associated IP multicast address and UDP port. An endpoint may be a member of multiple CoAP groups by subscribing to multiple IP multicast groups and/or listening on multiple UDP ports. Group membership(s) of an endpoint may dynamically change over time. A device sending a CoAP multicast message to a CoAP group is not necessarily itself a member of this CoAP group: it is a member only if it also has a CoAP endpoint listening on the group's associated IP multicast address and UDP port. A CoAP group can be encoded within a Group URI. This is defined as a CoAP URI that has the "coap" scheme and includes in the authority part either an IP multicast address or a group hostname (e.g., a Group Fully Qualified Domain Name (FQDN)) that can be resolved to an IP multicast address. A Group URI also contains an optional UDP port number in the authority part. Group URIs follow the regular CoAP URI syntax (see Section 6 of [RFC7252]).

2.1.2. Application Group

Besides CoAP groups, that have relevance at the level of IP networks and CoAP endpoints, there are also application groups. An application group is a set of CoAP server endpoints that share a common set of CoAP resources. An endpoint may be a member of multiple application groups. An application group has relevance at the application level - for example an application group could denote all lights in an office room or all sensors in a hallway. A client endpoint that sends a group communication message to an application group is not necessarily itself a member of this application group. There can be a one-to-one or a one-to-many relation between a CoAP group and application group(s). An application group identifier is optionally encoded explicitly in the CoAP request. If not explicitly encoded, the application group is implicitly derived by the receiver, based on information in the CoAP request. See Section 2.2.1 for more details on identifying the application group.

2.1.3. Security Group

For secure group communication, a security group is required. A security group is a group of endpoints that each store group security material, such that they can mutually exchange secured messages and verify secured messages. So, a client endpoint needs to be a member of a security group in order to send a valid secured group communication message to this group. An endpoint may be a member of multiple security groups. There can be a one-to-one or a one-to-many relation between security groups and CoAP groups. Also, there can be a one-to-one or a one-to-many relation between security groups and application groups. A special security group named "NoSec" identifies group communication without any security at the transport layer and/or application layer.

2.1.4. Relations Between Group Types

Using the above group type definitions, a CoAP group communication message sent by an endpoint can be represented as a tuple that contains one instance of each group type:

(application group, CoAP group, security group)

A special note is appropriate about the possible relation between security groups and application groups.

On one hand, multiple application groups may use the same security group. Thus, the same group security material is used to protect the messages targeting any of those application groups. In this case, a CoAP endpoint is supposed to know the exact application group to

refer to for each message, based on, e.g., the used server port number, the targeted resource, or the content and structure of the message payload.

On the other hand, a single application group may use multiple security groups. Thus, different messages targeting the resources of the application group can be protected with different security material. This can be convenient, for example, if the security groups differ with respect to the cryptographic algorithms and related parameters they use. In this case, a CoAP client can join just one of the security groups, based on what it supports and prefers, while a CoAP server in the application group would rather have to join all of them.

Beyond this particular case, applications should be greatly careful in associating a same application group to multiple security groups. In particular, it is NOT RECOMMENDED using different security groups to reflect different access policies for resources in a same application group. That is, being a member of a security group actually grants access only to exchanged secure messages, while access to resources in the application group belongs to a separate security domain, and has to be separately enforced by leveraging the resource properties or through dedicated access control credentials assessed by separate means.

Figure 1 summarizes the relations between the different types of groups described above in UML class diagram notation. The items in square brackets are optionally defined.

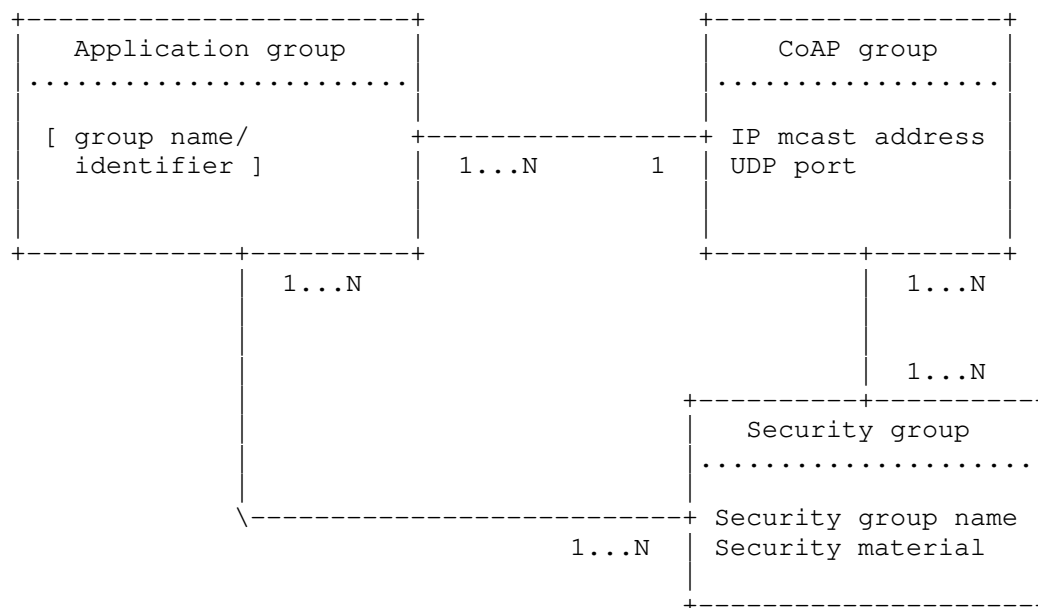


Figure 1: Relation Among Different Group Types

Figure 2 provides a deployment example of the relations between the different types of groups. It shows six CoAP servers (Srv1-Srv6) and their respective resources hosted (/resX). There are three application groups (1, 2, 3) and two security groups (1, 2). Security Group 1 is used by both Application Group 1 and 2. Three clients (Cli1, Cli2, Cli3) are configured with security material for Security Group 1. One client (Cli4) is configured with security material for Security Group 2. All the shown application groups use the same CoAP group (not shown in the figure), i.e. one specific multicast IP address and UDP port on which all the shown resources are hosted for each server.

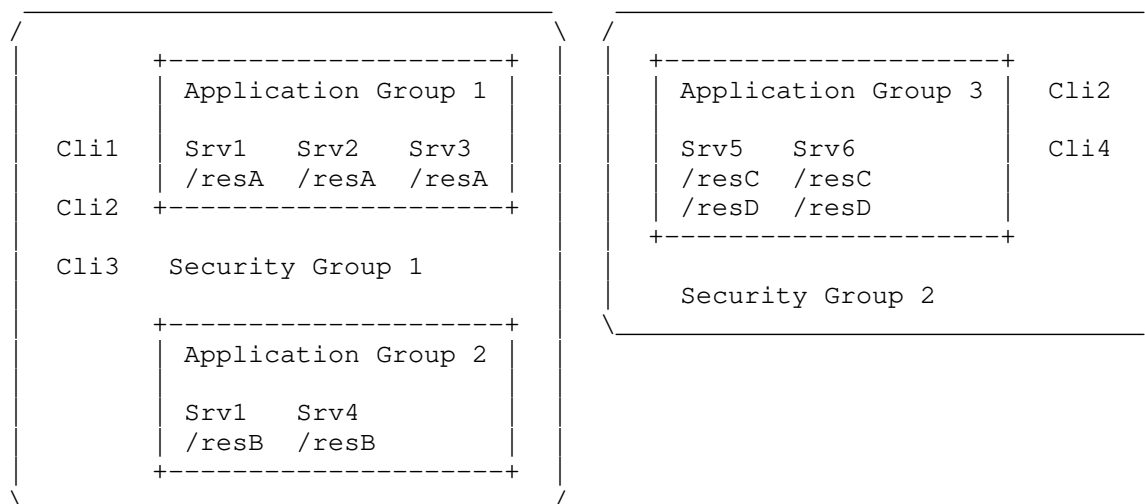


Figure 2: Deployment Example of Different Group Types

2.2. Group Configuration

2.2.1. Group Naming

A CoAP group is identified and named by the authority component in the Group URI, which includes host (possibly an IP multicast address literal) and an optional UDP port number. It is recommended to configure an endpoint with an IP multicast address literal, instead of a hostname, when configuring a CoAP group membership. This is because DNS infrastructure may not be deployed in many constrained networks. In case a group hostname is configured, it can be uniquely mapped to an IP multicast address via DNS resolution - if DNS client functionality is available in the endpoint being configured and the DNS service is supported in the network. Some examples of hierarchical CoAP group FQDN naming (and scoping) for a building control application are shown in Section 2.2 of [RFC7390].

An application group can be named in many ways through different types of identifiers, such as numbers, URIs or other strings. An application group name or identifier, if explicitly encoded in a CoAP request, is typically included in the path component or in the query component of a Group URI. It may also be encoded using the Uri-Host Option [RFC7252] in case application group members implement a virtual CoAP server specific to that application group. The application group can then be identified by the value of the Uri-Host Option and each virtual server serves one specific application group. However, encoding the application group in the Uri-Host Option is not

the preferred method because in this case the application group cannot be encoded in a Group URI, and also the Uri-Host Option is being used for another purpose than encoding the host part of a URI as intended by [RFC7252] - which is potentially confusing. Appendix A of [I-D.ietf-core-resource-directory] shows an example registration of an application group into a Resource Directory, along with the CoAP group it uses and the resources supported by the application group. In this example an application group identifier is not explicitly encoded in the RD nor in CoAP requests made to the group, but it implicitly follows from the CoAP group used for the request. So there is a one-to-one binding between the CoAP group and the application group. The "NoSec" security group is used.

A best practice for encoding application group into a Group URI is to use one URI path component to identify the application group and use the following URI paths component(s) to identify the resource within this application group. For example, /<groupname>/res1 or /base/<groupname>/res1/res2 conform to this practice. An application group identifier (like <groupname>) should be as short as possible when used in constrained networks.

A security group is identified by a stable and invariant string used as group name, which is generally not related with other kinds of group identifiers, specific to the chosen security solution. The "NoSec" security group name MUST be only used to represent the case of group communication without any security. It is typically characterized by the absence of any security group name, identifier, or security-related data structures in the CoAP message.

2.2.2. Group Creation and Membership

To create a CoAP group, a configuring entity defines an IP multicast address (or hostname) for the group and optionally a UDP port number in case it differs from the default CoAP port 5683. Then, it configures one or more devices as listeners to that IP multicast address, with a CoAP endpoint listening on the group's associated UDP port. These endpoints/devices are the group members. The configuring entity can be, for example, a local application with pre-configuration, a user, a software developer, a cloud service, or a local commissioning tool. Also, the devices sending CoAP requests to the group in the role of CoAP client need to be configured with the same information, even though they are not necessarily group members. One way to configure a client is to supply it with a CoAP Group URI. The IETF does not define a mandatory, standardized protocol to accomplish CoAP group creation. [RFC7390] defines an experimental protocol for configuration of group membership for unsecured group communication, based on JSON-formatted configuration resources. For

IPv6 CoAP groups, common multicast address ranges that are used to configure group addresses from are `ff1x::/16` and `ff3x::/16`.

To create an application group, a configuring entity may configure a resource (name) or set of resources on CoAP endpoints, such that a CoAP request with Group URI sent by a configured CoAP client will be processed by one or more CoAP servers that have the matching URI path configured. These servers are the application group members.

To create a security group, a configuring entity defines an initial subset of the related security material. This comprises a set of group properties including the cryptographic algorithms and parameters used in the group, as well as additional information relevant throughout the group life-cycle, such as the security group name and description. This task MAY be entrusted to a dedicated administrator, that interacts with a Group Manager as defined in Section 4. After that, further security material to protect group communications have to be generated, as compatible with the specified group configuration.

To participate in a security group, CoAP endpoints have to be configured with the group security material used to protect communications in the associated application/CoAP groups. The part of the process that involves secure distribution of group security material MAY use standardized communication with a Group Manager as defined in Section 4. For unsecure group communication using the "NoSec" security group, any CoAP endpoint may become a group member at any time: there is no (central) configuring entity that needs to provide the security material for this group. This means that group creation and membership cannot be tightly controlled for the "NoSec" group.

The configuration of groups and membership may be performed at different moments in the life-cycle of a device; for example during product (software) creation, in the factory, at a reseller, on-site during first deployment, or on-site during a system reconfiguration operation.

2.2.3. Group Discovery

It is possible for CoAP endpoints to discover application groups as well as CoAP groups, by using the RD-Groups usage pattern of the CoRE Resource Directory (RD), as defined in Appendix A of [I-D.ietf-core-resource-directory]. In particular, an application group can be registered to the RD, specifying the reference IP multicast address, hence its associated CoAP group. The registration is typically performed by a Commissioning Tool. Later on, CoAP

endpoints can discover the registered application groups and related CoAP group, by using the lookup interface of the RD.

CoAP endpoints can also discover application groups by performing a multicast discovery query using the /.well-known/core resource. Such a request may be sent to a known CoAP group multicast address associated to application group(s), or to the All CoAP Nodes multicast address.

When secure communication is provided with Group OSCORE (see Section 4), the approach described in [I-D.tiloca-core-oscore-discovery] and also based on the RD can be used, in order to discover the security group to join.

In particular, the responsible OSCORE Group Manager registers its own security groups to the RD, as links to its own corresponding resources for joining the security groups [I-D.ietf-ace-key-groupcomm-oscore]. Later on, CoAP endpoints can discover the registered security groups and related application groups, by using the lookup interface of the RD, and then join the security group through the respective Group Manager.

2.2.4. Group Maintenance

Maintenance of a group includes any necessary operations to cope with changes in a system, such as: adding group members, removing group members, changing group security material, reconfiguration of UDP port and/or IP multicast address, reconfiguration of the Group URI, renaming of application groups, splitting of groups, or merging of groups.

For unsecured group communication (see Section 3), addition/removal of CoAP group members is simply done by configuring these devices to start/stop listening to the group IP multicast address on the group's UDP port.

For secured group communication (see Section 4), the protocol Group OSCORE [I-D.ietf-core-oscore-groupcomm] is mandatory to implement. When using Group OSCORE, CoAP endpoints participating in group communication are also members of a corresponding OSCORE security group, and thus share common security material. Additional related maintenance operations are discussed in Section 4.1.

2.3. CoAP Usage

2.3.1. Request/Response Model

A CoAP client is an endpoint able to transmit CoAP requests and receive CoAP responses. Since the underlying UDP transport supports multiplexing by means of UDP port number, there can be multiple independent CoAP clients operational on a single host. On each UDP port, an independent CoAP client can be hosted. Each independent CoAP client sends requests that use the associated endpoint's UDP port number as the UDP source port of the request.

All CoAP requests that are sent via IP multicast **MUST** be Non-confirmable (Section 8.1 of [RFC7252]). The Message ID in an IP multicast CoAP message is used for optional message deduplication by both clients and servers, as detailed in Section 4.5 of [RFC7252].

A server sends back a unicast response to the CoAP group request - but the server **MAY** suppress the response for various reasons (Section 8.2 of [RFC7252]). This document adds the requirement that a server **SHOULD** suppress the response in case of error or in case there is nothing useful to respond, unless the application related to a particular resource requires such a response to be made for that resource. The unicast responses received by the CoAP client may be a mixture of success (e.g., 2.05 Content) and failure (e.g., 4.04 Not Found) codes, depending on the individual server processing results.

The CoAP No-Response Option [RFC7967] can be used by a client to influence the default response suppression on the server side. It is **RECOMMENDED** for a server to support this option only on selected resources where it is useful in the application context. If the Option is supported on a resource, it **MUST** override the default response suppression of that resource.

Any default response suppression by a server **SHOULD** be performed consistently, as follows: if a request on a resource produces a particular Response Code and this response is not suppressed, then another request on the same resource that produces a response of the same Response Code class is also not suppressed. For example, if a 4.05 Method Not Allowed error response code is suppressed by default on a resource, then a 4.15 Unsupported Content-Format error response code is also suppressed by default for that resource.

A CoAP client **MAY** repeat a multicast request using the same Token value and same Message ID value, in order to ensure that enough (or all) group members have been reached with the request. This is useful in case a number of group members did not respond to the initial request and the client suspects that the request did not reach these group members. However, in case one or more servers did receive the initial request but the response to that request was

lost, this repeat does not help to retrieve the lost response(s) if the server(s) implement the optional Message ID based deduplication (Section 4.5 of [RFC7252]).

A CoAP client MAY also repeat a multicast request using the same Token value and a different Message ID, in which case all servers that received the initial request will again process the repeated request since it appears within a new CoAP message. This is useful in case a client suspects that one or more response(s) to its original request were lost and the client needs to collect more, or even all, responses from group members, even if this comes at the cost of the overhead of certain group members responding twice (once to the original request, and once to the repeated request with different Message ID).

The CoAP client can distinguish the origin of multiple server responses by the source IP address of the UDP message containing the CoAP response and/or any other available application-specific source identifiers contained in the CoAP response, such as an application-level unique ID associated to the server. If secure communication is provided with Group OSCORE (see Section 4), additional security-related identifiers enable the client to retrieve the right security material for decrypting each response and authenticating its source.

While processing a response, the source endpoint of the response is not matched to the destination endpoint of the request, since for a multicast request these will never match. This is specified in Section 8.2 of [RFC7252]. It implies also that a server MAY respond from a UDP port number that differs from the destination UDP port number of the request, yet a CoAP server normally SHOULD respond from the UDP port number that equals the destination port of the request - following the convention for UDP-based protocols. In case a single client has sent multiple group requests and concurrent CoAP transactions are ongoing, the responses received by that client are matched to an active request using only the Token value. Due to UDP level multiplexing, the UDP destination port of the response MUST match to the client endpoint's UDP port value, i.e. to the UDP source port of the client's request.

For multicast CoAP requests, there are additional constraints on the reuse of Token values at the client, compared to the unicast case defined in [RFC7252] and updated by [I-D.ietf-core-echo-request-tag]. Since for multicast CoAP the number of responses is not bound a priori, the client cannot use the reception of a response as a trigger to "free up" a Token value for reuse. Reusing a Token value too early could lead to incorrect response/request matching on the client, and would be a protocol error. Therefore, the time between

reuse of Token values used in multicast requests MUST be greater than:

$$\text{MIN_TOKEN_REUSE_TIME} = (\text{NON_LIFETIME} + \text{MAX_LATENCY} + \text{MAX_SERVER_RESPONSE_DELAY})$$

where NON_LIFETIME and MAX_LATENCY are defined in Section 4.8 of [RFC7252]. This specification defines MAX_SERVER_RESPONSE_DELAY as in [RFC7390], that is: the expected maximum response delay over all servers that the client can send a multicast request to. This delay includes the maximum Leisure time period as defined in Section 8.2 of [RFC7252]. However, CoAP does not define a time limit for the server response delay. Using the default CoAP parameters, the Token reuse time MUST be greater than 250 seconds plus MAX_SERVER_RESPONSE_DELAY. A preferred solution to meet this requirement is to generate a new unique Token for every new multicast request, such that a Token value is never reused. If a client has to reuse Token values for some reason, and also MAX_SERVER_RESPONSE_DELAY is unknown, then using MAX_SERVER_RESPONSE_DELAY = 250 seconds is a reasonable guideline. The time between Token reuses is in that case set to a value greater than 500 seconds.

When securing Group CoAP communications with Group OSCORE [I-D.ietf-core-oscore-groupcomm], secure binding between requests and responses is ensured (see Section 4). Thus, a client may reuse a Token value after it has been freed up, as discussed above for the multicast case and considering a reuse time greater than MIN_TOKEN_REUSE_TIME. If an alternative security protocol for Group CoAP is defined in the future and it does not ensure secure binding between requests and responses, a client MUST follow the Token processing requirements for the unicast case discussed above, as defined in [I-D.ietf-core-echo-request-tag].

Another method to more easily meet the above constraint is to instantiate multiple CoAP clients at multiple UDP ports on the same host. The Token values only have to be unique within the context of a single CoAP client, so using multiple clients can make it easier to meet the constraint.

Since a client sending a multicast request with a Token T will accept multiple responses with the same Token T, there is a risk that the same server sends multiple responses with the same Token T back to the client. For example, this server might not implement the optional CoAP message deduplication based on Message ID, or it might be a malicious/compromised server acting out of specification. To mitigate issues with multiple responses from one server bound to a same multicast request, the client has to ensure that, as long as the CoAP Token used for a multicast request is retained, at most one

response to that request per server is accepted, with the exception of Observe notifications [RFC7641] (see Section 2.3.5).

To this end, upon receiving a response corresponding to a multicast request, the client **MUST** perform the following actions. First, the client checks whether it previously received a valid response to this request from the same originating server of the just-received response. If the check yields a positive match and the response is not an Observe notification (i.e., it does not include an Observe option), the client **SHALL** stop processing the response. Upon eventually freeing up the Token value of a multicast request for possible reuse, the client **MUST** also delete the list of responding servers associated to that request.

2.3.2. Port and URI Path Selection

A server that is a member of a CoAP group listens for CoAP messages on the group's IP multicast address, usually on the CoAP default UDP port 5683, or another non-default UDP port if configured. Regardless of the method for selecting the port number, the same port number **MUST** be used across all CoAP servers that are members of a CoAP group and across all CoAP clients performing the requests to that group. The URI Path used in the request is preferably a path that is known to be supported across all group members. However there are valid use cases where a request is known to be successful for a subset of the CoAP group, for example only members of a specific application group, while those group members for which the request is unsuccessful (for example because they are outside the application group) either ignore the multicast request or respond with an error status code.

One way to create multiple CoAP groups is using different UDP ports with the same IP multicast address, in case the devices' network stack only supports a limited number of multicast address subscriptions. However, it must be taken into account that this incurs additional processing overhead on each CoAP server participating in at least one of these groups: messages to groups that are not of interest to the node are only discarded at the higher transport (UDP) layer instead of directly at the network (IP) layer.

Port 5684 is reserved for DTLS-secured CoAP and **MUST NOT** be used for any CoAP group communication.

For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port 5683 **MUST** be supported (see Section 7.1 of [RFC7252]) for the "All CoAP Nodes" multicast group as detailed in Section 2.4.

2.3.3. Proxy Operation

CoAP enables a client to request a forward-proxy to process a CoAP request on its behalf, as described in Section 5.7.2 and 8.2.2 of [RFC7252]. For this purpose, the client specifies either the request group URI as a string in the Proxy-URI option or it uses the Proxy-Scheme option with the group URI constructed from the usual Uri-* options. The forward-proxy then resolves the group URI to a destination CoAP group, multicasts the CoAP request, receives the responses and forwards all the individual (unicast) responses back to the client.

However, there are certain issues and limitations with this approach:

- o The CoAP client component that sent a unicast CoAP request to the proxy may be expecting only one (unicast) response, as usual for a CoAP unicast request. Instead, it receives multiple (unicast) responses, potentially leading to fault conditions in the component or to discarding any received responses following the first one. This issue may occur even if the application calling the CoAP client component is aware that the forward-proxy is going to execute a CoAP group URI request.
- o Each individual CoAP response received by the client will appear to originate (based on its IP source address) from the CoAP Proxy, and not from the server that produced the response. This makes it impossible for the client to identify the server that produced each response, unless the server identity is contained as a part of the response payload or inside a CoAP Option in the response.

A method based on this approach and addressing the issues raised above is defined in [I-D.tiloca-core-groupcomm-proxy].

An alternative solution is for the proxy to collect all the individual (unicast) responses to a CoAP group request and then send back only a single (aggregated) response to the client. However, this solution brings up new issues:

- o The proxy does not know how many members there are in the CoAP group or how many group members will actually respond. Also, the proxy does not know for how long to collect responses before sending back the aggregated response to the client. A CoAP client that is not using a Proxy might face the same problems in collecting responses to a multicast request. However, the client itself would typically have application-specific rules or knowledge on how to handle this situation, while an application-agnostic CoAP Proxy would typically not have this knowledge. For example, a CoAP client could monitor incoming responses and use

this information to decide how long to continue collecting responses - which is something a proxy cannot do.

- o There is no default format defined in CoAP for aggregation of multiple responses into a single response. Such a format could be standardized based on, for example, the multipart content-format [RFC8710].

Due to the above issues, it is RECOMMENDED that a CoAP Proxy only processes a group URI request if it is explicitly enabled to do so. The default response (if the function is not explicitly enabled) to a group URI request is 5.01 (Not Implemented). Furthermore, a proxy SHOULD be explicitly configured (e.g. by white-listing and/or client authentication) to allow proxied CoAP multicast requests only from specific client(s).

The operation of HTTP-to-CoAP proxies for multicast CoAP requests is specified in Section 8.4 and 10.1 of [RFC8075]. In this case, the "application/http" media type is used to let the proxy return multiple CoAP responses - each translated to a HTTP response - back to the HTTP client. Of course, in this case the HTTP client sending a group URI to the proxy needs to be aware that it is going to receive this format, and needs to be able to decode it into the responses of multiple CoAP servers. Also, the IP source address of each CoAP response cannot be determined anymore from the "application/http" response. The HTTP client still identifies the CoAP servers by other means such as application-specific information in the response payload.

2.3.4. Congestion Control

CoAP group requests may result in a multitude of responses from different nodes, potentially causing congestion. Therefore, both the sending of IP multicast requests and the sending of the unicast CoAP responses to these multicast requests should be conservatively controlled.

CoAP [RFC7252] reduces IP multicast-specific congestion risks through the following measures:

- o A server may choose not to respond to an IP multicast request if there is nothing useful to respond to, e.g., error or empty response (see Section 8.2 of [RFC7252]).
- o A server should limit the support for IP multicast requests to specific resources where multicast operation is required (Section 11.3 of [RFC7252]).

- o An IP multicast request **MUST** be Non-confirmable (Section 8.1 of [RFC7252]).
- o A response to an IP multicast request **SHOULD** be Non-confirmable (Section 5.2.3 of [RFC7252]).
- o A server does not respond immediately to an IP multicast request and should first wait for a time that is randomly picked within a predetermined time interval called the Leisure (Section 8.2 of [RFC7252]).

Additional guidelines to reduce congestion risks defined in this document are as follows:

- o A server in a constrained network should only support group communication with GET and FETCH for resources that are small. This can consist, for example, in having the payload of the response as limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size, so that it fits into a single link-layer frame in case IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) (see Section 4 of [RFC4944]) is used.
- o A server **SHOULD** minimize the payload size of a response to a multicast GET or FETCH on `"/.well-known/core"` by using hierarchy in arranging link descriptions for the response. An example of this is given in Section 5 of [RFC6690].
- o A server **MAY** minimize the payload size of a response to a multicast GET or FETCH (e.g., on `"/.well-known/core"`) by using CoAP block-wise transfers [RFC7959] in case the payload is long, returning only a first block of the CoRE Link Format description. For this reason, a CoAP client sending an IP multicast CoAP request to `"/.well-known/core"` **SHOULD** support block-wise transfers. See also Section 2.3.6.
- o A client **SHOULD** be configured to use CoAP groups with the smallest possible IP multicast scope that fulfills the application needs. As an example, site-local scope is always preferred over global scope IP multicast if this fulfills the application needs. Similarly, realm-local scope is always preferred over site-local scope if this fulfills the application needs.

2.3.5. Observing Resources

The CoAP Observe Option [RFC7641] is a protocol extension of CoAP, that allows a CoAP client to retrieve a representation of a resource and automatically keep this representation up-to-date over a longer period of time. The client gets notified when the representation has

changed. [RFC7641] does not mention whether the Observe Option can be combined with CoAP multicast.

This section updates [RFC7641] with the use of the Observe Option in a CoAP multicast GET request, and defines normative behavior for both client and server. Consistently with Section 2.4 of [RFC8132], it is also possible to use the Observe Option in a CoAP multicast FETCH request.

Multicast Observe is a useful way to start observing a particular resource on all members of a CoAP group at the same time. Group members that do not have this particular resource or do not allow the GET or FETCH method on it will either respond with an error status - 4.04 Not Found or 4.05 Method Not Allowed, respectively - or will silently suppress the response following the rules of Section 2.3.1, depending on server-specific configuration.

A client that sends a multicast GET or FETCH request with the Observe Option MAY repeat this request using the same Token value and the same Observe Option value, in order to ensure that enough (or all) members of the CoAP group have been reached with the request. This is useful in case a number of group members did not respond to the initial request. The client MAY additionally use the same Message ID in the repeated request to avoid that group members that had already received the initial request would respond again. Note that using the same Message ID in a repeated request will not be helpful in case of loss of a response message, since the server that responded already will consider the repeated request as a duplicate message. On the other hand, if the client uses a different, fresh Message ID in the repeated request, then all the group members that receive this new message will typically respond again, which increases the network load.

A client that has sent a multicast GET or FETCH request with the Observe Option MAY follow up by sending a new unicast CON request with the same Token value and same Observe Option value to a particular server, in order to ensure that the particular server receives the request. This is useful in case a specific group member, that was expected to respond to the initial group request, did not respond to the initial request. In this case, the client always uses a Message ID that differs from the initial multicast message.

Furthermore, consistently with Section 3.3.1 of [RFC7641] and following its guidelines, a client MAY at any time send a new multicast GET or FETCH request with the same Token value and same Observe Option value as the original request. This allows the client

to verify that it has an up-to-date representation of an observed resource and/or to re-register its interest to observe a resource.

In the above client behaviors, the Token value is kept identical to the initial request to avoid that a client is included in more than one entry in the list of observers (Section 4.1 of [RFC7641]).

Before repeating a request as specified above, the client SHOULD wait for at least the expected round-trip time plus the Leisure time period defined in Section 8.2 of [RFC7252], to give the server time to respond.

A server that receives a GET or FETCH request with the Observe Option, for which request processing is successful, SHOULD respond to this request and not suppress the response. A server that adds a client to the list of observers for a resource due to an Observe request MUST respond to this request and not suppress it.

A server SHOULD have a mechanism to verify liveness of its observing clients and the continued interest of these clients in receiving the observe notifications. This can be implemented by sending notifications occasionally using a Confirmable message. See Section 4.5 of [RFC7641] for details. This requirement overrides the regular behavior of sending Non-Confirmable notifications in response to a Non-Confirmable request.

A client can use the unicast cancellation methods of Section 3.6 of [RFC7641] and stop the ongoing observation of a particular resource on members of a CoAP group. This can be used to remove specific observed servers, or even all servers in the group. In addition, a client MAY explicitly deregister from all those servers at once, by sending a multicast GET or FETCH request that includes the Token value of the observation to be cancelled and includes an Observe Option with the value set to 1 (deregister). In case not all the servers in the CoAP group received this deregistration request, either the unicast cancellation methods can be used or the multicast deregistration request MAY be repeated upon receiving another observe response from a server.

For observing a group of servers through a CoAP-to-CoAP proxy, the limitations stated in Section 2.3.3 apply. The method defined in [I-D.tiloca-core-groupcomm-proxy] enables group communication through proxies and addresses those limitations.

2.3.6. Block-Wise Transfer

Section 2.8 of [RFC7959] specifies how a client can use block-wise transfer (Block2 Option) in a multicast GET request to limit the size of the initial response of each server. Consistently with Section 2.5 of [RFC8132], the same can be done with a multicast FETCH request.

The client has to use unicast for any further request, separately addressing each different server, in order to retrieve more blocks of the resource from that server, if any. Also, a server (member of a targeted CoAP group) that needs to respond to a multicast request with a particularly large resource can use block-wise transfer (Block2 Option) at its own initiative, to limit the size of the initial response. Again, a client would have to use unicast for any further requests to retrieve more blocks of the resource.

A solution for multicast block-wise transfer using the Block1 Option is not specified in [RFC7959] nor in the present document. Such a solution would be useful for multicast FETCH/PUT/POST/PATCH/iPATCH requests, to efficiently distribute a large request payload as multiple blocks to all members of a CoAP group. Multicast usage of Block1 is non-trivial due to potential message loss (leading to missing blocks or missing confirmations), and potential diverging block size preferences of different members of the CoAP group.

2.4. Transport

In this document only UDP is considered as a transport protocol, both over IPv4 and IPv6. Therefore, [RFC8323] (CoAP over TCP, TLS, and WebSockets) is not in scope as a transport for group communication.

2.4.1. UDP/IPv6 Multicast Transport

CoAP group communication can use UDP over IPv6 as a transport protocol, provided that IPv6 multicast is enabled. IPv6 multicast MAY be supported in a network only for a limited scope. For example, Section 2.5.2 describes the potential limited support of RPL for multicast, depending on how the protocol is configured.

For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port 5683 MUST be supported as per Section 7.1 and 12.8 of [RFC7252] for the "All CoAP Nodes" multicast group. An IPv6 CoAP server SHOULD support the "All CoAP Nodes" groups with at least link-local (2), admin-local (4) and site-local (5) scopes. An IPv6 CoAP server on a 6LoWPAN node (see Section 2.4.3) SHOULD also support the realm-local (3) scope.

Note that a client sending an IPv6 multicast CoAP message to a port that is not supported by the server will not receive an ICMPv6 Port Unreachable error message from that server, because the server does not send it in this case, per Section 2.4 of [RFC4443].

2.4.2. UDP/IPv4 Multicast Transport

CoAP group communication can use UDP over IPv4 as a transport protocol, provided that IPv4 multicast is enabled. For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port 5683 MUST be supported as per Section 7.1 and 12.8 of [RFC7252], for the "All CoAP Nodes" IPv4 multicast group.

Note that a client sending an IPv4 multicast CoAP message to a port that is not supported by the server will not receive an ICMP Port Unreachable error message from that server, because the server does not send it in this case, per Section 3.2.2 of [RFC1122].

2.4.3. 6LoWPAN

In 6LoWPAN [RFC4944] networks, IPv6 packets (up to 1280 bytes) may be fragmented into smaller IEEE 802.15.4 MAC frames (up to 127 bytes), if the packet size requires this. Every 6LoWPAN IPv6 router that receives a multi-fragment packet reassembles the packet and refragments it upon transmission. Since the loss of a single fragment implies the loss of the entire IPv6 packet, the performance in terms of packet loss and throughput of multi-fragment multicast IPv6 packets is typically far worse than the performance of single-fragment IPv6 multicast packets. For this reason, a CoAP request sent over multicast in 6LoWPAN networks SHOULD be sized in such a way that it fits in a single IEEE 802.15.4 MAC frame, if possible.

On 6LoWPAN networks, multicast groups can be defined with realm-local scope [RFC7346]. Such a realm-local group is restricted to the local 6LoWPAN network/subnet. In other words, a multicast request to that group does not propagate beyond the 6LoWPAN network segment where the request originated. For example, a multicast discovery request can be sent to the realm-local "All CoAP Nodes" IPv6 multicast group (see Section 2.4.1) in order to discover only CoAP servers on the local 6LoWPAN network.

2.5. Interworking with Other Protocols

2.5.1. MLD/MLDv2/IGMP/IGMPv3

CoAP nodes that are IP hosts (i.e., not IP routers) are generally unaware of the specific IP multicast routing/forwarding protocol being used in their network. When such a host needs to join a

specific (CoAP) multicast group, it requires a way to signal to IP multicast routers which IP multicast address(es) it needs to listen to.

The MLDv2 protocol [RFC3810] is the standard IPv6 method to achieve this; therefore, this method SHOULD be used by members of a CoAP group to subscribe to its multicast IPv6 address, on IPv6 networks that support it. CoAP server nodes then act in the role of MLD Multicast Address Listener. Constrained IPv6 networks that implement either RPL (see Section 2.5.2) or MPL (see Section 2.5.3) typically do not support MLD as they have their own mechanisms defined.

The IGMPv3 protocol [RFC3376] is the standard IPv4 method to signal multicast group subscriptions. This SHOULD be used by members of a CoAP group to subscribe to its multicast IPv4 address on IPv4 networks.

The guidelines from [RFC6636] on the tuning of MLD for mobile and wireless networks may be useful when implementing MLD in constrained networks.

2.5.2. RPL

RPL [RFC6550] is an IPv6 based routing protocol suitable for low-power, lossy networks (LLNs). In such a context, CoAP is often used as an application protocol.

If only RPL is used in a network for routing and its optional multicast support is disabled, there will be no IP multicast routing available. Any IPv6 multicast packets in this case will not propagate beyond a single hop (to direct neighbors in the LLN). This implies that any CoAP group request will be delivered to link-local nodes only, for any scope value ≥ 2 used in the IPv6 destination address.

RPL supports (see Section 12 of [RFC6550]) advertisement of IP multicast destinations using Destination Advertisement Object (DAO) messages and subsequent routing of multicast IPv6 packets based on this. It requires the RPL mode of operation to be 3 (Storing mode with multicast support).

In this mode, RPL DAO can be used by a CoAP node that is either an RPL router or RPL Leaf Node, to advertise its CoAP group membership to parent RPL routers. Then, RPL will route any IP multicast CoAP requests over multiple hops to those CoAP servers that are group members.

The same DAO mechanism can be used to convey CoAP group membership information to an edge router (e.g., 6LBR), in case the edge router is also the root of the RPL Destination-Oriented Directed Acyclic Graph (DODAG). This is useful because the edge router then learns which IP multicast traffic it needs to pass through from the backbone network into the LLN subnet. In LLNs, such ingress filtering helps to avoid congestion of the resource-constrained network segment, due to IP multicast traffic from the high-speed backbone IP network.

2.5.3. MPL

The Multicast Protocol for Low-Power and Lossy Networks (MPL) [RFC7731] can be used for propagation of IPv6 multicast packets throughout a defined network domain, over multiple hops. MPL is designed to work in LLNs and can operate alone or in combination with RPL. The protocol involves a predefined group of MPL Forwarders to collectively distribute IPv6 multicast packets throughout their MPL Domain. An MPL Forwarder may be associated to multiple MPL Domains at the same time. Non-Forwarders will receive IPv6 multicast packets from one or more of their neighboring Forwarders. Therefore, MPL can be used to propagate a CoAP multicast request to all group members.

However, a CoAP multicast request to a group that originated outside of the MPL Domain will not be propagated by MPL - unless an MPL Forwarder is explicitly configured as an ingress point that introduces external multicast packets into the MPL Domain. Such an ingress point could be located on an edge router (e.g., 6LBR). The method to configure which multicast groups are to be propagated into the MPL Domain could be:

- o Manual configuration on the ingress MPL Forwarder.
- o A protocol to register multicast groups at an ingress MPL Forwarder. This could be a protocol offering features similar to MLDv2.

3. Unsecured Group Communication

CoAP group communication can operate in CoAP NoSec (No Security) mode, without using application-layer and transport-layer security mechanisms. The NoSec mode uses the "coap" scheme, and is defined in Section 9 of [RFC7252]. The conceptual "NoSec" security group as defined in Section 2.1 is used for unsecured group communication. Before using this mode of operation, the security implications (Section 5.1) must be well understood.

4. Secured Group Communication using Group OSCORE

The application-layer protocol Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] provides end-to-end encryption, integrity and replay protection of CoAP messages exchanged between two CoAP endpoints. These can act both as CoAP Client as well as CoAP Server, and share an OSCORE Security Context used to protect and verify exchanged messages. The use of OSCORE does not affect the URI scheme and OSCORE can therefore be used with any URI scheme defined for CoAP.

OSCORE uses COSE

[I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] to perform encryption operations and protect a CoAP message in a COSE object, by using an Authenticated Encryption with Associated Data (AEAD) algorithm. In particular, OSCORE takes as input an unprotected CoAP message and transforms it into a protected CoAP message transporting the COSE object.

OSCORE makes it possible to selectively protect different parts of a CoAP message in different ways, while still allowing intermediaries (e.g., CoAP proxies) to perform their intended functionalities. That is, some message parts are encrypted and integrity protected; other parts are only integrity protected to be accessible to, but not modifiable by, proxies; and some parts are kept as plain content to be both accessible to and modifiable by proxies. Such differences especially concern the CoAP options included in the unprotected message.

Group OSCORE [I-D.ietf-core-oscore-groupcomm] builds on OSCORE, and provides end-to-end security of CoAP messages exchanged between members of an OSCORE group, while fulfilling the same security requirements.

In particular, Group OSCORE protects CoAP requests sent over IP multicast by a CoAP client, as well as multiple corresponding CoAP responses sent over IP unicast by different CoAP servers. However, the same security material can also be used to protect CoAP requests sent over IP unicast to a single CoAP server in the OSCORE group, as well as the corresponding responses.

Group OSCORE ensures source authentication of all messages exchanged within the OSCORE group, by means of two possible methods.

The first method, namely group mode, relies on digital signatures. That is, sender devices sign their outgoing messages using their own private key, and embed the signature in the protected CoAP message.

The second method, namely pairwise mode, relies on a symmetric key, which is derived from a pairwise shared secret computed from the asymmetric keys of the message sender and recipient. This method is intended for one-to-one messages sent in the group, such as all responses as individually sent by servers, as well as requests addressed to an individual server.

A Group Manager is responsible for one or multiple OSCORE groups. In particular, the Group Manager acts as repository of public keys of group members; manages, renews and provides security material in the group; and handles the join process of new group members.

As defined in [I-D.ietf-ace-oscore-gm-admin], an administrator entity can interact with the Group Manager to create OSCORE groups and specify their configuration (see Section 2.2.2). During the lifetime of the OSCORE group, the administrator can further interact with the Group Manager, in order to possibly update the group configuration and eventually delete the group.

As recommended in [I-D.ietf-core-oscore-groupcomm], a CoAP endpoint can join an OSCORE group by using the method described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

A CoAP endpoint can discover OSCORE groups and retrieve information to join them through their Group Managers by using the method described in [I-D.ietf-core-oscore-discovery] and based on the CoRE Resource Directory [I-D.ietf-core-resource-directory].

If security is required, CoAP group communication as described in this specification MUST use Group OSCORE. In particular, a CoAP group as defined in Section 2.1 and using secure group communication is associated to an OSCORE security group, which includes:

- o All members of the CoAP group, i.e. the CoAP endpoints configured (also) as CoAP servers and listening to the group's multicast IP address.
- o All further CoAP endpoints configured only as CoAP clients, that send (multicast) CoAP requests to the CoAP group.

4.1. Secure Group Maintenance

Additional key management operations on the OSCORE group are required, depending also on the security requirements of the application (see Section 5.2). That is:

- o Adding new members to a CoAP group or enabling new client-only endpoints to interact with that group require also that each of such members/endpoints join the corresponding OSCORE group. By doing so, they are securely provided with the necessary cryptographic material. In case backward security is needed, this also requires to first renew such material and distribute it to the current members/endpoints, before new ones are added and join the OSCORE group.
- o In case forward security is needed, removing members from a CoAP group or stopping client-only endpoints from interacting with that group requires removing such members/endpoints from the corresponding OSCORE group. To this end, new cryptographic material is generated and securely distributed only to the remaining members/endpoints. This ensures that only the members/endpoints intended to remain are able to continue participating in secure group communication, while the evicted ones are not able to.

The key management operations mentioned above are entrusted to the Group Manager responsible for the OSCORE group [I-D.ietf-core-oscore-groupcomm], and it is RECOMMENDED to perform them according to the approach described in [I-D.ietf-ace-key-groupcomm-oscore].

5. Security Considerations

This section provides security considerations for CoAP group communication using IP multicast.

5.1. CoAP NoSec Mode

CoAP group communication, if not protected, is vulnerable to all the attacks mentioned in Section 11 of [RFC7252] for IP multicast.

Thus, for sensitive and mission-critical applications (e.g., health monitoring systems and alarm monitoring systems), it is NOT RECOMMENDED to deploy CoAP group communication in NoSec mode.

Without application-layer security, CoAP group communication SHOULD only be deployed in applications that are non-critical, and that do not involve or may have an impact on sensitive data and personal sphere. These include, e.g., read-only temperature sensors deployed in non-sensitive environments, where the client reads out the values but does not use the data to control actuators or to base an important decision on.

Discovery of devices and resources is a typical use case where NoSec mode is applied, since the devices involved do not have yet configured any mutual security relations at the time the discovery takes place.

5.2. Group OSCORE

Group OSCORE provides end-to-end application-level security. This has many desirable properties, including maintaining security assurances while forwarding traffic through intermediaries (proxies). Application-level security also tends to more cleanly separate security from the dynamics of group membership (e.g., the problem of distributing security keys across large groups with many members that come and go).

For sensitive and mission-critical applications, CoAP group communication **MUST** be protected by using Group OSCORE as specified in [I-D.ietf-core-oscore-groupcomm]. The same security considerations from Section 10 of [I-D.ietf-core-oscore-groupcomm] hold for this specification.

5.2.1. Group Key Management

A key management scheme for secure revocation and renewal of group security material, namely group rekeying, should be adopted in OSCORE groups. In particular, the key management scheme should preserve backward and forward security in the OSCORE group, if the application requires so (see Section 3.1 of [I-D.ietf-core-oscore-groupcomm]).

Group policies should also take into account the time that the key management scheme requires to rekey the group, on one hand, and the expected frequency of group membership changes, i.e. nodes' joining and leaving, on the other hand.

In fact, it may be desirable to not rekey the group upon every single membership change, in case members' joining and leaving are frequent, and at the same time a single group rekeying instance takes a non negligible time to complete.

In such a case, the Group Manager may consider to rekey the group, e.g., after a minimum number of nodes has joined or left the group within a pre-defined time interval, or according to communication patterns with predictable intervals of network inactivity. This would prevent paralyzing communications in the group, when a slow rekeying scheme is used and frequently invoked.

This comes at the cost of not continuously preserving backward and forward security, since group rekeying might not occur upon every

single group membership change. That is, most recently joined nodes would have access to the security material used prior to their join, and thus be able to access past group communications protected with that security material. Similarly, until the group is rekeyed, most recently left nodes would preserve access to group communications protected with the retained security material.

5.2.2. Source Authentication

Both the group mode and the pairwise mode of Group OSCORE ensure source authentication of messages exchanged by CoAP endpoints through CoAP group communication.

To this end, outgoing messages are either countersigned by the message sender endpoint with its own private key (group mode), or protected with a symmetric key, which is in turn derived using the asymmetric keys of the message sender and recipient (pairwise mode).

Thus, both modes allow a recipient CoAP endpoint to verify that a message has actually been originated by a specific and identified member of the OSCORE group.

Appendix F of [I-D.ietf-core-oscore-groupcomm] discusses a number of cases where a recipient CoAP endpoint may skip the verification of countersignatures in messages protected with the group mode, possibly on a per-message basis. However, this is NOT RECOMMENDED. That is, a CoAP endpoint receiving a message secured with the group mode of Group OSCORE SHOULD always verify the countersignature.

5.2.3. Countering Attacks

As discussed below, Group OSCORE addresses a number of security attacks mentioned in Section 11 of [RFC7252], with particular reference to their execution over IP multicast.

- o Since Group OSCORE provides end-to-end confidentiality and integrity of request/response messages, proxies in multicast settings cannot break message protection, and thus cannot act as man-in-the-middle beyond their legitimate duties (see Section 11.2 of [RFC7252]). In fact, intermediaries such as proxies are not assumed to have access to the OSCORE Security Context used by group members. Also, with the notable addition of countersignatures for the group mode, Group OSCORE protects messages using the same constructions of OSCORE (see Sections 8.1 and 8.3 of [I-D.ietf-core-oscore-groupcomm]), and especially processes CoAP options according to the same classification in U/I/E classes.

- o Group OSCORE protects against amplification attacks (see Section 11.3 of [RFC7252]), which are made e.g. by injecting (small) requests over IP multicast from the (spoofed) IP address of a victim client, and thus triggering the transmission of several (much bigger) responses back to that client. In fact, upon receiving a request over IP multicast as protected with Group OSCORE in group mode, a server is able to verify whether the request is fresh and originates from the alleged sender in the OSCORE group, by verifying the countersignature included in the request using the public key of that sender (see Section 8.2 of [I-D.ietf-core-oscore-groupcomm]). Furthermore, as also discussed in Section 8 of [I-D.ietf-core-oscore-groupcomm], it is recommended that servers failing to decrypt and verify an incoming message do not send back any error message.
- o Group OSCORE limits the impact of attacks based on IP spoofing also over IP multicast (see Section 11.4 of [RFC7252]). In fact, requests and corresponding responses sent in the OSCORE group can be correctly generated only by legitimate group members.

Within an OSCORE group, the shared symmetric-key security material strictly provides only group-level authentication (see Section 10.1 of [I-D.ietf-core-oscore-groupcomm]). However, source authentication of messages is also ensured, both in the group mode by means of countersignatures (see Sections 8.1 and 8.3 of [I-D.ietf-core-oscore-groupcomm]), and in the pairwise mode by using additionally derived pairwise keys (see Sections 9.1 and 9.3 of [I-D.ietf-core-oscore-groupcomm]). Thus, recipient endpoints can verify a message to be originated by the alleged, identifiable sender in the OSCORE group.

Note that the server may additionally rely on the Echo option for CoAP described in [I-D.ietf-core-echo-request-tag], in order to verify the aliveness and reachability of the client sending a request from a particular IP address.

- o Group OSCORE does not require group members to be equipped with a good source of entropy for generating security material (see Section 11.6 of [RFC7252]), and thus does not contribute to create an attack vector against such (constrained) CoAP endpoints. In particular, the symmetric keys used for message encryption and decryption are derived through the same HMAC-based HKDF scheme used for OSCORE (see Section 3.2 of [RFC8613]). Besides, the OSCORE Master Secret used in such derivation is securely generated by the Group Manager responsible for the OSCORE group, and securely provided to the CoAP endpoints when they join the group.

- o Group OSCORE prevents to make any single group member a target for subverting security in the whole OSCORE group (see Section 11.6 of [RFC7252]), even though all group members share (and can derive) the same symmetric-key security material used in the OSCORE group (see Section 10.1 of [I-D.ietf-core-oscore-groupcomm]). In fact, source authentication is always ensured for exchanged CoAP messages, as verifiable to be originated by the alleged, identifiable sender in the OSCORE group. This relies on including a countersignature computed with a node's individual private key (in the group mode), or on protecting messages with a pairwise symmetric key, which is in turn derived from the asymmetric keys of the sender and recipient CoAP endpoints (in the pairwise mode).

5.3. Replay of Non Confirmable Messages

Since all requests sent over IP multicast are Non-confirmable, a client might not be able to know if an adversary has actually captured one of its transmitted requests and later re-injected it in the group as a replay to the server nodes. In fact, even if the servers sent back responses to the replayed request, the client would not have a valid matching request anymore to suspect of the attack.

If Group OSCORE is used, such a replay attack on the servers is prevented, since a client protects every different request with a different Sequence Number value, which is in turn included as Partial IV in the protected message and takes part in the construction of the AEAD cipher nonce. Thus, a server would be able to detect the replayed request, by checking the conveyed Partial IV against its own replay window in the OSCORE Recipient Context associated to the client.

This requires a server to have a synchronized, up to date view of the sequence number used by the client. If such synchronization is lost, e.g. due to a reboot, or suspected so, the server should use one of the methods described in Appendix E of [I-D.ietf-core-oscore-groupcomm], such as the one based on the Echo option for CoAP described in [I-D.ietf-core-echo-request-tag], in order to (re-)synchronize with the client's sequence number.

5.4. Use of CoAP No-Response Option

When CoAP group communication is used in CoAP NoSec (No Security) mode (see Section 3), the CoAP No-Response Option [RFC7967] could be misused by a malicious client to evoke as much responses from servers to a multicast request as possible, by using the value '0' - Interested in all responses. This even overrides the default behaviour of a CoAP server to suppress the response in case there is

nothing of interest to respond with. Therefore, this option can be used to perform an amplification attack.

A proposed mitigation is to only allow this Option to relax the standard suppression rules for a resource in case the Option is sent by an authenticated client. If sent by an unauthenticated client, the Option can be used to expand the classes of responses suppressed compared to the default rules but not to reduce the classes of responses suppressed.

5.5. 6LoWPAN

In a 6LoWPAN network, a multicast IPv6 packet may be fragmented prior to transmission. A 6LoWPAN Router that forwards a fragmented packet can have a relatively high impact on the occupation of the wireless channel and on the memory load of the local node due to packet buffer occupation. For example, the MPL [RFC7731] protocol requires an MPL Forwarder to store the packet for a longer duration, to allow multiple forwarding transmissions to neighboring Forwarders. If only one of the fragments is not received correctly by an MPL Forwarder, the receiver needs to discard all received fragments and it needs to receive all the packet fragments again on a future occasion.

For these reasons, a fragmented IPv6 multicast packet is a possible attack vector in a Denial of Service (DoS) amplification attack. See Section 11.3 of [RFC7252] for more details on amplification. To mitigate the risk, applications sending multicast IPv6 requests to 6LoWPAN hosted CoAP servers SHOULD limit the size of the request to avoid 6LoWPAN fragmentation. A 6LoWPAN Router or multicast forwarder SHOULD deprioritize forwarding for multi-fragment 6LoWPAN multicast packets. Also, a 6LoWPAN Border Router SHOULD implement multicast packet filtering to prevent unwanted multicast traffic from entering a 6LoWPAN network from the outside. For example, it could filter out all multicast packet for which there is no known multicast listener on the 6LoWPAN network.

5.6. Wi-Fi

In a home automation scenario using Wi-Fi, Wi-Fi security should be enabled to prevent rogue nodes from joining. The Customer Premises Equipment (CPE) that enables access to the Internet should also have its IP multicast filters set so that it enforces multicast scope boundaries to isolate local multicast groups from the rest of the Internet (e.g., as per [RFC6092]). In addition, the scope of IP multicast transmissions and listeners should be site-local (5) or smaller. For site-local scope, the CPE will be an appropriate multicast scope boundary point.

5.7. Monitoring

5.7.1. General Monitoring

CoAP group communication can be used to control a set of related devices: for example, simultaneously turn on all the lights in a room. This intrinsically exposes the group to some unique monitoring risks that devices not in a group are not as vulnerable to. For example, assume an attacker is able to physically see a set of lights turn on in a room. Then the attacker can correlate an observed CoAP group communication message to the observed coordinated group action – even if the CoAP message is (partly) encrypted. This will give the attacker side-channel information to plan further attacks (e.g., by determining the members of the group some network topology information may be deduced).

5.7.2. Pervasive Monitoring

A key additional threat consideration for group communication is pervasive monitoring [RFC7258]. CoAP group communication solutions that are built on top of IP multicast need to pay particular heed to these dangers. This is because IP multicast is easier to intercept (and to secretly record) compared to IP unicast. Also, CoAP traffic is meant for the Internet of Things. This means that CoAP multicast may be used for the control and monitoring of critical infrastructure (e.g., lights, alarms, etc.) that may be prime targets for attack.

For example, an attacker may attempt to record all the CoAP traffic going over a smart grid (i.e., networked electrical utility) and try to determine critical nodes for further attacks. For example, the source node (controller) sends out CoAP group communication messages which easily identifies it as a controller. CoAP multicast traffic is inherently more vulnerable (compared to unicast) as the same packet may be replicated over many links, leading to a higher probability of packet capture by a pervasive monitoring system.

One mitigation is to restrict the scope of IP multicast to the minimal scope that fulfills the application need. Thus, for example, site-local IP multicast scope is always preferred over global scope IP multicast if this fulfills the application needs.

Even if all CoAP multicast traffic is encrypted/protected, an attacker may still attempt to capture this traffic and perform an off-line attack in the future.

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. Normative References

- [I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work in progress), September 2020.
- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-10 (work in progress), July 2020.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-10 (work in progress), November 2020.
- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", draft-ietf-cose-rfc8152bis-algs-12 (work in progress), September 2020.
- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", draft-ietf-cose-rfc8152bis-struct-14 (work in progress), September 2020.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [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>>.
- [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>>.

- [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>>.
- [RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <<https://www.rfc-editor.org/info/rfc4443>>.
- [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>>.
- [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>>.
- [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>>.

- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

7.2. Informative References

- [Californium]
Eclipse Foundation, "Eclipse Californium", March 2019, <<https://github.com/eclipse/californium/tree/2.0.x/californium-core/src/main/java/org/eclipse/californium/core>>.
- [Go-OCF] Open Connectivity Foundation (OCF), "Implementation of CoAP Server & Client in Go", March 2019, <<https://github.com/go-ocf/go-coap>>.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-09 (work in progress), November 2020.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-35 (work in progress), June 2020.
- [I-D.ietf-ace-oscore-gm-admin]
Tiloca, M., Hoeglund, R., Stok, P., Palombini, F., and K. Hartke, "Admin Interface for the OSCORE Group Manager", draft-ietf-ace-oscore-gm-admin-01 (work in progress), November 2020.
- [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-09 (work in progress), September 2019.

- [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-25 (work in progress), July 2020.
- [I-D.tiloca-core-groupcomm-proxy]
Tiloca, M. and E. Dijk, "Proxy Operations for CoAP Group Communication", draft-tiloca-core-groupcomm-proxy-02 (work in progress), November 2020.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-07 (work in progress), November 2020.
- [RFC6092] Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <<https://www.rfc-editor.org/info/rfc6092>>.
- [RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <<https://www.rfc-editor.org/info/rfc6550>>.
- [RFC6636] Asaeda, H., Liu, H., and Q. Wu, "Tuning the Behavior of the Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) for Routers in Mobile and Wireless Networks", RFC 6636, DOI 10.17487/RFC6636, May 2012, <<https://www.rfc-editor.org/info/rfc6636>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<https://www.rfc-editor.org/info/rfc7258>>.
- [RFC7346] Droms, R., "IPv6 Multicast Address Scopes", RFC 7346, DOI 10.17487/RFC7346, August 2014, <<https://www.rfc-editor.org/info/rfc7346>>.
- [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>>.

- [RFC7731] Hui, J. and R. Kelsey, "Multicast Protocol for Low-Power and Lossy Networks (MPL)", RFC 7731, DOI 10.17487/RFC7731, February 2016, <<https://www.rfc-editor.org/info/rfc7731>>.
- [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>>.
- [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>>.
- [RFC8710] Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for the Constrained Application Protocol (CoAP)", RFC 8710, DOI 10.17487/RFC8710, February 2020, <<https://www.rfc-editor.org/info/rfc8710>>.

Appendix A. Use Cases

To illustrate where and how CoAP-based group communication can be used, this section summarizes the most common use cases. These use cases include both secured and non-secured CoAP usage. Each subsection below covers one particular category of use cases for CoRE. Within each category, a use case may cover multiple application areas such as home IoT, commercial building IoT (sensing and control), industrial IoT/control, or environmental sensing.

A.1. Discovery

Discovery of physical devices in a network, or discovery of information entities hosted on network devices, are operations that are usually required in a system during the phases of setup or (re)configuration. When a discovery use case involves devices that need to interact without having been configured previously with a common security context, unsecured CoAP communication is typically used. Discovery may involve a request to a directory server, which provides services to aid clients in the discovery process. One particular type of directory server is the CoRE Resource Directory [I-D.ietf-core-resource-directory]; and there may be other types of directories that can be used with CoAP.

A.1.1. Distributed Device Discovery

Device discovery is the discovery and identification of networked devices - optionally only devices of a particular class, type, model, or brand. Group communication is used for distributed device discovery, if a central directory server is not used. Typically in distributed device discovery, a multicast request is sent to a particular address (or address range) and multicast scope of interest, and any devices configured to be discoverable will respond back. For the alternative solution of centralized device discovery a central directory server is accessed through unicast, in which case group communication is not needed. This requires that the address of the central directory is either preconfigured in each device or configured during operation using a protocol.

In CoAP, device discovery can be implemented by CoAP resource discovery requesting (GET) a particular resource that the sought device class, type, model or brand is known to respond to. It can also be implemented using CoAP resource discovery (Section 7 of [RFC7252]) and the CoAP query interface defined in Section 4 of [RFC6690] to find these particular resources. Also, a multicast GET request to /.well-known/core can be used to discover all CoAP devices.

A.1.2. Distributed Service Discovery

Service discovery is the discovery and identification of particular services hosted on network devices. Services can be identified by one or more parameters such as ID, name, protocol, version and/or type. Distributed service discovery involves group communication to reach individual devices hosting a particular service; with a central directory server not being used.

In CoAP, services are represented as resources and service discovery is implemented using resource discovery (Section 7 of [RFC7252]) and the CoAP query interface defined in Section 4 of [RFC6690].

A.1.3. Directory Discovery

This use case is a specific sub-case of Distributed Service Discovery (Appendix A.1.2), in which a device needs to identify the location of a Directory on the network to which it can e.g. register its own offered services, or to which it can perform queries to identify and locate other devices/services it needs to access on the network. Section 3.3 of [RFC7390] shows an example of discovering a CoRE Resource Directory using CoAP group communication. As defined in [I-D.ietf-core-resource-directory], a resource directory is a web entity that stores information about web resources and implements

REST interfaces for registration and lookup of those resources. For example, a device can register itself to a resource directory to let it be found by other devices and/or applications.

A.2. Operational Phase

Operational phase use cases describe those operations that occur most frequently in a networked system, during its operational lifetime and regular operation. Regular usage is when the applications on networked devices perform the tasks they were designed for and exchange of application-related data using group communication occurs. Processes like system reconfiguration, group changes, system/device setup, extra group security changes, etc. are not part of regular operation.

A.2.1. Actuator Group Control

Group communication can be beneficial to control actuators that need to act in synchrony, as a group, with strict timing (latency) requirements. Examples are office lighting, stage lighting, street lighting, or audio alert/Public Address systems. Sections 3.4 and 3.5 of [RFC7390] show examples of lighting control of a group of 6LoWPAN-connected lights.

A.2.2. Device Group Status Request

To properly monitor the status of systems, there may be a need for ad-hoc, unplanned status updates. Group communication can be used to quickly send out a request to a (potentially large) number of devices for specific information. Each device then responds back with the requested data. Those devices that did not respond to the request can optionally be polled again via reliable unicast communication to complete the dataset. The device group may be defined e.g. as "all temperature sensors on floor 3", or "all lights in wing B". For example, it could be a status request for device temperature, most recent sensor event detected, firmware version, network load, and/or battery level.

A.2.3. Network-wide Query

In some cases a whole network or subnet of multiple IP devices needs to be queried for status or other information. This is similar to the previous use case except that the device group is not defined in terms of its function/type but in terms of its network location. Technically this is also similar to distributed service discovery (Appendix A.1.2) where a query is processed by all devices on a network - except that the query is not about services offered by the device, but rather specific operational data is requested.

A.2.4. Network-wide / Group Notification

In some cases a whole network, or subnet of multiple IP devices, or a specific target group needs to be notified of a status change or other information. This is similar to the previous two use cases except that the recipients are not expected to respond with some information. Unreliable notification can be acceptable in some use cases, in which a recipient does not respond with a confirmation of having received the notification. In such a case, the receiving CoAP server does not have to create a CoAP response. If the sender needs confirmation of reception, the CoAP servers can be configured for that resource to respond with a 2.xx success status after processing a notification request successfully.

A.3. Software Update

Multicast can be useful to efficiently distribute new software (firmware, image, application, etc.) to a group of multiple devices. In this case, the group is defined in terms of device type: all devices in the target group are known to be capable of installing and running the new software. The software is distributed as a series of smaller blocks that are collected by all devices and stored in memory. All devices in the target group are usually responsible for integrity verification of the received software; which can be done per-block or for the entire software image once all blocks have been received. Due to the inherent unreliability of CoAP multicast, there needs to be a backup mechanism (e.g. implemented using CoAP unicast) by which a device can individually request missing blocks of a whole software image/entity. Prior to multicast software update, the group of recipients can be separately notified that there is new software available and coming, using the above network-wide or group notification.

Appendix B. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

B.1. Version -01 to -02

- o Clarified relation between security groups and application groups.
- o Considered also FETCH for requests over IP multicast.
- o More details on Observe re-registration.
- o More details on Proxy intermediaries.
- o More details on servers changing port number in the response.

- o Usage of the Uri-Host Option to indicate an application group.
- o Response suppression based on classes of error codes.

B.2. Version -00 to -01

- o Clarifications on group memberships for the different group types.
- o Simplified description of Token reusage, compared to the unicast case.
- o More details on the rationale for response suppression.
- o Clarifications of creation and management of security groups.
- o Clients more knowledgeable than proxies about stopping receiving responses.
- o Cancellation of group observations.
- o Clarification on multicast scope to use.
- o Both the group mode and pairwise mode of Group OSCORE are considered.
- o Updated security considerations.
- o Editorial improvements.

Acknowledgments

The authors sincerely thank Thomas Fossati and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Esko Dijk
IoTconsultancy.nl

Utrecht
Netherlands

Email: esko.dijk@iotconsultancy.nl

Chonggang Wang
InterDigital
1001 E Hector St, Suite 300
Conshohocken PA 19428
United States

Email: Chonggang.Wang@InterDigital.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

CoRE Working Group
Internet-Draft
Obsoletes: 7390 (if approved)
Updates: 7252, 7641 (if approved)
Intended status: Standards Track
Expires: 8 September 2022

E. Dijk
IoTconsultancy.nl
C. Wang
InterDigital
M. Tiloca
RISE AB
7 March 2022

Group Communication for the Constrained Application Protocol (CoAP)
draft-ietf-core-groupcomm-bis-06

Abstract

This document specifies the use of the Constrained Application Protocol (CoAP) for group communication, including the use of UDP/IP multicast as the default underlying data transport. Both unsecured and secured CoAP group communication are specified. Security is achieved by use of the Group Object Security for Constrained RESTful Environments (Group OSCORE) protocol. The target application area of this specification is any group communication use cases that involve resource-constrained devices or networks that support CoAP. This document replaces RFC 7390, while it updates RFC 7252 and RFC 7641.

Discussion Venues

This note is to be removed before publishing as an RFC.

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

Source for this draft and an issue tracker can be found at <https://github.com/core-wg/groupcomm-bis> (<https://github.com/core-wg/groupcomm-bis>).

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
1.1. Scope	5
1.2. Terminology	5
1.3. Changes to Other Documents	6
2. Group Definition and Group Configuration	7
2.1. Group Definition	7
2.1.1. CoAP Group	7
2.1.2. Application Group	7
2.1.3. Security Group	8
2.1.4. Relations Between Group Types	8
2.2. Group Configuration	11
2.2.1. Group Naming	11
2.2.2. Group Creation and Membership	17
2.2.3. Group Discovery	18
2.2.4. Group Maintenance	25
3. CoAP Usage in Group Communication	26
3.1. Request/Response Model	26
3.1.1. General	26
3.1.2. Response Suppression	27
3.1.3. Repeating a Request	27
3.1.4. Request/Response Matching and Distinguishing Responses	28
3.1.5. Token Reuse	28
3.1.6. Client Handling of Multiple Responses With Same Token	29

3.2.	Caching	30
3.2.1.	Freshness Model	31
3.2.2.	Validation Model	31
3.3.	URI Path Selection	32
3.4.	Port Selection for UDP Transport	33
3.5.	Proxy Operation	33
3.5.1.	Forward-Proxies	33
3.5.2.	Reverse-Proxies	35
3.6.	Congestion Control	37
3.7.	Observing Resources	38
3.8.	Block-Wise Transfer	40
3.9.	Transport Protocols	41
3.9.1.	UDP/IPv6 Multicast Transport	41
3.9.2.	UDP/IPv4 Multicast Transport	42
3.9.3.	6LoWPAN	42
3.9.4.	Other Transports	42
3.10.	Interworking with Other Protocols	43
3.10.1.	MLD/MLDv2/IGMP/IGMPv3	43
3.10.2.	RPL	43
3.10.3.	MPL	44
4.	Unsecured Group Communication	45
5.	Secured Group Communication using Group OSCORE	45
5.1.	Group OSCORE	45
5.2.	Secure Group Maintenance	47
5.3.	Proxy Security	48
6.	Security Considerations	49
6.1.	CoAP NoSec Mode	49
6.2.	Group OSCORE	50
6.2.1.	Group Key Management	51
6.2.2.	Source Authentication	51
6.2.3.	Countering Attacks	52
6.3.	Risk of Amplification	54
6.4.	Replay of Non-Confirmable Messages	55
6.5.	Use of CoAP No-Response Option	56
6.6.	6LoWPAN and MPL	56
6.7.	Wi-Fi	57
6.8.	Monitoring	57
6.8.1.	General Monitoring	57
6.8.2.	Pervasive Monitoring	58
7.	IANA Considerations	58
8.	References	58
8.1.	Normative References	58
8.2.	Informative References	61
Appendix A.	Use Cases	64
A.1.	Discovery	64
A.1.1.	Distributed Device Discovery	64
A.1.2.	Distributed Service Discovery	65
A.1.3.	Directory Discovery	65

A.2. Operational Phase	65
A.2.1. Actuator Group Control	65
A.2.2. Device Group Status Request	66
A.2.3. Network-wide Query	66
A.2.4. Network-wide / Group Notification	66
A.3. Software Update	66
Appendix B. Examples of Message Exchanges	67
Appendix C. Document Updates	73
C.1. Version -05 to -06	73
C.2. Version -04 to -05	74
C.3. Version -03 to -04	74
C.4. Version -02 to -03	74
C.5. Version -01 to -02	75
C.6. Version -00 to -01	75
Acknowledgments	76
Authors' Addresses	76

1. Introduction

This document specifies group communication using the Constrained Application Protocol (CoAP) [RFC7252], together with UDP/IP multicast as the default transport for CoAP group communication messages. CoAP is a RESTful communication protocol that is used in resource-constrained nodes, and in resource-constrained networks where packet sizes should be small. This area of use is summarized as Constrained RESTful Environments (CoRE).

One-to-many group communication can be achieved in CoAP, by a client using UDP/IP multicast data transport to send multicast CoAP request messages. In response, each server in the addressed group sends a response message back to the client over UDP/IP unicast. Notable CoAP implementations supporting group communication include the framework "Eclipse Californium" 2.0.x [Californium] from the Eclipse Foundation and the "Implementation of CoAP Server & Client in Go" [Go-OCF] from the Open Connectivity Foundation (OCF).

Both unsecured and secured CoAP group communication are specified in this document. Security is achieved by using Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm], which in turn builds on Object Security for Constrained Restful Environments (OSCORE) [RFC8613]. This method provides end-to-end application-layer security protection of CoAP messages, by using CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs].

This documents replaces and obsoletes [RFC7390], while it updates both [RFC7252] and [RFC7641]. A summary of the changes and additions to these documents is provided in Section 1.3.

All sections in the body of this document are normative, while appendices are informative. For additional background about use cases for CoAP group communication in resource-constrained devices and networks, see Appendix A.

1.1. Scope

For group communication, only those solutions that use CoAP messages over a "one-to-many" (i.e., non-unicast) transport protocol are in the scope of this document. There are alternative methods to achieve group communication using CoAP, using unicast only. One example is Publish-Subscribe [I-D.ietf-core-coap-pubsub] which uses a central broker server that CoAP clients access via unicast communication. These alternative methods may be usable for the same or similar use cases as the ones targeted in this document.

This document defines UDP/IP multicast as the default transport protocol for CoAP group requests, as in [RFC7252]. Other transport protocols (which may include broadcast, non-IP multicast, geocast, etc.) are not described in detail and are left for future work. Although UDP/IP multicast transport is assumed in most of the text in this document, we expect many of the considerations for UDP/IP multicast can be re-used for alternative transport protocols.

Furthermore, this document defines Group OSCORE [I-D.ietf-core-oscore-groupcomm] as the default group communication security solution for CoAP. Security solutions for group communication and configuration other than Group OSCORE are left for future work. General principles for secure group configuration are in scope.

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.

This specification requires readers to be familiar with CoAP terminology [RFC7252]. Terminology related to group communication is defined in Section 2.1.

In addition, the following terms are extensively used.

- * Group URI - This is defined as a CoAP URI that has the "coap" scheme and includes in the authority component either an IP multicast address or a group hostname (e.g., a Group Fully

Qualified Domain Name (FQDN)) that can be resolved to an IP multicast address. A group URI also contains an optional UDP port number in the authority component. Group URIs follow the regular CoAP URI syntax (see Section 6 of [RFC7252]).

- * Security material - This refers to any security keys, counters or parameters stored in a device that are required to participate in secure group communication with other devices.

1.3. Changes to Other Documents

This document obsoletes and replaces [RFC7390] as follows.

- * It defines separate definitions for CoAP groups, application groups and security groups, together with high-level guidelines on their configuration (see Section 2).
- * It defines the use of Group OSCORE [I-D.ietf-core-oscore-groupcomm] as the security protocol to protect group communication for CoAP, together with high-level guidelines on secure group maintenance (see Section 5).
- * It updates all the guidelines about using group communication for CoAP (see Section 3).
- * It strongly discourages unsecured group communication for CoAP based on the "NoSec" mode (see Section 4 and Section 6.1) and highlights the risk of amplification attacks (see Section 6.3).

This document updates [RFC7252] as follows.

- * It updates the request/response model for group communication, as to response suppression (see Section 3.1.2) and token reuse time (see Section 3.1.5).
- * It updates the freshness model and validation model to use for cached responses (see Section 3.2.1 and Section 3.2.2).

This document updates [RFC7641] as follows.

- * It defines the use of the CoAP Observe Option in group requests, for both the GET method and the FETCH method [RFC8132], together with normative behavior for both CoAP clients and CoAP servers (see Section 3.7).

2. Group Definition and Group Configuration

In the following, different group types are first defined in Section 2.1. Then, Group configuration, including group creation and maintenance by an application, user or commissioning entity is considered in Section 2.2.

2.1. Group Definition

Three types of groups and their mutual relations are defined in this section: CoAP group, application group, and security group.

2.1.1. CoAP Group

A CoAP group is defined as a set of CoAP endpoints, where each endpoint is configured to receive CoAP group messages that are sent to the group's associated IP multicast address and UDP port. That is, CoAP groups have relevance at the level of IP networks and CoAP endpoints.

An endpoint may be a member of multiple CoAP groups, by subscribing to multiple IP multicast addresses. A node may be a member of multiple CoAP groups, by hosting multiple CoAP server endpoints on different UDP ports. Group membership(s) of an endpoint or node may dynamically change over time. A node or endpoint sending a CoAP group message to a CoAP group is not necessarily itself a member of this CoAP group: it is a member only if it also has a CoAP endpoint listening on the group's associated IP multicast address and UDP port.

A CoAP group is identified by information encoded within a group URI. Further details on identifying a CoAP group are provided in Section 2.2.1.1.

2.1.2. Application Group

An application group is a set of CoAP server endpoints (hosted on different nodes) that share a common set of CoAP resources. That is, an application group has relevance at the application level. For example, an application group could denote all lights in an office room or all sensors in a hallway.

An endpoint may be a member of multiple application groups. A client endpoint that sends a group communication message to an application group is not necessarily itself a member of this application group.

There can be a one-to-one or a one-to-many relation between a CoAP group and application group(s). Such relations are discussed in more detail in Section 2.1.4.

An application group name may be explicitly encoded in the group URI of a CoAP request, for example in the URI path component. If this is not the case, the application group is implicitly derived by the receiver, e.g., based on information in the CoAP request or other contextual information. Further details on identifying an application group are provided in Section 2.2.1.2.

2.1.3. Security Group

For secure group communication, a security group is required. A security group comprises endpoints storing shared group security material, such that they can use it to protect and verify mutually exchanged messages.

That is, a client endpoint needs to be a member of a security group in order to send a valid secured group communication message to that group. A server endpoint needs to be a member of a security group in order to receive and correctly verify a secured group communication message sent to that group. An endpoint may be a member of multiple security groups.

There can be a many-to-many relation between security groups and CoAP groups, but often it is one-to-one. Also, there can be a many-to-many relation between security groups and application groups, but often it is one-to-one. Such relations are discussed in more detail in Section 2.1.4.

A special security group named "NoSec" identifies group communication without any security at the transport layer nor at the CoAP layer. Further details on identifying a security group are provided in Section 2.2.1.3.

2.1.4. Relations Between Group Types

Using the above group type definitions, a CoAP group communication message sent by an endpoint can be represented as a tuple that contains one instance of each group type:

(application group, CoAP group, security group)

A special note is appropriate about the possible relation between security groups and application groups.

On one hand, multiple application groups may use the same security group. Thus, the same group security material is used to protect the messages targeting any of those application groups. This has the benefit that typically less storage, configuration and updating are required for security material. In this case, a CoAP endpoint is supposed to know the exact application group to refer to for each message that is sent or received, based on, e.g., the used server port number, the targeted resource, or the content and structure of the message payload.

On the other hand, a single application group may use multiple security groups. Thus, different messages targeting the resources of the application group can be protected with different security material. This can be convenient, for example, if the security groups differ with respect to the cryptographic algorithms and related parameters they use. In this case, a CoAP client can join just one of the security groups, based on what it supports and prefers, while a CoAP server in the application group would rather have to join all of them.

Beyond this particular case, applications should be careful in associating a same application group to multiple security groups. In particular, it is NOT RECOMMENDED to use different security groups to reflect different access policies for resources in a same application group. That is, being a member of a security group actually grants access only to exchange secured messages and enables authentication of group members, while access control (authorization) to use resources in the application group belongs to a separate security domain. It has to be separately enforced by leveraging the resource properties or through dedicated access control credentials assessed by separate means.

Figure 1 summarizes the relations between the different types of groups described above in UML class diagram notation. The class attributes in square brackets are optionally defined.

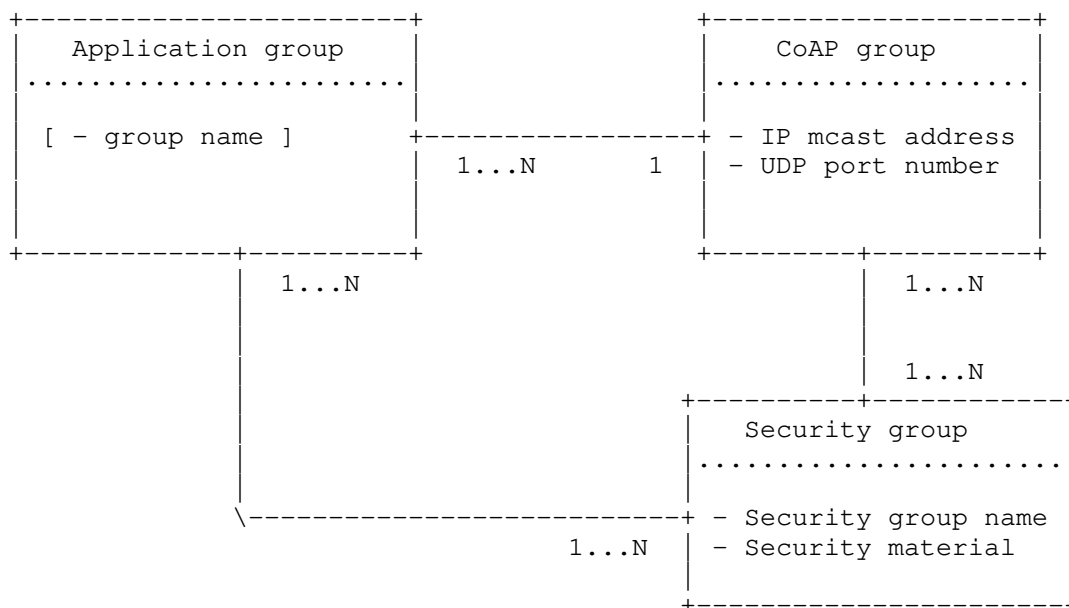


Figure 1: Relations Among Different Group Types

Figure 2 provides a deployment example of the relations between the different types of groups. It shows six CoAP servers (Srv1-Srv6) and their respective resources hosted (/resX). There are three application groups (1, 2, 3) and two security groups (1, 2). Security Group 1 is used by both Application Group 1 and 2. Three clients (Cli1, Cli2, Cli3) are configured with security material for Security Group 1. Two clients (Cli2, Cli4) are configured with security material for Security Group 2. All the shown application groups use the same CoAP group (not shown in the figure), i.e., one specific multicast IP address and UDP port number on which all the shown resources are hosted for each server.

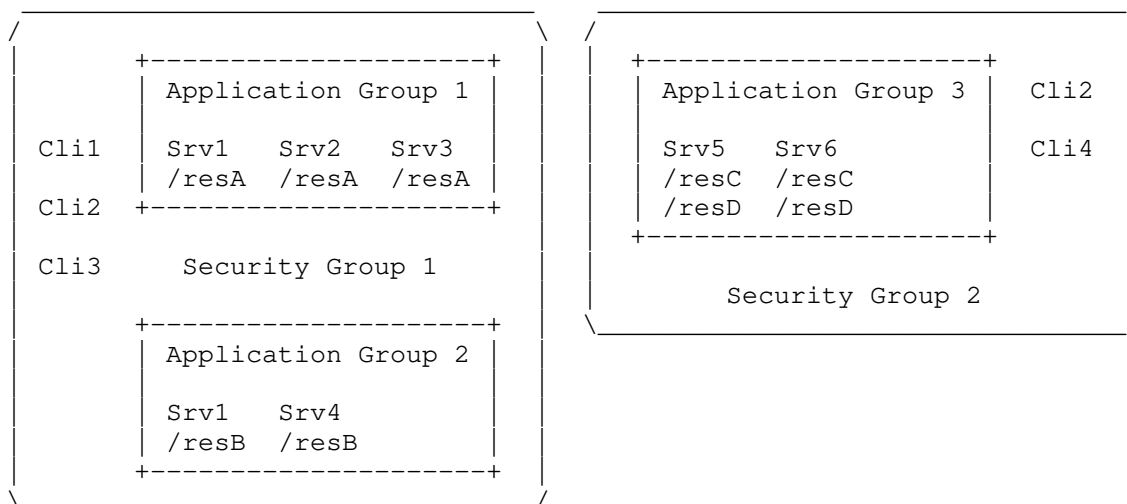


Figure 2: Deployment Example of Different Group Types

2.2. Group Configuration

The following defines how groups of different types are named, created, discovered and maintained.

2.2.1. Group Naming

Different types of group are named as specified below, separately for CoAP groups, application groups and security groups.

2.2.1.1. CoAP Groups

A CoAP group is identified and named by the authority component in the group URI (see Section 2.1.1), which includes the host subcomponent (possibly an IP multicast address literal) and an optional UDP port number. Note that the two authority components <HOST> and <HOST>:5683 both identify the same CoAP group, whose members listen to the CoAP default port number 5683.

When configuring a CoAP group membership, it is recommended to configure an endpoint with an IP multicast address literal, instead of a group hostname. This is because DNS infrastructure may not be deployed in many constrained networks. In case a group hostname is configured, it can be uniquely mapped to an IP multicast address via DNS resolution, if DNS client functionality is available in the endpoint being configured and the DNS service is supported in the network.

Examples of hierarchical CoAP group FQDN naming (and scoping) for a building control application were shown in Section 2.2 of [RFC7390].

2.2.1.2. Application Groups

An application group can be named in many ways through different types of identifiers, such as name string, (integer) number, URI or other types of string. The decision of whether and how exactly an application group name is encoded and transported is application specific.

The following defines a number of possible methods to use. The shown examples consider a CoAP group identified by the group hostname `grp.example.org`. Its members are CoAP servers listening to the associated IP multicast address `ff35:30:2001:db8:f1::8000:1` and port number 5685. Note that a group hostname is used here to have better-readable examples: in practice, an implementation would likely use an IP address literal as the host component of the Group URI in order to reduce the size of the CoAP request. In particular, the Uri-Host Option can be fully elided in this case. Also note that the Uri-Port Option does not appear in the examples, since the port number 5685 is already included in the CoAP request's UDP header (which is not shown in the examples).

An application group name can be explicitly encoded in a group URI. In such a case, it can be encoded within one of the following URI components.

- * URI path component - This is the most common and RECOMMENDED method to encode the application group name. When using this method in constrained networks, an application group name `GROUPNAME` should be as short as possible.

A best practice for doing so is to use a URI path component such that: i) it includes a path segment as delimiter with a designated value, e.g., `"gp"`, followed by ii) a path segment with value the name of the application group, followed by iii) the path segment(s) that identify the targeted resource within the application group. For example, both `/gp/GROUPNAME/res1` and `/base/gp/GROUPNAME/res1/res2` conform to this practice. Just like application group names, the path segment used as delimiter should be as short as possible in constrained networks.

A full-fledged example is provided in Figure 3. Another example of a compact CoAP request is provided in Figure 4, in which an IPv6 literal address and the default CoAP port number 5683 are used in the authority component. Also the resource structure is different in this example.

Application group name: gp1

Group URI: coap://grp.example.org:5685/gp/gp1/light?foo=bar

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: gp

Uri-Path: gp1

Uri-Path: light

Uri-Query: foo=bar

Figure 3: Example of application group name in URI path 1/2

Application group name: gp1

Group URI: coap://[ff35:30:2001:db8:f1::8000:1]/g/gp1/li

CoAP group request

Header: POST (T=NON, Code=0.02, MID=0x7d41)

Uri-Path: g

Uri-Path: gp1

Uri-Path: li

Figure 4: Example of application group name in URI path 2/2

- * URI query component - This method can use the following formats. In either case, when using this method in constrained networks, an application group name GROUPNAME should be as short as possible.

- As a first alternative, the URI query component consists of only one parameter, which has no value and has the name of the application group as its own identifier. That is, the query component ?GROUPNAME conforms to this format.

A full-fledged example is provided in Figure 5.

- As a second alternative, the URI query component includes a query parameter as designated indicator, e.g., "gp", with value the name of the application group. That is, assuming "gp" to be used as designated indicator, both the query components ?gp=GROUPNAME and ?par1=v1&gp=GROUPNAME conform to this format.

A full-fledged example is provided in Figure 6.

Application group name: gpl

Group URI: coap://grp.example.org:5685/light?gpl

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: gpl

Figure 5: Example of application group name in URI query (1/2)

Application group name: gpl

Group URI: coap://grp.example.org:5685/light?foo=bar&gp=gpl

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: foo=bar

Uri-Query: gp=gpl

Figure 6: Example of application group name in URI query (2/2)

- * URI authority component - If this method is used, the application group is identified by the authority component as a whole.

In particular, the application group has the same name of the CoAP group expressed by the group URI (see Section 2.2.1.1). Thus, this method can only be used if there is a one-to-one mapping between CoAP groups and application groups (see Section 2.1.4).

A full-fledged example is provided in Figure 7.

Application group name: grp.example.org:5685

Group URI: coap://grp.example.org:5685/light?foo=bar

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: foo=bar

Figure 7: Example of application group name in URI authority

- * URI host subcomponent - If this method is used, the application group is identified solely by the host subcomponent of the authority component.

Since an application group can be associated with only one CoAP group (see Section 2.1.4), using this method implies that any two CoAP groups cannot differ only by the port subcomponent of the URI authority component.

A full-fledged example is provided in Figure 8.

Application group name: grp.example.org

Group URI: coap://grp.example.org:5685/light?foo=bar

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: foo=bar

Figure 8: Example of application group name in URI host

- * URI port subcomponent - By using this method, the application group is uniquely identified by the destination port number encoded in the port subcomponent of the authority component.

Since an application group can be associated with only one CoAP group (see Section 2.1.4), using this method implies that any two CoAP groups cannot differ only by their host subcomponent of the URI authority component.

A full-fledged example is provided in Figure 9.

Application group name: grp1, as inferable from port number 5685

Group URI: coap://grp.example.org:5685/light?foo=bar

CoAP group request

Header: GET(T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: foo=bar

Figure 9: Example of application group name in URI port

Alternatively, there are also methods to encode the application group name within the CoAP request, even though it is not encoded within the group URI. An example of such method is summarized below.

- * The application group name can be encoded in a new (e.g., custom, application-specific) CoAP Option, which the client adds to the CoAP request before sending it out.

Upon receiving the request as a member of the targeted CoAP group, each CoAP server would, by design, understand this Option, decode it and treat the result as an application group name.

A full-fledged example is provided in Figure 10.

Application group name: grp1

Group URI: coap://grp.example.org:5685/light?foo=bar

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: foo=bar

App-Group-Name: grp1 // new (e.g., custom) CoAP option

Figure 10: Example of application group name in a new CoAP Option

Furthermore, it is possible to encode the application group name neither in the group URI nor within a CoAP request, thus yielding the most compact representation on the wire. In this case, each CoAP server needs to determine the right application group based on contextual information, such as the client identity and/or the target resource. For example, each application group on a server could support a unique set of resources, such that it does not overlap with the set of resources of any other application group.

Finally, Appendix A of [I-D.ietf-core-resource-directory] provides an example of application group registered to a Resource Directory (RD), along with the CoAP group it uses and the resources it supports. In that example, an application group name "lights" is encoded in the "ep" (endpoint) attribute of the RD registration entry. At the same time, the CoAP group is ff35:30:2001:db8:f1::8000:1 and the "NoSec" security group is used.

2.2.1.3. Security Groups

A security group is identified by a stable and invariant string used as group name. This is generally not related with other kinds of group identifiers that may be specific of the used security solution.

The "NoSec" security group name MUST be only used to represent the case of group communication without any security. This typically results in CoAP messages that do not include any security group name, identifier, or security-related data structures.

2.2.2. Group Creation and Membership

To create a CoAP group, a configuring entity defines an IP multicast address (or hostname) for the group and optionally a UDP port number in case it differs from the default CoAP port number 5683. Then, it configures one or more devices as listeners to that IP multicast address, with a CoAP endpoint listening on the group's associated UDP port. These endpoints/devices are the group members. The configuring entity can be, for example, a local application with pre-configuration, a user, a software developer, a cloud service, or a local commissioning tool. Also, the devices sending CoAP requests to the group in the role of CoAP client need to be configured with the same information, even though they are not necessarily group members. One way to configure a client is to supply it with a group URI. The IETF does not define a mandatory protocol to accomplish CoAP group creation. [RFC7390] defined an experimental protocol for configuration of group membership for unsecured group communication, based on JSON-formatted configuration resources. For IPv6 CoAP groups, common multicast address ranges that are used to configure group addresses from are `fflx::/16` and `ff3x::/16`.

To create an application group, a configuring entity may configure a resource (name) or a set of resources on CoAP endpoints, such that a CoAP request sent to a group URI by a configured CoAP client will be processed by one or more CoAP servers that have the matching URI path configured. These servers are the members of the application group.

To create a security group, a configuring entity defines an initial subset of the related security material. This comprises a set of group properties including the cryptographic algorithms and parameters used in the group, as well as additional information relevant throughout the group life-cycle, such as the security group name and description. This task MAY be entrusted to a dedicated administrator, that interacts with a Group Manager as defined in Section 5. After that, further security materials to protect group communications have to be generated, compatible with the specified group configuration.

To participate in a security group, CoAP endpoints have to be configured with the group security material used to protect communications in the associated application/CoAP groups. The part of the process that involves secure distribution of group security material MAY use standardized communication with a Group Manager as defined in Section 5. For unsecure group communication using the "NoSec" security group, any CoAP endpoint may become a group member at any time: there is no configuring entity that needs to provide security material for this group, as there is no security material for it. This means that group creation and membership cannot be tightly controlled for the "NoSec" group.

The configuration of groups and membership may be performed at different moments in the life-cycle of a device; for example during product (software) creation, in the factory, at a reseller, on-site during first deployment, or on-site during a system reconfiguration operation.

2.2.3. Group Discovery

The following describes how a CoAP endpoint can discover groups by different means, i.e., by using a Resource Directory or directly from the CoAP servers that are members of such groups.

2.2.3.1. Discovery through a Resource Directory

It is possible for CoAP endpoints to discover application groups as well as CoAP groups, by using the RD-Groups usage pattern of the CoRE Resource Directory (RD), as defined in Appendix A of [I-D.ietf-core-resource-directory].

In particular, an application group can be registered to the RD, specifying the reference IP multicast address of its associated CoAP group. The registration of groups to the RD is typically performed by a Commissioning Tool. Later on, CoAP endpoints can discover the registered application groups and related CoAP group(s), by using the lookup interface of the RD.

When secure communication is provided with Group OSCORE (see Section 5), the approach described in [I-D.tiloca-core-oscore-discovery] also based on the RD can be used, in order to discover the security group to join.

In particular, the responsible OSCORE Group Manager registers its security groups to the RD, as links to its own corresponding resources for joining the security groups [I-D.ietf-ace-key-groupcomm-oscore]. Later on, CoAP endpoints can discover the registered security groups and related application groups, by using the lookup interface of the RD, and then join the security group through the respective Group Manager.

2.2.3.2. Discovery from the CoAP Servers

It is possible for CoAP endpoints to discover application groups and CoAP groups from the CoAP servers that are members of such groups, by using a GET request targeting the `/.well-known/core` resource.

As shown below, such a GET request may be sent to the IP multicast address of an already known CoAP group associated with one or more application groups; or to the "All CoAP Nodes" multicast address, thus targeting all reachable CoAP servers in any CoAP group. Also, the GET request may specify a query component, in order to filter the application groups of interest.

These particular details concerning the GET request depend on the specific discovery action intended by the client and on application-specific means used to encode names of application groups and CoAP groups, e.g., in group URIs and/or CoRE target attributes used with resource links.

The following provides examples of methods to discover application groups and CoAP groups, building on the following assumptions. First, application group names are encoded in the path component of Group URIs (see Section 2.2.1.2), using the path segment "gp" as designated delimiter. Second, the type of an application group is encoded in the CoRE Link Format attribute "rt" of a group resource with a value "g.<GROUPTYPE>". Third, a CoAP group is used and is identified by the URI authority `grp.example.org:5685`.

- * A CoAP client can discover all the application groups associated with a specific CoAP group.

This is achieved by sending the GET request above to the IP multicast address of the CoAP group, and specifying a wildcarded group type "g._" as resource type in the URI query parameter "rt". For example, the request can use a Group URI with path and query components `/.well-known/core?rt=g._`, so that the query matches any application group resource type. Alternatively, the request can use a Group URI with path and query components `/.well-known/core?href=/gp/*`, so that the query matches any application group resources and also matches any sub-resources of those.

Through the corresponding responses, the query result is a list of resources at CoAP servers that are members of the specified CoAP group and have at least one application group associated with the CoAP group. That is, the client gains knowledge of: i) the set of servers that are members of the specified CoAP group and member of any of the associated application groups; ii) for each of those servers, the name of the application groups where the server is a member and that are associated with the CoAP group.

A full-fledged example is provided in Figure 11.

Each of the servers S1 and S2 is identified by the IP source address of the CoAP response. If the client wishes to discover resources that a particular server hosts within a particular application group, it may use unicast discovery request(s) to this server i.e., to its respective unicast IP address. Alternatively the client may use the discovered group resource type (e.g., `rt=g.light`) to infer which resources are present below the group resource.

```
// Request to all members of the CoAP group
Req: GET coap://grp.example.org:5685/.well-known/core?rt=g.*
```

```
// Response from server S1, as member of:
//   - The CoAP group "grp.example.org:5685"
//   - The application group "gp1"
Res: 2.05 Content
Content-Format: 40
Payload:
</gp/gp1>;rt=g.light
```

```
// Response from server S2, as member of:
//   - The CoAP group "grp.example.org:5685"
//   - The application groups "gp1" and "gp2"
Res: 2.05 Content
Content-Format: 40
Payload:
</gp/gp1>;rt=g.light,
</gp/gp2>;rt=g.temp
```

Figure 11: Discovery of application groups associated with a CoAP group

- * A CoAP client can discover the CoAP servers that are members of a specific application group, the CoAP group associated with the application group, and optionally the resources that those servers host for each application group.

This is achieved by sending the GET request above to the "All CoAP Nodes" IP multicast address (see Section 12.8 of [RFC7252]), with a particular chosen scope (e.g., site-local or realm-local) if IPv6 is used. Also, the request specifies the application group name of interest in the URI query component, as defined in Section 2.2.1.2. For example, the request can use a Group URI with path and query components `"/.well-known/core?href=/gp/gpl"` to specify the application group with name `"gpl"`.

Through the corresponding responses, the query result is a list of resources at CoAP servers that are members of the specified application group and for each application group the associated CoAP group. That is, the client gains knowledge of: i) the set of servers that are members of the specified application group and of the associated CoAP group; ii) for each of those servers, optionally the resources it hosts within the application group.

A full-fledged example is provided in Figure 12. Note that the servers need to respond now with an absolute URI and not a relative URI like in the previous example, because the group resource `"gpl"` is hosted at the authority indicated by the CoAP group (`grp.example.org:5685`) and not by the authority that is serving the Link Format document (which is `[ff03::fd]:5683`).

If the client wishes to discover resources that a particular server hosts within a particular application group, it may use unicast discovery request(s) to this server.

```
// Request to realm-local members of the application group "gpl"
Req: GET coap://[ff03::fd]/.well-known/core?href=/gp/gpl
```

```
// CoAP response from server S1, as member of:
// - The CoAP group "grp.example.org:5685"
// - The application group "gpl"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gpl>;rt=g.light
```

```
// CoAP response from server S2, as member of:
// - The CoAP group "grp.example.org:5685"
// - The application groups "gpl"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gpl>;rt=g.light
```

Figure 12: Discovery of members of an application group, together with the associated CoAP group

- * A CoAP client can discover the CoAP servers that are members of any application group of a specific type, the CoAP group associated with those application groups, and optionally the resources that those servers host as members of those application groups.

This is achieved by sending the GET request above to the "All CoAP Nodes" IP multicast address (see Section 12.8 of [RFC7252]), with a particular chosen scope (e.g., site-local or realm-local) if IPv6 is used. Also, the request can specify the application group type of interest in the URI query component as value of a query parameter "rt". For example, the request can use a Group URI with path and query components `"/.well-known/core?rt=TypeA"` to specify the application group type "TypeA".

Through the corresponding responses, the query result is a list of resources at CoAP servers that are members of any application group of the specified type and of the CoAP group associated with each of those application groups. That is, the client gains knowledge of: i) the set of servers that are members of the application groups of the specified type and of the associated CoAP group; ii) optionally for each of those servers, the resources it hosts within each of those application groups.

A full-fledged example is provided in Figure 13.

If the client wishes to discover resources that a particular server hosts within a particular application group, it may use unicast discovery request(s) to this server.

```

// Request to realm-local members of application groups
// with group type "g.temp"
Req: GET coap://[ff03::fd]/.well-known/core?rt=g.temp

// Response from server S1, as member of:
//   - The CoAP group "grp.example.org:5685"
//   - The application group "gp1" of type "g.temp"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=g.temp

// Response from server S2, as member of:
//   - The CoAP group "grp.example.org:5685"
//   - The application groups "gp1" and "gp2" of type "g.temp"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=g.temp,
<coap://grp.example.org:5685/gp/gp2>;rt=g.temp

```

Figure 13: Discovery of members of application groups of a specified type, and of the associated CoAP group

- * A CoAP client can discover the CoAP servers that are members of any application group configured in the 6LoWPAN wireless mesh network of the client, the CoAP group associated with each application group, and optionally the resources that those servers host as members of the application group.

This is achieved by sending the GET request above with a query specifying a wildcarded group type in the URI query parameter for "rt". For example, a Group URI with path and query components `"/.well-known/core?rt=g.*"`, so that the query matches any application group type. The request is sent to the "All CoAP Nodes" IP multicast address (see Section 12.8 of [RFC7252]), with a particular chosen scope if IPv6 is used. In this example the scope is realm-local to address all servers in the current 6LoWPAN wireless mesh network of the client.

Through the corresponding responses, the query result is a list of group resources hosted by any server in the mesh network. Each group resource denotes one application group membership of a server. The CoAP group per application group is obtained as the authority of each returned link.

A full-fledged example is provided in Figure 14.

If the client wishes to discover resources that a particular server hosts within a particular application group, it may use unicast discovery request(s) to this server.

```
// Request to realm-local members of any application group
Req: GET coap://[ff03::fd]/.well-known/core?rt=g.*

// Response from server S1, as member of:
//   - The CoAP groups "grp.example.org:5685" and "grp2.example.org"
//   - The application groups "gp1" and "gp5"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=g.light,
<coap://grp2.example.org/gp/gp5>;rt=g.lock

// Response from server S2, as member of:
//   - The CoAP group "grp.example.org:5685"
//   - The application groups "gp1" and "gp2"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=g.light,
<coap://grp.example.org:5685/gp/gp2>;rt=g.light

// Response from server S3, as member of:
//   - The CoAP group "grp2.example.org"
//   - The application group "gp5"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp2.example.org/gp/gp5>;rt=g.lock
```

Figure 14: Discovery of the resources and members of any application group, and of the associated CoAP group

Note that all the above examples are application-specific. That is, there is currently no standard way of encoding names of application groups and CoAP groups in group URIs and/or CoRE target attributes used with resource links. In particular, the discovery of groups through the RD mentioned in Section 2.2.3.1 is only defined for use with an RD, i.e., not directly with CoAP servers as group members.

For example, some applications may use the "rt" attribute on a parent resource to denote support for a particular REST API to access child resources, as detailed in the example in Figure 15. In this example a custom Link Format attribute "gpt" is used to denote the group type within the scope of the application/system. An alternative, shorter

encoding (not shown in the figure) is to use only the value "1" for each "gpt" attribute, to denote that the resource is of type application group. In that case information about the semantics/API of the group resource is disclosed only via the "rt" attribute as shown in the figure.

```
// Request to realm-local members of any application group
Req: GET coap://[ff03::fd]/.well-known/core?gpt=*

// Response from server S1, as member of:
//   - The CoAP groups "grp.example.org:5685" and "grp2.example.org"
//   - The application groups "gp1" and "gp5"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=oic.d.light;gpt=light,
<coap://grp2.example.org/gp/gp5>;rt=oic.d.smartlock;gpt=lock

// Response from server S2, as member of:
//   - The CoAP group "grp.example.org:5685"
//   - The application groups "gp1" and "gp2"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=oic.d.light;gpt=light,
<coap://grp.example.org:5685/gp/gp2>;rt=oic.d.light;gpt=light

// Response from server S3, as member of:
//   - The CoAP group "grp2.example.org"
//   - The application group "gp5"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp2.example.org/gp/gp5>;rt=oic.d.smartlock;gpt=lock
```

Figure 15: Example of using a custom 'gpt' link attribute to denote group type

2.2.4. Group Maintenance

Maintenance of a group includes any necessary operations to cope with changes in a system, such as: adding group members, removing group members, changing group security material, reconfiguration of UDP port number and/or IP multicast address, reconfiguration of the group URI, renaming of application groups, splitting of groups, or merging of groups.

For unsecured group communication (see Section 4) i.e., the "NoSec" security group, addition/removal of CoAP group members is simply done by configuring these devices to start/stop listening to the group IP multicast address on the group's UDP port.

For secured group communication (see Section 5), the maintenance operations of the protocol Group OSCORE [I-D.ietf-core-oscore-groupcomm] MUST be implemented. When using Group OSCORE, CoAP endpoints participating in group communication are also members of a corresponding OSCORE security group, and thus share common security material. Additional related maintenance operations are discussed in Section 5.2.

3. CoAP Usage in Group Communication

This section specifies the usage of CoAP in group communication, both unsecured and secured. This includes additional support for protocol extensions, such as Observe (see Section 3.7) and block-wise transfer (see Section 3.8).

How CoAP group messages are carried over various transport layers is the subject of Section 3.9. Finally, Section 3.10 covers the interworking of CoAP group communication with other protocols that may operate in the same network.

3.1. Request/Response Model

3.1.1. General

A CoAP client is an endpoint able to transmit CoAP requests and receive CoAP responses. Since the underlying UDP transport supports multiplexing by means of UDP port number, there can be multiple independent CoAP clients operational on a single host. On each UDP port, an independent CoAP client can be hosted. Each independent CoAP client sends requests that use the associated endpoint's UDP port number as the UDP source port number of the request.

All CoAP requests that are sent via IP multicast MUST be Non-confirmable, see Section 8.1 of [RFC7252]. The Message ID in an IP multicast CoAP message is used for optional message deduplication by both clients and servers, as detailed in Section 4.5 of [RFC7252]. A server sends back a unicast response to a CoAP group request. The unicast responses received by the CoAP client may be a mixture of success (e.g., 2.05 Content) and failure (e.g., 4.04 Not Found) codes, depending on the individual server processing results.

3.1.2. Response Suppression

A server MAY suppress its response for various reasons given in Section 8.2 of [RFC7252]. This document adds the requirement that a server SHOULD suppress the response in case of error or in case there is nothing useful to respond, unless the application related to a particular resource requires such a response to be made for that resource.

The CoAP No-Response Option [RFC7967] can be used by a client to influence the default response suppression on the server side. It is RECOMMENDED for a server to support this option only on selected resources where it is useful in the application context. If the option is supported on a resource, it MUST override the default response suppression of that resource.

Any default response suppression by a server SHOULD be performed consistently, as follows: if a request on a resource produces a particular Response Code and this response is not suppressed, then another request on the same resource that produces a response of the same Response Code class is also not suppressed. For example, if a 4.05 Method Not Allowed error response code is suppressed by default on a resource, then a 4.15 Unsupported Content-Format error response code is also suppressed by default for that resource.

3.1.3. Repeating a Request

A CoAP client MAY repeat a group request using the same Token value and same Message ID value, in order to ensure that enough (or all) group members have been reached with the request. This is useful in case a number of group members did not respond to the initial request and the client suspects that the request did not reach these group members. However, in case one or more servers did receive the initial request but the response to that request was lost, this repeat does not help to retrieve the lost response(s) if the server(s) implement the optional Message ID based deduplication (Section 4.5 of [RFC7252]).

A CoAP client MAY repeat a group request using the same Token value and a different Message ID, in which case all servers that received the initial request will again process the repeated request since it appears within a new CoAP message. This is useful in case a client suspects that one or more response(s) to its original request were lost and the client needs to collect more, or even all, responses from group members, even if this comes at the cost of the overhead of certain group members responding twice (once to the original request, and once to the repeated request with different Message ID).

3.1.4. Request/Response Matching and Distinguishing Responses

A CoAP client can distinguish the origin of multiple server responses by the source IP address of the message containing the CoAP response and/or any other available application-specific source identifiers contained in the CoAP response payload or CoAP response options, such as an application-level unique ID associated with the server. If secure communication is provided with Group OSCORE (see Section 5), additional security-related identifiers in the CoAP response enable the client to retrieve the right security material for decrypting each response and authenticating its source.

While processing a response on the client, the source endpoint of the response is not matched to the destination endpoint of the request, since for a group request these will never match. This is specified in Section 8.2 of [RFC7252], with reference to IP multicast.

Also, when UDP transport is used, this implies that a server MAY respond from a UDP port number that differs from the destination UDP port number of the request, although a CoAP server normally SHOULD respond from the UDP port number that equals the destination port number of the request -- following the convention for UDP-based protocols.

In case a single client has sent multiple group requests and concurrent CoAP transactions are ongoing, the responses received by that client are matched to an active request using only the Token value. Due to UDP level multiplexing, the UDP destination port number of the response MUST match to the client endpoint's UDP port number, i.e., to the UDP source port number of the client's request.

3.1.5. Token Reuse

For CoAP group requests, there are additional constraints on the reuse of Token values at the client, compared to the unicast case defined in [RFC7252] and updated by [RFC9175]. Since for CoAP group requests the number of responses is not bound a priori, the client cannot use the reception of a response as a trigger to "free up" a Token value for reuse.

Reusing a Token value too early could lead to incorrect response/request matching on the client, and would be a protocol error. Therefore, the time between reuse of Token values for different group requests MUST be greater than:

$$\text{MIN_TOKEN_REUSE_TIME} = (\text{NON_LIFETIME} + \text{MAX_LATENCY} + \text{MAX_SERVER_RESPONSE_DELAY})$$

where NON_LIFETIME and MAX_LATENCY are defined in Section 4.8 of [RFC7252]. This specification defines MAX_SERVER_RESPONSE_DELAY as was done in [RFC7390], that is: the expected maximum response delay over all servers that the client can send a CoAP group request to. This delay includes the maximum Leisure time period as defined in Section 8.2 of [RFC7252]. However, CoAP does not define a time limit for the server response delay. Using the default CoAP parameters, the Token reuse time MUST be greater than 250 seconds plus MAX_SERVER_RESPONSE_DELAY.

A preferred solution to meet this requirement is to generate a new unique Token for every new group request, such that a Token value is never reused. If a client has to reuse Token values for some reason, and also MAX_SERVER_RESPONSE_DELAY is unknown, then using MAX_SERVER_RESPONSE_DELAY = 250 seconds is a reasonable guideline. The time between Token reuses is in that case set to a value greater than MIN_TOKEN_REUSE_TIME = 500 seconds.

When securing CoAP group communication with Group OSCORE [I-D.ietf-core-oscore-groupcomm], secure binding between requests and responses is ensured (see Section 5). Thus, a client may reuse a Token value after it has been freed up, as discussed above and considering a reuse time greater than MIN_TOKEN_REUSE_TIME. If an alternative security protocol for CoAP group communication is used which does not ensure secure binding between requests and responses, a client MUST follow the Token processing requirements as defined in [RFC9175].

Another method to more easily meet the above constraint is to instantiate multiple CoAP clients at multiple UDP ports on the same host. The Token values only have to be unique within the context of a single CoAP client, so using multiple clients can make it easier to meet the constraint.

3.1.6. Client Handling of Multiple Responses With Same Token

Since a client sending a group request with a Token T will accept multiple responses with the same Token T, it is possible in particular that the same server sends multiple responses with the same Token T back to the client. For example, this server might not implement the optional CoAP message deduplication based on Message ID; or it might be acting out of specification as a malicious, compromised or faulty server.

When this happens, the client normally processes at the CoAP layer each of those responses to the same request coming from the same server. If the processing of a response is successful, the client delivers this response to the application as usual.

Then, the application is in a better position to decide what to do, depending on the available context information. For instance, it might accept and process all the responses from the same server, even if they are not Observe notifications (i.e., they do not include an Observe option). Alternatively, the application might accept and process only one of those responses, such as the most recent one from that server, e.g., when this can trigger a change of state within the application.

3.2. Caching

CoAP endpoints that are members of a CoAP group MAY cache responses to a group request as defined in Section 5.6 of [RFC7252]. In particular, these same rules apply to determine the set of request options used as "Cache-Key".

Furthermore, building on what is defined in Section 8.2.1 of [RFC7252]:

- * A client sending a GET or FETCH group request MAY update a cache with the responses from the servers in the CoAP group. Then, the client uses both cached-still-fresh and new responses as the result of the group request.
- * A client sending a GET or FETCH group request MAY use a response received from a server, to satisfy a subsequent sent request intended to that server on the related unicast request URI. In particular, the unicast request URI is obtained by replacing the authority component of the request URI with the transport-layer source address of the cached response message.
- * A client MAY revalidate a cached response by making a GET or FETCH request on the related unicast request URI.

Note that, in the presence of proxies, doing any of the above (optional) unicast requests requires the client to distinguish the different responses to a group request, as well as to distinguish the different origin servers that responded. This in turn requires additional means to provide the client with information about the origin server of each response, e.g., using the forward-proxying method defines in [I-D.tiloca-core-groupcomm-proxy].

The following subsections define the freshness model and validation model to use for cached responses, which update the models defined in Sections 5.6.1 and 5.6.2 of [RFC7252], respectively.

3.2.1. Freshness Model

For caching of group communication responses at client endpoints, the same freshness model relying on the Max-Age Option as defined in Section 5.6.1 of [RFC7252] applies, and the multicast caching rules of Section 8.2.1 of [RFC7252] apply except for the one discussed below.

In Section 8.2.1 of [RFC7252] it is stated that, regardless of the presence of cached responses to the group request, the client endpoint will always send out a new group request onto the network because new group members may have joined the group since the last group request to the same group/resource. That is, a request is never served from cached responses only. This document updates [RFC7252] by adding the following exception case, where a client endpoint MAY serve a request by using cached responses only, and not send out a new group request onto the network:

- * The client knows all current CoAP server group members; and, for each group member, the client's cache currently stores a fresh response.

How the client in the case above determines the current CoAP server group members is out of scope for this document. It may be, for example, via a group manager server, or by observing group join requests, or observing IGMP/MLD multicast group join messages, etc.

For caching at proxies, the freshness model defined in [I-D.tiloca-core-groupcomm-proxy] can be used.

3.2.2. Validation Model

For validation of cached group communication responses at client endpoints, the multicast validation rules in Section 8.2.1 of [RFC7252] apply, except for the last paragraph which states "A GET request to a multicast group MUST NOT contain an ETag option". This document updates [RFC7252] by allowing a group request to contain ETag Options as specified below.

For validation at proxies, the validation model defined in [I-D.tiloca-core-groupcomm-proxy] can be used.

3.2.2.1. ETag Option in a Group Request/Response

A client endpoint MAY include one or more ETag Options in a GET or FETCH group request to validate one or more stored responses it has cached. In case two or more servers in the group have responded to a previous request to the same resource with an identical ETag value, it is the responsibility of the client to handle this case. In particular, if the client wishes to validate, using a group request, a response from server 1 with an ETag value N, while it does not wish to validate a response from server 2 with the same ETag value N, there is no way to achieve this. In such cases of identical ETag values returned by two or more servers, the client, by default, SHOULD NOT include an ETag Option in a group request containing that ETag value.

A server endpoint MUST process an ETag Option in a GET or FETCH group request in the same way it processes an ETag Option for a unicast request. A server endpoint that includes an ETag Option in a response to a group request SHOULD construct the ETag Option value in such a way that the value will be unique to this particular server with a high probability. This can be done, for example, by embedding a compact ID of the server within the ETag value, where the ID is unique (or unique with a high probability) in the scope of the group.

Note: a legacy CoAP server might treat an ETag Option in a group request as an unrecognized option per Sections 5.4 and 8.2.1 of [RFC7252], causing it to ignore this (elective) ETag Option regardless of its value, and process the request normally as if that ETag Option was not included.

3.3. URI Path Selection

The URI Path used in a group request is preferably a path that is known to be supported across all group members. However there are valid use cases where a group request is known to be successful only for a subset of the CoAP group, for example only members of a specific application group, while those group members for which the request is unsuccessful (for example because they are outside the application group) either ignore the group request or respond with an error status code.

3.4. Port Selection for UDP Transport

A server that is a member of a CoAP group listens for CoAP request messages on the group's IP multicast address, usually on the CoAP default UDP port number 5683, or another non-default UDP port number if configured. Regardless of the method for selecting the port number, the same port number **MUST** be used across all CoAP servers that are members of a CoAP group and across all CoAP clients sending group requests to that group.

One way to create multiple CoAP groups is using different UDP ports with the same IP multicast address, in case the devices' network stack only supports a limited number of multicast address subscriptions. However, it must be taken into account that this incurs additional processing overhead on each CoAP server participating in at least one of these groups: messages to groups that are not of interest to the node are only discarded at the higher transport (UDP) layer instead of directly at the network (IP) layer. Also, a constrained network may be additionally burdened in this case with multicast traffic that is eventually discarded at the UDP layer by most nodes.

The port number 5684 is reserved for DTLS-secured unicast CoAP and **MUST NOT** be used for any CoAP group communication.

For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port number 5683 **MUST** be supported (see Section 7.1 of [RFC7252]) for the "All CoAP Nodes" multicast group as detailed in Section 3.9.

3.5. Proxy Operation

This section defines how proxies operate in a group communication scenario. In particular, Section 3.5.1 defines operations of forward-proxies, while Section 3.5.2 defines operations of reverse-proxies. Security operations for a proxy are discussed later in Section 5.3.

3.5.1. Forward-Proxies

CoAP enables a client to request a forward-proxy to process a CoAP request on its behalf, as described in Sections 5.7.2 and 8.2.2 of [RFC7252].

When intending to reach a CoAP group through a proxy, the client sends a unicast CoAP group request to the proxy. This request specifies the group URI where the request has to be forwarded to, either as a string in the Proxy-URI Option, or through the Proxy-

Scheme Option with the group URI constructed from the usual Uri-Options. Then, the forward-proxy resolves the group URI to a destination CoAP group, i.e., it sends (e.g., multicasts) the CoAP group request to the group URI, receives the responses and forwards all the individual (unicast) responses back to the client.

However, there are certain issues and limitations with this approach:

- * The CoAP client component that has sent the unicast CoAP group request to the proxy may be expecting only one (unicast) response, as usual for a CoAP unicast request. Instead, it receives multiple (unicast) responses, potentially leading to fault conditions in the component or to discarding any received responses following the first one. This issue may occur even if the application calling the CoAP client component is aware that the forward-proxy is going to forward the CoAP group request to the group URI.
- * Each individual CoAP response received by the client will appear to originate (based on its IP source address) from the CoAP Proxy, and not from the server that produced the response. This makes it impossible for the client to identify the server that produced each response, unless the server identity is contained as a part of the response payload or inside a CoAP option in the response.
- * The proxy does not necessarily know how many members there are in the CoAP group or how many group members will actually respond. Also, the proxy does not know for how long to collect responses before it stops forwarding them to the client. A CoAP client that is not using a Proxy might face the same problems in collecting responses to a group request. However, the client itself would typically have application-specific rules or knowledge on how to handle this situation, while an application-agnostic CoAP Proxy would typically not have this knowledge. For example, a CoAP client could monitor incoming responses and use this information to decide how long to continue collecting responses - which is something a proxy cannot do.

A forward-proxying method using this approach and addressing the issues raised above is defined in [I-D.tiloca-core-groupcomm-proxy].

An alternative solution is for the proxy to collect all the individual (unicast) responses to a CoAP group request and then send back only a single (aggregated) response to the client. However, this solution brings up new issues:

- * Like for the approach discussed above, the proxy does not know for how long to collect responses before sending back the aggregated response to the client. Analogous considerations apply to this approach too, both on the client and proxy side.
- * There is no default format defined in CoAP for aggregation of multiple responses into a single response. Such a format could be standardized based on, for example, the multipart content-format [RFC8710].

Due to the above issues, it is RECOMMENDED that a CoAP Proxy processes a request to be forwarded to a group URI only if it is explicitly enabled to do so. If such functionality is not explicitly enabled, the default response returned to the client is 5.01 Not Implemented. Furthermore, a proxy SHOULD be explicitly configured (e.g., by allow-listing and/or client authentication) to allow proxied CoAP group requests only from specific client(s).

The operation of HTTP-to-CoAP proxies for multicast CoAP requests is specified in Sections 8.4 and 10.1 of [RFC8075]. In this case, the "application/http" media type is used to let the proxy return multiple CoAP responses -- each translated to a HTTP response -- back to the HTTP client. Of course, in this case the HTTP client sending a group URI to the proxy needs to be aware that it is going to receive this format, and needs to be able to decode it into the responses of multiple CoAP servers. Also, the IP source address of each CoAP response cannot be determined anymore from the "application/http" response. The HTTP client still identifies the CoAP servers by other means such as application-specific information in the response payload.

3.5.2. Reverse-Proxies

CoAP enables the use of a reverse-proxy, as an endpoint that stands in for one or more other server(s), and satisfies requests on behalf of these, doing any necessary translations (see Section 5.7.3 of [RFC7252]).

In a group communication scenario, a reverse-proxy can rely on its configuration and/or on information in a request from a client, in order to determine that a group request has to be sent to a group of servers over a one-to-many transport such as IP/UDP multicast.

For example, specific resources on the reverse-proxy could be allocated, each to a specific application group and/or CoAP group. Or alternatively, the application group and/or CoAP group in question could be encoded as URI path segments. The URI path encodings for a reverse-proxy may also use a URI mapping template as described in Section 5.4 of [RFC8075].

Furthermore, the reverse-proxy can actually stand in for (and thus prevent to directly reach) only the whole set of servers in the group, or also for each of those individual servers (e.g., if acting as firewall).

For a reverse-proxy that sends a request to a group of servers, the considerations as defined in Section 5.7.3 of [RFC7252] hold, with the following additions:

- * The three issues and limitations defined in Section 3.5.1 for a forward proxy apply to a reverse-proxy as well, and have to be addressed, e.g., using the signaling method defined in [I-D.tiloca-core-groupcomm-proxy] or other means.
- * A reverse-proxy MAY have preconfigured time duration(s) that are used for the collecting of server responses and forwarding these back to the client. These duration(s) may be set as global configuration or resource-specific configurations. If there is such preconfiguration, then an explicit signaling of the time period in the client's request as defined in [I-D.tiloca-core-groupcomm-proxy] is not necessarily needed.
- * A client that is configured to access a reverse-proxy resource (i.e., one that triggers a CoAP group communication request) SHOULD be configured also to handle potentially multiple responses with the same Token value caused by a single request.

That is, the client needs to preserve the Token value used for the request also after the reception of the first response forwarded back by the proxy (see Section 3.1.6) and keep the request open to potential further responses with this Token. This requirement can be met by a combination of client implementation and proper proxied group communication configuration on the client.

- * A client might re-use a Token value in a valid new request to the reverse-proxy, while the reverse-proxy still has an ongoing group communication request for this client with the same Token value (i.e., its time period for response collection has not ended yet).

If this happens, the reverse-proxy MUST stop the ongoing request and associated response forwarding, it MUST NOT forward the new request to the group of servers, and it MUST send a 4.00 Bad Request error response to the client. The diagnostic payload of the error response SHOULD indicate to the client that the resource is a reverse-proxy resource, and that for this reason immediate Token re-use is not possible.

If the reverse-proxy supports the signalling protocol of [I-D.tiloca-core-groupcomm-proxy] it can include a Multicast-Signaling Option in the error response to convey the reason for the error in a machine-readable way.

For the operation of HTTP-to-CoAP reverse proxies, see the last paragraph of Section 3.5.1 which applies also to the case of reverse-proxies.

3.6. Congestion Control

CoAP group requests may result in a multitude of responses from different nodes, potentially causing congestion. Therefore, both the sending of CoAP group requests and the sending of the unicast CoAP responses to these group requests should be conservatively controlled.

CoAP [RFC7252] reduces IP multicast-specific congestion risks through the following measures:

- * A server may choose not to respond to an IP multicast request if there is nothing useful to respond to, e.g., error or empty response (see Section 8.2 of [RFC7252]).
- * A server should limit the support for IP multicast requests to specific resources where multicast operation is required (Section 11.3 of [RFC7252]).
- * An IP multicast request MUST be Non-confirmable (Section 8.1 of [RFC7252]).
- * A response to an IP multicast request SHOULD be Non-confirmable (Section 5.2.3 of [RFC7252]).
- * A server does not respond immediately to an IP multicast request and should first wait for a time that is randomly picked within a predetermined time interval called the Leisure (Section 8.2 of [RFC7252]).

This document also defines these measures to be applicable to alternative transports (other than IP multicast), if not defined otherwise. Additional guidelines to reduce congestion risks defined in this document are as follows:

- * A server in a constrained network SHOULD only support group requests for resources that have a small representation (where the representation may be retrieved via a GET, FETCH or POST method in the request). For example, "small" can be defined as a response payload limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size, so that it fits into a single link-layer frame in case IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN, see Section 3.9.3) is used on the constrained network.
- * A server SHOULD minimize the payload size of a response to a group GET or FETCH request on `"/.well-known/core"` by using hierarchy in arranging link descriptions for the response. An example of this is given in Section 5 of [RFC6690].
- * A server MAY minimize the payload size of a response to a group GET or FETCH request (e.g., on `"/.well-known/core"`) by using CoAP block-wise transfers [RFC7959] in case the payload is long, returning only a first block of the CoRE Link Format description. For this reason, a CoAP client sending a CoAP group request to `"/.well-known/core"` SHOULD support block-wise transfers. See also Section 3.8.
- * A client SHOULD be configured to use CoAP groups with the smallest possible IP multicast scope that fulfills the application needs. As an example, site-local scope is always preferred over global scope IP multicast if this fulfills the application needs. Similarly, realm-local scope is always preferred over site-local scope if this fulfills the application needs.

3.7. Observing Resources

The CoAP Observe Option [RFC7641] is a protocol extension of CoAP, that allows a CoAP client to retrieve a representation of a resource and automatically keep this representation up-to-date over a longer period of time. The client gets notified when the representation has changed. [RFC7641] does not mention whether the Observe Option can be combined with CoAP (multicast) group communication.

This section updates [RFC7641] with the use of the Observe Option in a CoAP GET group request, and defines normative behavior for both client and server. Consistent with Section 2.4 of [RFC8132], it is also possible to use the Observe Option in a CoAP FETCH group request.

Multicast Observe is a useful way to start observing a particular resource on all members of a CoAP group at the same time. Group members that do not have this particular resource or do not allow the GET or FETCH method on it will either respond with an error status -- 4.04 Not Found or 4.05 Method Not Allowed, respectively -- or will silently suppress the response following the rules of Section 3.1.2, depending on server-specific configuration.

A client that sends a group GET or FETCH request with the Observe Option MAY repeat this request using the same Token value and the same Observe Option value, in order to ensure that enough (or all) members of the CoAP group have been reached with the request. This is useful in case a number of group members did not respond to the initial request. The client MAY additionally use the same Message ID in the repeated request to avoid that group members that had already received the initial request would respond again. Note that using the same Message ID in a repeated request will not be helpful in case of loss of a response message, since the server that responded already will consider the repeated request as a duplicate message. On the other hand, if the client uses a different, fresh Message ID in the repeated request, then all the group members that receive this new message will typically respond again, which increases the network load.

A client that has sent a group GET or FETCH request with the Observe Option MAY follow up by sending a new unicast CON request with the same Token value and same Observe Option value to a particular server, in order to ensure that the particular server receives the request. This is useful in case a specific group member, that was expected to respond to the initial group request, did not respond to the initial request. In this case, the client MUST use a Message ID that differs from the initial group request message.

Furthermore, consistent with Section 3.3.1 of [RFC7641] and following its guidelines, a client MAY at any time send a new group/multicast GET or FETCH request with the same Token value and same Observe Option value as the original request. This allows the client to verify that it has an up-to-date representation of an observed resource and/or to re-register its interest to observe a resource.

In the above client behaviors, the Token value is kept identical to the initial request to avoid that a client is included in more than one entry in the list of observers (Section 4.1 of [RFC7641]).

Before repeating a request as specified above, the client SHOULD wait for at least the expected round-trip time plus the Leisure time period defined in Section 8.2 of [RFC7252], to give the server time to respond.

A server that receives a GET or FETCH request with the Observe Option, for which request processing is successful, SHOULD respond to this request and not suppress the response. If a server adds a client (as a new entry) to the list of observers for a resource due to an Observe request, the server SHOULD respond to this request and SHOULD NOT suppress the response. An exception to the above is the overriding of response suppression according to a CoAP No-Response Option [RFC7967] specified by the client in the GET or FETCH request (see Section 3.1.2).

A server SHOULD have a mechanism to verify liveness of its observing clients and the continued interest of these clients in receiving the observe notifications. This can be implemented by sending notifications occasionally using a Confirmable message (see Section 4.5 of [RFC7641] for details). This requirement overrides the regular behavior of sending Non-confirmable notifications in response to a Non-confirmable request.

A client can use the unicast cancellation methods of Section 3.6 of [RFC7641] and stop the ongoing observation of a particular resource on members of a CoAP group. This can be used to remove specific observed servers, or even all servers in the group (using serial unicast to each known group member). In addition, a client MAY explicitly deregister from all those servers at once, by sending a group/multicast GET or FETCH request that includes the Token value of the observation to be cancelled and includes an Observe Option with the value set to 1 (deregister). In case not all the servers in the CoAP group received this deregistration request, either the unicast cancellation methods can be used at a later point in time or the group/multicast deregistration request MAY be repeated upon receiving another observe response from a server.

For observing a group of servers through a CoAP-to-CoAP proxy, the limitations stated in Section 3.5 apply. The method defined in [I-D.tiloca-core-groupcomm-proxy] enables group communication including resource observation through proxies and addresses those limitations.

3.8. Block-Wise Transfer

Section 2.8 of [RFC7959] specifies how a client can use block-wise transfer (Block2 Option) in a multicast GET request to limit the size of the initial response of each server. Consistent with Section 2.5 of [RFC8132], the same can be done with a multicast FETCH request.

The client has to use unicast for any further request, separately addressing each different server, in order to retrieve more blocks of the resource from that server, if any. Also, a server (member of a

targeted CoAP group) that needs to respond to a group request with a particularly large resource can use block-wise transfer (Block2 Option) at its own initiative, to limit the size of the initial response. Again, a client would have to use unicast for any further requests to retrieve more blocks of the resource.

A solution for group/multicast block-wise transfer using the Block1 Option is not specified in [RFC7959] nor in the present document. Such a solution would be useful for group FETCH/PUT/POST/PATCH/iPATCH requests, to efficiently distribute a large request payload as multiple blocks to all members of a CoAP group. Multicast usage of Block1 is non-trivial due to potential message loss (leading to missing blocks or missing confirmations), and potential diverging block size preferences of different members of the CoAP group.

[I-D.ietf-core-new-block] specifies an alternative method for CoAP block-wise transfer. It specifies that "servers MUST ignore multicast requests that contain the Q-Block2 Option".

3.9. Transport Protocols

In this document UDP, both over IPv4 and IPv6, is considered as the default transport protocol for CoAP group communication.

3.9.1. UDP/IPv6 Multicast Transport

CoAP group communication can use UDP over IPv6 as a transport protocol, provided that IPv6 multicast is enabled. IPv6 multicast MAY be supported in a network only for a limited scope. For example, Section 3.10.2 describes the potential limited support of RPL for multicast, depending on how the protocol is configured.

For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port number 5683 MUST be supported as per Sections 7.1 and 12.8 of [RFC7252] for the "All CoAP Nodes" multicast group. An IPv6 CoAP server SHOULD support the "All CoAP Nodes" groups with at least link-local (2), admin-local (4) and site-local (5) scopes. An IPv6 CoAP server on a 6LoWPAN node (see Section 3.9.3) SHOULD also support the realm-local (3) scope.

Note that a client sending an IPv6 multicast CoAP message to a port number that is not supported by the server will not receive an ICMPv6 Port Unreachable error message from that server, because the server does not send it in this case, per Section 2.4 of [RFC4443].

3.9.2. UDP/IPv4 Multicast Transport

CoAP group communication can use UDP over IPv4 as a transport protocol, provided that IPv4 multicast is enabled. For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port number 5683 MUST be supported as per Sections 7.1 and 12.8 of [RFC7252], for the "All CoAP Nodes" IPv4 multicast group.

Note that a client sending an IPv4 multicast CoAP message to a port number that is not supported by the server will not receive an ICMP Port Unreachable error message from that server, because the server does not send it in this case, per Section 3.2.2 of [RFC1122].

3.9.3. 6LoWPAN

In 6LoWPAN [RFC4944] [RFC6282] networks, IPv6 packets (up to 1280 bytes) may be fragmented into smaller IEEE 802.15.4 MAC frames (up to 127 bytes), if the packet size requires this. Every 6LoWPAN IPv6 router that receives a multi-fragment packet reassembles the packet and refragments it upon transmission. Since the loss of a single fragment implies the loss of the entire IPv6 packet, the performance in terms of packet loss and throughput of multi-fragment multicast IPv6 packets is typically far worse than the performance of single-fragment IPv6 multicast packets. For this reason, a CoAP request sent over multicast in 6LoWPAN networks SHOULD be sized in such a way that it fits in a single IEEE 802.15.4 MAC frame, if possible.

On 6LoWPAN networks, multicast groups can be defined with realm-local scope [RFC7346]. Such a realm-local group is restricted to the local 6LoWPAN network/subnet. In other words, a multicast request to that group does not propagate beyond the 6LoWPAN network segment where the request originated. For example, a multicast discovery request can be sent to the realm-local "All CoAP Nodes" IPv6 multicast group (see Section 3.9.1) in order to discover only CoAP servers on the local 6LoWPAN network.

3.9.4. Other Transports

CoAP group communication may be used over transports other than UDP/IP multicast. For example broadcast, non-UDP multicast, geocast, serial unicast, etc. In such cases the particular considerations for UDP/IP multicast in this document may need to be applied to that particular transport.

Because it supports unicast only, [RFC8323] (CoAP over TCP, TLS, and WebSockets) is not in scope as a transport for CoAP group communication.

3.10. Interworking with Other Protocols

3.10.1. MLD/MLDv2/IGMP/IGMPv3

CoAP nodes that are IP hosts (i.e., not IP routers) are generally unaware of the specific IP multicast routing/forwarding protocol being used in their network. When such a host needs to join a specific (CoAP) multicast group, it requires a way to signal to IP multicast routers which IP multicast address(es) it needs to listen to.

The MLDv2 protocol [RFC3810] is the standard IPv6 method to achieve this; therefore, this method SHOULD be used by members of a CoAP group to subscribe to its multicast IPv6 address, on IPv6 networks that support it. CoAP server nodes then act in the role of MLD Multicast Address Listener. MLDv2 uses link-local communication between Listeners and IP multicast routers. Constrained IPv6 networks that implement either RPL (see Section 3.10.2) or MPL (see Section 3.10.3) typically do not support MLDv2 as they have their own mechanisms defined for subscribing to multicast groups.

The IGMPv3 protocol [RFC3376] is the standard IPv4 method to signal multicast group subscriptions. This SHOULD be used by members of a CoAP group to subscribe to its multicast IPv4 address on IPv4 networks.

The guidelines from [RFC6636] on the tuning of MLD for mobile and wireless networks may be useful when implementing MLD in constrained networks.

3.10.2. RPL

RPL [RFC6550] is an IPv6 based routing protocol suitable for low-power, lossy networks (LLNs). In such a context, CoAP is often used as an application protocol.

If only RPL is used in a network for routing and its optional multicast support is disabled, there will be no IP multicast routing available. Any IPv6 multicast packets in this case will not propagate beyond a single hop (to direct neighbors in the LLN). This implies that any CoAP group request will be delivered to link-local nodes only, for any scope value ≥ 2 used in the IPv6 destination address.

RPL supports (see Section 12 of [RFC6550]) advertisement of IP multicast destinations using Destination Advertisement Object (DAO) messages and subsequent routing of multicast IPv6 packets based on this. It requires the RPL mode of operation to be 3 (Storing mode with multicast support).

In this mode, RPL DAO can be used by a CoAP node that is either an RPL router or RPL Leaf Node, to advertise its CoAP group membership to parent RPL routers. Then, RPL will route any IP multicast CoAP requests over multiple hops to those CoAP servers that are group members.

The same DAO mechanism can be used to convey CoAP group membership information to an edge router (e.g., 6LBR), in case the edge router is also the root of the RPL Destination-Oriented Directed Acyclic Graph (DODAG). This is useful because the edge router then learns which IP multicast traffic it needs to pass through from the backbone network into the LLN subnet, and which traffic not. In LLNs, such ingress filtering helps to avoid congestion of the resource-constrained network segment, due to IP multicast traffic from the high-speed backbone IP network.

3.10.3. MPL

The Multicast Protocol for Low-Power and Lossy Networks (MPL) [RFC7731] can be used for propagation of IPv6 multicast packets throughout a defined network domain, over multiple hops. MPL is designed to work in LLNs and can operate alone or in combination with RPL. The protocol involves a predefined group of MPL Forwarders to collectively distribute IPv6 multicast packets throughout their MPL Domain. An MPL Forwarder may be associated with multiple MPL Domains at the same time. Non-Forwarders will receive IPv6 multicast packets from one or more of their neighboring Forwarders. Therefore, MPL can be used to propagate a CoAP multicast group request to all group members.

However, a CoAP multicast request to a group that originated outside of the MPL Domain will not be propagated by MPL - unless an MPL Forwarder is explicitly configured as an ingress point that introduces external multicast packets into the MPL Domain. Such an ingress point could be located on an edge router (e.g., 6LBR). Methods to configure which multicast groups are to be propagated into the MPL Domain could be:

- * Manual configuration on each ingress MPL Forwarder.

- * MLDv2 protocol, which works only in case all CoAP servers joining a group are in link-local communication range of an ingress MPL Forwarder. This is typically not the case on mesh networks.
- * A new/custom protocol to register multicast groups at an ingress MPL Forwarder. This could be for example a CoAP-based protocol offering multicast group subscription features similar to MLDv2.

For security and performance reasons also other filtering criteria may be defined at an ingress MPL Forwarder. See Section 6.6 for more details.

4. Unsecured Group Communication

CoAP group communication can operate in CoAP NoSec (No Security) mode, without using application-layer and transport-layer security mechanisms. The NoSec mode uses the "coap" scheme, and is defined in Section 9 of [RFC7252]. The conceptual "NoSec" security group as defined in Section 2.1 is used for unsecured group communication.

It is NOT RECOMMENDED to use CoAP group communication in NoSec mode, even in case of non-sensitive and non-critical applications.

Before possibly and exceptionally using the NoSec mode, the security implications in Section 6.1 must be very well considered and understood, especially as to the risk and impact of amplification attacks (see Section 6.3).

5. Secured Group Communication using Group OSCORE

This section defines how CoAP group communication can be secured. In particular, Section 5.1 describes how the Group OSCORE security protocol [I-D.ietf-core-oscore-groupcomm] can be used to protect messages exchanged in a CoAP group, while Section 5.2 provides guidance on required maintenance operations for OSCORE groups used as security groups.

5.1. Group OSCORE

The application-layer protocol Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] provides end-to-end encryption, integrity and replay protection of CoAP messages exchanged between two CoAP endpoints. These can act both as CoAP Client as well as CoAP Server, and share an OSCORE Security Context used to protect and verify exchanged messages. The use of OSCORE does not affect the URI scheme and OSCORE can therefore be used with any URI scheme defined for CoAP.

OSCORE uses COSE [I-D.ietf-cose-rfc8152bis-struct] [I-D.ietf-cose-rfc8152bis-algs] to perform encryption operations and protect a CoAP message carried in a COSE object, by using an Authenticated Encryption with Associated Data (AEAD) algorithm. In particular, OSCORE takes as input an unprotected CoAP message and transforms it into a protected CoAP message transporting the COSE object.

OSCORE makes it possible to selectively protect different parts of a CoAP message in different ways, while still allowing intermediaries (e.g., CoAP proxies) to perform their intended functionalities. That is, some message parts are encrypted and integrity protected; other parts are only integrity protected to be accessible to, but not modifiable by, proxies; and some parts are kept as plain content to be both accessible to and modifiable by proxies. Such differences especially concern the CoAP options included in the unprotected message.

Group OSCORE [I-D.ietf-core-oscore-groupcomm] builds on OSCORE, and provides end-to-end security of CoAP messages exchanged between members of an OSCORE group, while fulfilling the same security requirements.

In particular, Group OSCORE protects CoAP group requests sent by a CoAP client, e.g., over UDP/IP multicast, as well as multiple corresponding CoAP responses sent as (IP) unicast by different CoAP servers. However, the same security material can also be used to protect CoAP requests sent over (IP) unicast to a single CoAP server in the OSCORE group, as well as the corresponding responses.

Group OSCORE ensures source authentication of all messages exchanged within the OSCORE group, by means of two possible methods.

The first method, called group mode, relies on digital signatures. That is, sender devices sign their outgoing messages using their own private key, and embed the signature in the protected CoAP message.

The second method, called pairwise mode, relies on a symmetric key, which is derived from a pairwise shared secret computed from the asymmetric keys of the message sender and recipient. This method is intended for one-to-one messages sent in the group, such as all responses individually sent by servers, as well as requests addressed to an individual server.

A Group Manager is responsible for managing one or multiple OSCORE groups. In particular, the Group Manager acts as repository of the group members' authentication credentials including the corresponding public keys; manages, renews and provides security material in the group; and handles the join process of new group members.

As defined in [I-D.ietf-ace-oscore-gm-admin], an administrator entity can interact with the Group Manager to create OSCORE groups and specify their configuration (see Section 2.2.2). During the lifetime of the OSCORE group, the administrator can further interact with the Group Manager, in order to possibly update the group configuration and eventually delete the group.

As recommended in [I-D.ietf-core-oscore-groupcomm], a CoAP endpoint can join an OSCORE group by using the method described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

A CoAP endpoint can discover OSCORE groups and retrieve information to join them through their respective Group Managers by using the method described in [I-D.ietf-core-oscore-discovery] and based on the CoRE Resource Directory [I-D.ietf-core-resource-directory].

If security is required, CoAP group communication as described in this specification MUST use Group OSCORE. In particular, a CoAP group as defined in Section 2.1 and using secure group communication is associated with an OSCORE security group, which includes:

- * All members of the CoAP group, i.e., the CoAP endpoints configured to receive CoAP group messages sent to the particular group and -- in case of IP multicast transport -- are listening to the group's multicast IP address on the group's UDP port.
- * All further CoAP endpoints configured only as CoAP clients, that may send CoAP group requests to the CoAP group.

5.2. Secure Group Maintenance

As part of group maintenance operations (see Section 2.2.4), additional key management operations are required for an OSCORE group, also depending on the security requirements of the application (see Section 6.2.1). Specifically:

- * Adding new members to a CoAP group or enabling new client-only endpoints to interact with that group require also that each of such members/endpoints join the corresponding OSCORE group. When this happens, they are securely provided with the security material to use in that OSCORE group.

Applications may need backward security. That is, they may require that, after having joined an OSCORE group, a new group member cannot access messages exchanged in the group prior to its joining, even if it has recorded them.

In such a case, new security material to use in the OSCORE group has first to be generated and distributed to the current members of that group, before new endpoints are also provided with that new security material upon their joining.

- * Removing members from a CoAP group or stopping client-only endpoints from interacting with that group requires removing such members/endpoints from the corresponding OSCORE group. To this end, new security material is generated and securely distributed only to the remaining members of the OSCORE group, together with the list of former members removed from that group.

This ensures forward security in the OSCORE group. That is, it ensures that only the members intended to remain in the OSCORE group are able to continue participating in the secure communications within that group, while the evicted ones are not able to after the distribution and installation of the new security material.

Also, this ensures that the members intended to remain in the OSCORE group are able to confidently assert the group membership of other sender nodes, when receiving protected messages in the OSCORE group after the distribution and installation of the new security material (see Section 3.2 of [I-D.ietf-core-oscore-groupcomm]).

The key management operations mentioned above are entrusted to the Group Manager responsible for the OSCORE group [I-D.ietf-core-oscore-groupcomm], and it is RECOMMENDED to perform them as defined in [I-D.ietf-ace-key-groupcomm-oscore].

5.3. Proxy Security

Different solutions may be selected for secure group communication via a proxy depending on proxy type, use case and deployment requirements. In this section the options based on Group OSCORE are listed.

For a client performing a group communication request via a forward-proxy, end-to-end security should be implemented. The client then creates a group request protected with Group OSCORE and unicasts this to the proxy. The proxy adapts the request from a forward-proxy request to a regular request and multicasts this adapted request to the indicated CoAP group. During the adaptation, the security provided by Group OSCORE persists, in either case of using the group mode or using the pairwise mode. The first leg of communication from client to proxy can optionally be further protected, e.g., by using (D)TLS and/or OSCORE.

For a client performing a group communication request via a reverse-proxy, either end-to-end-security or hop-by-hop security can be implemented. The case of end-to-end security is the same as for the forward-proxy case.

The case of hop-by-hop security is only possible if the proxy can be completely trusted and it is configured as a member of the OSCORE security group(s) that it needs to access, on behalf of clients. The first leg of communication between client and proxy is then protected with a security method for CoAP unicast, such as (D)TLS, OSCORE or a combination of such methods. The second leg between proxy and servers is protected using Group OSCORE. This can be useful in applications where for example the origin client does not implement Group OSCORE, or the group management operations are confined to a particular network domain and the client is outside this domain.

For all the above cases, more details on using Group OSCORE are defined in [I-D.tiloca-core-groupcomm-proxy].

6. Security Considerations

This section provides security considerations for CoAP group communication, in general and for the particular transport of IP multicast.

6.1. CoAP NoSec Mode

CoAP group communication, if not protected, is vulnerable to all the attacks mentioned in Section 11 of [RFC7252] for IP multicast. Moreover, as also discussed in [I-D.mattsson-t2trg-amplification-attacks], the NoSec mode is susceptible to source IP address spoofing, hence amplification attacks are especially feasible to perform and greatly effective in their impact, since a single request can result in multiple responses from multiple servers (see Section 6.3).

Therefore, even in case of non-sensitive and non-critical applications, it is generally NOT RECOMMENDED to use CoAP group communication in NoSec mode, also in order to prevent an easy proliferation of high-volume amplification attacks as further discussed in Section 6.3.

Exceptionally, early discovery of devices and resources is a typical use case where NoSec mode is applied. In such a situation, the querying devices do not have yet configured any mutual security relations at the time they perform the discovery. Also, high-volume and harmful amplifications can be prevented through appropriate and conservative configurations, since only a few CoAP servers are expected to be configured for responding to the group requests sent for discovery (see Section 6.3).

CoAP group communication in NoSec mode SHOULD NOT be deployed for sensitive and mission-critical applications (e.g., health monitoring systems and alarm monitoring systems).

CoAP group communication without application-layer security SHOULD be deployed only in applications that are non-critical and that do not involve or may have an impact on sensitive data and personal sphere. These include, e.g., read-only temperature sensors deployed in non-sensitive environments, where the client reads out the values but does not use the data to control actuators or to base an important decision on.

6.2. Group OSCORE

Group OSCORE provides end-to-end application-level security. This has many desirable properties, including maintaining security assurances while forwarding traffic through intermediaries (proxies). Application-level security also tends to more cleanly separate security from the dynamics of group membership (e.g., the problem of distributing security keys across large groups with many members that come and go).

For sensitive and mission-critical applications, CoAP group communication MUST be protected by using Group OSCORE as specified in [I-D.ietf-core-oscore-groupcomm]. The same security considerations from Section 11 of [I-D.ietf-core-oscore-groupcomm] hold for this specification.

6.2.1. Group Key Management

A key management scheme for secure revocation and renewal of group security material, namely group rekeying, is required to be adopted in OSCORE groups. The key management scheme has to preserve forward security in the OSCORE group, as well as backward security if this is required by the application (see Section 5.2). In particular, the key management scheme **MUST** comply with the functional steps defined in Section 3.2 of [I-D.ietf-core-oscore-groupcomm].

Group policies should also take into account the time that the key management scheme requires to rekey the group, on one hand, and the expected frequency of group membership changes, i.e., nodes' joining and leaving, on the other hand.

That is, it may be desirable to not rekey the group upon every single membership change, in case members' joining and leaving are frequent, and at the same time a single group rekeying instance takes a non-negligible time to complete.

In such a case, the Group Manager may cautiously consider to rekey the group, e.g., after a minimum number of nodes has joined or left the group within a pre-defined time interval, or according to communication patterns with predictable time intervals of network inactivity. This would prevent paralyzing communications in the group, when a slow rekeying scheme is used and frequently invoked.

At the same, the security implications of delaying the rekeying process have to be carefully considered and understood, before enforcing such group policies.

In fact, this comes at the cost of not continuously preserving backward and forward security, since group rekeying might not occur upon every single group membership change. That is, most recently joined nodes would have access to the security material used prior to their joining, and thus be able to access past group communications protected with that security material. Similarly, until the group is rekeyed, most recently left nodes would preserve access to group communications protected with the retained security material.

6.2.2. Source Authentication

Both the group mode and the pairwise mode of Group OSCORE ensure source authentication of messages exchanged by CoAP endpoints through CoAP group communication.

To this end, outgoing messages are either countersigned by the message sender endpoint with its own private key (group mode), or protected with a symmetric key, which is in turn derived using the asymmetric keys of the message sender and recipient (pairwise mode).

Thus, both modes allow a recipient CoAP endpoint to verify that a message has actually been originated by a specific and identified member of the OSCORE group.

6.2.3. Countering Attacks

As discussed below, Group OSCORE addresses a number of security attacks mentioned in Section 11 of [RFC7252], with particular reference to their execution over IP multicast.

- * Since Group OSCORE provides end-to-end confidentiality and integrity of request/response messages, proxies capable of group communication cannot break message protection, and thus cannot act as man-in-the-middle beyond their legitimate duties (see Section 11.2 of [RFC7252]). In fact, intermediaries such as proxies are not assumed to have access to the OSCORE Security Context used by group members. Also, with the notable addition of signatures for the group mode, Group OSCORE protects messages using the same procedure as OSCORE (see Sections 8 and 9 of [I-D.ietf-core-oscore-groupcomm]), and especially processes CoAP options according to the same classification in U/I/E classes.
- * Group OSCORE limits the feasibility and impact of amplification attacks, see Section 6.3 of this document and Section 11.3 of [RFC7252].

In fact, upon receiving a group request protected with Group OSCORE in group mode, a server is able to verify whether the request is not a replay and originates from the alleged sender in the OSCORE group, by verifying the signature included in the request using the public key of that sender (see Section 8.2 of [I-D.ietf-core-oscore-groupcomm]). Furthermore, as also discussed in Section 8 of [I-D.ietf-core-oscore-groupcomm], it is recommended that servers failing to decrypt and verify an incoming message do not send back any error message.

This limits an adversary to leveraging an intercepted group request protected with Group OSCORE, and then altering the source address to be the one of the intended amplification victim.

Furthermore, the adversary needs to consider a group request that specifically targets a resource for which the CoAP servers are configured to respond. While this can be often correctly assumed

or inferable from the application context, it is not explicit from the group request itself, since Group OSCORE protects the Uri-Path and Uri-Query CoAP Options conveying the respective components of the target URI.

As a further mitigation against amplification attacks, a server can also rely on the Echo Option for CoAP defined in [RFC9175] and include it in a response to a group request. By doing so, the server can assert that the alleged sender of the group request (i.e., the CoAP client associated with a certain authentication credential including the corresponding public key) is indeed reachable at the claimed source address, especially if this differs from the one used in previous group requests from the same CoAP client. Although responses including the Echo Option do still result in amplification, this is limited in volume compared to when all servers reply with a full-fledged response.

- * Group OSCORE limits the impact of attacks based on IP spoofing also over IP multicast (see Section 11.4 of [RFC7252]). In fact, requests and corresponding responses sent in the OSCORE group can be correctly generated only by legitimate group members.

Within an OSCORE group, the shared symmetric-key security material strictly provides only group-level authentication. However, source authentication of messages is also ensured, both in the group mode by means of signatures (see Sections 8.1 and 8.3 of [I-D.ietf-core-oscore-groupcomm]), and in the pairwise mode by using additionally derived pairwise keys (see Sections 9.3 and 9.5 of [I-D.ietf-core-oscore-groupcomm]). Thus, recipient endpoints can verify a message to be originated by the alleged, identifiable sender in the OSCORE group.

As noted above, the server may additionally rely on the Echo Option for CoAP defined in [RFC9175], in order to verify the aliveness and reachability of the client sending a request from a particular IP address.

- * Group OSCORE does not require group members to be equipped with a good source of entropy for generating security material (see Section 11.6 of [RFC7252]), and thus does not contribute to create an entropy-related attack vector against such (constrained) CoAP endpoints. In particular, the symmetric keys used for message encryption and decryption are derived through the same HMAC-based HKDF scheme used for OSCORE (see Section 3.2 of [RFC8613]). Besides, the OSCORE Master Secret used in such derivation is securely generated by the Group Manager responsible for the OSCORE group, and securely provided to the CoAP endpoints when they join the group.

- * Group OSCORE prevents to make any single group member a target for subverting security in the whole OSCORE group (see Section 11.6 of [RFC7252]), even though all group members share (and can derive) the same symmetric-key security material used in the OSCORE group. In fact, source authentication is always ensured for exchanged CoAP messages, as verifiable to be originated by the alleged, identifiable sender in the OSCORE group. This relies on including a signature computed with a node's individual private key (in the group mode), or on protecting messages with a pairwise symmetric key, which is in turn derived from the asymmetric keys of the sender and recipient CoAP endpoints (in the pairwise mode).

6.3. Risk of Amplification

Section 11.3 of [RFC7252] highlights that CoAP group requests may be used for accidentally or deliberately performing Denial of Service attacks, especially in the form of a high-volume amplification attack, by using all the servers in the CoAP group as attack vectors.

That is, following a group request sent to a CoAP group, each of the servers in the group may reply with a response which is likely larger in size than the group request. Thus, an attacker sending a single group request may achieve a high amplification factor, i.e., a high ratio between the size of the group request and the total size of the corresponding responses intended to the attack victim.

Thus, consistently with Section 11.3 of [RFC7252], a server in a CoAP group:

- * SHOULD limit the support for CoAP group requests only to the group resources of the application group(s) using that CoAP group;
- * SHOULD NOT accept group requests that can not be authenticated in some way;
- * SHOULD NOT provide large amplification factors through its responses to a non-authenticated group request, and can possibly rely on CoAP block-wise transfers [RFC7959] to reduce the amount of amplification.

Amplifications attacks using CoAP are further discussed in [I-D.mattsson-t2trg-amplification-attacks], which also highlights how the amplification factor would become even higher when CoAP group communication is combined with resource observation [RFC7641]. That is, a single group request may result in multiple notification responses from each of the responding servers, throughout the observation lifetime.

Thus, consistently with Section 7 of [RFC7641], a server in a CoAP group MUST strictly limit the number of notifications it sends between receiving acknowledgments that confirm the actual interest of the client in continuing the observation.

Moreover, it is especially easy to perform an amplification attack when the NoSec mode is used. Therefore, even in case of non-sensitive and non-critical applications, it is generally NOT RECOMMENDED to use CoAP group communication in NoSec mode, in order to prevent an easy proliferation of high-volume amplification attacks.

Exceptions should be carefully limited to use cases and accesses to a group resource that have a specific, narrow and well understood scope, and where only a few CoAP servers (or, ideally, only one) would possibly respond to a group request.

A relevant exceptional example is a CoAP client performing the discovery of hosts such as a group manager or a Resource Directory [I-D.ietf-core-resource-directory], by probing for them through a group request sent to the CoAP group. This early, unprotected step is relevant for a CoAP client that does not know the address of such hosts in advance, and that does not have yet configured a mutual security relation with them. In this kind of deployments, such a discovery procedure does not result in a considerable and harmful amplification, since only the few CoAP servers object of discovery are going to respond to the group request targeting that specific resource. In particular, those hosts can be the only CoAP servers in that specific CoAP group (hence listening for group requests sent to that group), and/or the only CoAP servers explicitly configured to respond to group requests targeting specific group resources.

With the exception of such particular use cases, group communications MUST be secured using Group OSCORE [I-D.ietf-core-oscore-groupcomm], see Section 5. As discussed in Section 6.2.3, this limits the feasibility and impact of amplification attacks.

6.4. Replay of Non-Confirmable Messages

Since all requests sent over IP multicast are Non-confirmable, a client might not be able to know if an adversary has actually captured one of its transmitted requests and later re-injected it in the group as a replay to the server nodes. In fact, even if the servers sent back responses to the replayed request, the client would typically not have a valid matching request active anymore so this attack would not accomplish anything in the client.

If Group OSCORE is used, such a replay attack on the servers is prevented, since a client protects every different request with a different Sequence Number value, which is in turn included as Partial IV in the protected message and takes part in the construction of the AEAD cipher nonce. Thus, a server would be able to detect the replayed request, by checking the conveyed Partial IV against its own replay window in the OSCORE Recipient Context associated with the client.

This requires a server to have a synchronized, up to date view of the sequence number used by the client. If such synchronization is lost, e.g., due to a reboot, or suspected so, the server should use the challenge-response synchronization method described in Appendix E of [I-D.ietf-core-oscore-groupcomm] and based on the Echo Option for CoAP defined in [RFC9175], in order to (re-)synchronize with the client's sequence number.

6.5. Use of CoAP No-Response Option

When CoAP group communication is used in CoAP NoSec (No Security) mode (see Section 4), the CoAP No-Response Option [RFC7967] could be misused by a malicious client to evoke as much responses from servers to a group request as possible, by using the value '0' - Interested in all responses. This even overrides the default behavior of a CoAP server to suppress the response in case there is nothing of interest to respond with. Therefore, this option can be used to perform an amplification attack (see Section 6.3).

A proposed mitigation is to only allow this option to relax the standard suppression rules for a resource in case the option is sent by an authenticated client. If sent by an unauthenticated client, the option can be used to expand the classes of responses suppressed compared to the default rules but not to reduce the classes of responses suppressed.

6.6. 6LoWPAN and MPL

In a 6LoWPAN network, a multicast IPv6 packet may be fragmented prior to transmission. A 6LoWPAN Router that forwards a fragmented packet may have a relatively high impact on the occupation of the wireless channel and may locally experience high memory load due to packet buffering. For example, the MPL [RFC7731] protocol requires an MPL Forwarder to store the packet for a longer duration, to allow multiple forwarding transmissions to neighboring Forwarders. If one or more of the fragments are not received correctly by an MPL Forwarder during its packet reassembly time window, the Forwarder discards all received fragments and at a future point in time it needs to receive again all the packet fragments (this time, possibly

from another neighboring MPL Forwarder).

For these reasons, a fragmented IPv6 multicast packet is a possible attack vector in a Denial of Service (DoS) amplification attack. See Section 6.3 of this document and Section 11.3 of [RFC7252] for more details on amplification. To mitigate the risk, applications sending multicast IPv6 requests to 6LoWPAN hosted CoAP servers SHOULD limit the size of the request to avoid 6LoWPAN fragmentation of the request packet. A 6LoWPAN Router or (MPL) multicast forwarder SHOULD deprioritize forwarding for multi-fragment 6LoWPAN multicast packets. 6LoWPAN Border Routers are typical ingress points where multicast traffic enters into a 6LoWPAN network. Specific MPL Forwarders (whether located on a 6LBR or not) may also be configured as ingress points. Any such ingress point SHOULD implement multicast packet filtering to prevent unwanted multicast traffic from entering a 6LoWPAN network from the outside. For example, it could filter out all multicast packets for which there is no known multicast listener on the 6LoWPAN network. See Section 3.10 for protocols that allow multicast listeners to signal which groups they would like to listen to. As part of multicast packet filtering, the ingress point SHOULD implement a filtering criterium based on the size of the multicast packet. Ingress multicast packets above a defined size may then be dropped or deprioritized.

6.7. Wi-Fi

In a home automation scenario using Wi-Fi, Wi-Fi security should be enabled to prevent rogue nodes from joining. The Customer Premises Equipment (CPE) that enables access to the Internet should also have its IP multicast filters set so that it enforces multicast scope boundaries to isolate local multicast groups from the rest of the Internet (e.g., as per [RFC6092]). In addition, the scope of IP multicast transmissions and listeners should be site-local (5) or smaller. For site-local scope, the CPE will be an appropriate multicast scope boundary point.

6.8. Monitoring

6.8.1. General Monitoring

CoAP group communication can be used to control a set of related devices: for example, simultaneously turn on all the lights in a room. This intrinsically exposes the group to some unique monitoring risks that devices not in a group are not as vulnerable to. For example, assume an attacker is able to physically see a set of lights turn on in a room. Then the attacker can correlate an observed CoAP group communication message to the observed coordinated group action -- even if the CoAP message is (partly) encrypted.

This will give the attacker side-channel information to plan further attacks (e.g., by determining the members of the group some network topology information may be deduced).

6.8.2. Pervasive Monitoring

A key additional threat consideration for group communication is pervasive monitoring [RFC7258]. CoAP group communication solutions that are built on top of IP multicast need to pay particular heed to these dangers. This is because IP multicast is easier to intercept compared to IP unicast. Also, CoAP traffic is typically used for the Internet of Things. This means that CoAP (multicast) group communication may be used for the control and monitoring of critical infrastructure (e.g., lights, alarms, HVAC, electrical grid, etc.) that may be prime targets for attack.

For example, an attacker may attempt to record all the CoAP traffic going over a smart grid (i.e., networked electrical utility) and try to determine critical nodes for further attacks. For example, the source node (controller) sends out CoAP group communication messages which easily identifies it as a controller. CoAP multicast traffic is inherently more vulnerable compared to unicast, as the same packet may be replicated over many more links, leading to a higher probability of packet capture by a pervasive monitoring system.

One mitigation is to restrict the scope of IP multicast to the minimal scope that fulfills the application need. See the congestion control recommendations in the last bullet of Section 3.6 to minimize the scope. Thus, for example, realm-local IP multicast scope is always preferred over site-local scope IP multicast if this fulfills the application needs.

Even if all CoAP multicast traffic is encrypted/protected, an attacker may still attempt to capture this traffic and perform an off-line attack in the future.

7. IANA Considerations

This document has no actions for IANA.

8. References

8.1. Normative References

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P.,
and J. Park, "Group OSCORE - Secure Group Communication
for CoAP", Work in Progress, Internet-Draft, draft-ietf-

core-oscore-groupcomm-14, 7 March 2022,
<<https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-14.txt>>.

[I-D.ietf-cose-rfc8152bis-algs]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.

[I-D.ietf-cose-rfc8152bis-struct]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.

[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>>.

[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>>.

[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>>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <<https://www.rfc-editor.org/info/rfc4443>>.

[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>>.

- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

- [RFC9175] Amsüss, C., Preuß Mattsson, J., and G. Selander, "Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing", RFC 9175, DOI 10.17487/RFC9175, February 2022, <<https://www.rfc-editor.org/info/rfc9175>>.

8.2. Informative References

- [Californium]
Eclipse Foundation, "Eclipse Californium", March 2019, <<https://github.com/eclipse/californium/tree/2.0.x/californium-core/src/main/java/org/eclipse/californium/core>>.
- [Go-OCF] Open Connectivity Foundation (OCF), "Implementation of CoAP Server & Client in Go", March 2019, <<https://github.com/go-ocf/go-coap>>.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-13, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-13.txt>>.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-46, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>>.
- [I-D.ietf-ace-oscore-gm-admin]
Tiloca, M., Höglund, R., Stok, P. V. D., and F. Palombini, "Admin Interface for the OSCORE Group Manager", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-gm-admin-05, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-gm-admin-05.txt>>.
- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-09, 30 September 2019, <<https://www.ietf.org/archive/id/draft-ietf-core-coap-pubsub-09.txt>>.

[I-D.ietf-core-new-block]

Boucadair, M. and J. Shallow, "Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission", Work in Progress, Internet-Draft, draft-ietf-core-new-block-14, 26 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-new-block-14.txt>>.

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

Amsüss, C., Shelby, Z., Kostér, M., Bormann, C., and P. V. D. Stok, "CoRE Resource Directory", Work in Progress, Internet-Draft, draft-ietf-core-resource-directory-28, 7 March 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-resource-directory-28.txt>>.

[I-D.mattsson-t2trg-amplification-attacks]

Mattsson, J. P., Selander, G., and C. Amsüss, "Amplification Attacks Using the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-mattsson-t2trg-amplification-attacks-00, 11 February 2022, <<https://www.ietf.org/archive/id/draft-mattsson-t2trg-amplification-attacks-00.txt>>.

[I-D.tiloca-core-groupcomm-proxy]

Tiloca, M. and E. Dijk, "Proxy Operations for CoAP Group Communication", Work in Progress, Internet-Draft, draft-tiloca-core-groupcomm-proxy-06, 7 March 2022, <<https://www.ietf.org/archive/id/draft-tiloca-core-groupcomm-proxy-06.txt>>.

[I-D.tiloca-core-oscore-discovery]

Tiloca, M., Amsuess, C., and P. V. D. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-11, 7 March 2022, <<https://www.ietf.org/archive/id/draft-tiloca-core-oscore-discovery-11.txt>>.

[RFC6092]

Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <<https://www.rfc-editor.org/info/rfc6092>>.

- [RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <<https://www.rfc-editor.org/info/rfc6550>>.
- [RFC6636] Asaeda, H., Liu, H., and Q. Wu, "Tuning the Behavior of the Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) for Routers in Mobile and Wireless Networks", RFC 6636, DOI 10.17487/RFC6636, May 2012, <<https://www.rfc-editor.org/info/rfc6636>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<https://www.rfc-editor.org/info/rfc7258>>.
- [RFC7346] Droms, R., "IPv6 Multicast Address Scopes", RFC 7346, DOI 10.17487/RFC7346, August 2014, <<https://www.rfc-editor.org/info/rfc7346>>.
- [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>>.
- [RFC7731] Hui, J. and R. Kelsey, "Multicast Protocol for Low-Power and Lossy Networks (MPL)", RFC 7731, DOI 10.17487/RFC7731, February 2016, <<https://www.rfc-editor.org/info/rfc7731>>.
- [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>>.
- [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>>.
- [RFC8710] Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for the Constrained Application Protocol (CoAP)", RFC 8710, DOI 10.17487/RFC8710, February 2020, <<https://www.rfc-editor.org/info/rfc8710>>.

Appendix A. Use Cases

To illustrate where and how CoAP-based group communication can be used, this section summarizes the most common use cases. These use cases include both secured and non-secured CoAP usage. Each subsection below covers one particular category of use cases for CoRE. Within each category, a use case may cover multiple application areas such as home IoT, commercial building IoT (sensing and control), industrial IoT/control, or environmental sensing.

A.1. Discovery

Discovery of physical devices in a network, or discovery of information entities hosted on network devices, are operations that are usually required in a system during the phases of setup or (re)configuration. When a discovery use case involves devices that need to interact without having been configured previously with a common security context, unsecured CoAP communication is typically used. Discovery may involve a request to a directory server, which provides services to aid clients in the discovery process. One particular type of directory server is the CoRE Resource Directory [I-D.ietf-core-resource-directory]; and there may be other types of directories that can be used with CoAP.

A.1.1. Distributed Device Discovery

Device discovery is the discovery and identification of networked devices -- optionally only devices of a particular class, type, model, or brand. Group communication is used for distributed device discovery, if a central directory server is not used. Typically in distributed device discovery, a multicast request is sent to a particular address (or address range) and multicast scope of interest, and any devices configured to be discoverable will respond back. For the alternative solution of centralized device discovery a central directory server is accessed through unicast, in which case group communication is not needed. This requires that the address of the central directory is either preconfigured in each device or configured during operation using a protocol.

In CoAP, device discovery can be implemented by CoAP resource discovery requesting (GET) a particular resource that the sought device class, type, model or brand is known to respond to. It can also be implemented using CoAP resource discovery (Section 7 of [RFC7252]) and the CoAP query interface defined in Section 4 of [RFC6690] to find these particular resources. Also, a multicast GET request to /.well-known/core can be used to discover all CoAP devices.

A.1.2. Distributed Service Discovery

Service discovery is the discovery and identification of particular services hosted on network devices. Services can be identified by one or more parameters such as ID, name, protocol, version and/or type. Distributed service discovery involves group communication to reach individual devices hosting a particular service; with a central directory server not being used.

In CoAP, services are represented as resources and service discovery is implemented using resource discovery (Section 7 of [RFC7252]) and the CoAP query interface defined in Section 4 of [RFC6690].

A.1.3. Directory Discovery

This use case is a specific sub-case of Distributed Service Discovery (Appendix A.1.2), in which a device needs to identify the location of a Directory on the network to which it can e.g., register its own offered services, or to which it can perform queries to identify and locate other devices/services it needs to access on the network. Section 3.3 of [RFC7390] showed an example of discovering a CoRE Resource Directory using CoAP group communication. As defined in [I-D.ietf-core-resource-directory], a resource directory is a web entity that stores information about web resources and implements REST interfaces for registration and lookup of those resources. For example, a device can register itself to a resource directory to let it be found by other devices and/or applications.

A.2. Operational Phase

Operational phase use cases describe those operations that occur most frequently in a networked system, during its operational lifetime and regular operation. Regular usage is when the applications on networked devices perform the tasks they were designed for and exchange of application-related data using group communication occurs. Processes like system reconfiguration, group changes, system/device setup, extra group security changes, etc. are not part of regular operation.

A.2.1. Actuator Group Control

Group communication can be beneficial to control actuators that need to act in synchrony, as a group, with strict timing (latency) requirements. Examples are office lighting, stage lighting, street lighting, or audio alert/Public Address systems. Sections 3.4 and 3.5 of [RFC7390] showed examples of lighting control of a group of 6LoWPAN-connected lights.

A.2.2. Device Group Status Request

To properly monitor the status of systems, there may be a need for ad-hoc, unplanned status updates. Group communication can be used to quickly send out a request to a (potentially large) number of devices for specific information. Each device then responds back with the requested data. Those devices that did not respond to the request can optionally be polled again via reliable unicast communication to complete the dataset. The device group may be defined e.g., as "all temperature sensors on floor 3", or "all lights in wing B". For example, it could be a status request for device temperature, most recent sensor event detected, firmware version, network load, and/or battery level.

A.2.3. Network-wide Query

In some cases a whole network or subnet of multiple IP devices needs to be queried for status or other information. This is similar to the previous use case except that the device group is not defined in terms of its function/type but in terms of its network location. Technically this is also similar to distributed service discovery (Appendix A.1.2) where a query is processed by all devices on a network - except that the query is not about services offered by the device, but rather specific operational data is requested.

A.2.4. Network-wide / Group Notification

In some cases a whole network, or subnet of multiple IP devices, or a specific target group needs to be notified of a status change or other information. This is similar to the previous two use cases except that the recipients are not expected to respond with some information. Unreliable notification can be acceptable in some use cases, in which a recipient does not respond with a confirmation of having received the notification. In such a case, the receiving CoAP server does not have to create a CoAP response. If the sender needs confirmation of reception, the CoAP servers can be configured for that resource to respond with a 2.xx success status after processing a notification request successfully.

A.3. Software Update

Group communication can be useful to efficiently distribute new software (firmware, image, application, etc.) to a group of multiple devices. In this case, the group is defined in terms of device type: all devices in the target group are known to be capable of installing and running the new software. The software is distributed as a series of smaller blocks that are collected by all devices and stored in memory. All devices in the target group are usually responsible

for integrity verification of the received software; which can be done per-block or for the entire software image once all blocks have been received. Due to the inherent unreliability of CoAP multicast, there needs to be a backup mechanism (e.g., implemented using CoAP unicast) by which a device can individually request missing blocks of a whole software image/entity. Prior to a multicast software update, the group of recipients can be separately notified that there is new software available and coming, using the above network-wide or group notification.

Appendix B. Examples of Message Exchanges

This section provides examples of different message exchanges when CoAP is used with group communication. The examples consider:

- * A client with address ADDR_CLIENT and port number PORT_CLIENT.
- * A CoAP group associated with the IP multicast address ADDR_GRP and port number PORT_GRP.
- * An application group "gp1" associated with the CoAP group above.
- * Three servers A, B and C, all of which are members of the CoAP group above and of the application group "gp1". Each server X (with X equal to A, B or C): listens to its own address ADDR_X and port number PORT_X; and listens to the address ADDR_GRP and port number PORT_GRP. For each server its PORT_X may be different from PORT_GRP or may be equal to it, in general.

In Figure 16, the client sends a Non-confirmable GET request to the CoAP group, targeting the resource "temperature" in the application group "gp1". All servers reply with a 2.05 Content response, although the response from server B is lost. As source port number of their response, servers A and B use the destination port number of the request, i.e, PORT_GRP. Instead, server C uses its own port number PORT_C.

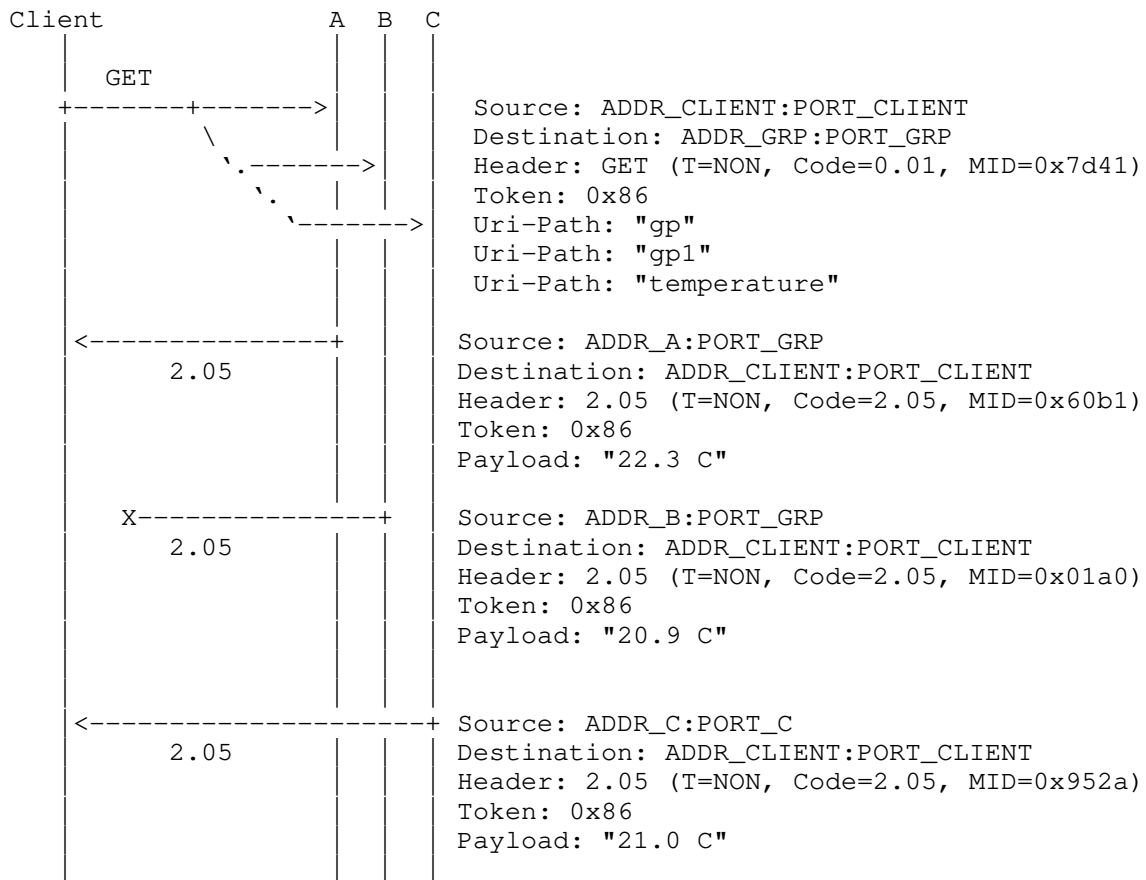
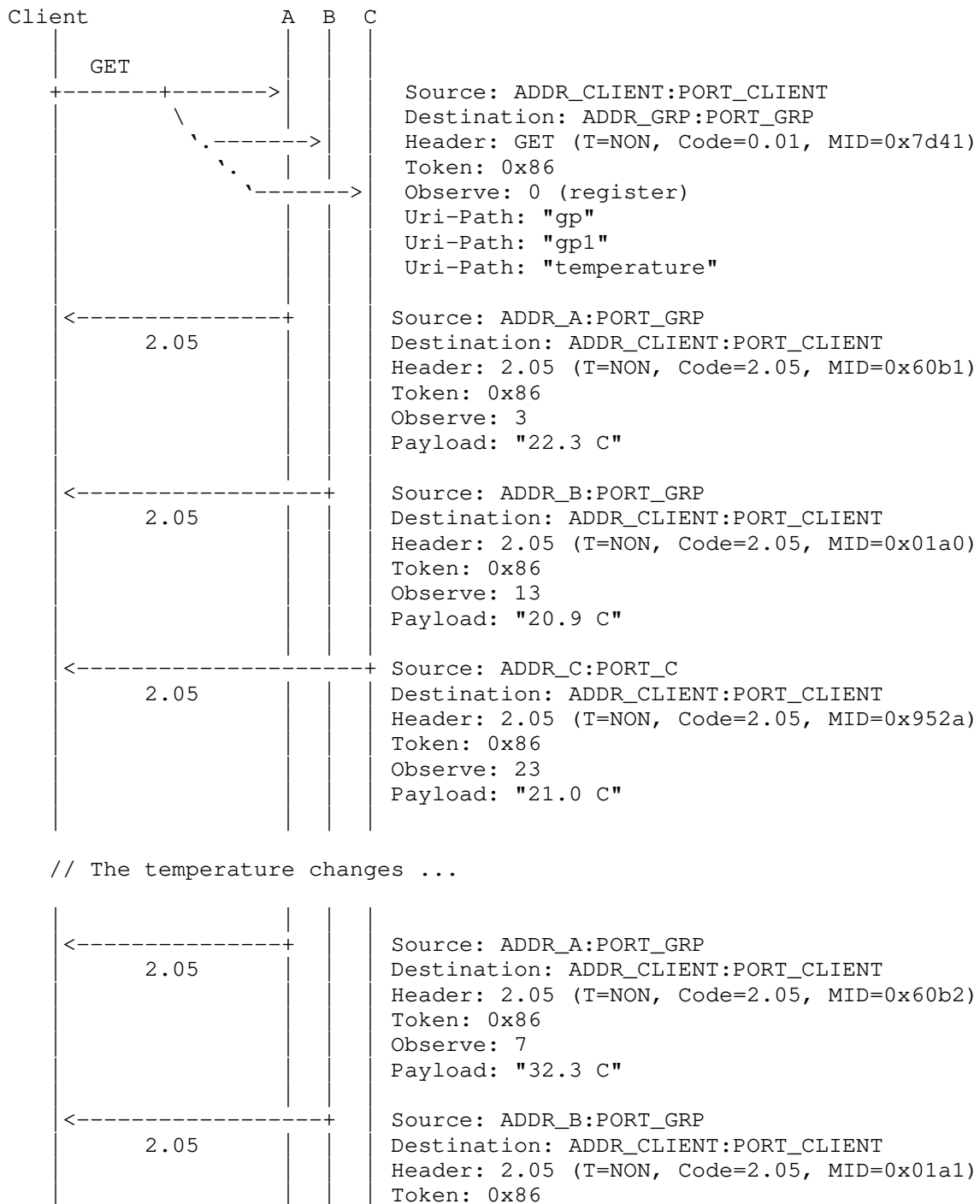


Figure 16: Example of Non-confirmable group request, followed by Non-confirmable Responses

In Figure 17, the client sends a Non-confirmable GET request to the CoAP group, targeting and requesting to observe the resource "temperature" in the application group "gp1". All servers reply with a 2.05 Content notification response. As source port number of their response, servers A and B use the destination port number of the request, i.e, PORT_GRP. Instead, server C uses its own port number PORT_C. Some time later, all servers send a 2.05 Content notification response, with payload the new representation of the "temperature" resource.



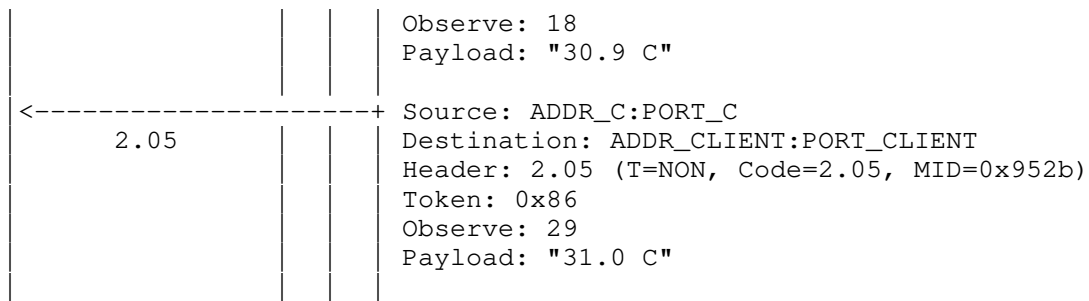
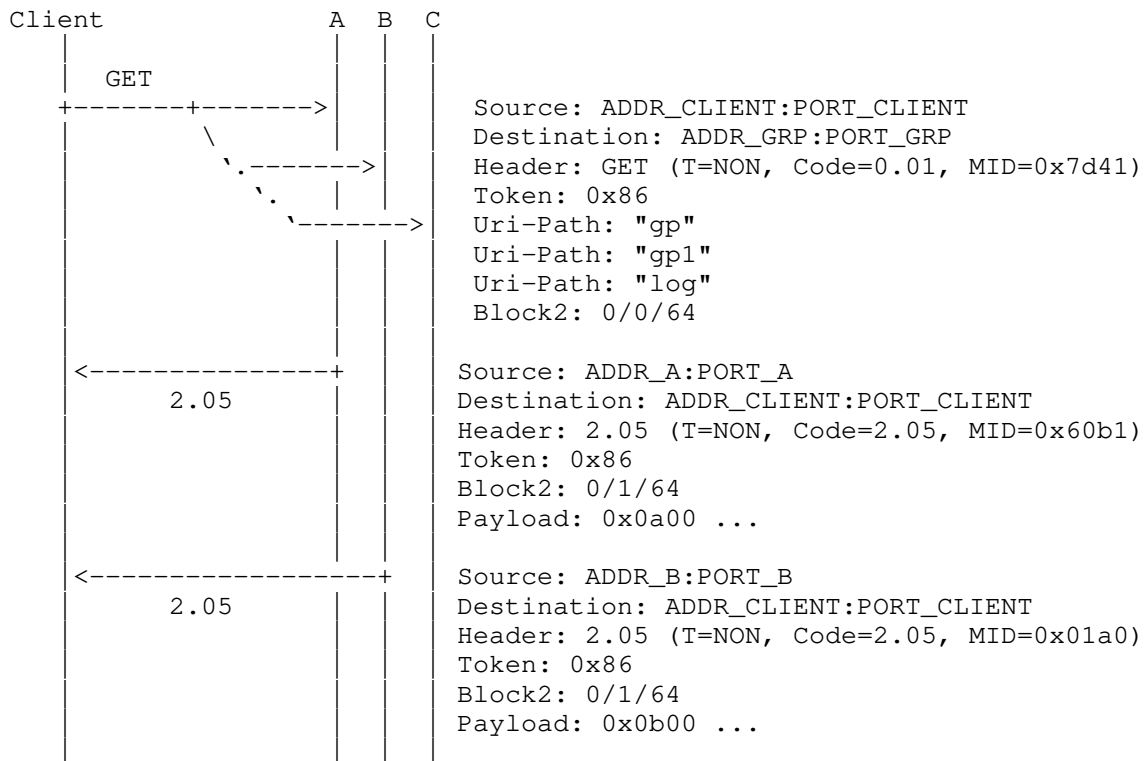
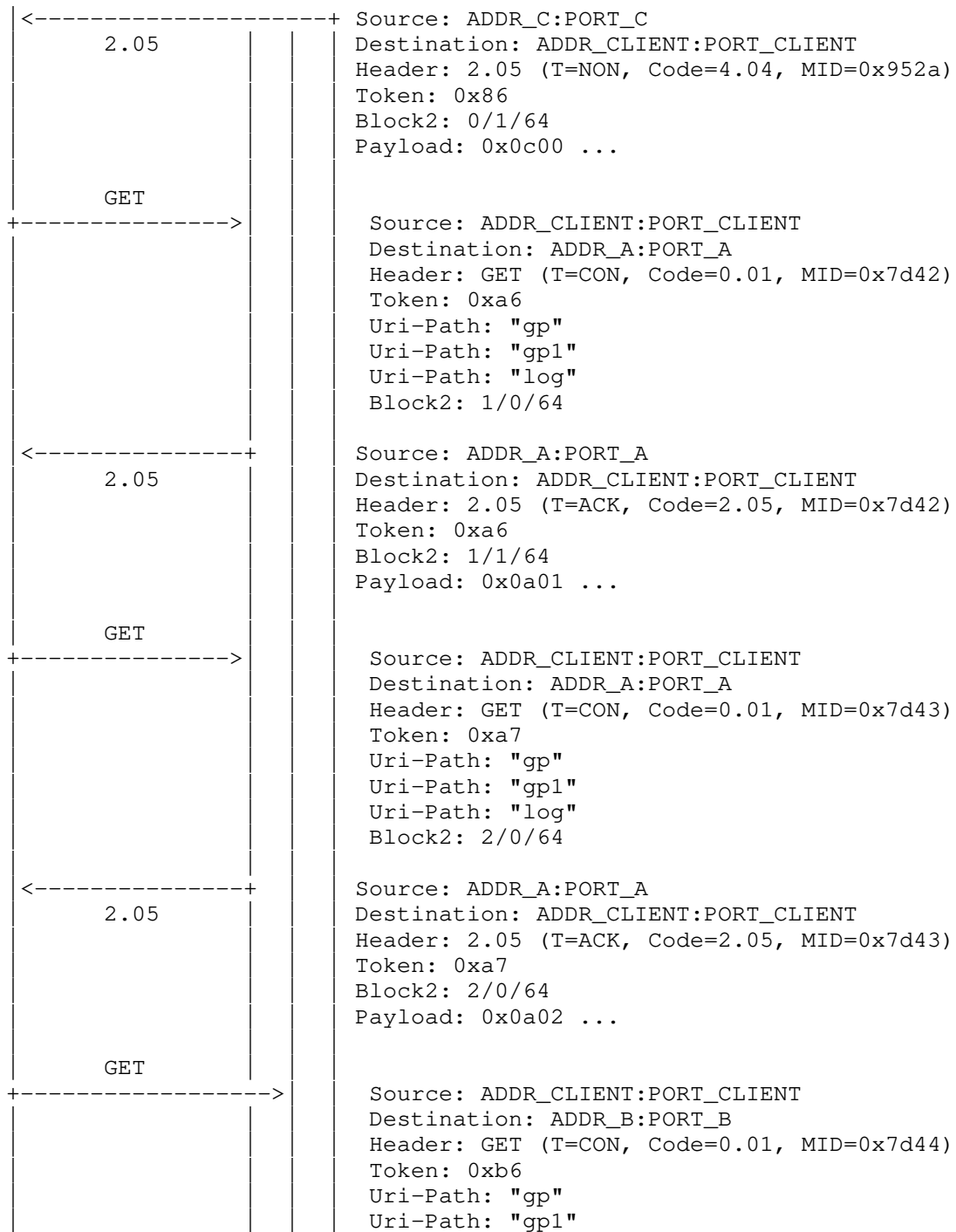
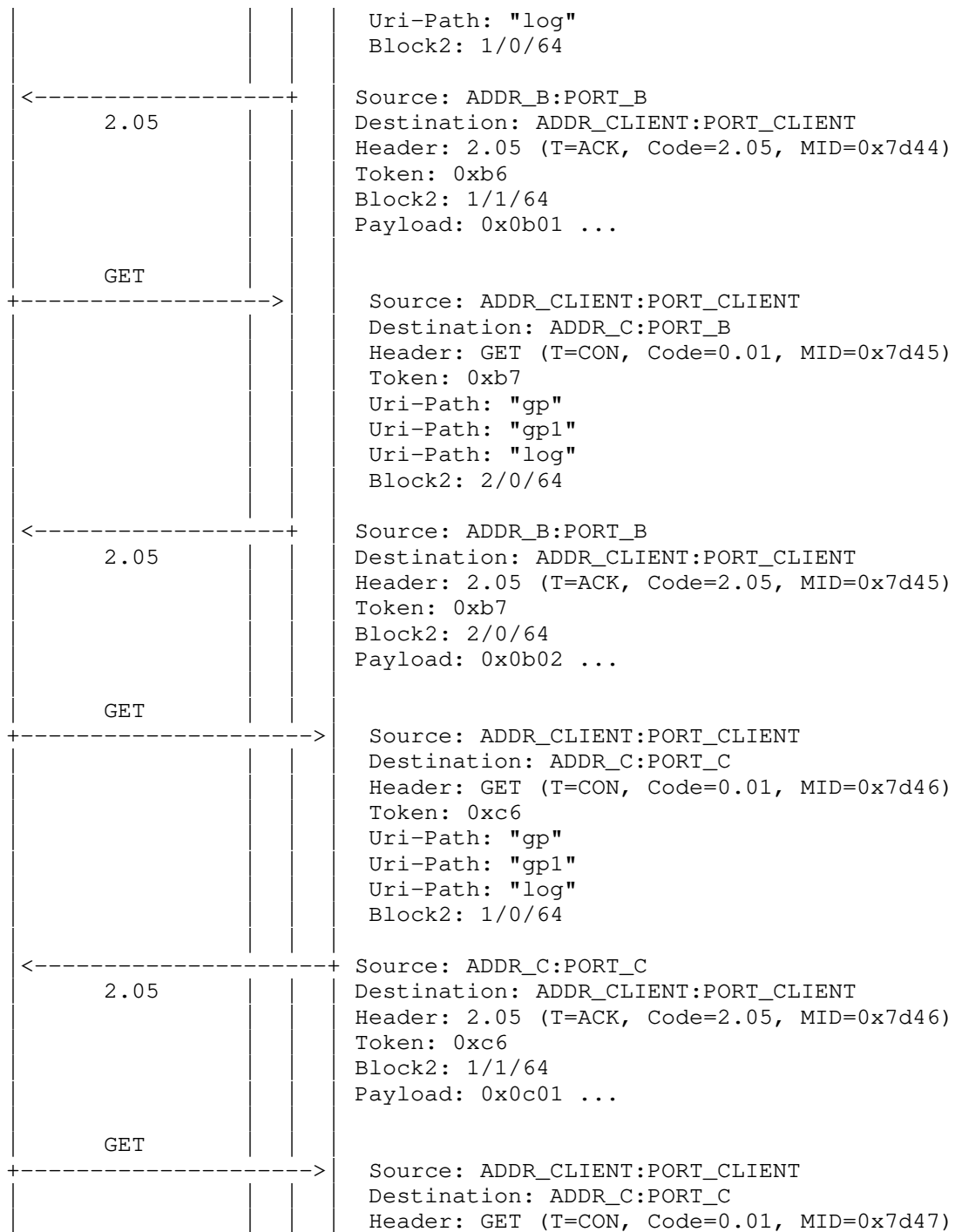


Figure 17: Example of Non-confirmable Observe group request, followed by Non- confirmable Responses as Observe notifications

In Figure 18, the client sends a Non-confirmable GET request to the CoAP group, targeting the resource "log" in the application group "gp1", and requesting a blockwise transfer. All servers reply with a 2.05 Content response including the first block. As source port number of its response, each server uses its own port number. After obtaining the first block, the client requests the following blocks separately from each server, by means of unicast exchanges.







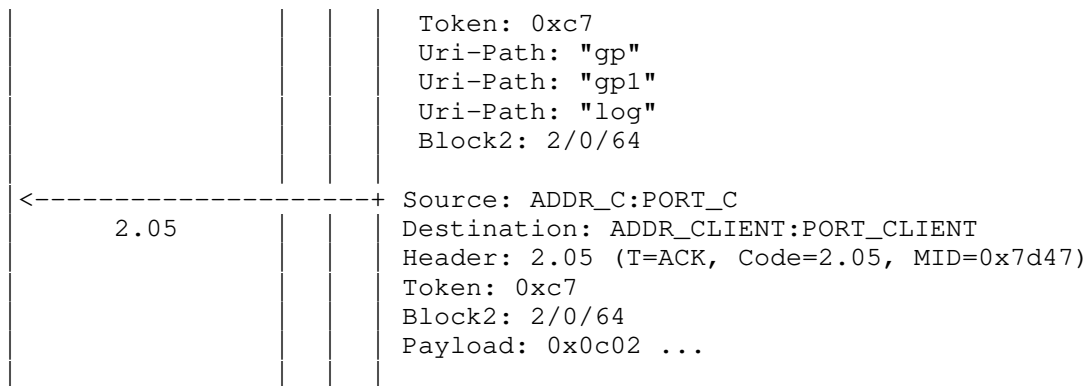


Figure 18: Example of Non-confirmable group request starting a blockwise transfer, followed by Non-confirmable Responses with the first block. The transfer continues over confirmable unicast exchanges

Appendix C. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

C.1. Version -05 to -06

- * Harmonized use of "group URI".
- * Clarifications about different group types.
- * Revised methods to perform group naming.
- * Revised methods to discover application groups and CoAP groups.
- * Explicit difference between "authentication credential" and "public key".
- * Added examples of application group naming.
- * Added examples of application/CoAP group discovery.
- * Added examples of message exchanges.
- * Reference to draft-mattsson-core-coap-attacks replaced with reference to draft-mattsson-t2trg-amplification-attacks.
- * Editorial improvements.

C.2. Version -04 to -05

- * Clarified changes to other documents.
- * Clarified relation between different group types.
- * Clarified discovery of application groups.
- * Discussed methods to express application group names in requests.
- * Revised and extended text on the NoSec mode and amplification attacks.
- * Rephrased backward/forward security as properties.
- * Removed appendix on Multi-ETag Option for response revalidation.
- * Editorial improvements.

C.3. Version -03 to -04

- * Multi-ETag Option for response revalidation moved to appendix.
- * ETag Option usage added.
- * Q-Block Options added in the block-wise transfer section.
- * Caching at proxies moved to draft-tiloca-core-groupcomm-proxy.
- * Client-Proxy response revalidation with the Group-ETag Option moved to draft-tiloca-core-groupcomm-proxy.
- * Security considerations on amplification attacks.
- * Generalized transport protocols to include others than UDP/IP multicast; and security protocols other than Group OSCORE.
- * Overview of security cases with proxies.
- * Editorial improvements.

C.4. Version -02 to -03

- * Multiple responses from same server handled at the application.
- * Clarifications about issues with forward-proxies.
- * Operations for reverse-proxies.

- * Caching of responses at proxies.
- * Client-Server response revalidation, with Multi-ETag Option.
- * Client-Proxy response revalidation, with the Group-ETag Option.

C.5. Version -01 to -02

- * Clarified relation between security groups and application groups.
- * Considered also FETCH for requests over IP multicast.
- * More details on Observe re-registration.
- * More details on Proxy intermediaries.
- * More details on servers changing port number in the response.
- * Usage of the Uri-Host Option to indicate an application group.
- * Response suppression based on classes of error codes.

C.6. Version -00 to -01

- * Clarifications on group memberships for the different group types.
- * Simplified description of Token reuse, compared to the unicast case.
- * More details on the rationale for response suppression.
- * Clarifications of creation and management of security groups.
- * Clients more knowledgeable than proxies about stopping receiving responses.
- * Cancellation of group observations.
- * Clarification on multicast scope to use.
- * Both the group mode and pairwise mode of Group OSCORE are considered.
- * Updated security considerations.
- * Editorial improvements.

Acknowledgments

The authors sincerely thank Christian Amsuess, Carsten Bormann, Thomas Fossati, Rikard Hoeglund, Jaime Jimenez, John Mattsson and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Esko Dijk
IoTconsultancy.nl

Utrecht
Netherlands
Email: esko.dijk@iotconsultancy.nl

Chonggang Wang
InterDigital
1001 E Hector St, Suite 300
Conshohocken, PA 19428
United States
Email: Chonggang.Wang@InterDigital.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

CORE
Internet-Draft
Intended status: Standards Track
Expires: May 2, 2021

M. Boucadair
Orange
J. Shallow
October 29, 2020

Constrained Application Protocol (CoAP) Block-Wise Transfer Options for
Faster Transmission
draft-ietf-core-new-block-02

Abstract

This document specifies alternative Constrained Application Protocol (CoAP) Block-Wise transfer options: Q-Block1 and Q-Block2 Options.

These options are similar to the CoAP Block1 and Block2 Options, not a replacement for them, but do enable faster transmission rates for large amounts of data with less packet interchanges as well as supporting faster recovery should any of the blocks get lost in transmission.

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 May 2, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Existing CoAP Block-Wise Transfer Options	3
1.2. Alternative CoAP Block-Wise Transfer Options	3
1.3. Updated CoAP Response Code (4.08)	4
1.4. Applicability Scope	5
2. Terminology	5
3. The Q-Block1 and Q-Block2 Options	6
3.1. Properties of Q-Block1 and Q-Block2 Options	6
3.2. Structure of Q-Block1 and Q-Block2 Options	7
3.3. Using the Q-Block1 Option	8
3.4. Using the Q-Block2 Option	9
3.5. Working with Observe and Q-Block2 Options	11
3.6. Working with Size1 and Size2 Options	11
3.7. Use of Q-Block1 and Q-Block2 Options Together	11
4. The Use of 4.08 (Request Entity Incomplete) Response Code . .	11
5. The Use of Tokens	13
6. Congestion Control	13
7. Caching Considerations	14
8. HTTP-Mapping Considerations	15
9. Examples of Selective Block Recovery	15
9.1. Q-Block1 Option: Non-Confirmable Example	16
9.2. Q-Block2 Option: Non-Confirmable Example	17
10. IANA Considerations	19
10.1. New CoAP Options	19
10.2. New Content Format	20
11. Security Considerations	20
12. Acknowledgements	20
13. References	20
13.1. Normative References	20
13.2. Informative References	22
Appendix A. Examples with Confirmable Messages	22
A.1. Q-Block1 Option	22
A.2. Q-Block2 Option	24
Authors' Addresses	25

1. Introduction

1.1. Existing CoAP Block-Wise Transfer Options

The Constrained Application Protocol (CoAP) [RFC7252], although inspired by HTTP, was designed to use UDP instead of TCP. The message layer of CoAP over UDP includes support for reliable delivery, simple congestion control, and flow control. [RFC7959] introduced the CoAP Block1 and Block2 Options to handle data records that cannot fit in a single IP packet, so not having to rely on IP fragmentation and further updated by [RFC8323] for use over TCP, TLS, and Websockets.

The CoAP Block1 and Block2 Options work well in environments where there are no or minimal packet losses. These options operate synchronously where each block has to be requested and can only ask for (or send) the next block when the request for the previous block has completed. Packet, and hence block transmission rate, is controlled by Round Trip Times (RTTs).

There is a requirement for these blocks of data to be transmitted at higher rates under network conditions where there may be asymmetrical transient packet loss. An example is when a network is subject to a Distributed Denial of Service (DDoS) attack and there is a need for DDoS mitigation agents relying upon CoAP to communicate with each other (e.g., [I-D.ietf-dots-telemetry]). As a reminder, [RFC7959] recommends use of Confirmable (CON) responses to handle potential packet loss; which does not work with a flooded pipe DDoS situation.

1.2. Alternative CoAP Block-Wise Transfer Options

This document introduces the CoAP Q-Block1 and Q-Block2 Options. These options are similar in operation to the CoAP Block1 and Block2 Options respectively, they are not a replacement for them, but have the following benefits:

- o They can operate in environments where packet loss is highly asymmetrical.
- o They enable faster transmissions of sets of blocks of data with less packet interchanges.
- o They support faster recovery should any of the Blocks get lost in transmission.
- o They support sending an entire body using Non-confirmable (NON) without requiring a response from the peer.

There are the following disadvantages over using CoAP Block 1 and Block2 Options:

- o Loss of lock-stepping so payloads are not always received in the correct (block ascending) order.
- o Additional congestion control measures need to be put in place.

Using NON messages, the faster transmissions occur as all the Blocks can be transmitted serially (as are IP fragmented packets) without having to wait for an acknowledgement or next request from the remote CoAP peer. Recovery of missing Blocks is faster in that multiple missing Blocks can be requested in a single CoAP packet. Even if there is asymmetrical packet loss, a body can still be sent and received by the peer whether the body compromises of a single or multiple payloads assuming no recovery is required.

Note that the same performance benefits can be applied to Confirmable messages if the value of NSTART is increased from 1 (Section 4.7 of [RFC7252]). However, the asymmetrical packet loss is not a benefit here. Some sample examples with Confirmable messages are provided in Appendix A.

There is little, if any, benefit of using these options with CoAP running over a reliable connection [RFC8323]. In this case, there is no differentiation between Confirmable and NON as they are not used.

A CoAP endpoint can acknowledge all or a subset of the blocks. Concretely, the receiving CoAP endpoint informs the CoAP endpoint sender either successful receipt or reports on all blocks in the body that have been not yet been received. The CoAP endpoint sender will then retransmit only the blocks that have been lost in transmission.

Q-Block1 and Q-Block2 Options can be used instead of Block1 and Block2 Options respectively when the different transmission semantics are required. If the option is not supported by a peer, then transmissions can fall back to using Block1 and Block2 respectively.

The deviations from Block1 and Block2 Options are specified in Section 3. Pointers to appropriate [RFC7959] sections are provided.

The specification refers to the base CoAP methods defined in Section 5.8 of [RFC7252] and the new CoAP methods, FETCH, PATCH, and iPATCH introduced in [RFC8132].

1.3. Updated CoAP Response Code (4.08)

This document updates the 4.08 (Request Entity Incomplete) by defining an additional message format for reporting on payloads using the Q-Block1 Option that are not received by the server.

See Section 4 for more details.

1.4. Applicability Scope

The block-wise transfer specified in [RFC7959] covers the general case, but falls short in situations where packet loss is highly asymmetrical. The mechanism specified in this document provides roughly similar features to the Block1/Block2 Options. It provides additional properties that are tailored towards the intended use case. Concretely, this mechanism primarily targets applications such as DDoS Open Threat Signaling (DOTS) that can't use Confirmable (CON) responses to handle potential packet loss and that support application-specific mechanisms to assess whether the remote peer is able to handle the messages sent by a CoAP endpoint (e.g., DOTS heartbeats in Section 4.7 of [RFC8782]).

The mechanism includes guards to prevent a CoAP agent from overloading the network by adopting an aggressive sending rate. These guards **MUST** be followed in addition to the existing CoAP congestion control as specified in Section 4.7 of [RFC7252]. See Section 6 for more details.

This mechanism is not intended for general CoAP usage, and any use outside the intended use case should be carefully weighed against the loss of interoperability with generic CoAP applications. It is hoped that the experience gained with this mechanism can feed future extensions of the block-wise mechanism that will both generally applicable and serve this particular use case.

It is not recommended that these options are used in a NoSec security mode (Section 9 of [RFC7252]) as the source endpoint needs to be trusted.

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.

Readers should be familiar with the terms and concepts defined in [RFC7252].

The terms "payload" and "body" are defined in [RFC7959]. The term "payload" is thus used for the content of a single CoAP message (i.e., a single block being transferred), while the term "body" is

used for the entire resource representation that is being transferred in a block-wise fashion.

3. The Q-Block1 and Q-Block2 Options

3.1. Properties of Q-Block1 and Q-Block2 Options

The properties of Q-Block1 and Q-Block2 Options are shown in Table 1. The formatting of this table follows the one used in Table 4 of [RFC7252] (Section 5.10). The C, U, N, and R columns indicate the properties Critical, Unsafe, NoCacheKey, and Repeatable defined in Section 5.4 of [RFC7252]. Only C and U columns are marked for the Q-Block1 Option. C, U, and R columns are marked for the Q-Block2 Option.

Number	C	U	N	R	Name	Format	Length	Default
TBA1	x	x			Q-Block1	uint	0-3	(none)
TBA2	x	x		x	Q-Block2	uint	0-3	(none)

Table 1: CoAP Q-Block1 and Q-Block2 Option Properties

The Q-Block1 and Q-Block2 Options can be present in both the request and response messages. The Q-Block1 Option pertains to the request payload and the Q-Block2 Option pertains to the response payload. The Content-Format Option applies to the body, not to the payload (i.e., it must be the same for all payloads of the same body).

Q-Block1 Option is useful with the payload-bearing POST, PUT, PATCH, and iPATCH requests and their responses (2.01 and 2.04).

Q-Block2 Option is useful with GET, POST, PUT, FETCH, PATCH, and iPATCH requests and their payload-bearing responses (2.01, 2.03, 2.04, and 2.05) (Section 5.5 of [RFC7252]).

A CoAP endpoint (or proxy) MUST support either both or neither of the Q-Block1 and Q-Block2 Options.

To indicate support for Q-Block2 responses, the CoAP client MUST include the Q-Block2 Option in a GET or similar request, the Q-Block2 Option in a PUT or similar request, or the Q-Block1 Option in a PUT or similar so that the server knows that the client supports this Q-Block2 functionality should it need to send back a body that spans multiple payloads. Otherwise, the server would use the Block2 Option (if supported) to send back a message body that is too large to fit into a single IP packet [RFC7959].

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

Implementation of the Q-Block1 and Q-Block2 Options is intended to be optional. However, when it is present in a CoAP message, it **MUST** be processed (or the message rejected). Therefore, Q-Block1 and Q-Block2 Options are identified as Critical options.

The Q-Block1 and Q-Block2 Options are unsafe to forward. That is, a CoAP proxy that does not understand the Q-Block1 (or Q-Block2) Option **MUST** reject the request or response that uses either option.

The Q-Block2 Option is repeatable when requesting re-transmission of missing Blocks, but not otherwise. Except that case, any request carrying multiple Q-Block1 (or Q-Block2) Options **MUST** be handled following the procedure specified in Section 5.4.5 of [RFC7252].

The Q-Block1 and Q-Block2 Options, like the Block1 and Block2 Options, are both a class E and a class U in terms of OSCORE processing (see Section 4.1 of [RFC8613]): The Q-Block1 (or Q-Block2) Option **MAY** be an Inner or Outer option. The Inner and Outer values are therefore independent of each other. The Inner option is encrypted and integrity protected between clients and servers, 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.

3.2. Structure of Q-Block1 and Q-Block2 Options

The structure of Q-Block1 and Q-Block2 Options follows the structure defined in Section 2.2 of [RFC7959].

There is no default value for the Q-Block1 and Q-Block2 Options. Absence of one of these options is equivalent to an option value of 0 with respect to the value of block number (NUM) and more bit (M) that could be given in the option, i.e., it indicates that the current block is the first and only block of the transfer (block number is set to 0, M is unset). However, in contrast to the explicit value 0, which would indicate a size of the block (SZX) of 0, and thus a size value of 16 bytes, there is no specific explicit size implied by the absence of the option -- the size is left unspecified. (As for any uint, the explicit value 0 is efficiently indicated by a zero-length

option; this, therefore, is different in semantics from the absence of the option).

3.3. Using the Q-Block1 Option

The Q-Block1 Option is used when the client wants to send a large amount of data to the server using the POST, PUT, PATCH, or iPATCH methods where the data and headers do not fit into a single packet.

When Q-Block1 Option is used, the client MUST include a single Request-Tag Option [I-D.ietf-core-echo-request-tag]. The Request-Tag value MUST be the same for all of the blocks in the body of data that is being transferred. It is also used to identify a particular block of a body that needs to be re-transmitted. The Request-Tag is opaque in nature, but it is RECOMMENDED that the client treats it as an unsigned integer of 8 bytes in length. An implementation may want to consider limiting this to 4 bytes to reduce packet overhead size. The server still treats it as an opaque entity. The Request-Tag value MUST be different for distinct bodies or sets of blocks of data and SHOULD be incremented whenever a new body of data is being transmitted for a CoAP session between peers. The initial Request-Tag value SHOULD be randomly generated by the client.

For Confirmable transmission, the server MUST continue to acknowledge each packet. NSTART will also need to be increased from the default (1) to get faster transmission rates.

Each individual payload of the body is treated as a new request (see Section 5).

A 2.01 (Created) or 2.04 (Changed) Response Code indicates successful receipt of the entire body.

The 2.31 (Continue) Response is not used in the current version of the specification.

A 4.00 (Bad Request) Response Code MUST be returned if the request does not include a Request-Tag Option but does include a Q-Block1 option.

A 4.02 (Bad Option) Response Code MUST be returned if the server does not support the Q-Block1 Option.

A 4.13 (Request Entity Too Large) Response Code can be returned under similar conditions to those discussed in Section 2.9.3 of [RFC7959].

A 4.08 (Request Entity Incomplete) Response Code returned without Content-Type "application/missing-blocks+cbor-seq" (Section 10.2) is handled as in Section 2.9.2 [RFC7959].

A 4.08 (Request Entity Incomplete) Response Code returned with Content-Type "application/missing-blocks+cbor-seq" indicates that some of the payloads are missing and need to be resent. The client then re-transmits the missing payloads using the Request-Tag and Q-Block1 to specify the block number, SZX, and M bit as appropriate. The Request-Tag value to use is determined from the payload of the 4.08 (Request Entity Incomplete) Response Code. If the client does not recognize the Request-Tag, the client can ignore this response.

If the server has not received all the payloads of a body, but one or more payloads have been received, it SHOULD wait for up to MAX_TRANSMIT_SPAN (Section 4.8.2 of [RFC7252]) before sending the 4.08 (Request Entity Incomplete) Response Code. However, this time MAY be reduced to two times ACK_TIMEOUT before sending a 4.08 (Request Entity Incomplete) Response Code to cover the situation where MAX_PAYLOADS has been triggered by the client causing a break in transmission.

If the client transmits a new body of data with a new Request-Tag to the same resource on a server, the server MUST remove any partially received body held for a previous Request-Tag for that resource.

If the server receives a duplicate block with the same Request-Tag, it SHOULD silently ignore the packet.

A server SHOULD only maintain a partial body (missing payloads) for up to EXCHANGE_LIFETIME (Section 4.8.2 of [RFC7252]).

3.4. Using the Q-Block2 Option

In a request for any block number, the M bit unset indicates the request is just for that block. If the M bit is set, this indicates that this is a request for this block and for all of the remaining blocks within the body. If the server receives multiple requests (implied or otherwise) for the same block, it MUST only send back one instance of that block.

The payloads sent back from the server as a response MUST all have the same ETag (Section 5.10.6 of [RFC7252]) for the same body. The server MUST NOT use the same ETag value for different representations of a resource.

The ETag is opaque in nature, but it is RECOMMENDED that the server treats it as an unsigned integer of 8 bytes in length. An

implementation may want to consider limiting this to 4 bytes to reduce packet overhead size. The client still treats it as an opaque entity. The ETag value MUST be different for distinct bodies or sets of blocks of data and SHOULD be incremented whenever a new body of data is being transmitted for a CoAP session between peers. The initial ETag value SHOULD be randomly generated by the server.

If the client detects that some of the payloads are missing, the missing payloads are requested by issuing a new GET, POST, PUT, FETCH, PATCH, or iPATCH request that contains one or more Q-Block2 Options that define the missing blocks with the M bit unset.

The requested missing block numbers MUST have an increasing block number in each additional Q-Block2 Option with no duplicates. The server SHOULD respond with a 4.00 (Bad Request) if this is the case.

The ETag Option MUST NOT be used in the request as the server could respond with a 2.03 (Valid Response) with no payload. If the server responds with a different ETag Option value (as the resource representation has changed), then the client SHOULD drop all the payloads for the current body that are no longer valid.

The client may elect to request the missing blocks or just ignore the partial body. It SHOULD wait for up to MAX_TRANSMIT_SPAN (Section 4.8.2 of [RFC7252]) before issuing a GET, POST, PUT, FETCH, PATCH, or iPATCH request for the missing blocks. However, this time MAY be reduced to two times ACK_TIMEOUT before sending the request to cover the situation where MAX_PAYLOADS has been triggered by the server causing a break in transmission.

With NON transmission, the client only needs to indicate that some of the payloads are missing by issuing a GET, POST, PUT, FETCH, PATCH, or iPATCH request for the missing blocks.

For Confirmable transmission, the client SHOULD continue to acknowledge each packet as well as issuing a separate GET, POST, PUT, FETCH, PATCH, or iPATCH for the missing blocks.

If the server transmits a new body of data (e.g., a triggered Observe) with a new ETag to the same client as an additional response, the client MUST remove any partially received body held for a previous ETag.

If the client receives a duplicate block with the same ETag, it SHOULD silently ignore the packet.

A client SHOULD only maintain a partial body (missing payloads) for up to EXCHANGE_LIFETIME (Section 4.8.2 of [RFC7252]) or as defined by the Max-Age Option whichever is the less.

If there is insufficient space to create a response PDU with a block size of 16 bytes (SZX = 0) to reflect back all the request options as appropriate, a 4.13 (Request Entity Too Large) is returned without the Size2 Option.

3.5. Working with Observe and Q-Block2 Options

As the blocks of the body are sent without waiting for acknowledgement of the individual blocks, the Observe value [RFC7641] MUST be the same for all the blocks of the same body.

If the client requests missing blocks, this is treated as a new request. The Observe value may change but MUST still be reported. If the ETag value changes then the previously received partial body should be destroyed and the whole body re-requested.

3.6. Working with Size1 and Size2 Options

Section 4 of [RFC7959] defines two CoAP options: Size1 for indicating the size of the representation transferred in requests and Size2 for indicating the size of the representation transferred in responses.

The Size1 or Size2 option values MUST exactly represent the size of the data on the body so that any missing data can easily be determined.

The Size1 Option MUST be used with the Q-Block1 Option when used in a request. The Size2 Option MUST be used with the Q-Block2 Option when used in a response.

If Size1 or Size2 Options are used, they MUST be used in all payloads of the body and MUST have the same value.

3.7. Use of Q-Block1 and Q-Block2 Options Together

The behavior is similar to the one defined in Section 3.3 of [RFC7959] with Q-Block1 substituted for Block1 and Q-Block2 for Block2.

4. The Use of 4.08 (Request Entity Incomplete) Response Code

4.08 (Request Entity Incomplete) Response Code has a new Content-Type "application/missing-blocks+cbor-seq" used to indicate that the

server has not received all of the blocks of the request body that it needs to proceed.

Likely causes are the client has not sent all blocks, some blocks were dropped during transmission, or the client has sent them sufficiently long ago that the server has already discarded them.

The data payload of the 4.08 (Request Entity Incomplete) Response Code is encoded as a CBOR Sequence [RFC8742]. First is CBOR encoded Request-Tag followed by 1 or more missing CBOR encoded missing block numbers. The missing block numbers MUST be unique in each 4.08 (Request Entity Incomplete) when created by the server; the client SHOULD drop any duplicates in the same 4.08 (Request Entity Incomplete) message.

The Content-Format Option (Section 5.10.3 of [RFC7252]) MUST be used in the 4.08 (Request Entity Incomplete) Response Code. It MUST be set to "application/missing-blocks+cbor-seq" (see Section 10.2).

The Concise Data Definition Language [RFC8610] for the data describing these missing blocks is as follows:

```
; This defines an array, the elements of which are to be used
; in a CBOR Sequence:
payload = [request-tag, + missing-block-number]
request-tag = bstr
; A unique block number not received:
missing-block-number = uint
```

Figure 1: Structure of the Missing Blocks Payload

If the size of the 4.08 (Request Entity Incomplete) response packet is larger than that defined by Section 4.6 [RFC7252], then the number of missing blocks MUST be limited so that the response can fit into a single packet. If this is the case, then the server can send subsequent 4.08 (Request Entity Incomplete) responses containing the missing blocks on receipt of a new request providing a missing payload with the same Request-Tag.

The missing blocks MUST be reported in ascending order without any duplicates. The client SHOULD silently drop 4.08 (Request Entity Incomplete) responses not adhering with this behavior.

Implementation Note: Updating the payload without overflowing the overall packet size as each block number can be of varying length needs consideration. It is possible to use Indefinite-Length Arrays (Section 2.2.1 of [RFC7049]), limit the array count to 23 (Undefined value) so that the array data byte can be updated with

the overall length once the payload length is confirmed or limited to MAX_PAYLOADS count. Limiting the count to MAX_PAYLOADS means that Congestion Control is less likely to be invoked on the server.

5. The Use of Tokens

Each new request MUST use a unique Token (Section 4 of [I-D.ietf-core-echo-request-tag]). Additional responses may use the same Token.

Implementation Note: To minimize on the number of tokens that have to be tracked by clients, it is recommended that the bottom 32 bits is kept the same for the same body and the upper 32 bits contains the individual payload number.

Servers continue to treat the token as a unique opaque entity. If an individual payload has to be resent (e.g., requested upon packet loss), then the retransmitted packet is treated as a new request (i.e., the bottom 32 bits must change).

6. Congestion Control

PROBING_RATE parameter in CoAP indicates the average data rate that must not be exceeded by a CoAP endpoint in sending to a peer endpoint that does not respond. The body of blocks will be subjected to PROBING_RATE (Section 4.7 of [RFC7252]).

Each NON 4.08 (Request Entity Incomplete) Response Codes is subjected to PROBING_RATE.

Each NON GET or similar request using Q-Block2 Option is subjected to PROBING_RATE.

As the sending of many payloads of a single body may itself cause congestion, it is RECOMMENDED that after transmission of every set of MAX_PAYLOADS payloads of a single body, a delay is introduced of ACK_TIMEOUT (Section 4.8.2 of [RFC7252]) before the next set of payload transmissions to manage potential congestion issues. MAX_PAYLOADS should be configurable with a default value of 10.

Note: The default value is chosen for reasons similar to those discussed in Section 5 of [RFC6928].

For NON transmissions, it is permissible, but not required, to send the ultimate payload of a MAX_PAYLOADS set as a Confirmable packet. If a Confirmable packet is used, then the transmitting peer MUST wait

for the ACK to be returned before sending the next set of payloads, which can be in time terms less than the ACK_TIMEOUT delay.

Also, for NON transmissions, it is permissible, but not required, to send a Confirmable packet for the final payload of a body transfer (that is, M bit unset). If a Confirmable packet is used, then the transmitting peer MUST wait for the appropriate response to be returned for successful transmission, or respond to requests for the missing blocks (if any).

The sending of the set of missing blocks is subject to MAX_PAYLOADS.

Note: A delay of ACK_TIMEOUT after every transmission of MAX_PAYLOADS blocks may be observed even if the peer agent is able to handle more blocks without experiencing an overload. This delay can be reduced by using CON for the MAX_PAYLOADS packet to trigger sending the next set of data when the ACK is received. Nevertheless, this behavior is likely to create other timeout issues in a lossy environment (e.g., unidirectional loss as in DDoS pipe flooding). The use of NON is thus superior but requires an additional signal in the MAX_PAYLOADS packet to seek for a 2.31 (Continue) from the peer if it is ready to receive the next set of blocks.

7. Caching Considerations

Caching block based information is not straight forward in a proxy. For Q-Block1 and Q-Block2 Options, it is expected that the proxy will reassemble the body (using any appropriate recovery options for packet loss) before passing on the body to the appropriate CoAP endpoint. The onward transmission of the body does not require the use of the Q-Block1 or Q-Block2 Options as these options may not be supported in that link. This means that the proxy must fully support the Q-Block1 and Q-Block2 Options.

How the body is cached in the initial CoAP client (Q-Block1) or ultimate CoAP server (Q-Block2) is implementation specific.

As the entire body is being cached in the proxy, the Q-Block1 and Q-Block2 Options are not part of the cache key.

For Q-Block2 responses, the ETag Option value is associated with the data (and onward transmitted to the CoAP client), but is not part of the cache key.

For requests with Q-Block1 Option, the Request-Tag Option is associated with the build up of the body from successive payloads,

but is not part of the cache key. For the onward transmission of the body using CoAP, a new Request-Tag SHOULD be generated and used.

It is possible that two or more CoAP clients are concurrently updating the same resource through a common proxy to the same CoAP server using Q-Block1 (or Block1) Option. If this is the case, the first client to complete building the body causes that body to start transmitting to the CoAP server with an appropriate Request-Tag value. When the next client completes building the body, any existing partial body transmission to the CoAP server is terminated and the new body representation transmission starts with a new Request-Tag value.

A proxy that supports Q-Block2 Option MUST be prepared to receive a GET or similar message indicating one or more missing blocks. The proxy will serve from its cache the missing blocks that are available in its cache in the same way a server would send all the appropriate Q-Block2s. If the cache key matching body is not available in the cache, the proxy MUST request the entire body from the CoAP server using the information in the cache key.

How long a CoAP endpoint (or proxy) keeps the body in its cache is implementation specific (e.g., it may be based on Max-Age).

8. HTTP-Mapping Considerations

As a reminder, the basic normative requirements on HTTP/CoAP mappings are defined in Section 10 of [RFC7252]. The implementation guidelines for HTTP/CoAP mappings are elaborated in [RFC8075].

The rules defined in Section 5 of [RFC7959] are to be followed.

9. Examples of Selective Block Recovery

This section provides some sample flows to illustrate the use of Q-Block1 and Q-Block2 Options. Figure 2 lists the conventions that are used in the following subsections.

T: Token value
 O: Observe Option value
 M: Message ID
 RT: Request-Tag
 ET: ETag
 QB1: Q-Block1 Option values NUM/More/SZX
 QB2: Q-Block2 Option values NUM/More/SZX
 \: Trimming long lines
 [[]]: Comments
 -->X: Message loss
 X<--: Message loss

Figure 2: Notations Used in the Figures

9.1. Q-Block1 Option: Non-Confirmable Example

Figure 3 depicts an example of a NON PUT request conveying Q-Block1 Option. All the blocks are received by the server.

CoAP Client	CoAP Server
+----->	NON PUT /path M:0x01 T:0xf0 RT=10 QB1:0/1/1024
+----->	NON PUT /path M:0x02 T:0xf1 RT=10 QB1:1/1/1024
+----->	NON PUT /path M:0x03 T:0xf2 RT=10 QB1:2/1/1024
+----->	NON PUT /path M:0x04 T:0xf3 RT=10 QB1:3/0/1024
<-----+	NON 2.04 M:0xf1 T:0xf3
...	

Figure 3: Example of NON Request with Q-Block1 Option (Without Loss)

Consider now a scenario where a new body of data is to be sent by the client, but some blocks are dropped in transmission as illustrated in Figure 4.

CoAP Client	CoAP Server
+----->	NON PUT /path M:0x05 T:0xe0 RT=11 QB1:0/1/1024
+--->X	NON PUT /path M:0x06 T:0xe1 RT=11 QB1:1/1/1024
+--->X	NON PUT /path M:0x07 T:0xe2 RT=11 QB1:2/1/1024
+----->	NON PUT /path M:0x08 T:0xe3 RT=11 QB1:3/0/1024
...	

Figure 4: Example of NON Request with Q-Block1 Option (With Loss)

The server realizes that some blocks are missing and asks for the missing ones in one go (Figure 5). It does so by indicating which blocks have been received in the data portion of the response.

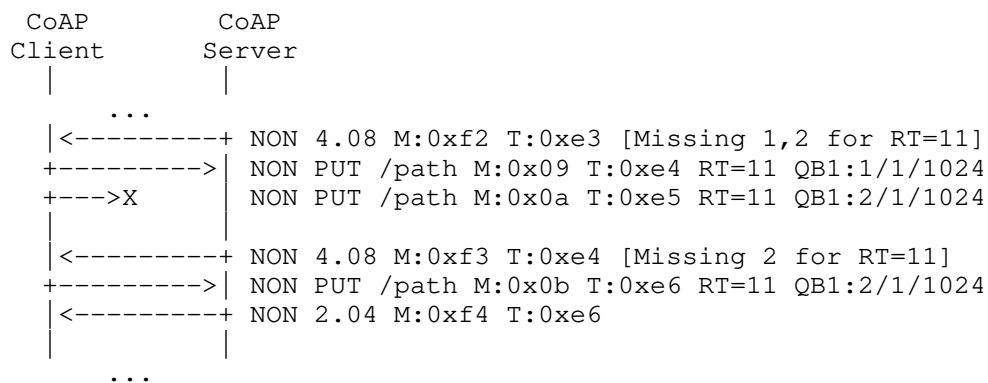


Figure 5: Example of NON Request with Q-Block1 Option (Blocks Recovery)

Under high levels of traffic loss, the client can elect not to retry sending missing blocks of data. This decision is implementation specific.

9.2. Q-Block2 Option: Non-Confirmable Example

Figure 6 illustrates the example of Q-Block2 Option. The client sends a NON GET carrying an Observe and a Q-Block2 Options. The Q-Block2 Option indicates a size hint (1024 bytes). This request is replied by the server using four (4) blocks that are transmitted to the client without any loss. Each of these blocks carries a Q-Block2 Option. The same process is repeated when an Observe is triggered, but no loss is experienced by any of the notification blocks.

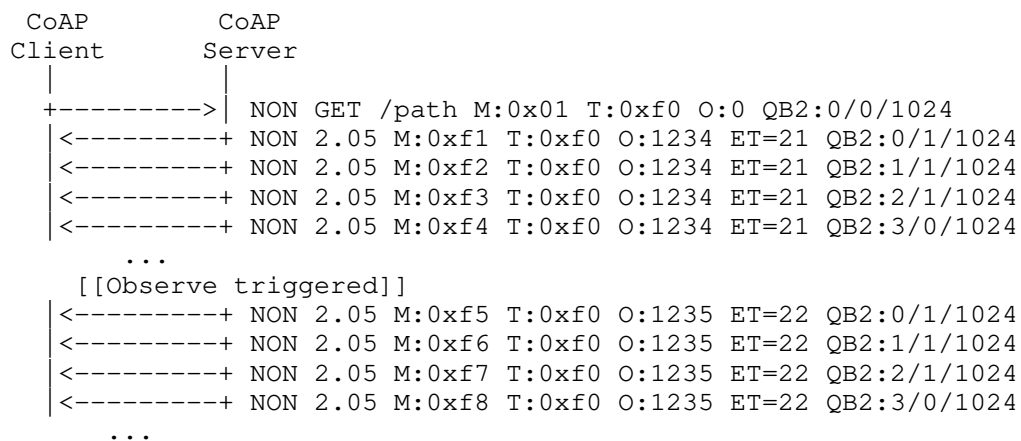


Figure 6: Example of NON Notifications with Q-Block2 Option (Without Loss)

Figure 7 shows the example of an Observe that is triggered but for which some notification blocks are lost. The client detects the missing blocks and request their retransmission. It does so by indicating the blocks that were successfully received.

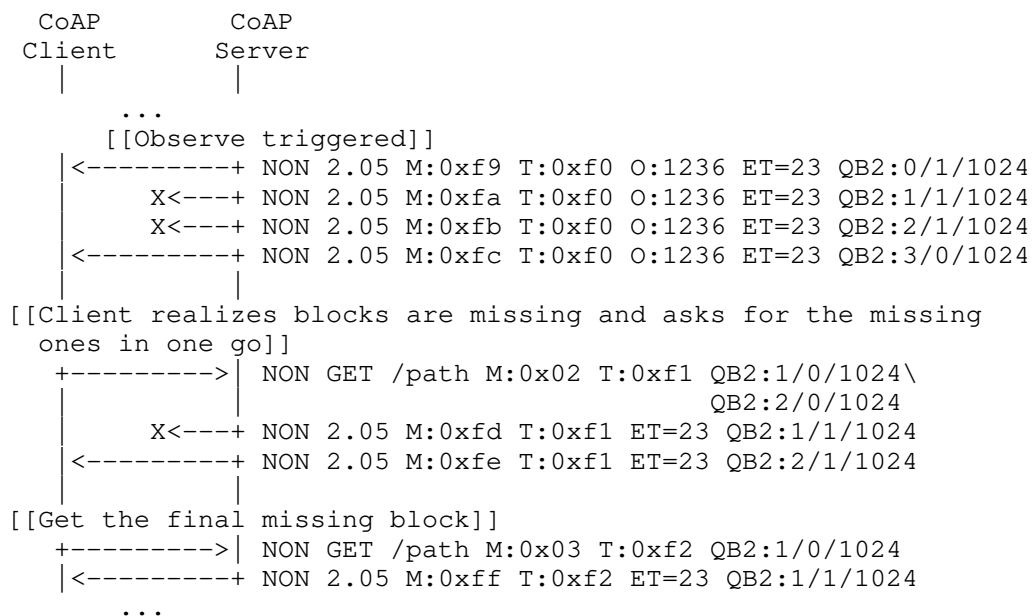


Figure 7: Example of NON Notifications with Q-Block2 Option (Blocks Recovery)

Under high levels of traffic loss, the client can elect not to retry getting missing blocks of data. This decision is implementation specific.

10. IANA Considerations

10.1. New CoAP Options

IANA is requested to add the following entries to the "CoAP Option Numbers" sub-registry [Options]:

Number	Name	Reference
TBA1	Q-Block1	[RFCXXXX]
TBA2	Q-Block2	[RFCXXXX]

Table 2: CoAP Q-Block1 and Q-Block2 Option Numbers

This document suggests 19 (TBA1) and 51 (TBA2) as a values to be assigned for the new option numbers.

10.2. New Content Format

This document requests IANA to register the CoAP Content-Format ID for the "application/missing-blocks+cbor-seq" media type in the "CoAP Content-Formats" registry [Format]:

- o Media Type: application/missing-blocks+cbor-seq
- o Encoding: -
- o Id: TBD3
- o Reference: [RFCXXXX]

11. Security Considerations

Security considerations discussed in Section 7 of [RFC7959] should be taken into account.

Security considerations discussed in Sections 11.3 and 11.4 of [RFC7252] should be taken into account. In particular, it is NOT RECOMMENDED that the NoSec security mode is used if the Q-Block1 and Q-Block2 Options are to be used.

Security considerations related to the use of Request-Tag are discussed in Section 5 of [I-D.ietf-core-echo-request-tag].

12. Acknowledgements

Thanks to Achim Kraus, Jim Schaad, and Michael Richardson for the comments on the mailing list.

Special thanks to Christian Amsuess and Carsten Bormann for their suggestions and several reviews, which improved this specification significantly.

Some text from [RFC7959] is reused for readers convenience.

13. References

13.1. Normative References

- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-10 (work in progress), July 2020.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <<https://www.rfc-editor.org/info/rfc8742>>.

13.2. Informative References

- [Format] , <<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>>.
- [I-D.ietf-dots-telemetry]
Boucadair, M., Reddy, K. T., Doron, E., chenmeiling, c., and J. Shallow, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Telemetry", draft-ietf-dots-telemetry-13 (work in progress), October 2020.
- [Options] , <<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#option-numbers>>.
- [RFC6928] Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <<https://www.rfc-editor.org/info/rfc6928>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8782] Reddy, K. T., Ed., Boucadair, M., Ed., Patil, P., Mortensen, A., and N. Teague, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", RFC 8782, DOI 10.17487/RFC8782, May 2020, <<https://www.rfc-editor.org/info/rfc8782>>.

Appendix A. Examples with Confirmable Messages

These examples assume NSTART has been increased to at least 4.

The notations provided in Figure 2 are used in the following subsections.

A.1. Q-Block1 Option

Let's now consider the use Q-Block1 Option with a CON request as shown in Figure 8. All the blocks are acknowledged (ACK).

```

CoAP Client      CoAP Server
|               |
+----->| CON PUT /path M:0x01 T:0xf0 RT=10 QB1:0/1/1024
+----->| CON PUT /path M:0x02 T:0xf1 RT=10 QB1:1/1/1024
+----->| CON PUT /path M:0x03 T:0xf2 RT=10 QB1:2/1/1024
+----->| CON PUT /path M:0x04 T:0xf3 RT=10 QB1:3/0/1024
|         |
|<-----+ ACK 0.00 M:0x01
|<-----+ ACK 0.00 M:0x02
|<-----+ ACK 0.00 M:0x03
|<-----+ ACK 2.04 M:0x04

```

Figure 8: Example of CON Request with Q-Block1 Option (Without Loss)

Now, suppose that a new body of data is to sent but with some blocks dropped in transmission as illustrated in Figure 9. The client will retry sending blocks for which no ACK was received.

```

CoAP Client      CoAP Server
|               |
+----->| CON PUT /path M:0x05 T:0xf4 RT=11 QB1:0/1/1024
+---->X| CON PUT /path M:0x06 T:0xf5 RT=11 QB1:1/1/1024
+---->X| CON PUT /path M:0x07 T:0xf6 RT=11 QB1:2/1/1024
+----->| CON PUT /path M:0x08 T:0xf7 RT=11 QB1:3/1/1024
|         |
|<-----+ ACK 0.00 M:0x05
|<-----+ ACK 0.00 M:0x08
|
[[The client retries sending packets not acknowledged]]
+----->| CON PUT /path M:0x06 T:0xf5 RT=11 QB1:1/1/1024
+---->X| CON PUT /path M:0x07 T:0xf6 RT=11 QB1:2/1/1024
|<-----+ ACK 0.00 M:0x06
|
[[The client retransmits messages not acknowledged
(exponential backoff)]]
+---->?| CON PUT /path M:0x07 T:0xf6 RT=11 QB1:2/1/1024
|
[[Either transmission failure (acknowledge retry timeout)
or successfully transmitted.]]

```

Figure 9: Example of CON Request with Q-Block1 Option (Blocks Recovery)

It is implementation dependent as to whether a CoAP session is terminated following acknowledge retry timeout, or whether the CoAP session continues to be used under such adverse traffic conditions.

If there is likely to be the possibility of network transient losses, then the use of Non-confirmable traffic should be considered.

A.2. Q-Block2 Option

An example of the use of Q-Block2 Option with Confirmable messages is shown in Figure 10.

```

Client      Server
|           |
+----->| CON GET /path M:0x01 T:0xf0 O:0 QB2:0/0/1024
|         |
|         +-----+ ACK 2.05 M:0x01 T:0xf0 O:1234 ET=21 QB2:0/1/1024
|         |
|         +-----+ ACK 2.05 M:0xe1 T:0xf0 O:1234 ET=21 QB2:1/1/1024
|         |
|         +-----+ ACK 2.05 M:0xe2 T:0xf0 O:1234 ET=21 QB2:2/1/1024
|         |
|         +-----+ ACK 2.05 M:0xe3 T:0xf0 O:1234 ET=21 QB2:3/0/1024
|         |
|         ...
|         [[Observe triggered]]
|         |
|         +-----+ CON 2.05 M:0xe4 T:0xf0 O:1235 ET=22 QB2:0/1/1024
|         |
|         +-----+ CON 2.05 M:0xe5 T:0xf0 O:1235 ET=22 QB2:1/1/1024
|         |
|         +-----+ CON 2.05 M:0xe6 T:0xf0 O:1235 ET=22 QB2:2/1/1024
|         |
|         +-----+ CON 2.05 M:0xe7 T:0xf0 O:1235 ET=22 QB2:3/0/1024
|         |
|         +----->+ ACK 0.00 M:0xe4
|         |
|         +----->+ ACK 0.00 M:0xe5
|         |
|         +----->+ ACK 0.00 M:0xe6
|         |
|         +----->+ ACK 0.00 M:0xe7
|         |
|         ...
|         [[Observe triggered]]
|         |
|         +-----+ CON 2.05 M:0xe8 T:0xf0 O:1236 ET=23 QB2:0/1/1024
|         |
|         X<---+ CON 2.05 M:0xe9 T:0xf0 O:1236 ET=23 QB2:1/1/1024
|         |
|         X<---+ CON 2.05 M:0xea T:0xf0 O:1236 ET=23 QB2:2/1/1024
|         |
|         +-----+ CON 2.05 M:0xeb T:0xf0 O:1236 ET=23 QB2:3/0/1024
|         |
|         +----->+ ACK 0.00 M:0xe8
|         |
|         +----->+ ACK 0.00 M:0xeb
|         |
|         [[Server retransmits messages not acknowledged]]
|         |
|         +-----+ CON 2.05 M:0xe9 T:0xf0 O:1236 ET=23 QB2:1/1/1024
|         |
|         X<---+ CON 2.05 M:0xea T:0xf0 O:1236 ET=23 QB2:2/1/1024
|         |
|         +----->+ ACK 0.00 M:0xe9
|         |
|         [[Server retransmits messages not acknowledged
|         (exponential backoff)]]
|         |
|         X<---+ CON 2.05 M:0xea T:0xf0 O:1236 ET=23 QB2:2/1/1024
|         |
|         [[Either transmission failure (acknowledge retry timeout)
|         or successfully transmitted.]]

```

Figure 10: Example of CON Notifications with Q-Block2 Option

It is implementation-dependent as to whether a CoAP session is terminated following acknowledge retry timeout, or whether the CoAP session continues to be used under such adverse traffic conditions.

If there is likely to be the possibility of network transient losses, then the use of Non-confirmable traffic should be considered.

Authors' Addresses

Mohamed Boucadair
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Jon Shallow
United Kingdom

Email: supjps-ietf@jpshallow.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 27, 2021

M. Boucadair
Orange
J. Shallow
May 26, 2021

Constrained Application Protocol (CoAP) Block-Wise Transfer Options
Supporting Robust Transmission
draft-ietf-core-new-block-14

Abstract

This document specifies alternative Constrained Application Protocol (CoAP) Block-Wise transfer options: Q-Block1 and Q-Block2 Options.

These options are similar to, but distinct from, the CoAP Block1 and Block2 Options defined in RFC 7959. Q-Block1 and Q-Block2 Options are not intended to replace Block1 and Block2 Options, but rather have the goal of supporting Non-confirmable messages for large amounts of data with fewer packet interchanges. Also, the Q-Block1 and Q-Block2 Options support faster recovery should any of the blocks get lost in transmission.

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 November 27, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Alternative CoAP Block-Wise Transfer Options	5
3.1. CoAP Response Code (4.08) Usage	7
3.2. Applicability Scope	7
4. The Q-Block1 and Q-Block2 Options	8
4.1. Properties of Q-Block1 and Q-Block2 Options	8
4.2. Structure of Q-Block1 and Q-Block2 Options	10
4.3. Using the Q-Block1 Option	11
4.4. Using the Q-Block2 Option	15
4.5. Using Observe Option	17
4.6. Using Size1 and Size2 Options	17
4.7. Using Q-Block1 and Q-Block2 Options Together	18
4.8. Using Q-Block2 Option With Multicast	18
5. The Use of 4.08 (Request Entity Incomplete) Response Code	18
6. The Use of Tokens	19
7. Congestion Control for Unreliable Transports	20
7.1. Confirmable (CON)	20
7.2. Non-confirmable (NON)	20
8. Caching Considerations	25
9. HTTP-Mapping Considerations	26
10. Examples with Non-confirmable Messages	26
10.1. Q-Block1 Option	27
10.1.1. A Simple Example	27
10.1.2. Handling MAX_PAYLOADS Limits	27
10.1.3. Handling MAX_PAYLOADS with Recovery	27
10.1.4. Handling Recovery with Failure	29
10.2. Q-Block2 Option	30
10.2.1. A Simple Example	30
10.2.2. Handling MAX_PAYLOADS Limits	31
10.2.3. Handling MAX_PAYLOADS with Recovery	32
10.2.4. Handling Recovery using M-bit Set	33
10.3. Q-Block1 and Q-Block2 Options	34
10.3.1. A Simple Example	34
10.3.2. Handling MAX_PAYLOADS Limits	35
10.3.3. Handling Recovery	36
11. Security Considerations	38
12. IANA Considerations	39
12.1. CoAP Option Numbers Registry	39

12.2. Media Type Registration	39
12.3. CoAP Content-Formats Registry	40
13. References	41
13.1. Normative References	41
13.2. Informative References	42
Appendix A. Examples with Confirmable Messages	43
A.1. Q-Block1 Option	43
A.2. Q-Block2 Option	45
Appendix B. Examples with Reliable Transports	47
B.1. Q-Block1 Option	47
B.2. Q-Block2 Option	47
Acknowledgements	48
Authors' Addresses	48

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252], although inspired by HTTP, was designed to use UDP instead of TCP. The message layer of CoAP over UDP includes support for reliable delivery, simple congestion control, and flow control. CoAP supports two message types (Section 1.2 of [RFC7252]): Confirmable (CON) and Non-confirmable (NON) messages. Unlike NON messages, every CON message will elicit an acknowledgement or a reset.

The CoAP specification recommends that a CoAP message should fit within a single IP packet (i.e., avoid IP fragmentation). To handle data records that cannot fit in a single IP packet, [RFC7959] introduced the concept of block-wise transfer and the companion CoAP Block1 and Block2 Options. However, this concept is designed to work exclusively with Confirmable messages (Section 1 of [RFC7959]). Note that the block-wise transfer was further updated by [RFC8323] for use over TCP, TLS, and WebSockets.

The CoAP Block1 and Block2 Options work well in environments where there are no, or minimal, packet losses. These options operate synchronously, i.e., each individual block has to be requested. A CoAP endpoint can only ask for (or send) the next block when the transfer of the previous block has completed. Packet transmission rate, and hence block transmission rate, is controlled by Round Trip Times (RTTs).

There is a requirement for blocks of data larger than a single IP datagram to be transmitted under network conditions where there may be asymmetrical transient packet loss (e.g., acknowledgment responses may get dropped). An example is when a network is subject to a Distributed Denial of Service (DDoS) attack and there is a need for DDoS mitigation agents relying upon CoAP to communicate with each other (e.g., [RFC8782][I-D.ietf-dots-telemetry]). As a reminder,

[RFC7959] recommends the use of CON responses to handle potential packet loss. However, such a recommendation does not work with a flooded pipe DDoS situation (e.g., [RFC8782]).

This document introduces the CoAP Q-Block1 and Q-Block2 Options which allow block-wise transfer to work with series of Non-confirmable messages, instead of lock-stepping using Confirmable messages (Section 3). In other words, this document provides a missing piece of [RFC7959], namely the support of block-wise transfer using Non-confirmable where an entire body of data can be transmitted without the requirement that intermediate acknowledgments be received from the peer (but recovery is available should it be needed).

Similar to [RFC7959], this specification does not remove any of the constraints posed by the base CoAP specification [RFC7252] it is strictly layered on top of.

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.

Readers should be familiar with the terms and concepts defined in [RFC7252], [RFC7959], and [RFC8132]. Particularly, the document uses the following key concepts:

Token: is used to match responses to requests independently from the underlying messages (Section 5.3.1 of [RFC7252]).

ETag: is used as a resource-local identifier for differentiating between representations of the same resource that vary over time (Section 5.10.6 of [RFC7252]).

The terms "payload" and "body" are defined in [RFC7959]. The term "payload" is thus used for the content of a single CoAP message (i.e., a single block being transferred), while the term "body" is used for the entire resource representation that is being transferred in a block-wise fashion.

Request-Tag refers to an option that allows a CoAP server to match message fragments belonging to the same request [I-D.ietf-core-echo-request-tag].

MAX_PAYLOADS is the maximum number of payloads that can be transmitted at any one time.

MAX_PAYLOADS_SET is the set of blocks identified by block numbers that, when divided by MAX_PAYLOADS, have the same numeric result. For example, if MAX_PAYLOADS is set to '10', a MAX_PAYLOADS_SET could be blocks #0 to #9, #10 to #19, etc. Depending on the overall data size, there could be fewer than MAX_PAYLOADS blocks in the final MAX_PAYLOADS_SET.

3. Alternative CoAP Block-Wise Transfer Options

This document introduces the CoAP Q-Block1 and Q-Block2 Options. These options are designed to work in particular with NON requests and responses.

Using NON messages, faster transmissions can occur as all the blocks can be transmitted serially (akin to fragmented IP packets) without having to wait for a response or next request from the remote CoAP peer. Recovery of missing blocks is faster in that multiple missing blocks can be requested in a single CoAP message. Even if there is asymmetrical packet loss, a body can still be sent and received by the peer whether the body comprises a single or multiple payloads, assuming no recovery is required.

A CoAP endpoint can acknowledge all or a subset of the blocks. Concretely, the receiving CoAP endpoint either informs the CoAP sender endpoint of successful reception or reports on all blocks in the body that have not yet been received. The CoAP sender endpoint will then retransmit only the blocks that have been lost in transmission.

Note that similar transmission rate benefits can be applied to Confirmable messages if the value of NSTART is increased from 1 (Section 4.7 of [RFC7252]). However, the use of Confirmable messages will not work effectively if there is asymmetrical packet loss. Some examples with Confirmable messages are provided in Appendix A.

There is little, if any, benefit of using these options with CoAP running over a reliable connection [RFC8323]. In this case, there is no differentiation between CON and NON as they are not used. Some examples using a reliable transport are provided in Appendix B.

Q-Block1 and Q-Block2 Options are similar in operation to the CoAP Block1 and Block2 Options, respectively. They are not a replacement for them, but have the following benefits:

- o They can operate in environments where packet loss is highly asymmetrical.

- o They enable faster transmissions of sets of blocks of data with fewer packet interchanges.
- o They support faster recovery should any of the blocks get lost in transmission.
- o They support sending an entire body using NON messages without requiring that an intermediate response be received from the peer.

There are the following disadvantages over using CoAP Block1 and Block2 Options:

- o Loss of lock-stepping so payloads are not always received in the correct (block ascending) order.
- o Additional congestion control measures need to be put in place for NON messages (Section 7.2).
- o To reduce the transmission times for CON transmission of large bodies, NSTART needs to be increased from 1, but this affects congestion control and incurs a requirement to re-tune other parameters (Section 4.7 of [RFC7252]). Such tuning is out of scope of this document.
- o Mixing of NON and CON during requests/responses using Q-Block is not supported.
- o The Q-Block Options do not support stateless operation/random access.
- o Proxying of Q-Block is limited to caching full representations.
- o There is no multicast support.

Q-Block1 and Q-Block2 Options can be used instead of Block1 and Block2 Options when the different transmission properties are required. If the new options are not supported by a peer, then transmissions can fall back to using Block1 and Block2 Options (Section 4.1).

The deviations from Block1 and Block2 Options are specified in Section 4. Pointers to appropriate [RFC7959] sections are provided.

The specification refers to the base CoAP methods defined in Section 5.8 of [RFC7252] and the new CoAP methods, FETCH, PATCH, and iPATCH introduced in [RFC8132].

The No-Response Option [RFC7967] was considered but was abandoned as it does not apply to Q-Block2 responses. A unified solution is defined in the document.

3.1. CoAP Response Code (4.08) Usage

This document adds a media type for the 4.08 (Request Entity Incomplete) response defining an additional message format for reporting on payloads using the Q-Block1 Option that are not received by the server.

See Section 5 for more details.

3.2. Applicability Scope

The block-wise transfer specified in [RFC7959] covers the general case using Confirmable messages, but falls short in situations where packet loss is highly asymmetrical or there is no need for an acknowledgement. In other words, there is a need for Non-confirmable support.

The mechanism specified in this document provides roughly similar features to the Block1/Block2 Options. It provides additional properties that are tailored towards the intended use case of Non-confirmable transmission. Concretely, this mechanism primarily targets applications such as DDoS Open Threat Signaling (DOTS) that cannot use CON requests/responses because of potential packet loss and that support application-specific mechanisms to assess whether the remote peer is not overloaded and thus is able to process the messages sent by a CoAP endpoint (e.g., DOTS heartbeats in Section 4.7 of [RFC8782]). Other use cases are when an application sends data but has no need for an acknowledgement of receipt and, any data transmission loss is not critical.

The mechanism includes guards to prevent a CoAP agent from overloading the network by adopting an aggressive sending rate. These guards MUST be followed in addition to the existing CoAP congestion control as specified in Section 4.7 of [RFC7252]. See Section 7 for more details.

Any usage outside the primary use case of Non-confirmable with block transfers should be carefully weighed against the potential loss of interoperability with generic CoAP applications (See the disadvantages listed in Section 3). It is hoped that the experience gained with this mechanism can feed future extensions of the block-wise mechanism that will both be generally applicable and serve this particular use case.

It is not recommended that these options are used in a NoSec security mode (Section 9 of [RFC7252]) as the source endpoint needs to be trusted. Using OSCORE [RFC8613] does provide a security context and, hence, a trust of the source endpoint that prepared the inner OSCORE content. However, even with OSCORE, using a NoSec security mode with these options may still be inadequate, for reasons discussed in Section 11.

4. The Q-Block1 and Q-Block2 Options

4.1. Properties of Q-Block1 and Q-Block2 Options

The properties of the Q-Block1 and Q-Block2 Options are shown in Table 1. The formatting of this table follows the one used in Table 4 of [RFC7252] (Section 5.10). The C, U, N, and R columns indicate the properties Critical, UnSafe, NoCacheKey, and Repeatable defined in Section 5.4 of [RFC7252]. Only Critical and UnSafe columns are marked for the Q-Block1 Option. Critical, UnSafe, and Repeatable columns are marked for the Q-Block2 Option. As these options are UnSafe, NoCacheKey has no meaning and so is marked with a dash.

Number	C	U	N	R	Name	Format	Length	Default
TBA1	x	x	-		Q-Block1	uint	0-3	(none)
TBA2	x	x	-	x	Q-Block2	uint	0-3	(none)

Table 1: CoAP Q-Block1 and Q-Block2 Option Properties

The Q-Block1 and Q-Block2 Options can be present in both the request and response messages. The Q-Block1 Option pertains to the request payload and the Q-Block2 Option pertains to the response payload. When the Content-Format Option is present together with the Q-Block1 or Q-Block2 Option, the option applies to the body not to the payload (i.e., it must be the same for all payloads of the same body).

The Q-Block1 Option is useful with the payload-bearing, e.g., POST, PUT, FETCH, PATCH, and iPATCH requests and their responses.

The Q-Block2 Option is useful, e.g., with GET, POST, PUT, FETCH, PATCH, and iPATCH requests and their payload-bearing responses (response codes 2.01, 2.02, 2.04, and 2.05) (Section 5.5 of [RFC7252]).

A CoAP endpoint (or proxy) MUST support either both or neither of the Q-Block1 and Q-Block2 Options.

If the Q-Block1 Option is present in a request or the Q-Block2 Option is returned in a response, this indicates a block-wise transfer and describes how this specific block-wise payload forms part of the entire body being transferred. If it is present in the opposite direction, it provides additional control on how that payload will be formed or was processed.

To indicate support for Q-Block2 responses, the CoAP client MUST include the Q-Block2 Option in a GET or similar request (FETCH, for example), the Q-Block2 Option in a PUT or similar request (POST, for example), or the Q-Block1 Option in a PUT or similar request so that the server knows that the client supports this Q-Block functionality should it need to send back a body that spans multiple payloads. Otherwise, the server would use the Block2 Option (if supported) to send back a message body that is too large to fit into a single IP packet [RFC7959].

How a client decides whether it needs to include a Q-Block1 or Q-Block2 Option can be driven by a local configuration parameter, triggered by an application (DOTS, for example), etc. Such considerations are out of the scope of the document.

Implementation of the Q-Block1 and Q-Block2 Options is intended to be optional. However, when it is present in a CoAP message, it MUST be processed (or the message rejected). Therefore, Q-Block1 and Q-Block2 Options are identified as Critical options.

With CoAP over UDP, the way a request message is rejected for critical options depends on the message type. A Confirmable message with an unrecognized critical option is rejected with a 4.02 (Bad Option) response (Section 5.4.1 of [RFC7252]). A Non-confirmable message with an unrecognized critical option is either rejected with a Reset message or just silently ignored (Sections 5.4.1 and 4.3 of [RFC7252]). To reliably get a rejection message, it is therefore REQUIRED that clients use a Confirmable message for determining support for Q-Block1 and Q-Block2 Options. This CON message can be sent under the base CoAP congestion control setup specified in Section 4.7 of [RFC7252] (that is, NSTART does not need to be increased (Section 7.1)).

The Q-Block1 and Q-Block2 Options are unsafe to forward. That is, a CoAP proxy that does not understand the Q-Block1 (or Q-Block2) Option must reject the request or response that uses either option (See Section 5.7.1 of [RFC7252]).

The Q-Block2 Option is repeatable when requesting retransmission of missing blocks, but not otherwise. Except that case, any request

carrying multiple Q-Block1 (or Q-Block2) Options MUST be handled following the procedure specified in Section 5.4.5 of [RFC7252].

The Q-Block1 and Q-Block2 Options, like the Block1 and Block2 Options, are of both class E and class U for OSCORE processing (Table 2). The Q-Block1 (or Q-Block2) Option MAY be an Inner or Outer option (Section 4.1 of [RFC8613]). The Inner and Outer values are therefore independent of each other. The Inner option is encrypted and integrity protected between clients and servers, 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.

Number	Name	E	U
TBA1	Q-Block1	x	x
TBA2	Q-Block2	x	x

Table 2: OSCORE Protection of Q-Block1 and Q-Block2 Options

Note that if Q-Block1 or Q-Block2 Options are included in a packet as Inner options, Block1 or Block2 Options MUST NOT be included as Inner options. Similarly, there MUST NOT be a mix of Q-Block and Block for the Outer options. Messages that do not adhere with this behavior MUST be rejected with 4.02 (Bad Option). Q-Block and Block Options can be mixed across Inner and Outer options as these are handled independently of each other. For clarity, if OSCORE is not being used, there MUST NOT be a mix of Q-Block and Block Options in the same packet.

4.2. Structure of Q-Block1 and Q-Block2 Options

The structure of Q-Block1 and Q-Block2 Options follows the structure defined in Section 2.2 of [RFC7959].

There is no default value for the Q-Block1 and Q-Block2 Options. Absence of one of these options is equivalent to an option value of 0 with respect to the value of block number (NUM) and more bit (M) that could be given in the option, i.e., it indicates that the current block is the first and only block of the transfer (block number is set to 0, M is unset). However, in contrast to the explicit value 0, which would indicate a size of the block (SZX) of 0, and thus a size value of 16 bytes, there is no specific explicit size implied by the absence of the option -- the size is left unspecified. (As for any uint, the explicit value 0 is efficiently indicated by a zero-length

option; this, therefore, is different in semantics from the absence of the option).

4.3. Using the Q-Block1 Option

The Q-Block1 Option is used when the client wants to send a large amount of data to the server using the POST, PUT, FETCH, PATCH, or iPATCH methods where the data and headers do not fit into a single packet.

When Q-Block1 Option is used, the client MUST include a Request-Tag Option [I-D.ietf-core-echo-request-tag]. The Request-Tag value MUST be the same for all of the requests for the body of data that is being transferred. The Request-Tag is opaque, but the client MUST ensure that it is unique for every different body of transmitted data.

Implementation Note: It is suggested that the client treats the Request-Tag as an unsigned integer of 8 bytes in length. An implementation may want to consider limiting this to 4 bytes to reduce packet overhead size. The initial Request-Tag value should be randomly generated and then subsequently incremented by the client whenever a new body of data is being transmitted between peers.

Section 4.6 discusses the use of Size1 Option.

For Confirmable transmission, the server continues to acknowledge each packet, but a response is not required (whether separate or piggybacked) until successful receipt of the body by the server. For Non-confirmable transmission, no response is required until either the successful receipt of the body by the server or a timer expires with some of the payloads having not yet arrived. In the latter case, a "retransmit missing payloads" response is needed. For reliable transports (e.g., [RFC8323]), a response is not required until successful receipt of the body by the server.

Each individual message that carries a block of the body is treated as a new request (Section 6).

The client MUST send the payloads in order of increasing block number, starting from zero, until the body is complete (subject to any congestion control (Section 7)). Any missing payloads requested by the server must in addition be separately transmitted with increasing block numbers.

The following Response Codes are used:

2.01 (Created)

This Response Code indicates successful receipt of the entire body and that the resource was created. The token to use MUST be one of the tokens that were received in a request for this block-wise exchange. However, it is desirable to provide the one used in the last received request, since that will aid any troubleshooting. The client should then release all of the tokens used for this body. Note that the last received payload might not be the one with the highest block number.

2.02 (Deleted)

This Response Code indicates successful receipt of the entire body and that the resource was deleted when using POST (Section 5.8.2 [RFC7252]). The token to use MUST be one of the tokens that were received in a request for this block-wise exchange. However, it is desirable to provide the one used in the last received request. The client should then release all of the tokens used for this body.

2.04 (Changed)

This Response Code indicates successful receipt of the entire body and that the resource was updated. The token to use MUST be one of the tokens that were received in a request for this block-wise exchange. However, it is desirable to provide the one used in the last received request. The client should then release all of the tokens used for this body.

2.05 (Content)

This Response Code indicates successful receipt of the entire FETCH request body (Section 2 of [RFC8132]) and that the appropriate representation of the resource is being returned. The token to use MUST be one of the tokens that were received in a request for this block-wise exchange. However, it is desirable to provide the one used in the last received request.

If the FETCH request includes the Observe Option, then the server MUST use the same token as used for the 2.05 (Content) response for returning any Observe triggered responses so that the client can match them up.

The client should then release all of the tokens used for this body apart from the one used for tracking an observed resource.

2.31 (Continue)

This Response Code can be used to indicate that all of the blocks up to and including the Q-Block1 Option block NUM (all having the M bit set) have been successfully received. The token to use MUST be one of the tokens that were received in a request for this latest MAX_PAYLOADS_SET block-wise exchange. However, it is desirable to provide the one used in the last received request.

The client should then release all of the tokens used for this MAX_PAYLOADS_SET.

A response using this Response Code MUST NOT be generated for every received Q-Block1 Option request. It SHOULD only be generated when all the payload requests are Non-confirmable and a MAX_PAYLOADS_SET has been received by the server. More details about the motivations for this optimization are discussed in Section 7.2.

This Response Code SHOULD NOT be generated for CON as this may cause duplicated payloads to unnecessarily be sent.

4.00 (Bad Request)

This Response Code MUST be returned if the request does not include a Request-Tag Option or a Sizer Option but does include a Q-Block1 option.

4.02 (Bad Option)

This Response Code MUST be returned for a Confirmable request if the server does not support the Q-Block Options. Note that a reset message may be sent in case of Non-confirmable request.

4.08 (Request Entity Incomplete)

As a reminder, this Response Code returned without Content-Type "application/missing-blocks+cbor-seq" (Section 12.3) is handled as in Section 2.9.2 [RFC7959].

This Response Code returned with Content-Type "application/missing-blocks+cbor-seq" indicates that some of the payloads are missing and need to be resent. The client then retransmits the individual missing payloads using the same Request-Tag, Sizer, and, Q-Block1 Option to specify the same NUM, SZX, and M bit as sent initially in the original, but not received, packet.

The Request-Tag value to use is determined by taking the token in the 4.08 (Request Entity Incomplete) response, locating the matching client request, and then using its Request-Tag.

The token to use in the 4.08 (Request Entity Incomplete) response MUST be one of the tokens that were received in a request for this block-wise body exchange. However, it is desirable to provide the one used in the last received request. See Section 5 for further information.

If the server has not received all the blocks of a body, but one or more NON payloads have been received, it SHOULD wait for NON_RECEIVE_TIMEOUT (Section 7.2) before sending a 4.08 (Request Entity Incomplete) response.

4.13 (Request Entity Too Large)

This Response Code can be returned under similar conditions to those discussed in Section 2.9.3 of [RFC7959].

This Response Code can be returned if there is insufficient space to create a response PDU with a block size of 16 bytes (SZX = 0) to send back all the response options as appropriate. In this case, the Size1 Option is not included in the response.

Further considerations related to the transmission timings of 4.08 (Request Entity Incomplete) and 2.31 (Continue) Response Codes are discussed in Section 7.2.

If a server receives payloads with different Request-Tags for the same resource, it should continue to process all the bodies as it has no way of determining which is the latest version, or which body, if any, the client is terminating the transmission for.

If the client elects to stop the transmission of a complete body, and absent any local policy, the client MUST "forget" all tracked tokens associated with the body's Request-Tag so that a reset message is generated for the invalid token in the 4.08 (Request Entity Incomplete) response. The server on receipt of the reset message SHOULD delete the partial body.

If the server receives a duplicate block with the same Request-Tag, it MUST ignore the payload of the packet, but MUST still respond as if the block was received for the first time.

A server SHOULD maintain a partial body (missing payloads) for NON_PARTIAL_TIMEOUT (Section 7.2).

4.4. Using the Q-Block2 Option

In a request for any block number, the M bit unset indicates the request is just for that block. If the M bit is set, this has different meanings based on the NUM value:

NUM is zero: This is a request for the entire body.

'NUM modulo MAX_PAYLOADS' is zero, while NUM is not zero: This is used to confirm that the current MAX_PAYLOADS_SET (the latest block having block number NUM-1) has been successfully received and that, upon receipt of this request, the server can continue to send the next MAX_PAYLOADS_SET (the first block having block number NUM). This is the 'Continue' Q-Block-2 and conceptually has the same usage (i.e., continue sending the next set of data) as the use of 2.31 (Continue) for Q-Block1.

Any other value of NUM: This is a request for that block and for all of the remaining blocks in the current MAX_PAYLOADS_SET.

If the request includes multiple Q-Block2 Options and these options overlap (e.g., combination of M being set (this and later blocks) and being unset (this individual block)) resulting in an individual block being requested multiple times, the server MUST only send back one instance of that block. This behavior is meant to prevent amplification attacks.

The payloads sent back from the server as a response MUST all have the same ETag (Section 5.10.6 of [RFC7252]) for the same body. The server MUST NOT use the same ETag value for different representations of a resource.

The ETag is opaque, but the server MUST ensure that it is unique for every different body of transmitted data.

Implementation Note: It is suggested that the server treats the ETag as an unsigned integer of 8 bytes in length. An implementation may want to consider limiting this to 4 bytes to reduce packet overhead size. The initial ETag value should be randomly generated and then subsequently incremented by the server whenever a new body of data is being transmitted between peers.

Section 4.6 discusses the use of Size2 Option.

The client may elect to request any detected missing blocks or just ignore the partial body. This decision is implementation specific.

For NON payloads, the client SHOULD wait NON_RECEIVE_TIMEOUT (Section 7.2) after the last received payload before requesting retransmission of any missing blocks. Retransmission is requested by issuing a GET, POST, PUT, FETCH, PATCH, or iPATCH request that contains one or more Q-Block2 Options that define the missing block(s). Generally the M bit on the Q-Block2 Option(s) SHOULD be unset, although the M bit MAY be set to request this and later blocks from this MAX_PAYLOADS_SET, see Section 10.2.4 for an example of this in operation. Further considerations related to the transmission timing for missing requests are discussed in Section 7.2.

The missing block numbers requested by the client MUST have an increasing block number in each additional Q-Block2 Option with no duplicates. The server SHOULD respond with a 4.00 (Bad Request) to requests not adhering to this behavior. Note that the ordering constraint is meant to force the client to check for duplicates and remove them. This also helps with troubleshooting.

If the client receives a duplicate block with the same ETag, it MUST silently ignore the payload.

A client SHOULD maintain a partial body (missing payloads) for NON_PARTIAL_TIMEOUT (Section 7.2) or as defined by the Max-Age Option (or its default of 60 seconds (Section 5.6.1 of [RFC7252])), whichever is the less. On release of the partial body, the client should then release all of the tokens used for this body apart from the token that is used to track a resource that is being observed.

The ETag Option should not be used in the request for missing blocks as the server could respond with a 2.03 (Valid) response with no payload. It can be used in the request if the client wants to check the freshness of the locally cached body response.

The server SHOULD maintain a cached copy of the body when using the Q-Block2 Option to facilitate retransmission of any missing payloads.

If the server detects part way through a body transfer that the resource data has changed and the server is not maintaining a cached copy of the old data, then the transmission is terminated. Any subsequent missing block requests MUST be responded to using the latest ETag and Size2 Option values with the updated data.

If the server responds during a body update with a different ETag Option value (as the resource representation has changed), then the client should treat the partial body with the old ETag as no longer being fresh. The client may then request all of the new data by specifying Q-Block2 with block number '0' and the M bit set.

If the server transmits a new body of data (e.g., a triggered Observe notification) with a new ETag to the same client as an additional response, the client should remove any partially received body held for a previous ETag for that resource as it is unlikely the missing blocks can be retrieved.

If there is insufficient space to create a response PDU with a block size of 16 bytes (SZX = 0) to send back all the response options as appropriate, a 4.13 (Request Entity Too Large) is returned without the Size1 Option.

For Confirmable traffic, the server typically acknowledges the initial request using an ACK with a piggybacked payload, and then sends the subsequent payloads of the MAX_PAYLOADS_SET as CON responses. These CON responses are individually ACKed by the client. The server will detect failure to send a packet and SHOULD terminate the body transfer, but the client can issue, after a MAX_TRANSMIT_SPAN delay, a separate GET, POST, PUT, FETCH, PATCH, or iPATCH for any missing blocks as needed.

4.5. Using Observe Option

For a request that uses Q-Block1, the Observe value [RFC7641] MUST be the same for all the payloads of the same body. This includes any missing payloads that are retransmitted.

For a response that uses Q-Block2, the Observe value MUST be the same for all the payloads of the same body. This is different from Block2 usage where the Observe value is only present in the first block (Section 3.4 of [RFC7959]). This includes payloads transmitted following receipt of the 'Continue' Q-Block2 Option (Section 4.4) by the server. If a missing payload is requested by a client, then both the request and response MUST NOT include the Observe Option.

4.6. Using Size1 and Size2 Options

Section 4 of [RFC7959] defines two CoAP options: Size1 for indicating the size of the representation transferred in requests and Size2 for indicating the size of the representation transferred in responses.

For Q-Block1 and Q-Block2 Options, the Size1 or Size2 Option values MUST exactly represent the size of the data on the body so that any missing data can easily be determined.

The Size1 Option MUST be used with the Q-Block1 Option when used in a request and MUST be present in all payloads of the request, preserving the same value. The Size2 Option MUST be used with the

Q-Block2 Option when used in a response and MUST be present in all payloads of the response, preserving the same value.

4.7. Using Q-Block1 and Q-Block2 Options Together

The behavior is similar to the one defined in Section 3.3 of [RFC7959] with Q-Block1 substituted for Block1 and Q-Block2 for Block2.

4.8. Using Q-Block2 Option With Multicast

Servers MUST ignore multicast requests that contain the Q-Block2 Option. As a reminder, Block2 Option can be used as stated in Section 2.8 of [RFC7959].

5. The Use of 4.08 (Request Entity Incomplete) Response Code

4.08 (Request Entity Incomplete) Response Code has a new Content-Type "application/missing-blocks+cbor-seq" used to indicate that the server has not received all of the blocks of the request body that it needs to proceed. Such messages must not be treated by the client as a fatal error.

Likely causes are the client has not sent all blocks, some blocks were dropped during transmission, or the client has sent them sufficiently long ago that the server has already discarded them.

The new data payload of the 4.08 (Request Entity Incomplete) response with Content-Type set to "application/missing-blocks+cbor-seq" is encoded as a CBOR Sequence [RFC8742]. It comprises one or more missing block numbers encoded as CBOR unsigned integers [RFC8949]. The missing block numbers MUST be unique in each 4.08 (Request Entity Incomplete) response when created by the server; the client MUST ignore any duplicates in the same 4.08 (Request Entity Incomplete) response.

The Content-Format Option (Section 5.10.3 of [RFC7252]) MUST be used in the 4.08 (Request Entity Incomplete) response. It MUST be set to "application/missing-blocks+cbor-seq" (Section 12.3).

The Concise Data Definition Language [RFC8610] (and see Section 4.1 [RFC8742]) for the data describing these missing blocks is as follows:

```
; This defines an array, the elements of which are to be used
; in a CBOR Sequence:
payload = [+ missing-block-number]
; A unique block number not received:
missing-block-number = uint
```

Figure 1: Structure of the Missing Blocks Payload

This CDDL syntax MUST be followed.

It is desirable that the token to use for the response is the token that was used in the last block number received so far with the same Request-Tag value. Note that the use of any received token with the same Request-Tag would be acceptable, but providing the one used in the last received payload will aid any troubleshooting. The client will use the token to determine what was the previously sent request to obtain the Request-Tag value that was used.

If the size of the 4.08 (Request Entity Incomplete) response packet is larger than that defined by Section 4.6 [RFC7252], then the number of reported missing blocks MUST be limited so that the response can fit into a single packet. If this is the case, then the server can send subsequent 4.08 (Request Entity Incomplete) responses containing the missing other blocks on receipt of a new request providing a missing payload with the same Request-Tag.

The missing blocks MUST be reported in ascending order without any duplicates. The client SHOULD silently drop 4.08 (Request Entity Incomplete) responses not adhering with this behavior.

Implementation Note: Consider limiting the number of missing payloads to MAX_PAYLOADS to minimize congestion control being needed. The CBOR sequence does not include any array wrapper.

The 4.08 (Request Entity Incomplete) with Content-Type "application/missing-blocks+cbor-seq" SHOULD NOT be used when using Confirmable requests or a reliable connection [RFC8323] as the client will be able to determine that there is a transmission failure of a particular payload and hence that the server is missing that payload.

6. The Use of Tokens

Each new request generally uses a new Token (and sometimes must, see Section 4 of [I-D.ietf-core-echo-request-tag]). Additional responses to a request all use the token of the request they respond to.

Implementation Note: By using 8-byte tokens, it is possible to easily minimize the number of tokens that have to be tracked by

clients, by keeping the bottom 32 bits the same for the same body and the upper 32 bits containing the current body's request number (incrementing every request, including every re-transmit). This allows the client to be alleviated from keeping all the per-request-state, e.g., in Section 3 of [RFC8974]. However, if using NoSec, Section 5.2 of [RFC8974] needs to be considered for security implications.

7. Congestion Control for Unreliable Transports

The transmission of all the blocks of a single body over an unreliable transport MUST either all be Confirmable or all be Non-confirmable. This is meant to simplify the congestion control procedure.

As a reminder, there is no need for CoAP-specific congestion control for reliable transports [RFC8323].

7.1. Confirmable (CON)

Congestion control for CON requests and responses is specified in Section 4.7 of [RFC7252]. In order to benefit from faster transmission rates, NSTART will need to be increased from 1. However, the other CON congestion control parameters will need to be tuned to cover this change. This tuning is not specified in this document, given that the applicability scope of the current specification assumes that all requests and responses using Q-Block1 and Q-Block2 will be Non-confirmable (Section 3.2) apart from the initial Q-Block functionality negotiation.

Following the failure to transmit a packet due to packet loss after MAX_TRANSMIT_SPAN time (Section 4.8.2 of [RFC7252]), it is implementation specific as to whether there should be any further requests for missing data.

7.2. Non-confirmable (NON)

This document introduces new parameters MAX_PAYLOADS, NON_TIMEOUT, NON_TIMEOUT_RANDOM, NON_RECEIVE_TIMEOUT, NON_MAX_RETRANSMIT, NON_PROBING_WAIT, and NON_PARTIAL_TIMEOUT primarily for use with NON (Table 3).

Note: Randomness may naturally be provided based on the traffic profile, how PROBING_RATE is computed (as this is an average), and when the peer responds. Randomness is explicitly added for some of the congestion control parameters to handle situations where every thing is in sync when retrying.

MAX_PAYLOADS should be configurable with a default value of 10. Both CoAP endpoints MUST have the same value (otherwise there will be transmission delays in one direction) and the value MAY be negotiated between the endpoints to a common value by using a higher level protocol (out of scope of this document). This is the maximum number of payloads that can be transmitted at any one time.

Note: The default value of 10 is chosen for reasons similar to those discussed in Section 5 of [RFC6928], especially given the target application discussed in Section 3.2.

NON_TIMEOUT is used to compute the delay between sending MAX_PAYLOADS_SET for the same body. By default, NON_TIMEOUT has the same value as ACK_TIMEOUT (Section 4.8 of [RFC7252]).

NON_TIMEOUT_RANDOM is the initial actual delay between sending the first two MAX_PAYLOADS_SETs of the same body. The same delay is then used between the subsequent MAX_PAYLOADS_SETs. It is a random duration (not an integral number of seconds) between NON_TIMEOUT and (NON_TIMEOUT * ACK_RANDOM_FACTOR). ACK_RANDOM_FACTOR is set to 1.5 as discussed in Section 4.8 of [RFC7252].

NON_RECEIVE_TIMEOUT is the initial time to wait for a missing payload before requesting retransmission for the first time. Every time the missing payload is re-requested, the time to wait value doubles. The time to wait is calculated as:

$$\text{Time-to-Wait} = \text{NON_RECEIVE_TIMEOUT} * (2 ** (\text{Re-Request-Count} - 1))$$

NON_RECEIVE_TIMEOUT has a default value of twice NON_TIMEOUT. NON_RECEIVE_TIMEOUT MUST always be greater than NON_TIMEOUT_RANDOM by at least one second so that the sender of the payloads has the opportunity to start sending the next MAX_PAYLOADS_SET before the receiver times out.

NON_MAX_RETRANSMIT is the maximum number of times a request for the retransmission of missing payloads can occur without a response from the remote peer. After this occurs, the local endpoint SHOULD consider the body stale, remove any body, and release Tokens and Request-Tag on the client (or the ETag on the server). By default, NON_MAX_RETRANSMIT has the same value as MAX_RETRANSMIT (Section 4.8 of [RFC7252]).

NON_PROBING_WAIT is used to limit the potential wait needed when using PROBING_RATE. By default, NON_PROBING_WAIT is computed in a similar way as EXCHANGE_LIFETIME (Section 4.8.2 of [RFC7252]) but with ACK_TIMEOUT, MAX_RETRANSMIT, and PROCESSING_DELAY substituted

with NON_TIMEOUT, NON_MAX_RETRANSMIT, and NON_TIMEOUT_RANDOM, respectively:

$$\text{NON_PROBING_WAIT} = \text{NON_TIMEOUT} * ((2 ** \text{NON_MAX_RETRANSMIT}) - 1) * \text{ACK_RANDOM_FACTOR} + (2 * \text{MAX_LATENCY}) + \text{NON_TIMEOUT_RANDOM}$$

NON_PARTIAL_TIMEOUT is used for expiring partially received bodies. By default, NON_PARTIAL_TIMEOUT is computed in the same way as EXCHANGE_LIFETIME (Section 4.8.2 of [RFC7252]) but with ACK_TIMEOUT and MAX_RETRANSMIT substituted with NON_TIMEOUT and NON_MAX_RETRANSMIT, respectively:

$$\text{NON_PARTIAL_TIMEOUT} = \text{NON_TIMEOUT} * ((2 ** \text{NON_MAX_RETRANSMIT}) - 1) * \text{ACK_RANDOM_FACTOR} + (2 * \text{MAX_LATENCY}) + \text{NON_TIMEOUT}$$

Parameter Name	Default Value
MAX_PAYLOADS	10
NON_MAX_RETRANSMIT	4
NON_TIMEOUT	2 s
NON_TIMEOUT_RANDOM	between 2-3 s
NON_RECEIVE_TIMEOUT	4 s
NON_PROBING_WAIT	between 247-248 s
NON_PARTIAL_TIMEOUT	247 s

Table 3: Congestion Control Parameters

The PROBING_RATE parameter in CoAP indicates the average data rate that must not be exceeded by a CoAP endpoint in sending to a peer endpoint that does not respond. A single body will be subjected to PROBING_RATE (Section 4.7 of [RFC7252]), not the individual packets. If the wait time between sending bodies that are not being responded to based on PROBING_RATE exceeds NON_PROBING_WAIT, then the wait time is limited to NON_PROBING_WAIT.

Note: For the particular DOTS application, PROBING_RATE and other transmission parameters are negotiated between peers. Even when not negotiated, the DOTS application uses customized defaults as discussed in Section 4.5.2 of [RFC8782]. Note that MAX_PAYLOADS, NON_MAX_RETRANSMIT, NON_TIMEOUT, NON_PROBING_WAIT, and NON_PARTIAL_TIMEOUT can be negotiated between DOTS peers, e.g., as per [I-D.bosh-dots-quick-blocks]. When explicit values are configured for NON_PROBING_WAIT and NON_PARTIAL_TIMEOUT, these values are used without applying any jitter.

Each NON 4.08 (Request Entity Incomplete) response is subject to PROBING_RATE.

Each NON GET or FETCH request using a Q-Block2 Option is subject to PROBING_RATE.

As the sending of many payloads of a single body may itself cause congestion, after transmission of every MAX_PAYLOADS_SET of a single body, a delay MUST be introduced of NON_TIMEOUT_RANDOM before sending the next MAX_PAYLOADS_SET unless a 'Continue' is received from the peer for the current MAX_PAYLOADS_SET, in which case the next MAX_PAYLOADS_SET MAY start transmission immediately.

Note: Assuming 1500-byte packets and the MAX_PAYLOADS_SET having 10 payloads, this corresponds to $1500 * 10 * 8 = 120$ Kbits. With a delay of 2 seconds between MAX_PAYLOADS_SET, this indicates an average speed requirement of 60 Kbps for a single body should there be no responses. This transmission rate is further reduced by being subject to PROBING_RATE.

The sending of a set of missing blocks of a body is restricted to those in a MAX_PAYLOADS_SET at a time. In other words, a NON_TIMEOUT_RANDOM delay is still observed between each MAX_PAYLOAD_SET.

For Q-Block1 Option, if the server responds with a 2.31 (Continue) Response Code for the latest payload sent, then the client can continue to send the next MAX_PAYLOADS_SET without any further delay. If the server responds with a 4.08 (Request Entity Incomplete) Response Code, then the missing payloads SHOULD be retransmitted before going into another NON_TIMEOUT_RANDOM delay prior to sending the next set of payloads.

For the server receiving NON Q-Block1 requests, it SHOULD send back a 2.31 (Continue) Response Code on receipt of all of the MAX_PAYLOADS_SET to prevent the client unnecessarily delaying. If not all of the MAX_PAYLOADS_SET were received, the server SHOULD delay for NON_RECEIVE_TIMEOUT (exponentially scaled based on the repeat request count for a payload) before sending the 4.08 (Request Entity Incomplete) Response Code for the missing payload(s). If all of the MAX_PAYLOADS_SET were received and a 2.31 (Continue) had been sent, but no more payloads were received for NON_RECEIVE_TIMEOUT (exponentially scaled), the server SHOULD send a 4.08 (Request Entity Incomplete) response detailing the missing payloads after the block number that was indicated in the sent 2.31 (Continue). If the repeated response count of the 4.08 (Request Entity Incomplete) exceeds NON_MAX_RETRANSMIT, the server SHOULD discard the partial body and stop requesting the missing payloads.

It is likely that the client will start transmitting the next MAX_PAYLOADS_SET before the server times out on waiting for the last of the previous MAX_PAYLOADS_SET. On receipt of a payload from the next MAX_PAYLOADS_SET, the server SHOULD send a 4.08 (Request Entity Incomplete) Response Code indicating any missing payloads from any previous MAX_PAYLOADS_SET. Upon receipt of the 4.08 (Request Entity Incomplete) Response Code, the client SHOULD send the missing payloads before continuing to send the remainder of the MAX_PAYLOADS_SET and then go into another NON_TIMEOUT_RANDOM delay prior to sending the next MAX_PAYLOADS_SET.

For the client receiving NON Q-Block2 responses, it SHOULD send a 'Continue' Q-Block2 request (Section 4.4) for the next MAX_PAYLOADS_SET on receipt of all of the MAX_PAYLOADS_SET, to prevent the server unnecessarily delaying. Otherwise the client SHOULD delay for NON_RECEIVE_TIMEOUT (exponentially scaled based on the repeat request count for a payload), before sending the request for the missing payload(s). If the repeat request count for a missing payload exceeds NON_MAX_RETRANSMIT, the client SHOULD discard the partial body and stop requesting the missing payloads.

The server SHOULD recognize the 'Continue' Q-Block2 request as a continue request and just continue the transmission of the body (including Observe Option, if appropriate for an unsolicited response) rather than as a request for the remaining missing blocks.

It is likely that the server will start transmitting the next MAX_PAYLOADS_SET before the client times out on waiting for the last of the previous MAX_PAYLOADS_SET. Upon receipt of a payload from the new MAX_PAYLOADS_SET, the client SHOULD send a request indicating any missing payloads from any previous MAX_PAYLOADS_SET. Upon receipt of such request, the server SHOULD send the missing payloads before continuing to send the remainder of the MAX_PAYLOADS_SET and then go into another NON_TIMEOUT_RANDOM delay prior to sending the next MAX_PAYLOADS_SET.

The client does not need to acknowledge the receipt of the entire body.

Note: If there is asymmetric traffic loss causing responses to never get received, a delay of NON_TIMEOUT_RANDOM after every transmission of MAX_PAYLOADS_SET will be observed. The endpoint receiving the body is still likely to receive the entire body.

8. Caching Considerations

Caching block based information is not straight forward in a proxy. For Q-Block1 and Q-Block2 Options, for simplicity it is expected that the proxy will reassemble the body (using any appropriate recovery options for packet loss) before passing on the body to the appropriate CoAP endpoint. This does not preclude an implementation doing a more complex per payload caching, but how to do this is out of the scope of this document. The onward transmission of the body does not require the use of the Q-Block1 or Q-Block2 Options as these options may not be supported in that link. This means that the proxy must fully support the Q-Block1 and Q-Block2 Options.

How the body is cached in the CoAP client (for Q-Block1 transmissions) or the CoAP server (for Q-Block2 transmissions) is implementation specific.

As the entire body is being cached in the proxy, the Q-Block1 and Q-Block2 Options are removed as part of the block assembly and thus do not reach the cache.

For Q-Block2 responses, the ETag Option value is associated with the data (and onward transmitted to the CoAP client), but is not part of the cache key.

For requests with Q-Block1 Option, the Request-Tag Option is associated with the build up of the body from successive payloads, but is not part of the cache key. For the onward transmission of the body using CoAP, a new Request-Tag SHOULD be generated and used. Ideally this new Request-Tag should replace the client's request Request-Tag.

It is possible that two or more CoAP clients are concurrently updating the same resource through a common proxy to the same CoAP server using Q-Block1 (or Block1) Option. If this is the case, the first client to complete building the body causes that body to start transmitting to the CoAP server with an appropriate Request-Tag value. When the next client completes building the body, any existing partial body transmission to the CoAP server is terminated and the new body representation transmission starts with a new Request-Tag value. Note that it cannot be assumed that the proxy will always receive a complete body from a client.

A proxy that supports Q-Block2 Option MUST be prepared to receive a GET or similar request indicating one or more missing blocks. The proxy will serve from its cache the missing blocks that are available in its cache in the same way a server would send all the appropriate Q-Block2 responses. If a body matching the cache key is not

available in the cache, the proxy MUST request the entire body from the CoAP server using the information in the cache key.

How long a CoAP endpoint (or proxy) keeps the body in its cache is implementation specific (e.g., it may be based on Max-Age).

9. HTTP-Mapping Considerations

As a reminder, the basic normative requirements on HTTP/CoAP mappings are defined in Section 10 of [RFC7252]. The implementation guidelines for HTTP/CoAP mappings are elaborated in [RFC8075].

The rules defined in Section 5 of [RFC7959] are to be followed.

10. Examples with Non-confirmable Messages

This section provides some sample flows to illustrate the use of Q-Block1 and Q-Block2 Options with NON. Examples with CON are provided in Appendix A.

The examples in the following subsections assume MAX_PAYLOADS is set to 10 and NON_MAX_RETRANSMIT is set to 4.

Figure 2 lists the conventions that are used in the following subsections.

T: Token value
O: Observe Option value
M: Message ID
RT: Request-Tag
ET: ETag
QB1: Q-Block1 Option values NUM/More/Size
QB2: Q-Block2 Option values NUM/More/Size
Size: Actual block size encoded in SZX
 \: Trimming long lines
[[]]: Comments
-->X: Message loss (request)
X<--: Message loss (response)
...: Passage of time
Payload N: Corresponds to the CoAP message that conveys
 a block number (N-1) of a given block-wise exchange.

Figure 2: Notations Used in the Figures

10.1. Q-Block1 Option

10.1.1. A Simple Example

Figure 3 depicts an example of a NON PUT request conveying Q-Block1 Option. All the blocks are received by the server.

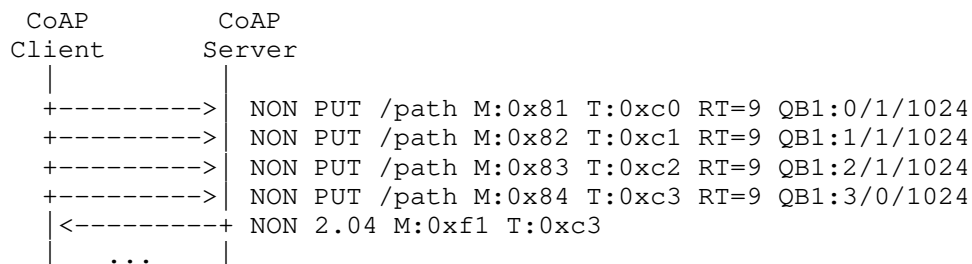


Figure 3: Example of NON Request with Q-Block1 Option (Without Loss)

10.1.2. Handling MAX_PAYLOADS Limits

Figure 4 depicts an example of a NON PUT request conveying Q-Block1 Option. The number of payloads exceeds MAX_PAYLOADS. All the blocks are received by the server.

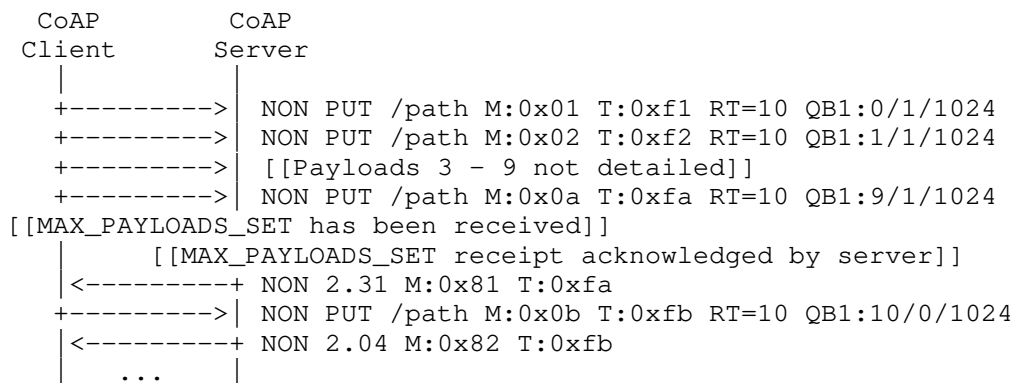


Figure 4: Example of MAX_PAYLOADS NON Request with Q-Block1 Option (Without Loss)

10.1.3. Handling MAX_PAYLOADS with Recovery

Consider now a scenario where a new body of data is to be sent by the client, but some blocks are dropped in transmission as illustrated in Figure 5.

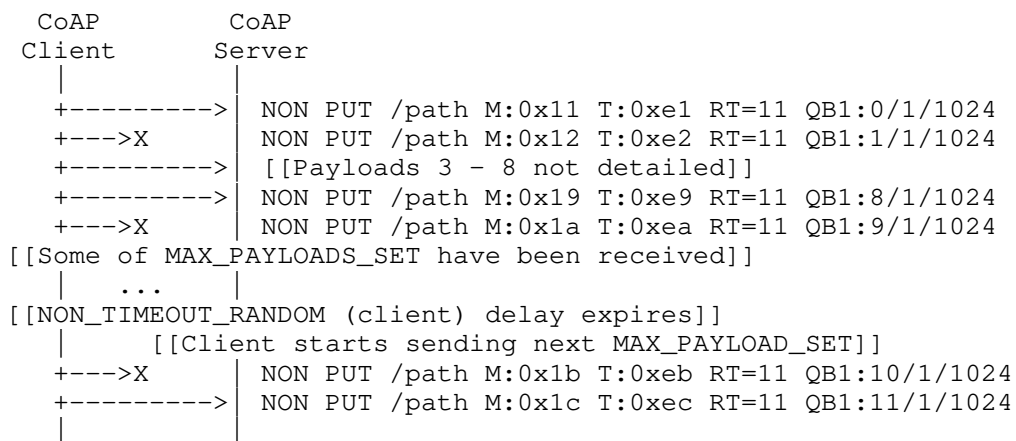


Figure 5: Example of MAX_PAYLOADS NON Request with Q-Block1 Option
(With Loss)

On seeing a payload from the next MAX_PAYLOAD_SET, the server realizes that some blocks are missing from the previous MAX_PAYLOADS_SET and asks for the missing blocks in one go (Figure 6). It does so by indicating which blocks from the previous MAX_PAYLOADS_SET have not been received in the data portion of the response (Section 5). The token used in the response should be the token that was used in the last received payload. The client can then derive the Request-Tag by matching the token with the sent request.

```

CoAP      CoAP
Client    Server
|         |
|<-----+ NON 4.08 M:0x91 T:0xec [Missing 1,9]
|         | [[Client responds with missing payloads]]
|+----->| NON PUT /path M:0x1d T:0xed RT=11 QBl:1/1/1024
|+----->| NON PUT /path M:0x1e T:0xee RT=11 QBl:9/1/1024
|         | [[Client continues sending next MAX_PAYLOAD_SET]]
|+----->| NON PUT /path M:0x1f T:0xef RT=11 QBl:12/0/1024
|         | ...
|         | [[NON_RECEIVE_TIMEOUT (server) delay expires]]
|         | [[The server realizes a block is still missing and asks
|         | for the missing one]]
|<-----+ NON 4.08 M:0x92 T:0xef [Missing 10]
|+----->| NON PUT /path M:0x20 T:0xf0 RT=11 QBl:10/1/1024
|<-----+ NON 2.04 M:0x93 T:0xf0
|         | ...

```

Figure 6: Example of NON Request with Q-Block1 Option (Blocks Recovery)

10.1.4. Handling Recovery with Failure

Figure 7 depicts an example of a NON PUT request conveying Q-Block1 Option where recovery takes place, but eventually fails.

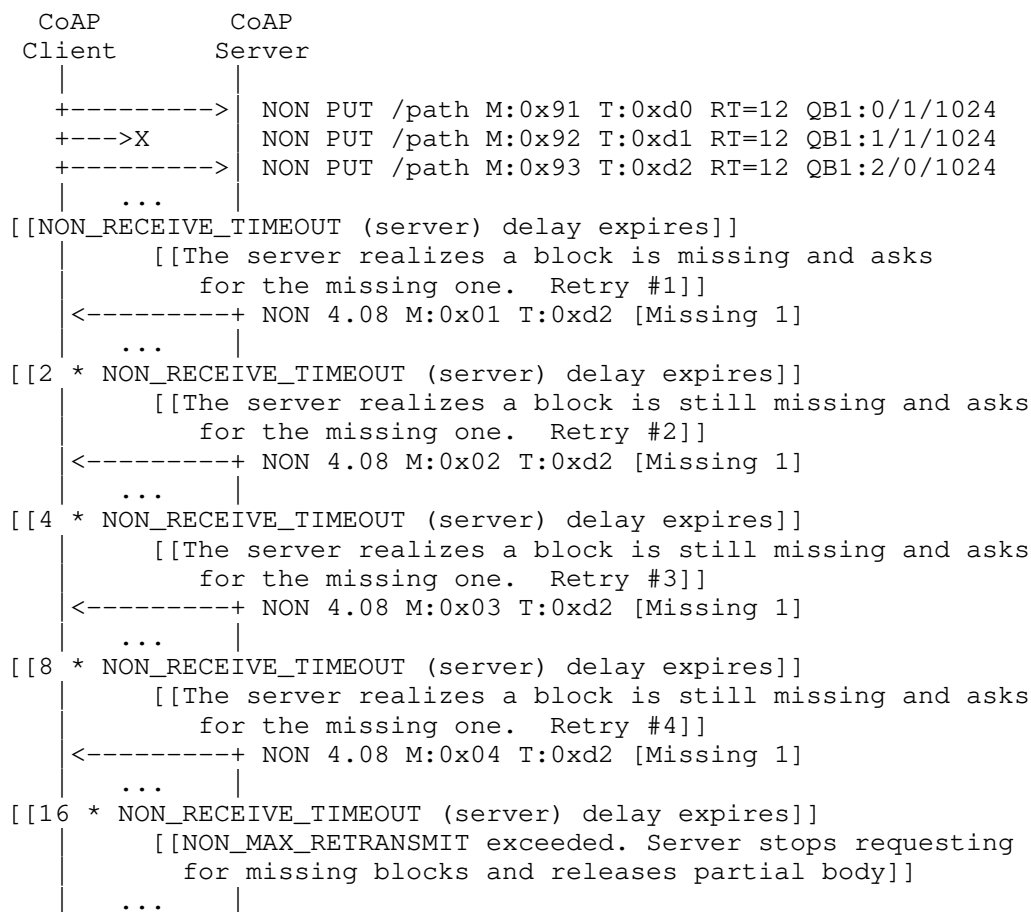


Figure 7: Example of NON Request with Q-Block1 Option (With Eventual Failure)

10.2. Q-Block2 Option

These examples include the Observe Option to demonstrate how that option is used. Note that the Observe Option is not required for Q-Block2; the observe detail can thus be ignored.

10.2.1. A Simple Example

Figure 8 illustrates the example of Q-Block2 Option. The client sends a NON GET carrying Observe and Q-Block2 Options. The Q-Block2 Option indicates a block size hint (1024 bytes). This request is replied to by the server using four (4) blocks that are transmitted to the client without any loss. Each of these blocks carries a

Q-Block2 Option. The same process is repeated when an Observe is triggered, but no loss is experienced by any of the notification blocks.

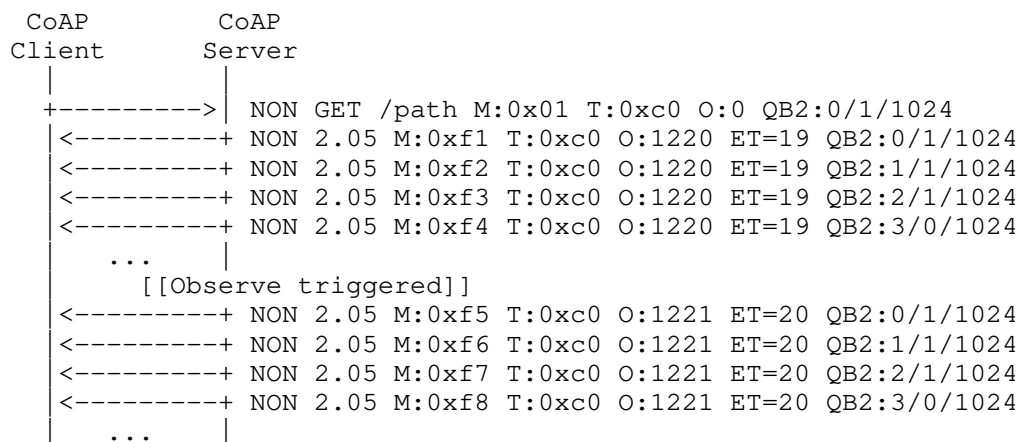


Figure 8: Example of NON Notifications with Q-Block2 Option (Without Loss)

10.2.2. Handling MAX_PAYLOADS Limits

Figure 9 illustrates the same as Figure 8 but this time has eleven (11) payloads which exceeds MAX_PAYLOADS. There is no loss experienced.

```

CoAP      CoAP
Client    Server
|         |
+----->| NON GET /path M:0x01 T:0xf0 O:0 QB2:0/1/1024
|         |
<-----+ NON 2.05 M:0x81 T:0xf0 O:1234 ET=21 QB2:0/1/1024
|         |
<-----+ NON 2.05 M:0x82 T:0xf0 O:1234 ET=21 QB2:1/1/1024
|         |
<-----+ [[Payloads 3 - 9 not detailed]]
|         |
<-----+ NON 2.05 M:0x8a T:0xf0 O:1234 ET=21 QB2:9/1/1024
|         |
[[MAX_PAYLOADS_SET has been received]]
|         |
|         | [[MAX_PAYLOADS_SET acknowledged by client using
|         | 'Continue' Q-Block2]]
+----->| NON GET /path M:0x02 T:0xf1 QB2:10/1/1024
|         |
<-----+ NON 2.05 M:0x8b T:0xf0 O:1234 ET=21 QB2:10/0/1024
|         |
|         | ...
|         | [[Observe triggered]]
|         |
<-----+ NON 2.05 M:0x91 T:0xf0 O:1235 ET=22 QB2:0/1/1024
|         |
<-----+ NON 2.05 M:0x92 T:0xf0 O:1235 ET=22 QB2:1/1/1024
|         |
<-----+ [[Payloads 3 - 9 not detailed]]
|         |
<-----+ NON 2.05 M:0x9a T:0xf0 O:1235 ET=22 QB2:9/1/1024
|         |
[[MAX_PAYLOADS_SET has been received]]
|         |
|         | [[MAX_PAYLOADS_SET acknowledged by client using
|         | 'Continue' Q-Block2]]
+----->| NON GET /path M:0x03 T:0xf2 QB2:10/1/1024
|         |
<-----+ NON 2.05 M:0x9b T:0xf0 O:1235 ET=22 QB2:10/0/1024
|         |
[[Body has been received]]
|         |
|         | ...

```

Figure 9: Example of NON Notifications with Q-Block2 Option (Without Loss)

10.2.3. Handling MAX_PAYLOADS with Recovery

Figure 10 shows the example of an Observe that is triggered but for which some notification blocks are lost. The client detects the missing blocks and requests their retransmission. It does so by indicating the blocks that are missing as one or more Q-Block2 Options.

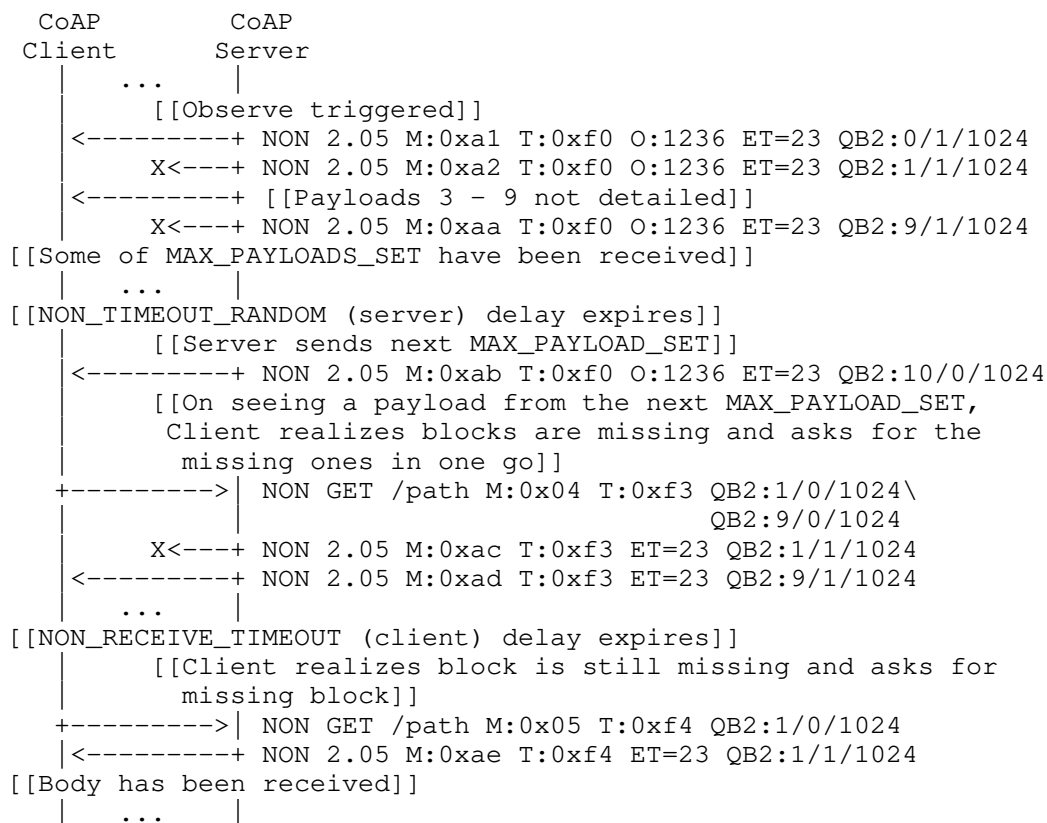


Figure 10: Example of NON Notifications with Q-Block2 Option (Blocks Recovery)

10.2.4. Handling Recovery using M-bit Set

Figure 11 shows the example of an Observe that is triggered but only the first two notification blocks reach the client. In order to retrieve the missing blocks, the client sends a request with a single Q-Block2 Option with the M bit set.

```

CoAP      CoAP
Client    Server
|         |
|         | ...
|         | [[Observe triggered]]
|         | <-----+ NON 2.05 M:0xb1 T:0xf0 O:1237 ET=24 QB2:0/1/1024
|         | <-----+ NON 2.05 M:0xb2 T:0xf0 O:1237 ET=24 QB2:1/1/1024
|         | X<----+ NON 2.05 M:0xb3 T:0xf0 O:1237 ET=24 QB2:2/1/1024
|         | X<----+ [[Payloads 4 - 9 not detailed]]
|         | X<----+ NON 2.05 M:0xb9 T:0xf0 O:1237 ET=24 QB2:9/1/1024
|         | [[Some of MAX_PAYLOADS_SET have been received]]
|         | ...
|         | [[NON_TIMEOUT_RANDOM (server) delay expires]]
|         | [[Server sends next MAX_PAYLOAD_SET]]
|         | X<----+ NON 2.05 M:0xba T:0xf0 O:1237 ET=24 QB2:10/0/1024
|         | ...
|         | [[NON_RECEIVE_TIMEOUT (client) delay expires]]
|         | [[Client realizes blocks are missing and asks for the
|         |      missing ones in one go by setting the M bit]]
|         | +----->| NON GET /path M:0x06 T:0xf5 QB2:2/1/1024
|         | <-----+ NON 2.05 M:0xbb T:0xf5 ET=24 QB2:2/1/1024
|         | <-----+ [[Payloads 3 - 9 not detailed]]
|         | <-----+ NON 2.05 M:0xc2 T:0xf5 ET=24 QB2:9/1/1024
|         | [[MAX_PAYLOADS_SET has been received]]
|         | [[MAX_PAYLOADS_SET acknowledged by client using 'Continue'
|         |      Q-Block2]]
|         | +----->| NON GET /path M:0x87 T:0xf6 QB2:10/1/1024
|         | <-----+ NON 2.05 M:0xc3 T:0xf0 O:1237 ET=24 QB2:10/0/1024
|         | [[Body has been received]]
|         | ...

```

Figure 11: Example of NON Notifications with Q-Block2 Option (Blocks Recovery with M bit Set)

10.3. Q-Block1 and Q-Block2 Options

10.3.1. A Simple Example

Figure 12 illustrates the example of a FETCH using both Q-Block1 and Q-Block2 Options along with an Observe Option. No loss is experienced.

```

CoAP      CoAP
Client    Server
|         |
+-----> | NON FETCH /path M:0x10 T:0x90 O:0 RT=30 QB1:0/1/1024
+-----> | NON FETCH /path M:0x11 T:0x91 O:0 RT=30 QB1:1/1/1024
+-----> | NON FETCH /path M:0x12 T:0x93 O:0 RT=30 QB1:2/0/1024
<-----+ | NON 2.05 M:0x60 T:0x93 O:1320 ET=90 QB2:0/1/1024
<-----+ | NON 2.05 M:0x61 T:0x93 O:1320 ET=90 QB2:1/1/1024
<-----+ | NON 2.05 M:0x62 T:0x93 O:1320 ET=90 QB2:2/1/1024
<-----+ | NON 2.05 M:0x63 T:0x93 O:1320 ET=90 QB2:3/0/1024
    ...   |
    [[Observe triggered]]
<-----+ | NON 2.05 M:0x64 T:0x93 O:1321 ET=91 QB2:0/1/1024
<-----+ | NON 2.05 M:0x65 T:0x93 O:1321 ET=91 QB2:1/1/1024
<-----+ | NON 2.05 M:0x66 T:0x93 O:1321 ET=91 QB2:2/1/1024
<-----+ | NON 2.05 M:0x67 T:0x93 O:1321 ET=91 QB2:3/0/1024
    ...   |

```

Figure 12: Example of NON FETCH with Q-Block1 and Q-Block2 Options
(Without Loss)

10.3.2. Handling MAX_PAYLOADS Limits

Figure 13 illustrates the same as Figure 12 but this time has eleven (11) payloads in both directions which exceeds MAX_PAYLOADS. There is no loss experienced.

```

CoAP      CoAP
Client    Server
|         |
+----->| NON FETCH /path M:0x30 T:0xa0 O:0 RT=10 QB1:0/1/1024
+----->| NON FETCH /path M:0x31 T:0xa1 O:0 RT=10 QB1:1/1/1024
+----->| [[Payloads 3 - 9 not detailed]]
+----->| NON FETCH /path M:0x39 T:0xa9 O:0 RT=10 QB1:9/1/1024
[[MAX_PAYLOADS_SET has been received]]
|         |
|         | [[MAX_PAYLOADS_SET acknowledged by server]]
<-----+| NON 2.31 M:0x80 T:0xa9
+----->| NON FETCH /path M:0x3a T:0xaa O:0 RT=10 QB1:10/0/1024
<-----+| NON 2.05 M:0x81 T:0xaa O:1334 ET=21 QB2:0/1/1024
<-----+| NON 2.05 M:0x82 T:0xaa O:1334 ET=21 QB2:1/1/1024
<-----+| [[Payloads 3 - 9 not detailed]]
<-----+| NON 2.05 M:0x8a T:0xaa O:1334 ET=21 QB2:9/1/1024
[[MAX_PAYLOADS_SET has been received]]
|         |
|         | [[MAX_PAYLOADS_SET acknowledged by client using
|         | 'Continue' Q-Block2]]
+----->| NON FETCH /path M:0x3b T:0xab QB2:10/1/1024
<-----+| NON 2.05 M:0x8b T:0xaa O:1334 ET=21 QB2:10/0/1024
|         |
|         | ...
|         | [[Observe triggered]]
<-----+| NON 2.05 M:0x8c T:0xaa O:1335 ET=22 QB2:0/1/1024
<-----+| NON 2.05 M:0x8d T:0xaa O:1335 ET=22 QB2:1/1/1024
<-----+| [[Payloads 3 - 9 not detailed]]
<-----+| NON 2.05 M:0x95 T:0xaa O:1335 ET=22 QB2:9/1/1024
[[MAX_PAYLOADS_SET has been received]]
|         |
|         | [[MAX_PAYLOADS_SET acknowledged by client using
|         | 'Continue' Q-Block2]]
+----->| NON FETCH /path M:0x3c T:0xac QB2:10/1/1024
<-----+| NON 2.05 M:0x96 T:0xaa O:1335 ET=22 QB2:10/0/1024
[[Body has been received]]
|         |
|         | ...

```

Figure 13: Example of NON FETCH with Q-Block1 and Q-Block2 Options
(Without Loss)

Note that as 'Continue' was used, the server continues to use the same token (0xaa) since the 'Continue' is not being used as a request for a new set of packets, but rather is being used to instruct the server to continue its transmission (Section 7.2).

10.3.3. Handling Recovery

Consider now a scenario where some blocks are lost in transmission as illustrated in Figure 14.

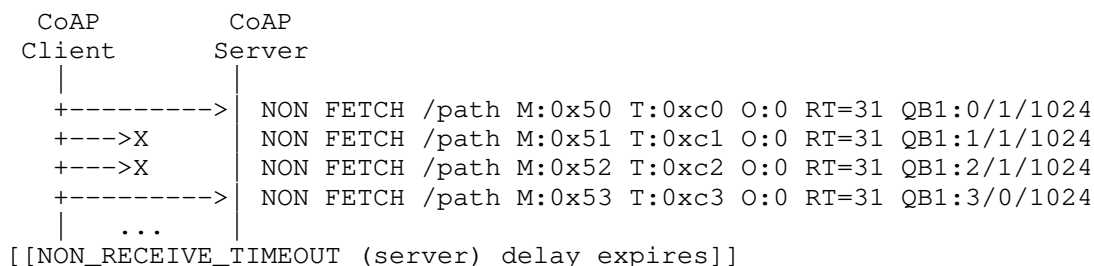


Figure 14: Example of NON FETCH with Q-Block1 and Q-Block2 Options
(With Loss)

The server realizes that some blocks are missing and asks for the missing blocks in one go (Figure 15). It does so by indicating which blocks have not been received in the data portion of the response. The token used in the response is the token that was used in the last received payload. The client can then derive the Request-Tag by matching the token with the sent request.

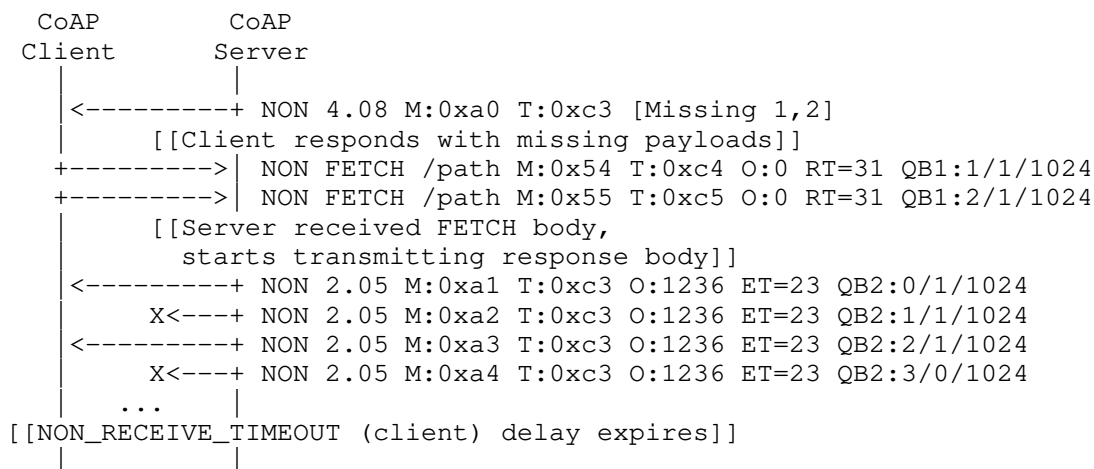


Figure 15: Example of NON Request with Q-Block1 Option (Server Recovery)

The client realizes that not all the payloads of the response have been returned. The client then asks for the missing blocks in one go (Figure 16). Note that, following Section 2.7 of [RFC7959], the FETCH request does not include the Q-Block1 or any payload.

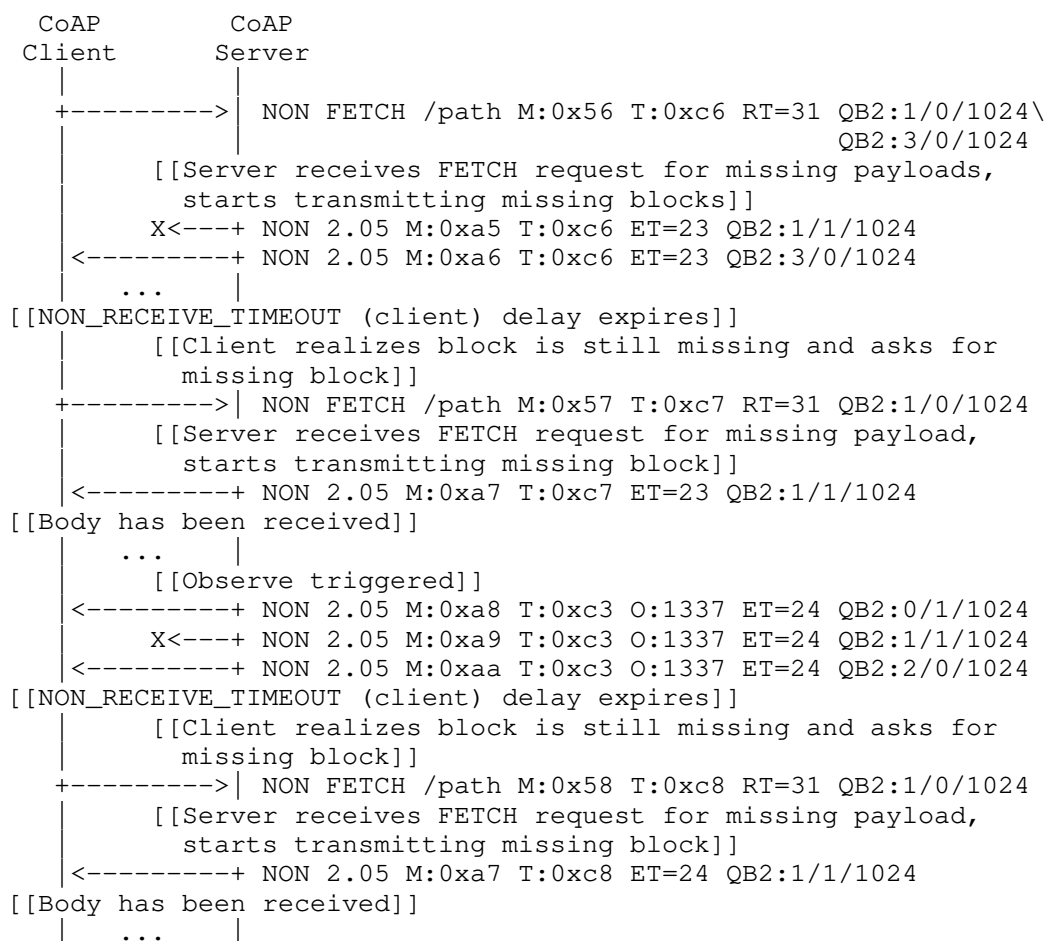


Figure 16: Example of NON Request with Q-Block1 Option (Client Recovery)

11. Security Considerations

Security considerations discussed in Section 7 of [RFC7959] should be taken into account.

Security considerations discussed in Sections 11.3 and 11.4 of [RFC7252] should be taken into account.

OSCORE provides end-to-end protection of all information that is not required for proxy operations and requires that a security context is set up (Section 3.1 of [RFC8613]). It can be trusted that the source endpoint is legitimate even if NoSec security mode is used. However,

an intermediary node can modify the unprotected outer Q-Block1 and/or Q-Block2 Options to cause a Q-Block transfer to fail or keep requesting all the blocks by setting the M bit and, thus, causing attack amplification. As discussed in Section 12.1 of [RFC8613], applications need to consider that certain message fields and messages types are not protected end-to-end and may be spoofed or manipulated. Therefore, it is NOT RECOMMENDED to use the NoSec security mode if either the Q-Block1 or Q-Block2 Options is used.

If OSCORE is not used, it is also NOT RECOMMENDED to use the NoSec security mode if either the Q-Block1 or Q-Block2 Options is used.

If NoSec is being used, Section D.5 of [RFC8613] discusses the security analysis and considerations for unprotected message fields even if OSCORE is not being used.

Security considerations related to the use of Request-Tag are discussed in Section 5 of [I-D.ietf-core-echo-request-tag].

12. IANA Considerations

RFC Editor Note: Please replace [RFCXXXX] with the RFC number to be assigned to this document.

12.1. CoAP Option Numbers Registry

IANA is requested to add the following entries to the "CoAP Option Numbers" sub-registry [Options] defined in [RFC7252] within the "Constrained RESTful Environments (CoRE) Parameters" registry:

Number	Name	Reference
TBA1	Q-Block1	[RFCXXXX]
TBA2	Q-Block2	[RFCXXXX]

Table 4: CoAP Q-Block1 and Q-Block2 Option Numbers

This document suggests 19 (TBA1) and 31 (TBA2) as values to be assigned for the new option numbers.

12.2. Media Type Registration

This document requests IANA to register the "application/missing-blocks+cbor-seq" media type in the "Media Types" registry [IANA-MediaTypes]. This registration follows the procedures specified in [RFC6838]:

Type name: application

Subtype name: missing-blocks+cbor-seq

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: Must be encoded as a CBOR sequence [RFC8742], as defined in Section 4 of [RFCXXXX].

Security considerations: See Section 10 of [RFCXXXX].

Interoperability considerations: N/A

Published specification: [RFCXXXX]

Applications that use this media type: Data serialization and deserialization. In particular, the type is used by applications relying upon block-wise transfers, allowing a server to specify non-received blocks and request for their retransmission, as defined in Section 4 of [RFCXXXX].

Fragment identifier considerations: N/A

Additional information: N/A

Person & email address to contact for further information: IETF, iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: none

Author: See Authors' Addresses section.

Change controller: IESG

Provisional registration? No

12.3. CoAP Content-Formats Registry

This document requests IANA to register the following CoAP Content-Format for the "application/missing-blocks+cbor-seq" media type in the "CoAP Content-Formats" registry [Format], defined in [RFC7252], within the "Constrained RESTful Environments (CoRE) Parameters" registry:

- o Media Type: application/missing-blocks+cbor-seq
- o Encoding: -
- o Id: TBA3
- o Reference: [RFCXXXX]

This document suggests 272 (TBA3) as a value to be assigned for the new content format number.

13. References

13.1. Normative References

- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J. P., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-12 (work in progress), February 2021.
- [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>>.
- [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>>.
- [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>>.

- [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>>.
- [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>>.
- [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>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <<https://www.rfc-editor.org/info/rfc8742>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

13.2. Informative References

- [Format] "CoAP Content-Formats", <<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>>.
- [I-D.bosh-dots-quick-blocks]
Boucadair, M. and J. Shallow, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Configuration Attributes for Faster Block Transmission", draft-bosh-dots-quick-blocks-01 (work in progress), January 2021.

- [I-D.ietf-dots-telemetry]
Boucadair, M., Reddy, T., Doron, E., Chen, M., and J. Shallow, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Telemetry", draft-ietf-dots-telemetry-15 (work in progress), December 2020.
- [IANA-MediaTypes]
IANA, "Media Types",
<<https://www.iana.org/assignments/media-types>>.
- [Options] "CoAP Option Numbers", <<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#option-numbers>>.
- [RFC6928] Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <<https://www.rfc-editor.org/info/rfc6928>>.
- [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>>.
- [RFC8782] Reddy, K. T., Ed., Boucadair, M., Ed., Patil, P., Mortensen, A., and N. Teague, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", RFC 8782, DOI 10.17487/RFC8782, May 2020, <<https://www.rfc-editor.org/info/rfc8782>>.
- [RFC8974] Hartke, K. and M. Richardson, "Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP)", RFC 8974, DOI 10.17487/RFC8974, January 2021, <<https://www.rfc-editor.org/info/rfc8974>>.

Appendix A. Examples with Confirmable Messages

The following examples assume NSTART has been increased to 3.

The notations provided in Figure 2 are used in the following subsections.

A.1. Q-Block1 Option

Let's now consider the use of Q-Block1 Option with a CON request as shown in Figure 17. All the blocks are acknowledged (ACK).

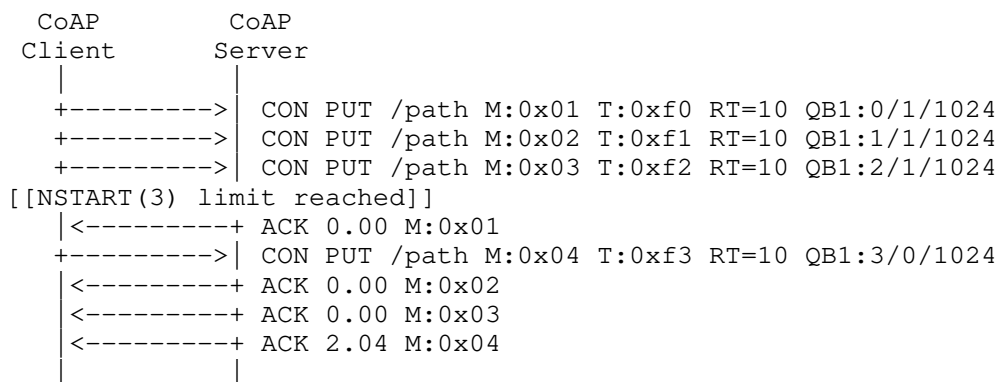


Figure 17: Example of CON Request with Q-Block1 Option (Without Loss)

Now, suppose that a new body of data is to be sent but with some blocks dropped in transmission as illustrated in Figure 18. The client will retry sending blocks for which no ACK was received.

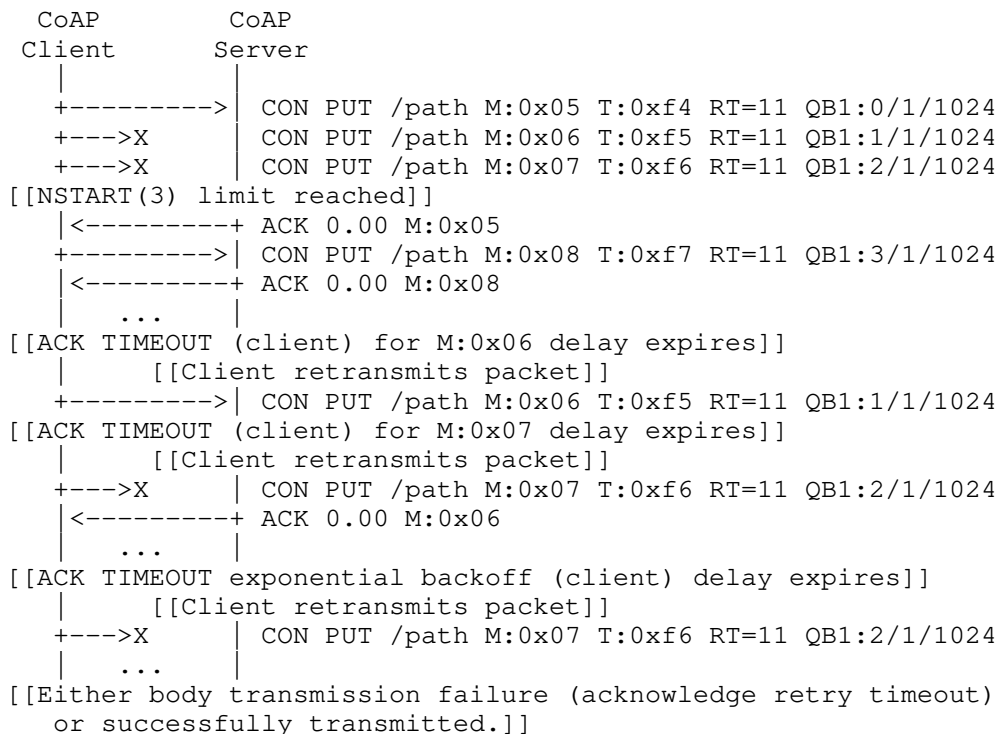


Figure 18: Example of CON Request with Q-Block1 Option (Blocks Recovery)

It is up to the implementation as to whether the application process stops trying to send this particular body of data on reaching `MAX_RETRANSMIT` for any payload, or separately tries to initiate the new transmission of the payloads that have not been acknowledged under these adverse traffic conditions.

If there is likely to be the possibility of transient network losses, then the use of NON should be considered.

A.2. Q-Block2 Option

An example of the use of Q-Block2 Option with Confirmable messages is shown in Figure 19.

```

Client      Server
+----->| CON GET /path M:0x01 T:0xf0 O:0 QB2:0/1/1024
<-----+ ACK 2.05 M:0x01 T:0xf0 O:1234 ET=21 QB2:0/1/1024
<-----+ CON 2.05 M:0xe1 T:0xf0 O:1234 ET=21 QB2:1/1/1024
<-----+ CON 2.05 M:0xe2 T:0xf0 O:1234 ET=21 QB2:2/1/1024
<-----+ CON 2.05 M:0xe3 T:0xf0 O:1234 ET=21 QB2:3/0/1024
----->+ ACK 0.00 M:0xe1
----->+ ACK 0.00 M:0xe2
----->+ ACK 0.00 M:0xe3
...      |
      | [[Observe triggered]]
<-----+ CON 2.05 M:0xe4 T:0xf0 O:1235 ET=22 QB2:0/1/1024
<-----+ CON 2.05 M:0xe5 T:0xf0 O:1235 ET=22 QB2:1/1/1024
<-----+ CON 2.05 M:0xe6 T:0xf0 O:1235 ET=22 QB2:2/1/1024
[[NSTART(3) limit reached]]
----->+ ACK 0.00 M:0xe4
<-----+ CON 2.05 M:0xe7 T:0xf0 O:1235 ET=22 QB2:3/0/1024
----->+ ACK 0.00 M:0xe5
----->+ ACK 0.00 M:0xe6
----->+ ACK 0.00 M:0xe7
...      |
      | [[Observe triggered]]
<-----+ CON 2.05 M:0xe8 T:0xf0 O:1236 ET=23 QB2:0/1/1024
      X<---+ CON 2.05 M:0xe9 T:0xf0 O:1236 ET=23 QB2:1/1/1024
      X<---+ CON 2.05 M:0xea T:0xf0 O:1236 ET=23 QB2:2/1/1024
[[NSTART(3) limit reached]]
----->+ ACK 0.00 M:0xe8
<-----+ CON 2.05 M:0xeb T:0xf0 O:1236 ET=23 QB2:3/0/1024
----->+ ACK 0.00 M:0xeb
...      |
[[ACK TIMEOUT (server) for M:0xe9 delay expires]]
      | [[Server retransmits packet]]
<-----+ CON 2.05 M:0xe9 T:0xf0 O:1236 ET=23 QB2:1/1/1024
[[ACK TIMEOUT (server) for M:0xea delay expires]]
      | [[Server retransmits packet]]
      X<---+ CON 2.05 M:0xea T:0xf0 O:1236 ET=23 QB2:2/1/1024
----->+ ACK 0.00 M:0xe9
...      |
[[ACK TIMEOUT exponential backoff (server) delay expires]]
      | [[Server retransmits packet]]
      X<---+ CON 2.05 M:0xea T:0xf0 O:1236 ET=23 QB2:2/1/1024
...      |
[[Either body transmission failure (acknowledge retry timeout)
or successfully transmitted.]]

```

Figure 19: Example of CON Notifications with Q-Block2 Option

It is up to the implementation as to whether the application process stops trying to send this particular body of data on reaching `MAX_RETRANSMIT` for any payload, or separately tries to initiate the new transmission of the payloads that have not been acknowledged under these adverse traffic conditions.

If there is likely to be the possibility of transient network losses, then the use of `NON` should be considered.

Appendix B. Examples with Reliable Transports

The notations provided in Figure 2 are used in the following subsections.

B.1. Q-Block1 Option

Let's now consider the use of Q-Block1 Option with a reliable transport as shown in Figure 20. There is no acknowledgment of packets at the CoAP layer, just the final result.

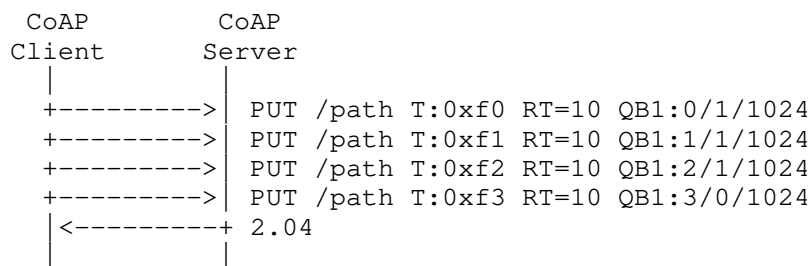


Figure 20: Example of Reliable Request with Q-Block1 Option

If there is likely to be the possibility of transient network losses, then the use of unreliable transport with `NON` should be considered.

B.2. Q-Block2 Option

An example of the use of Q-Block2 Option with a reliable transport is shown in Figure 21.

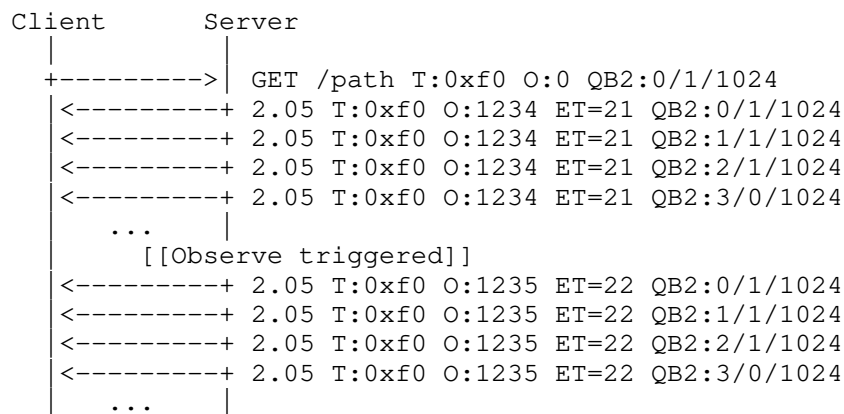


Figure 21: Example of Notifications with Q-Block2 Option

If there is likely to be the possibility of network transient losses, then the use of unreliable transport with NON should be considered.

Acknowledgements

Thanks to Achim Kraus, Jim Schaad, and Michael Richardson for their comments.

Special thanks to Christian Amsuess, Carsten Bormann, and Marco Tiloca for their suggestions and several reviews, which improved this specification significantly. Thanks to Francesca Palombini for the AD review.

Thanks to Pete Resnick for the Gen-ART review, Colin Perkins for the Tsvart review, and Emmanuel Baccelli for the Iotdir review. Thanks to Martin Duke, Eric Vyncke, Benjamin Kaduk, Roman Danyliw, John Scudder, and Lars Eggert for the IESG review.

Some text from [RFC7959] is reused for readers convenience.

Authors' Addresses

Mohamed Boucadair
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Jon Shallow
United Kingdom

Email: supjps-ietf@jpshallow.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

M. Tiloca
RISE AB
G. Selander
F. Palombini
Ericsson AB
J. Park
Universitaet Duisburg-Essen
November 02, 2020

Group OSCORE - Secure Group Communication for CoAP
draft-ietf-core-oscore-groupcomm-10

Abstract

This document defines Group Object Security for Constrained RESTful Environments (Group OSCORE), providing end-to-end security of CoAP messages exchanged between members of a group, e.g. sent over IP multicast. In particular, the described approach defines how OSCORE is used in a group communication setting to provide source authentication for CoAP group requests, sent by a client to multiple servers, and for protection of the corresponding CoAP responses.

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 May 6, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. Terminology	6
2. Security Context	7
2.1. Common Context	9
2.1.1. ID Context	9
2.1.2. Counter Signature Algorithm	9
2.1.3. Counter Signature Parameters	9
2.1.4. Secret Derivation Algorithm	10
2.1.5. Secret Derivation Parameters	10
2.2. Sender Context and Recipient Context	11
2.3. Pairwise Keys	12
2.3.1. Derivation of Pairwise Keys	12
2.3.2. Usage of Sequence Numbers	13
2.3.3. Security Context for Pairwise Mode	13
2.4. Update of Security Context	14
2.4.1. Loss of Mutable Security Context	14
2.4.2. Exhaustion of Sender Sequence Number	15
2.4.3. Retrieving New Security Context Parameters	16
3. The Group Manager	18
3.1. Management of Group Keying Material	19
3.2. Responsibilities of the Group Manager	20
4. The COSE Object	21
4.1. Counter Signature	21
4.2. The 'kid' and 'kid context' parameters	21
4.3. external_aad	22
4.3.1. external_aad for Encryption	22
4.3.2. external_aad for Signing	23
5. OSCORE Header Compression	24
5.1. Examples of Compressed COSE Objects	25
5.1.1. Examples in Group Mode	25
5.1.2. Examples in Pairwise Mode	26
6. Message Binding, Sequence Numbers, Freshness and Replay Protection	27
6.1. Update of Replay Window	27
7. Message Reception	28
8. Message Processing in Group Mode	29
8.1. Protecting the Request	29
8.1.1. Supporting Observe	30
8.2. Verifying the Request	31

8.2.1. Supporting Observe	32
8.3. Protecting the Response	32
8.3.1. Supporting Observe	33
8.4. Verifying the Response	34
8.4.1. Supporting Observe	34
9. Message Processing in Pairwise Mode	35
9.1. Pre-Conditions	36
9.2. Protecting the Request	36
9.3. Verifying the Request	37
9.4. Protecting the Response	37
9.5. Verifying the Response	38
10. Security Considerations	38
10.1. Group-level Security	39
10.2. Uniqueness of (key, nonce)	40
10.3. Management of Group Keying Material	40
10.4. Update of Security Context and Key Rotation	41
10.4.1. Late Update on the Sender	41
10.4.2. Late Update on the Recipient	42
10.5. Collision of Group Identifiers	42
10.6. Cross-group Message Injection	43
10.6.1. Attack Description	43
10.6.2. Attack Prevention in Group Mode	44
10.7. Group OSCORE for Unicast Requests	45
10.8. End-to-end Protection	46
10.9. Master Secret	46
10.10. Replay Protection	47
10.11. Client Aliveness	48
10.12. Cryptographic Considerations	48
10.13. Message Segmentation	49
10.14. Privacy Considerations	49
11. IANA Considerations	50
11.1. OSCORE Flag Bits Registry	50
12. References	50
12.1. Normative References	50
12.2. Informative References	52
Appendix A. Assumptions and Security Objectives	54
A.1. Assumptions	55
A.2. Security Objectives	56
Appendix B. List of Use Cases	57
Appendix C. Example of Group Identifier Format	60
Appendix D. Set-up of New Endpoints	60
Appendix E. Examples of Synchronization Approaches	61
E.1. Best-Effort Synchronization	61
E.2. Baseline Synchronization	62
E.3. Challenge-Response Synchronization	62
Appendix F. No Verification of Signatures in Group Mode	65
Appendix G. Example Values with COSE Capabilities	66
Appendix H. Document Updates	67

H.1. Version -09 to -10	67
H.2. Version -08 to -09	68
H.3. Version -07 to -08	69
H.4. Version -06 to -07	70
H.5. Version -05 to -06	71
H.6. Version -04 to -05	72
H.7. Version -03 to -04	72
H.8. Version -02 to -03	73
H.9. Version -01 to -02	74
H.10. Version -00 to -01	74
Acknowledgments	75
Authors' Addresses	75

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 [I-D.ietf-core-groupcomm-bis] addresses use cases where deployed devices benefit from a group communication model, for example to reduce latencies, improve performance and reduce bandwidth utilization. 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). This specification defines the security protocol for Group communication for CoAP [I-D.ietf-core-groupcomm-bis].

Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] describes a security protocol based on the exchange of protected CoAP messages. OSCORE builds on CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] and provides end-to-end encryption, integrity, replay protection and binding of response to request between a sender and a recipient, independent of transport also in the presence of intermediaries. To this end, a CoAP message is protected by including its payload (if any), certain options, and header fields in a COSE object, which replaces the authenticated and encrypted fields in the protected message.

This document defines Group OSCORE, providing the same end-to-end security properties as OSCORE in the case where CoAP requests have multiple recipients. In particular, the described approach defines how OSCORE is used in a group communication setting to provide source authentication for CoAP group requests, sent by a client to multiple servers, and for protection of the corresponding CoAP responses.

Just like OSCORE, Group OSCORE is independent of transport layer and works wherever CoAP does. Group communication for CoAP [I-D.ietf-core-groupcomm-bis] uses UDP/IP multicast as the underlying data transport.

As with OSCORE, it is possible to combine Group OSCORE with communication security on other layers. One example is the use of transport layer security, such as DTLS [RFC6347][I-D.ietf-tls-dtls13], between one client and one proxy (and vice versa), or between one proxy and one server (and vice versa), in order to protect the routing information of packets from observers. Note that DTLS does not define how to secure messages sent over IP multicast.

Group OSCORE defines two modes of operation:

- o In the group mode, Group OSCORE requests and responses are digitally signed with the private key of the sender and the signature is embedded in the protected CoAP message. The group mode supports all COSE algorithms as well as signature verification by intermediaries. This mode is defined in Section 8 and MUST be supported.
- o In the pairwise mode, two group members exchange Group OSCORE requests and responses over unicast, and the messages are protected with symmetric keys. These symmetric keys are derived from Diffie-Hellman shared secrets, calculated with the asymmetric keys of the sender and recipient, allowing for shorter integrity tags and therefore lower message overhead. This mode is defined in Section 9 and is OPTIONAL to support.

Both modes provide source authentication of CoAP messages. The application decides what mode to use, potentially on a per-message basis. Such decisions can be based, for instance, on pre-configured policies or dynamic assessing of the target recipient and/or resource, among other things. One important case is when requests are protected with the group mode, and responses with the pairwise mode. Since such responses convey shorter integrity tags instead of bigger, full-fledged signatures, this significantly reduces the message overhead in case of many responses to one request.

A special deployment of Group OSCORE is to use pairwise mode only. For example, consider the case of a constrained-node network [RFC7228] with a large number of CoAP endpoints and the objective to establish secure communication between any pair of endpoints with a small provisioning effort and message overhead. Since the total number of security associations that needs to be established grows with the square of the number of nodes, it is desirable to restrict

the provisioned keying material. Moreover, a key establishment protocol would need to be executed for each security association. One solution to this is to deploy Group OSCORE, with the endpoints being part of a group, and use the pairwise mode. This solution assumes a trusted third party called Group Manager (see Section 3), but has the benefit of restricting the symmetric keying material while distributing only the public key of each group member. After that, a CoAP endpoint can locally derive the OSCORE Security Context for the other endpoint in the group, and protect CoAP communications with very low overhead [I-D.ietf-lwig-security-protocol-comparison].

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", "client", "server", "sender" and "recipient"; group communication for CoAP [I-D.ietf-core-groupcomm-bis]; CBOR [I-D.ietf-cbor-7049bis]; COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] and related counter signatures [I-D.ietf-cose-countersign].

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" and "Master Secret", defined in [RFC8613].

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

This document refers also to the following terminology.

- o Keying material: data that is necessary to establish and maintain secure communication among endpoints. This includes, for instance, keys and IVs [RFC4949].
- o Group: a set of endpoints that share group keying material and security parameters (Common Context, see Section 2). Unless specified otherwise, the term group used in this specification refers thus to a "security group" (see Section 2.1 of [I-D.ietf-core-groupcomm-bis]), not to be confused with "CoAP group" or "application group".
- o Group Manager: entity responsible for a group. Each endpoint in a group communicates securely with the respective Group Manager,

which is neither required to be an actual group member nor to take part in the group communication. The full list of responsibilities of the Group Manager is provided in Section 3.2.

- o Silent server: member of a group that never sends protected responses in reply to requests. For CoAP group communications, requests are normally sent without necessarily expecting a response. A silent server may send unprotected responses, as error responses reporting an OSCORE error. Note that an endpoint can implement both a silent server and a client, i.e. the two roles are independent. An endpoint acting only as a silent server performs only Group OSCORE processing on incoming requests. Silent servers maintain less keying material and in particular do not have a Sender Context for the group. Since silent servers do not have a Sender ID, they cannot support the pairwise mode.
- o Group Identifier (Gid): identifier assigned to the group, unique within the set of groups of a given Group Manager.
- o Group request: CoAP request message sent by a client in the group to all servers in that group.
- o Source authentication: evidence that a received message in the group originated from a specific identified group member. This also provides assurance that the message was not tampered with by anyone, be it a different legitimate group member or an endpoint which is not a group member.

2. Security Context

This specification refers to a group as a set of endpoints sharing keying material and security parameters for executing the Group OSCORE protocol (see Section 1.1). Each endpoint which is member of a group maintains a Security Context as defined in Section 3 of [RFC8613], extended as follows (see Figure 1):

- o One Common Context, shared by all the endpoints in the group. Two new parameters are included in the Common Context, namely Counter Signature Algorithm and Counter Signature Parameters. These relate to the computation of counter signatures, when messages are protected using the group mode (see Section 8).

If the pairwise mode is supported, the Common Context is further extended with two new parameters, namely Secret Derivation Algorithm and Secret Derivation Parameters. These relate to the derivation of a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.3.1) to protect messages with the pairwise mode (see Section 9).

- o One Sender Context, extended with the endpoint's private key. The private key is used to sign the message in group mode, and for deriving the pairwise keys in pairwise mode (see Section 2.3). If the pairwise mode is supported, the Sender Context is also extended with the Pairwise Sender Keys associated to the other endpoints (see Section 2.3). The Sender Context is omitted if the endpoint is configured exclusively as silent server.
- o One Recipient Context for each endpoint from which messages are received. It is not necessary to maintain Recipient Contexts associated to endpoints from which messages are not (expected to be) received. The Recipient Context is extended with the public key of the associated endpoint, used to verify the signature in group mode and for deriving the pairwise keys in pairwise mode (see Section 2.3). If the pairwise mode is supported, then the Recipient Context is also extended with the Pairwise Recipient Key associated to the other endpoint (see Section 2.3).

Context Component	New Information Elements
Common Context	Counter Signature Algorithm Counter Signature Parameters *Secret Derivation Algorithm *Secret Derivation Parameters
Sender Context	Endpoint's own private key *Pairwise Sender Keys for the other endpoints
Each Recipient Context	Public key of the other endpoint *Pairwise Recipient Key of the other endpoint

Figure 1: Additions to the OSCORE Security Context. Optional additions are labeled with an asterisk.

Further details about the Security Context of Group OSCORE are provided in the remainder of this section. How the Security Context is established by the group members is out of scope for this specification, but if there is more than one Security Context applicable to a message, then the endpoints MUST be able to tell which Security Context was latest established.

The default setting for how to manage information about the group is described in terms of a Group Manager (see Section 3).

2.1. Common Context

The Common Context may be acquired from the Group Manager (see Section 3). The following sections define how the Common Context is extended, compared to [RFC8613].

2.1.1. ID Context

The ID Context parameter (see Sections 3.3 and 5.1 of [RFC8613]) in the Common Context SHALL contain the Group Identifier (Gid) of the group. The choice of the Gid format is application specific. An example of specific formatting of the Gid is given in Appendix C. The application needs to specify how to handle potential collisions between Gids (see Section 10.5).

2.1.2. Counter Signature Algorithm

Counter Signature Algorithm identifies the digital signature algorithm used to compute a counter signature on the COSE object (see Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign]), when messages are protected using the group mode (see Section 8).

This parameter is immutable once the Common Context is established. Counter Signature Algorithm MUST take value from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms]. The value is associated to a COSE key type, as specified in the "Capabilities" column of the "COSE Algorithms" Registry [COSE.Algorithms]. COSE capabilities for algorithms are defined in Section 8 of [I-D.ietf-cose-rfc8152bis-algs].

The EdDSA signature algorithm and the elliptic curve Ed25519 [RFC8032] are mandatory to implement. If elliptic curve signatures are used, it is RECOMMENDED to implement deterministic signatures with additional randomness as specified in [I-D.mattsson-cfrg-det-sigs-with-noise].

2.1.3. Counter Signature Parameters

Counter Signature Parameters identifies the parameters associated to the digital signature algorithm specified in Counter Signature Algorithm. This parameter is immutable once the Common Context is established.

This parameter is a CBOR array including the following two elements, whose exact structure and value depend on the value of Counter Signature Algorithm:

- o The first element is the array of COSE capabilities for Counter Signature Algorithm, as specified for that algorithm in the "Capabilities" column of the "COSE Algorithms" Registry [COSE.Algorithms] (see Section 8.1 of [I-D.ietf-cose-rfc8152bis-algs]).
- o The second element is the array of COSE capabilities for the COSE key type associated to Counter Signature Algorithm, as specified for that key type in the "Capabilities" column of the "COSE Key Types" Registry [COSE.Key.Types] (see Section 8.2 of [I-D.ietf-cose-rfc8152bis-algs]).

Examples of Counter Signature Parameters are in Appendix G.

2.1.4. Secret Derivation Algorithm

Secret Derivation Algorithm identifies the elliptic curve Diffie-Hellman algorithm used to derive a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.3.1) to protect messages with the pairwise mode (see Section 9).

This parameter is immutable once the Common Context is established. Secret Derivation Algorithm MUST take value from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms]. The value is associated to a COSE key type, as specified in the "Capabilities" column of the "COSE Algorithms" Registry [COSE.Algorithms]. COSE capabilities for algorithms are defined in Section 8 of [I-D.ietf-cose-rfc8152bis-algs].

For endpoints that support the pairwise mode, the ECDH-SS + HKDF-256 algorithm specified in Section 6.3.1 of [I-D.ietf-cose-rfc8152bis-algs] and the X25519 curve [RFC7748] are mandatory to implement.

2.1.5. Secret Derivation Parameters

Secret Derivation Parameters identifies the parameters associated to the elliptic curve Diffie-Hellman algorithm specified in Secret Derivation Algorithm. This parameter is immutable once the Common Context is established.

This parameter is a CBOR array including the following two elements, whose exact structure and value depend on the value of Secret Derivation Algorithm:

- o The first element is the array of COSE capabilities for Secret Derivation Algorithm, as specified for that algorithm in the

"Capabilities" column of the "COSE Algorithms" Registry [COSE.Algorithms] (see Section 8.1 of [I-D.ietf-cose-rfc8152bis-algs]).

- o The second element is the array of COSE capabilities for the COSE key type associated to Secret Derivation Algorithm, as specified for that key type in the "Capabilities" column of the "COSE Key Types" Registry [COSE.Key.Types] (see Section 8.2 of [I-D.ietf-cose-rfc8152bis-algs]).

Examples of Secret Derivation Parameters are in Appendix G.

2.2. Sender Context and Recipient Context

OSCORE specifies the derivation of Sender Context and Recipient Context, specifically of Sender/Recipient Keys and Common IV, from a set of input parameters (see Section 3.2 of [RFC8613]). This derivation applies also to Group OSCORE, and the mandatory-to-implement HKDF and AEAD algorithms are the same as in [RFC8613]. The Sender ID SHALL be unique for each endpoint in a group with a fixed Master Secret, Master Salt and Group Identifier (see Section 3.3 of [RFC8613]).

For Group OSCORE, the Sender Context and Recipient Context additionally contain asymmetric keys, as described previously in Section 2. The private/public key pair of the sender can, for example, be generated by the endpoint or provisioned during manufacturing.

With the exception of the public key of the sender endpoint, a receiver endpoint can derive a complete Security Context from a received Group OSCORE message and the Common Context. The public keys in the Recipient Contexts can be retrieved from the Group Manager (see Section 3) upon joining the group. A public key can alternatively be acquired from the Group Manager at a later time, for example the first time a message is received from a particular endpoint in the group (see Section 8.2 and Section 8.4).

For severely constrained devices, it may be not feasible to simultaneously handle the ongoing processing of a recently received message in parallel with the retrieval of the sender endpoint's public key. Such devices can be configured to drop a received message for which there is no (complete) Recipient Context, and retrieve the sender endpoint's public key in order to have it available to verify subsequent messages from that endpoint.

Furthermore, sufficiently large replay windows should be considered, to handle Partial IV values moving forward fast. This can happen

when a client engages in frequent or long sequences of one-to-one exchanges with servers in the group, such as a large number of block-wise transfers to a single server. When receiving following group requests from that client, other servers in the group may believe to have lost synchronization with the client's Sender Sequence Number. If these servers use an Echo exchange to re-gain synchronization (see Appendix E.3), this in itself may consume a considerable amount of client's Sender Sequence Numbers, hence later resulting in the servers possibly performing a new Echo exchange.

2.3. Pairwise Keys

Certain signature schemes, such as EdDSA and ECDSA, support a secure combined signature and encryption scheme. This section specifies the derivation of "pairwise keys", for use in the pairwise mode of Group OSCORE defined in Section 9.

2.3.1. Derivation of Pairwise Keys

Using the Group OSCORE Security Context (see Section 2), a group member can derive AEAD keys to protect point-to-point communication between itself and any other endpoint in the group. The same AEAD algorithm as in the group mode is used. The key derivation of these so-called pairwise keys follows the same construction as in Section 3.2.1 of [RFC8613]:

Pairwise Recipient Key = HKDF(Recipient Key, Shared Secret, info, L)
Pairwise Sender Key = HKDF(Sender Key, Shared Secret, info, L)

where:

- o The Pairwise Recipient Key is the AEAD key for processing incoming messages from endpoint X.
- o The Pairwise Sender Key is the AEAD key for processing outgoing messages addressed to endpoint X.
- o HKDF is the HKDF algorithm specified by Secret Derivation Algorithm from the Common Context (see Section 2.1.4).
- o The Shared Secret is computed as a static-static Diffie-Hellman shared secret [NIST-800-56A], where the endpoint uses its private key and the public key of the other endpoint X.
- o The Recipient Key and the public key are from the Recipient Context associated to endpoint X.
- o The Sender Key and private key are from the Sender Context.

- o info and L are defined as in Section 3.2.1 of [RFC8613].

If EdDSA asymmetric keys are used, the Edward coordinates are mapped to Montgomery coordinates using the maps defined in Sections 4.1 and 4.2 of [RFC7748], before using the X25519 and X448 functions defined in Section 5 of [RFC7748].

After establishing a partially or completely new Security Context (see Section 3.1 and Section 2.4), the old pairwise keys MUST be deleted. Since new Sender/Recipient Keys are derived from the new group keying material (see Section 2.2), every group member MUST use the new Sender/Recipient Keys when deriving new pairwise keys.

As long as any two group members preserve the same asymmetric keys, their Diffie-Hellman shared secret does not change across updates of the group keying material.

2.3.2. Usage of Sequence Numbers

When using any of its Pairwise Sender Keys, a sender endpoint including the 'Partial IV' parameter in the protected message MUST use the current fresh value of the Sender Sequence Number from its Sender Context (see Section 2.2). That is, the same Sender Sequence Number space is used for all outgoing messages protected with Group OSCORE, thus limiting both storage and complexity.

On the other hand, when combining group and pairwise communication modes, this may result in the Partial IV values moving forward more often. This can happen when a client engages in frequent or long sequences of one-to-one exchanges with servers in the group, by sending requests over unicast.

2.3.3. Security Context for Pairwise Mode

If the pairwise mode is supported, the Security Context additionally includes Secret Derivation Algorithm, Secret Derivation Parameters and the pairwise keys, as described at the beginning of Section 2.

The pairwise keys as well as the shared secrets used in their derivation (see Section 2.3.1) may be stored in memory or recomputed every time they are needed. The shared secret changes only when a public/private key pair used for its derivation changes, which results in the pairwise keys also changing. Additionally, the pairwise keys change if the Sender ID changes or if a new Security Context is established for the group (see Section 2.4.3). In order to optimize protocol performance, an endpoint may store the derived pairwise keys for easy retrieval.

In the pairwise mode, the Sender Context includes the Pairwise Sender Keys to use with the other endpoints (see Figure 1). In order to identify the right key to use, the Pairwise Sender Key for endpoint X may be associated to the Recipient ID of endpoint X, as defined in the Recipient Context (i.e. the Sender ID from the point of view of endpoint X). In this way, the Recipient ID can be used to lookup for the right Pairwise Sender Key. This association may be implemented in different ways, e.g. by storing the pair (Recipient ID, Pairwise Sender Key) or linking a Pairwise Sender Key to a Recipient Context.

2.4. Update of Security Context

It is RECOMMENDED that the immutable part of the Security Context is stored in non-volatile memory, or that it can otherwise be reliably accessed throughout the operation of the group, e.g. after a device reboots. However, also immutable parts of the Security Context may need to be updated, for example due to scheduled key renewal, new or re-joining members in the group, or the fact that the endpoint changes Sender ID (see Section 2.4.3).

On the other hand, the mutable parts of the Security Context are updated by the endpoint when executing the security protocol, but may nevertheless become outdated, e.g. due to loss of the mutable Security Context (see Section 2.4.1) or exhaustion of Sender Sequence Numbers (see Section 2.4.2).

If it is not feasible or practically possible to store and maintain up-to-date the mutable part in non-volatile memory (e.g., due to limited number of write operations), the endpoint MUST be able to detect a loss of the mutable Security Context.

When a loss of mutable Security Context is detected (e.g., following a reboot), the endpoint MUST NOT protect further messages using this Security Context to avoid reusing a nonce with the same AEAD key, and SHOULD instead retrieve new security parameters from the Group Manager (see Section 2.4.1).

2.4.1. Loss of Mutable Security Context

An endpoint that has lost its mutable Security Context, e.g. due to a reboot, needs to prevent the re-use of a nonce with the same AEAD key, and to handle incoming replayed messages.

To this end, after a loss of mutable Security Context, the endpoint SHOULD inform the Group Manager, retrieve new Security Context parameters from the Group Manager (see Section 2.4.3), and use them to derive a new Sender Context (see Section 2.2). In particular, regardless the exact actions taken by the Group Manager, the endpoint

resets its Sender Sequence Number to 0, and derives a new Sender Key. This is in turn used to possibly derive new Pairwise Sender Keys.

From then on, the endpoint MUST use its latest installed Sender Context to protect outgoing messages.

If an endpoint is not able to establish an updated Sender Context, e.g. because of lack of connectivity with the Group Manager, the endpoint MUST NOT protect further messages using the current Security Context.

In order to handle the update of Replay Window in Recipient Contexts, three approaches are discussed in Appendix E. In particular, the approach specified in Appendix E.3 and based on the Echo Option [I-D.ietf-core-echo-request-tag] is a variant of the approach defined in Appendix B.1.2 of [RFC8613] as applicable to Group OSCORE.

2.4.2. Exhaustion of Sender Sequence Number

An endpoint can eventually exhaust the Sender Sequence Number, which is incremented for each new outgoing message including a Partial IV. This is the case for group requests, Observe notifications [RFC7641] and, optionally, any other response.

Implementations MUST be able to detect an exhaustion of Sender Sequence Number, after the endpoint has consumed the largest usable value. If an implementation's integers support wrapping addition, the implementation MUST treat Sender Sequence Number as exhausted when a wrap-around is detected.

Upon exhausting the Sender Sequence Numbers, the endpoint MUST NOT use this Security Context to protect further messages including a Partial IV.

The endpoint SHOULD inform the Group Manager, retrieve new Security Context parameters from the Group Manager (see Section 2.4.3), and use them to derive a new Sender Context (see Section 2.2). In particular, regardless the exact actions taken by the Group Manager, the endpoint resets its Sender Sequence Number to 0, and derives a new Sender Key. This is in turn used to possibly derive new Pairwise Sender Keys.

From then on, the endpoint MUST use its latest installed Sender Context to protect outgoing messages.

2.4.3. Retrieving New Security Context Parameters

The Group Manager can assist an endpoint with an incomplete Sender Context to retrieve missing data of the Security Context and thereby become fully operational in the group again. The two main options for the Group Manager are described in this section: i) assignment of a new Sender ID to the endpoint (see Section 2.4.3.1); and ii) establishment of a new Security Context for the group (see Section 2.4.3.2). Update of Replay Window in Recipient Contexts is discussed in Section 6.1.

As group membership changes, or as group members get new Sender IDs (see Section 2.4.3.1) so do the relevant Recipient IDs that the other endpoints need to keep track of. As a consequence, group members may end up retaining stale Recipient Contexts, that are no longer useful to verify incoming secure messages.

The Recipient ID ('kid') SHOULD NOT be considered as a persistent and reliable indicator of a group member. Such an indication can be achieved only by using that member's public key, when verifying countersignatures of received messages (in group mode), or when verifying messages integrity-protected with pairwise keying material derived from asymmetric keys (in pairwise mode).

Furthermore, applications MAY define policies to: i) delete (long-)unused Recipient Contexts and reduce the impact on storage space; as well as ii) check with the Group Manager that a public key is currently the one associated to a 'kid' value, after a number of consecutive failed verifications.

2.4.3.1. New Sender ID for the Endpoint

The Group Manager may assign a new Sender ID to an endpoint, while leaving the Gid, Master Secret and Master Salt unchanged in the group. In this case, the Group Manager MUST assign a Sender ID that has never been assigned before in the group.

Having retrieved the new Sender ID, and potentially other missing data of the immutable Security Context, the endpoint can derive a new Sender Context (see Section 2.2). When doing so, the endpoint re-initializes the Sender Sequence Number in its Sender Context to 0.

From then on, the endpoint MUST use its latest installed Sender Context to protect outgoing messages.

The assignment of a new Sender ID may be the result of different processes. The endpoint may request a new Sender ID, e.g. because of exhaustion of Sender Sequence Numbers (see Section 2.4.2). An

endpoint may request to re-join the group, e.g. because of losing its mutable Security Context (see Section 2.4.1), and receive as response a new Sender ID together with the latest immutable Security Context.

For the other group members, the Recipient Context corresponding to the old Sender ID becomes stale (see Section 3.1).

2.4.3.2. New Security Context for the Group

The Group Manager may establish a new Security Context for the group (see Section 3.1). The Group Manager does not necessarily establish a new Security Context for the group if one member has an outdated Security Context (see Section 2.4.3.1), unless that was already planned or required for other reasons.

All the group members need to acquire new Security Context parameters from the Group Manager. Once having acquired new Security Context parameters, each group member performs the following actions.

- o From then on, it MUST NOT use the current Security Context to start processing new messages for the considered group.
- o It completes any ongoing message processing for the considered group.
- o It derives and install a new Security Context. In particular:
 - * It re-derives the keying material stored in its Sender Context and Recipient Contexts (see Section 2.2). The Master Salt used for the re-derivations is the updated Master Salt parameter if provided by the Group Manager, or the empty byte string otherwise.
 - * It resets to 0 its Sender Sequence Number in its Sender Context.
 - * It re-initializes the Replay Window of each Recipient Context.
 - * It resets to 0 the sequence number of each ongoing observation where it is an observer client and that it wants to keep active.

From then on, it can resume processing new messages for the considered group. In particular:

- o It MUST use its latest installed Sender Context to protect outgoing messages.

- o It SHOULD use its latest installed Recipient Contexts to process incoming messages, unless application policies admit to temporarily retain and use the old, recent, Security Context (see Section 10.4.1).

The distribution of a new Gid and Master Secret may result in temporarily misaligned Security Contexts among group members. In particular, this may result in a group member not being able to process messages received right after a new Gid and Master Secret have been distributed. A discussion on practical consequences and possible ways to address them, as well as on how to handle the old Security Context, is provided in Section 10.4.

3. The Group Manager

As with OSCORE, endpoints communicating with Group OSCORE need to establish the relevant Security Context. Group OSCORE endpoints need to acquire OSCORE input parameters, information about the group(s) and about other endpoints in the group(s). This specification is based on the existence of an entity called Group Manager which is responsible for the group, but does not mandate how the Group Manager interacts with the group members. The responsibilities of the Group Manager are compiled in Section 3.2.

It is RECOMMENDED to use a Group Manager as described in [I-D.ietf-ace-key-groupcomm-oscure], where the join process is based on the ACE framework for authentication and authorization in constrained environments [I-D.ietf-ace-oauth-authz].

The Group Manager assigns unique Group Identifiers (Gids) to different groups under its control, as well as unique Sender IDs (and thereby Recipient IDs) to the members of those groups. The Group Manager MUST NOT reassign a Sender ID within the same group, and MUST NOT reassign a Gid value to the same group. According to a hierarchical approach, the Gid value assigned to a group is associated to a dedicated space for the values of Sender ID and Recipient ID of the members of that group.

In addition, the Group Manager maintains records of the public keys of endpoints in a group, and provides information about the group and its members to other members and selected roles. Upon nodes' joining, the Group Manager collects such public keys and MUST verify proof-of-possession of the respective private key.

An endpoint acquires group data such as the Gid and OSCORE input parameters including its own Sender ID from the Group Manager, and provides information about its public key to the Group Manager, for example upon joining the group.

A group member can retrieve from the Group Manager the public key and other information associated to another group member, with which it can generate the corresponding Recipient Context. An application can configure a group member to asynchronously retrieve information about Recipient Contexts, e.g. by Observing [RFC7641] a resource at the Group Manager to get updates on the group membership.

The Group Manager MAY serve additional entities acting as signature checkers, e.g. intermediary gateways. These entities do not join a group as members, but can retrieve public keys of group members from the Group Manager, in order to verify counter signatures of group messages. A signature checker MUST be authorized for retrieving public keys of members in a specific group from the Group Manager. To this end, the same method mentioned above based on the ACE framework [I-D.ietf-ace-oauth-authz] can be used.

3.1. Management of Group Keying Material

In order to establish a new Security Context for a group, a new Group Identifier (Gid) for that group and a new value for the Master Secret parameter MUST be generated. When distributing the new Gid and Master Secret, the Group Manager MAY distribute also a new value for the Master Salt parameter, and SHOULD preserve the current value of the Sender ID of each group member.

The Group Manager MUST NOT reassign a Gid value to the same group. That is, each group can have a given Gid at most once during its lifetime. An example of Gid format supporting this operation is provided in Appendix C.

The Group Manager MUST NOT reassign a previously used Sender ID ('kid') with the same Gid, Master Secret and Master Salt. Even if Gid and Master Secret are renewed as described in this section, the Group Manager MUST NOT reassign an endpoint's Sender ID ('kid') within a same group (see Section 2.4.3.1).

If required by the application (see Appendix A.1), it is RECOMMENDED to adopt a group key management scheme, and securely distribute a new value for the Gid and 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 to preserve backward security and forward security in the group, if the application requires it.

The specific approach used to distribute new group data is out of the scope of this document. However, it is RECOMMENDED that the Group Manager supports the distribution of the new Gid and Master Secret

parameter to the group according to the Group Rekeying Process described in [I-D.ietf-ace-key-groupcomm-oscore].

3.2. Responsibilities of the Group Manager

The Group Manager is responsible for performing the following tasks:

1. Creating and managing OSCORE groups. This includes the assignment of a Gid to every newly created group, as well as ensuring uniqueness of Gids within the set of its OSCORE groups.
2. Defining policies for authorizing the joining of its OSCORE groups.
3. Handling the join process to add new endpoints as group members.
4. Establishing the Common Context part of the Security Context, and providing it to authorized group members during the join process, together with the corresponding Sender Context.
5. Generating and managing Sender IDs within its OSCORE groups, as well as assigning and providing them to new endpoints during the join process, or to current group members upon request of renewal. This includes ensuring that each Sender ID is unique within each of the OSCORE groups, and that it is not reassigned within the same group.
6. Defining communication policies for each of its OSCORE groups, and signalling them to new endpoints during the join process.
7. Renewing the Security Context of an OSCORE group upon membership change, by revoking and renewing common security parameters and keying material (rekeying).
8. 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.
9. Updating the Gid of its OSCORE groups, upon renewing the respective Security Context. This includes ensuring that the same Gid value is not reassigned to the same group.
10. Acting as key repository, in order to handle the public keys of the members of its OSCORE groups, and providing such public keys to other members of the same group upon request. The actual storage of public keys may be entrusted to a separate secure storage device or service.

11. Validating that the format and parameters of public keys of group members are consistent with the countersignature algorithm and related parameters used in the respective OSCORE group.

The Group Manager described in [I-D.ietf-ace-key-groupcomm-oscore] provides these functionalities.

4. The COSE Object

Building on Section 5 of [RFC8613], this section defines how to use COSE [I-D.ietf-cose-rfc8152bis-struct] to wrap and protect data in the original message. OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Associated Data (AEAD) algorithm. Unless otherwise specified, the following modifications apply for both the group mode and the pairwise mode of Group OSCORE.

4.1. Counter Signature

For the group mode only, the 'unprotected' field MUST additionally include the following parameter:

- o COSE_CounterSignature0: its value is set to the counter signature of the COSE object, computed by the sender as described in Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign], by using its private key and according to the Counter Signature Algorithm and Counter Signature Parameters in the Security Context.

In particular, the Countersign_structure contains the context text string "CounterSignature0", the external_aad as defined in Section 4.3.2 of this specification, and the ciphertext of the COSE object as payload.

4.2. The 'kid' and 'kid context' parameters

The value of the 'kid' parameter in the 'unprotected' field of response messages MUST be set to the Sender ID of the endpoint transmitting the message. That is, unlike in [RFC8613], the 'kid' parameter is always present in all messages, both requests and responses.

The value of the 'kid context' parameter in the 'unprotected' field of requests messages MUST be set to the ID Context, i.e. the Group Identifier value (Gid) of the group. That is, unlike in [RFC8613], the 'kid context' parameter is always present in requests.

4.3. external_aad

The external_aad of the Additional Authenticated Data (AAD) is different compared to OSCORE. In particular, there is one external_aad used for encryption (both in group mode and pairwise mode), and another external_aad used for signing (only in group mode).

4.3.1. external_aad for Encryption

The external_aad for encryption (see Section 4.3 of [I-D.ietf-cose-rfc8152bis-struct]), used both in group mode and pairwise mode, includes also the counter signature algorithm and related signature parameters, as well as the value of the 'kid context' in the COSE object of the request (see Figure 2).

```
external_aad = bstr .cbor aad_array

aad_array = [
  oscore_version : uint,
  algorithms : [alg_aead : int / tstr,
                alg_countersign : int / tstr,
                par_countersign : [countersign_alg_capab,
                                   countersign_key_type_capab],
                par_countersign_key : countersign_key_type_capab],
  request_kid : bstr,
  request_piv : bstr,
  options : bstr,
  request_kid_context : bstr
]
```

Figure 2: external_aad for Encryption

Compared with Section 5.4 of [RFC8613], the aad_array has the following differences.

- o The 'algorithms' array in the aad_array additionally includes:
 - * 'alg_countersign', which specifies Counter Signature Algorithm from the Common Context (see Section 2.1.2). This parameter MUST encode the value of Counter Signature Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this counter signature algorithm.
 - * 'par_countersign', which specifies the CBOR array Counter Signature Parameters from the Common Context (see Section 2.1.3). In particular:

- + `'countersign_alg_capab'` is the array of COSE capabilities for the countersignature algorithm indicated in `'alg_countersign'`. This is the first element of the CBOR array Counter Signature Parameters from the Common Context.
- + `'countersign_key_type_capab'` is the array of COSE capabilities for the COSE key type used by the countersignature algorithm indicated in `'alg_countersign'`. This is the second element of the CBOR array Counter Signature Parameters from the Common Context.
- * `'par_countersign_key'`, which specifies the parameters associated to the keys used with the countersignature algorithm indicated in `'alg_countersign'`. These parameters are encoded as a CBOR array `'countersign_key_type_capab'`, whose exact structure and value depend on the value of `'alg_countersign'`.

In particular, `'countersign_key_type_capab'` is the array of COSE capabilities for the COSE key type of the keys used with the countersignature algorithm. This is the second element of the CBOR array Counter Signature Parameters from the Common Context.

Examples of `'par_countersign_key'` are in Appendix G.

- o The new element `'request_kid_context'` contains the value of the `'kid context'` in the COSE object of the request (see Section 4.2).

4.3.2. external_aad for Signing

The `external_aad` for signing (see Section 4.3 of [I-D.ietf-cose-rfc8152bis-struct]) used in group mode is identical to the `external_aad` for encryption (see Section 4.3.1) with the addition of the OSCORE option (see Figure 3).

```
external_aad = bstr .cbor aad_array

aad_array = [
  oscore_version : uint,
  algorithms : [alg_aead : int / tstr,
                alg_countersign : int / tstr,
                par_countersign : [countersign_alg_capab,
                                   countersign_key_type_capab],
                par_countersign_key : countersign_key_type_capab],
  request_kid : bstr,
  request_piv : bstr,
  options : bstr,
  request_kid_context : bstr,
  OSCORE_option: bstr
]
```

Figure 3: external_aad for Signing

Compared with Section 5.4 of [RFC8613] the aad_array additionally includes:

- o the 'algorithms' array, as defined in the external_aad for encryption (see Section 4.3.1);
- o the 'request_kid_context' element, as defined in the external_aad for encryption (see Section 4.3.1);
- o the value of the OSCORE Option present in the protected message, encoded as a binary string.

Note for implementation: this construction requires the OSCORE option of the message to be generated before calculating the signature. Also, the aad_array needs to be large enough to contain the largest possible OSCORE option.

5. OSCORE Header Compression

The OSCORE header compression defined in Section 6 of [RFC8613] is used, with the following differences.

- o The payload of the OSCORE message SHALL encode the ciphertext of the COSE_Encrypt0 object. In the group mode, the ciphertext above is concatenated with the value of the COSE_CounterSignature0 of the COSE object, computed as described in Section 4.1.
- o This specification defines the usage of the sixth least significant bit, called the "Group Flag", in the first byte of the

OSCORE option containing the OSCORE flag bits. This flag bit is specified in Section 11.1.

- o The Group Flag MUST be set to 1 if the OSCORE message is protected using the group mode (see Section 8).
- o The Group Flag MUST be set to 0 if the OSCORE message is protected using the pairwise mode (see Section 9). The Group Flag MUST also be set to 0 for ordinary OSCORE messages processed according to [RFC8613].

5.1. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples of Group OSCORE used in group mode (see Section 5.1.1) or in pairwise mode (see Section 5.1.2).

The examples assume that the COSE_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the examples do not include the full CoAP unprotected message or the full Security Context, 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 5 and divided into two parts, since the object is transported in two CoAP fields: OSCORE option and payload.

The examples assume that the plaintext (see Section 5.3 of [RFC8613]) is 6 bytes long, and that the AEAD tag is 8 bytes long, hence resulting in a ciphertext which is 14 bytes long. When using the group mode, COUNTERSIGN denotes the COSE_CounterSignature0 byte string as described in Section 4, and is 64 bytes long.

5.1.1. Examples in Group Mode

- o Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c

Before compression (96 bytes):

```
[
  h'',
  { 4:h'25', 6:h'05', 10:h'44616c', 11:COUNTERSIGN },
  h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (85 bytes):

Flag byte: 0b00111001 = 0x39

Option Value: 39 05 03 44 61 6c 25 (7 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 COUNTERSIGN
(14 bytes + size of COUNTERSIGN)

- o Response with ciphertext = 0x60b035059d9ef5667c5a0710823b, kid = 0x52 and no Partial IV.

Before compression (88 bytes):

```
[  
  h'',  
  { 4:h'52', 11:COUNTERSIGN },  
  h'60b035059d9ef5667c5a0710823b'  
]
```

After compression (80 bytes):

Flag byte: 0b00101000 = 0x28

Option Value: 28 52 (2 bytes)

Payload: 60 b0 35 05 9d 9e f5 66 7c 5a 07 10 82 3b COUNTERSIGN
(14 bytes + size of COUNTERSIGN)

5.1.2. Examples in Pairwise Mode

- o Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c

Before compression (32 bytes):

```
[  
  h'',  
  { 4:h'25', 6:h'05', 10:h'44616c' },  
  h'aea0155667924dff8a24e4cb35b9'  
]
```

After compression (21 bytes):

Flag byte: 0b00011001 = 0x19

Option Value: 19 05 03 44 61 6c 25 (7 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

- o Response with ciphertext = 0x60b035059d9ef5667c5a0710823b, kid = 0x52 and no Partial IV.

Before compression (24 bytes):

```
[
  h'',
  { 4:h'52' },
  h'60b035059d9ef5667c5a0710823b'
]
```

After compression (16 bytes):

Flag byte: 0b00001000 = 0x08

Option Value: 08 52 (2 bytes)

Payload: 60 b0 35 05 9d 9e f5 66 7c 5a 07 10 82 3b (14 bytes)

6. Message Binding, Sequence Numbers, Freshness and Replay Protection

The requirements and properties described in Section 7 of [RFC8613] also apply to OSCORE used in group communication. In particular, Group OSCORE provides message binding of responses to requests, which enables absolute freshness of responses that are not notifications, relative freshness of requests and notification responses, and replay protection of requests.

6.1. Update of Replay Window

A new server joining a group may not be aware of the current Partial IVs (Sender Sequence Numbers of the clients). Hence, when receiving a request from a particular client for the first time, the new server is not able to verify if that request is a replay. The same holds when a server loses its mutable Security Context (see Section 2.4.1), for instance after a device reboot.

The exact way to address this issue is application specific, and depends on the particular use case and its replay requirements. The

list of methods to handle the update of a Replay Window is part of the group communication policy, and different servers can use different methods. Appendix E describes three possible approaches that can be considered to address the issue discussed above.

Furthermore, when the Group Manager establishes a new Security Context for the group (see Section 2.4.3.2), every server re-initializes the Replay Window in each of its Recipient Contexts.

7. Message Reception

Upon receiving a protected message, a recipient endpoint retrieves a Security Context as in [RFC8613]. An endpoint MUST be able to distinguish between a Security Context to process OSCORE messages as in [RFC8613] and a Security Context to process Group OSCORE messages as defined in this specification.

To this end, an endpoint can take into account the different structure of the Security Context defined in Section 2, for example based on the presence of Counter Signature Algorithm in the Common Context. Alternatively implementations can use an additional parameter in the Security Context, to explicitly signal that it is intended for processing Group OSCORE messages.

If either of the following two conditions holds, a recipient endpoint MUST discard the incoming protected message:

- o The Group Flag is set to 1, and the recipient endpoint can not retrieve a Security Context which is both valid to process the message and also associated to an OSCORE group.
- o The Group Flag is set to 0, and the recipient endpoint retrieves a Security Context which is both valid to process the message and also associated to an OSCORE group, but the endpoint does not support the pairwise mode.

Otherwise, if a Security Context associated to an OSCORE group and valid to process the message is retrieved, the recipient endpoint processes the message with Group OSCORE, using the group mode (see Section 8) if the Group Flag is set to 1, or the pairwise mode (see Section 9) if the Group Flag is set to 0.

Note that, if the Group Flag is set to 0, and the recipient endpoint retrieves a Security Context which is valid to process the message but is not associated to an OSCORE group, then the message is processed according to [RFC8613].

8. Message Processing in Group Mode

When using the group mode, messages are protected and processed as specified in [RFC8613], with the modifications described in this section. The security objectives of the group mode are discussed in Appendix A.2. The group mode **MUST** be supported.

During all the steps of the message processing, an endpoint **MUST** use the same Security Context for the considered group. That is, an endpoint **MUST NOT** install a new Security Context for that group (see Section 2.4.3.2) until the message processing is completed.

The group mode **MUST** be used to protect group requests intended for multiple recipients or for the whole group. This includes both requests directly addressed to multiple recipients, e.g. sent by the client over multicast, as well as requests sent by the client over unicast to a proxy, that forwards them to the intended recipients over multicast [I-D.ietf-core-groupcomm-bis].

As per [RFC7252][I-D.ietf-core-groupcomm-bis], group requests sent over multicast **MUST** be Non-Confirmable, and thus are not retransmitted by the CoAP messaging layer. Instead, applications should store such outgoing messages for a pre-defined, sufficient amount of time, in order to correctly perform possible retransmissions at the application layer. According to Section 5.2.3 of [RFC7252], responses to Non-Confirmable group requests **SHOULD** also be Non-Confirmable, but endpoints **MUST** be prepared to receive Confirmable responses in reply to a Non-Confirmable group request. Confirmable group requests are acknowledged in non-multicast environments, as specified in [RFC7252].

Furthermore, endpoints in the group locally perform error handling and processing of invalid messages according to the same principles adopted in [RFC8613]. However, a recipient **MUST** stop processing and silently reject any message which is malformed and does not follow the format specified in Section 4, or which is not cryptographically validated in a successful way. In either case, it is **RECOMMENDED** that the recipient does not send back any error message. This prevents servers from replying with multiple error messages to a client sending a group request, so avoiding the risk of flooding and possibly congesting the network.

8.1. Protecting the Request

A client transmits a secure group request as described in Section 8.1 of [RFC8613], with the following modifications.

- o In step 2, the Additional Authenticated Data is modified as described in Section 4 of this document.
- o In step 4, the encryption of the COSE object is modified as described in Section 4 of this document. The encoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, the Group Flag MUST be set to 1.
- o In step 5, the counter signature is computed and the format of the OSCORE message is modified as described in Section 4 and Section 5 of this document. In particular, the payload of the OSCORE message includes also the counter signature.

8.1.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds for each newly started observation.

- o If the client intends to keep the observation active beyond a possible change of Sender ID, the client MUST store the value of the 'kid' parameter from the original Observe request, and retain it for the whole duration of the observation. Even in case the client is individually rekeyed and receives a new Sender ID from the Group Manager (see Section 2.4.3.1), the client MUST NOT update the stored value associated to a particular Observe request.
- o If the client intends to keep the observation active beyond a possible change of ID Context following a group rekeying (see Section 3.1), then the following applies.
 - * The client MUST store the value of the 'kid context' parameter from the original Observe request, and retain it for the whole duration of the observation. Upon establishing a new Security Context with a new ID Context as Gid (see Section 2.4.3.2), the client MUST NOT update the stored value associated to a particular Observe request.
 - * The client MUST store an invariant identifier of the group, which is immutable even in case the Security Context of the group is re-established. For example, this invariant identifier can be the "group name" in [I-D.ietf-ace-key-groupcomm-oscure], where it is used for joining the group and retrieving the current group keying material from the Group Manager.

After a group rekeying, such an invariant information makes it simpler for the observer client to retrieve the current group

keying material from the Group Manager, in case the client has missed both the rekeying messages and the first observe notification protected with the new Security Context (see Section 8.3.1).

8.2. Verifying the Request

Upon receiving a secure group request with the Group Flag set to 1, following the procedure in Section 7, a server proceeds as described in Section 8.2 of [RFC8613], with the following modifications.

- o In step 2, the decoding of the compressed COSE object follows Section 5 of this document. In particular:
 - * If the server discards the request due to not retrieving a Security Context associated to the OSCORE group, the server MAY respond with a 4.02 (Bad Option) error. When doing so, the server MAY set an Outer Max-Age option with value zero, and MAY include a descriptive string as diagnostic payload.
 - * If the received 'kid context' matches an existing ID Context (Gid) but the received 'kid' does not match any Recipient ID in this Security Context, then the server MAY create a new Recipient Context for this Recipient ID and initialize it according to Section 3 of [RFC8613], and also retrieve the associated public key. Such a configuration is application specific. If the application does not specify dynamic derivation of new Recipient Contexts, then the server SHALL stop processing the request.
- o In step 4, the Additional Authenticated Data is modified as described in Section 4 of this document.
- o In step 6, the server also verifies the counter signature using the public key of the client from the associated Recipient Context. In particular:
 - * If the server does not have the public key of the client yet, the server MUST stop processing the request and MAY respond with a 5.03 (Service Unavailable) response. The response MAY include a Max-Age Option, indicating to the client the number of seconds after which to retry. If the Max-Age Option is not present, a retry time of 60 seconds will be assumed by the client, as default value defined in Section 5.10.5 of [RFC7252].
 - * If the signature verification fails, the server SHALL stop processing the request and MAY respond with a 4.00 (Bad

Request) response. If the verification fails, the same steps are taken as if the decryption had failed. In particular, the Replay Window is only updated if both the signature verification and the decryption succeed.

- o Additionally, if the used Recipient Context was created upon receiving this group request and the message is not verified successfully, the server MAY delete that Recipient Context. Such a configuration, which is specified by the application, mitigates attacks that aim at overloading the server's storage.

A server SHOULD NOT process a request if the received Recipient ID ('kid') is equal to its own Sender ID in its own Sender Context. For an example where this is not fulfilled, see Section 6.2.1 in [I-D.tiloca-core-observe-multicast-notifications].

8.2.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds for each newly started observation.

- o The server MUST store the value of the 'kid' parameter from the original Observe request, and retain it for the whole duration of the observation. The server MUST NOT update the stored value of a 'kid' parameter associated to a particular Observe request, even in case the observer client is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.4.3.1).
- o The server MUST store the value of the 'kid context' parameter from the original Observe request, and retain it for the whole duration of the observation, beyond a possible change of ID Context following a group rekeying (see Section 3.1). That is, upon establishing a new Security Context with a new ID Context as Gid (see Section 2.4.3.2), the server MUST NOT update the stored value associated to the ongoing observation.

8.3. Protecting the Response

If a server generates a CoAP message in response to a Group OSCORE request, then the server SHALL follow the description in Section 8.3 of [RFC8613], with the modifications described in this section.

Note that the server always protects a response with the Sender Context from its latest Security Context, and that establishing a new Security Context resets the Sender Sequence Number to 0 (see Section 3.1).

- o In step 2, the Additional Authenticated Data is modified as described in Section 4 of this document.
- o In step 3, if the server is using a different Security Context for the response compared to what was used to verify the request (see Section 3.1), then the server MUST include its Sender Sequence Number as Partial IV in the response and use it to build the AEAD nonce to protect the response. This prevents the AEAD nonce from the request from being reused.
- o In step 4, the encryption of the COSE object is modified as described in Section 4 of this document. The encoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, the Group Flag MUST be set to 1. If the server is using a different ID Context (Gid) for the response compared to what was used to verify the request (see Section 3.1), then the new ID Context MUST be included in the 'kid context' parameter of the response.
- o In step 5, the counter signature is computed and the format of the OSCORE message is modified as described in Section 5 of this document. In particular, the payload of the OSCORE message includes also the counter signature.

8.3.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds when protecting notifications for an ongoing observation.

- o The server MUST use the stored value of the 'kid' parameter from the original Observe request (see Section 8.2.1), as value for the 'request_kid' parameter in the two external_aad structures (see Section 4.3.1 and Section 4.3.2).
- o The server MUST use the stored value of the 'kid context' parameter from the original Observe request (see Section 8.2.1), as value for the 'request_kid_context' parameter in the two external_aad structures (see Section 4.3.1 and Section 4.3.2).

Furthermore, the server may have ongoing observations started by Observe requests protected with an old Security Context. After completing the establishment of a new Security Context, the server MUST protect the following notifications with the Sender Context of the new Security Context.

For each ongoing observation, the server MUST include in the first notification protected with the new Security Context also the 'kid context' parameter, which is set to the ID Context (Gid) of the new

Security Context. It is OPTIONAL for the server to include the ID Context (Gid) in the 'kid context' parameter also in further following notifications for those observations.

8.4. Verifying the Response

Upon receiving a secure response message with the Group Flag set to 1, following the procedure in Section 7, the client proceeds as described in Section 8.4 of [RFC8613], with the following modifications.

Note that a client may receive a response protected with a Security Context different from the one used to protect the corresponding group request, and that, upon the establishment of a new Security Context, the client re-initializes its replay windows in its Recipient Contexts (see Section 3.1).

- o In step 2, the decoding of the compressed COSE object is modified as described in Section 5 of this document. If the received 'kid context' matches an existing ID Context (Gid) but the received 'kid' does not match any Recipient ID in this Security Context, then the client MAY create a new Recipient Context for this Recipient ID and initialize it according to Section 3 of [RFC8613], and also retrieve the associated public key. If the application does not specify dynamic derivation of new Recipient Contexts, then the client SHALL stop processing the response.
- o In step 3, the Additional Authenticated Data is modified as described in Section 4 of this document.
- o In step 5, the client also verifies the counter signature using the public key of the server from the associated Recipient Context. If the verification fails, the same steps are taken as if the decryption had failed.
- o Additionally, if the used Recipient Context was created upon receiving this response and the message is not verified successfully, the client MAY delete that Recipient Context. Such a configuration, which is specified by the application, mitigates attacks that aim at overloading the client's storage.

8.4.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds when verifying notifications for an ongoing observation.

- o The client MUST use the stored value of the 'kid' parameter from the original Observe request (see Section 8.1.1), as value for the

'request_kid' parameter in the two external_aad structures (see Section 4.3.1 and Section 4.3.2).

- o The client MUST use the stored value of the 'kid context' parameter from the original Observe request (see Section 8.1.1), as value for the 'request_kid_context' parameter in the two external_aad structures (see Section 4.3.1 and Section 4.3.2).

This ensures that the client can correctly verify notifications, even in case it is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.4.3.1), as well as when it establishes a new Security Context with a new ID Context (Gid) following a group rekeying (see Section 3.1).

9. Message Processing in Pairwise Mode

When using the pairwise mode of Group OSCORE, messages are protected and processed as in Section 8, with the modifications described in this section. The security objectives of the pairwise mode are discussed in Appendix A.2.

The pairwise mode takes advantage of an existing Security Context for the group mode to establish a Security Context shared exclusively with any other member. In order to use the pairwise mode, the signature scheme of the group mode MUST support a combined signature and encryption scheme. This can be, for example, signature using ECDSA, and encryption using AES-CCM with a key derived with ECDH.

The pairwise mode does not support the use of additional entities acting as verifiers of source authentication and integrity of group messages, such as intermediary gateways (see Section 3).

The pairwise mode MAY be supported. An endpoint implementing only a silent server does not support the pairwise mode.

If the signature algorithm used in the group supports ECDH (e.g., ECDSA, EdDSA), the pairwise mode MUST be supported by endpoints that use the CoAP Echo Option [I-D.ietf-core-echo-request-tag] and/or block-wise transfers [RFC7959], for instance for responses after the first block-wise request, which possibly targets all servers in the group and includes the CoAP Block2 option (see Section 2.3.6 of [I-D.ietf-core-groupcomm-bis]). This prevents the attack described in Section 10.7, which leverages requests sent over unicast to a single group member and protected with the group mode.

The pairwise mode protects messages between two members of a group, essentially following [RFC8613], but with the following notable differences:

- o The 'kid' and 'kid context' parameters of the COSE object are used as defined in Section 4.2 of this document.
- o The external_aad defined in Section 4.3.1 of this document is used for the encryption process.
- o The Pairwise Sender/Recipient Keys used as Sender/Recipient keys are derived as defined in Section 2.3 of this document.

Senders MUST NOT use the pairwise mode to protect a message intended for multiple recipients. The pairwise mode is defined only between two endpoints and the keying material is thus only available to one recipient.

The Group Manager MAY indicate that the group uses also the pairwise mode, as part of the group data provided to candidate group members when joining the group.

9.1. Pre-Conditions

In order to protect an outgoing message in pairwise mode, the sender needs to know the public key and the Recipient ID for the recipient endpoint, as stored in the Recipient Context associated to that endpoint (see Pairwise Sender Context of Section 2.3.3).

Furthermore, the sender needs to know the individual address of the recipient endpoint. This information may not be known at any given point in time. For instance, right after having joined the group, a client may know the public key and Recipient ID for a given server, but not the addressing information required to reach it with an individual, one-to-one request.

To make addressing information of individual endpoints available, servers in the group MAY expose a resource to which a client can send a group request targeting a server or a set of servers, identified by their 'kid' value(s). The specified set may be empty, hence identifying all the servers in the group. Further details of such an interface are out of scope for this document.

9.2. Protecting the Request

When using the pairwise mode, the request is protected as defined in Section 8.1, with the following differences.

- o The Group Flag MUST be set to 0.
- o The used Sender Key is the Pairwise Sender Key (see Section 2.3).

- o The counter signature is not computed and therefore not included in the message. The payload of the protected request thus terminates with the encoded ciphertext of the COSE object, just like in [RFC8613].

Note that, like in the group mode, the external_aad for encryption is generated as in Section 4.3.1, and the Partial IV is the current fresh value of the client's Sender Sequence Number (see Section 2.3.2).

9.3. Verifying the Request

Upon receiving a request with the Group Flag set to 0, following the procedure in Section 7, the server MUST process it as defined in Section 8.2, with the following differences.

- o If the server discards the request due to not retrieving a Security Context associated to the OSCORE group or to not supporting the pairwise mode, the server MAY respond with a 4.02 (Bad Option) error. When doing so, the server MAY set an Outer Max-Age option with value zero, and MAY include a descriptive string as diagnostic payload.
- o If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.3.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.
- o The used Recipient Key is the Pairwise Recipient Key (see Section 2.3).
- o No verification of counter signature occurs, as there is none included in the message.

9.4. Protecting the Response

When using the pairwise mode, a response is protected as defined in Section 8.3, with the following differences.

- o The Group Flag MUST be set to 0.
- o The used Sender Key is the Pairwise Sender Key (see Section 2.3).
- o The counter signature is not computed and therefore not included in the message. The payload of the protected response thus terminates with the encoded ciphertext of the COSE object, just like in [RFC8613].

9.5. Verifying the Response

Upon receiving a response with the Group Flag set to 0, following the procedure in Section 7, the client MUST process it as defined in Section 8.4, with the following differences.

- o If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.3.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.
- o The used Recipient Key is the Pairwise Recipient Key (see Section 2.3).
- o No verification of counter signature occurs, as there is none included in the message.

10. Security Considerations

The same threat model discussed for OSCORE in Appendix D.1 of [RFC8613] holds for Group OSCORE. In addition, when using the group mode, source authentication of messages is explicitly ensured by means of counter signatures, as discussed in Section 10.1.

The same considerations on supporting Proxy operations discussed for OSCORE in Appendix D.2 of [RFC8613] hold for Group OSCORE.

The same considerations on protected message fields for OSCORE discussed in Appendix D.3 of [RFC8613] hold for Group OSCORE.

The same considerations on uniqueness of (key, nonce) pairs for OSCORE discussed in Appendix D.4 of [RFC8613] hold for Group OSCORE. This is further discussed in Section 10.2 of this document.

The same considerations on unprotected message fields for OSCORE discussed in Appendix D.5 of [RFC8613] hold for Group OSCORE, with the following difference. The counter signature included in a Group OSCORE message protected in group mode is computed also over the value of the OSCORE option, which is part of the Additional Authenticated Data used in the signing process. This is further discussed in Section 10.6 of this document.

As discussed in Section 6.2.3 of [I-D.ietf-core-groupcomm-bis], Group OSCORE addresses security attacks against CoAP listed in Sections 11.2-11.6 of [RFC7252], especially when run over IP multicast.

The rest of this section first discusses security aspects to be taken into account when using Group OSCORE. Then it goes through aspects covered in the security considerations of OSCORE (see Section 12 of [RFC8613]), and discusses how they hold when Group OSCORE is used.

10.1. Group-level Security

The group mode described in Section 8 relies on commonly shared group keying material to protect communication within a group. This has the following implications.

- o 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.
- o The AEAD algorithm provides only group authentication, i.e. it ensures that a message sent to a group has been sent by a member of that group, but not necessarily by the alleged sender. This is why source authentication of messages sent to a group is ensured through a counter signature, which is computed by the sender using its own private key and then appended to the message payload.

Instead, the pairwise mode described in Section 9 protects messages by using pairwise symmetric keys, derived from the static-static Diffie-Hellman shared secret computed from the asymmetric keys of the sender and recipient endpoint (see Section 2.3). Therefore, in the pairwise mode, the AEAD algorithm provides both pairwise data-confidentiality and source authentication of messages, without using counter signatures.

The long-term storing of the Diffie-Hellman shared secret is a potential security issue. In fact, if the shared secret of two group members is leaked, a third group member can exploit it to impersonate any of those two group members, by deriving and using their pairwise key. The possibility of such leakage should be contemplated, as more likely to happen than the leakage of a private key, which could be rather protected at a significantly higher level than generic memory, e.g. by using a Trusted Platform Module. Therefore, there is a trade-off between the maximum amount of time a same shared secret is stored and the frequency of its re-computing.

Note that, even if an endpoint is authorized to be a group member and to take part in group communications, there is a risk that it behaves inappropriately. For instance, it can forward the content of messages in the group 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), which results in a

practically limited risk and enables a prompt detection/reaction in case of misbehaving.

10.2. Uniqueness of (key, nonce)

The proof for uniqueness of (key, nonce) pairs in Appendix D.4 of [RFC8613] is also valid in group communication scenarios. That is, given an OSCORE group:

- o Uniqueness of Sender IDs within the group is enforced by the Group Manager, which never reassigns the same Sender ID within the same group.
- o The case A in Appendix D.4 of [RFC8613] concerns all group requests and responses including a Partial IV (e.g. Observe notifications). In this case, same considerations from [RFC8613] apply here as well.
- o The case B in Appendix D.4 of [RFC8613] concerns responses not including a Partial IV (e.g. single response to a group request). In this case, same considerations from [RFC8613] apply here as well.

As a consequence, each message encrypted/decrypted with the same Sender Key is processed by using a different (ID_PIV, PIV) pair. This means that nonces used by any fixed encrypting endpoint are unique. Thus, each message is processed with a different (key, nonce) pair.

10.3. Management of Group Keying Material

The approach described in this specification should take into account the risk of compromise of group members. In particular, this document specifies that a key management scheme for secure revocation and renewal of Security Contexts and group keying material should be adopted.

[I-D.ietf-ace-key-groupcomm-oscore] provides a simple rekeying scheme for renewing the Security Context in a group.

Alternative rekeying schemes which are more scalable with the group size may be needed in dynamic, large-scale groups where endpoints can join and leave at any time, in order to limit the impact on performance due to the Security Context and keying material update.

10.4. Update of Security Context and Key Rotation

A group member can receive a message shortly after the group has been rekeyed, and new security parameters and keying material have been distributed by the Group Manager.

This may result in a client using an old Security Context to protect a group request, and a server using a different new Security Context to protect a corresponding response. As a consequence, clients may receive a response protected with a Security Context different from the one used to protect the corresponding group request.

In particular, a server may first get a group request protected with the old Security Context, then install the new Security Context, and only after that produce a response to send back to the client. In such a case, as specified in Section 8.3, the server **MUST** protect the potential response using the new Security Context. Specifically, the server **MUST** include its Sender Sequence Number as Partial IV in the response and use it to build the AEAD nonce to protect the response. This prevents the AEAD nonce from the request from being reused with the new Security Context.

The client will process that response using the new Security Context, provided that it has installed the new security parameters and keying material before the message processing.

In case block-wise transfer [RFC7959] is used, the same considerations from Section 7.2 of [I-D.ietf-ace-key-groupcomm] hold.

Furthermore, as described below, a group rekeying may temporarily result in misaligned Security Contexts between the sender and recipient of a same message.

10.4.1. Late Update on the Sender

In this case, the sender protects a message using the old Security Context, i.e. before having installed the new Security Context. However, the recipient receives the message after having installed the new Security Context, and is thus unable to correctly process it.

A possible way to ameliorate this issue is to preserve the old, recent, Security Context for a maximum amount of time defined by the application. By doing so, the recipient can still try to process the received message using the old retained Security Context as second attempt. This makes particular sense when the recipient is a client, that would hence be able to process incoming responses protected with the old, recent, Security Context used to protect the associated group request. Instead, a recipient server would better and more

simply discard an incoming group request which is not successfully processed with the new Security Context.

This tolerance preserves the processing of secure messages throughout a long-lasting key rotation, as group rekeying processes may likely take a long time to complete, especially in large scale groups. On the other hand, a former (compromised) group member can abusively take advantage of this, and send messages protected with the old retained Security Context. Therefore, a conservative application policy should not admit the retention of old Security Contexts.

10.4.2. Late Update on the Recipient

In this case, the sender protects a message using the new Security Context, but the recipient receives that message before having installed the new Security Context. Therefore, the recipient would not be able to correctly process the message and hence discards it.

If the recipient installs the new Security Context shortly after that and the sender endpoint retransmits the message, the former will still be able to receive and correctly process the message.

In any case, the recipient should actively ask the Group Manager for an updated Security Context according to an application-defined policy, for instance after a given number of unsuccessfully decrypted incoming messages.

10.5. Collision of Group Identifiers

In case endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible for Group Identifiers of different groups to coincide.

This does not impair the security of the AEAD algorithm. In fact, as long as the Master Secret is different for different groups and this condition holds over time, AEAD keys are different among different groups.

The entity assigning an IP multicast address may help limiting the chances to experience such collisions of Group Identifiers. In particular, it may allow the Group Managers of groups using the same IP multicast address to share their respective list of assigned Group Identifiers currently in use.

10.6. Cross-group Message Injection

A same endpoint is allowed to and would likely use the same public/private key pair in multiple OSCORE groups, possibly administered by different Group Managers.

When a sender endpoint sends a message protected in pairwise mode to a recipient endpoint in an OSCORE group, a malicious group member may attempt to inject the message to a different OSCORE group also including the same endpoints (see Section 10.6.1).

This practically relies on altering the content of the OSCORE option, and having the same MAC in the ciphertext still correctly validating, which has a success probability depending on the size of the MAC.

As discussed in Section 10.6.2, the attack is practically infeasible if the message is protected in group mode, since the counter signature is bound also to the OSCORE option, through the Additional Authenticated Data used in the signing process (see Section 4.3.2).

10.6.1. Attack Description

Let us consider:

- o Two OSCORE groups G1 and G2, with ID Context (Group ID) Gid1 and Gid2, respectively. Both G1 and G2 use the AEAD cipher AES-CCM-16-64-128, i.e. the MAC of the ciphertext is 8 bytes in size.
- o A sender endpoint X which is member of both G1 and G2, and uses the same public/private key pair in both groups. The endpoint X has Sender ID Sid1 in G1 and Sender ID Sid2 in G2. The pairs (Sid1, Gid1) and (Sid2, Gid2) identify the same public key of X in G1 and G2, respectively.
- o A recipient endpoint Y which is member of both G1 and G2, and uses the same public/private key pair in both groups. The endpoint Y has Sender ID Sid3 in G1 and Sender ID Sid4 in G2. The pairs (Sid3, Gid1) and (Sid4, Gid2) identify the same public key of Y in G1 and G2, respectively.
- o A malicious endpoint Z is also member of both G1 and G2. Hence, Z is able to derive the symmetric keys associated to X in G1 and G2.

When X sends a message M1 addressed to Y in G1 and protected in pairwise mode, Z can intercept M1, and forge a valid message M2 to be injected in G2, making it appear as still sent by X to Y and valid to be accepted.

More in detail, Z intercepts and stops message M1, and forges a message M2 by changing the value of the OSCORE option from M1 as follows: the 'kid context' is changed from G1 to G2; and the 'kid' is changed from Sid1 to Sid2. Then, Z injects message M2 as addressed to Y in G2.

Upon receiving M2, there is a probability equal to 2^{-64} that Y successfully verifies the same unchanged MAC by using Sid2 as 'request_kid' and using the Pairwise Recipient Key associated to X in G2.

Note that Z does not know the pairwise keys of X and Y, since it does not know and is not able to compute their shared Diffie-Hellman secret. Therefore, Z is not able to check offline if a performed forgery is actually valid, before sending the forged message to G2.

10.6.2. Attack Prevention in Group Mode

When a Group OSCORE message is protected with the group mode, the counter signature is computed also over the value of the OSCORE option, which is part of the Additional Authenticated Data used in the signing process (see Section 4.3.2).

That is, the countersignature is computed also over: the ID Context (Group ID) and the Partial IV, which are always present in group requests; as well as the Sender ID of the message originator, which is always present in all group requests and responses.

Since the signing process takes as input also the ciphertext of the COSE_Encrypt0 object, the countersignature is bound not only to the intended OSCORE group, hence to the triplet (Master Secret, Master Salt, ID Context), but also to a specific Sender ID in that group and to its specific symmetric key used for AEAD encryption, hence to the quartet (Master Secret, Master Salt, ID Context, Sender ID).

This makes it practically infeasible to perform the attack described in Section 10.6.1, since it would require the adversary to additionally forge a valid countersignature that replaces the original one in the forged message M2.

If the countersignature did not cover the OSCORE option, the attack would be possible also in group mode, since the same unchanged countersignature from message M1 would be also valid in message M2. Also, the following attack simplifications would hold, since Z is able to derive the Sender/Recipient Keys of X and Y in G1 and G2.

- o If M2 is used as a request, Z can check offline if a performed forgery is actually valid before sending the forged message to G2.

That is, this attack would have a complexity of 2^{64} offline calculations.

- o If M2 is used as a response, Z can also change the response Partial IV, until the same unchanged MAC is successfully verified by using Sid2 as 'request_kid' and the symmetric key associated to X in G2. Since the Partial IV is 5 bytes in size, this requires 2^{40} operations to test all the Partial IVs, which can be done in real-time. Also, the probability that a single given message M1 can be used to forge a response M2 for a given request would be equal to 2^{-24} , since there are more MAC values (8 bytes in size) than Partial IV values (5 bytes in size).

Note that, by changing the Partial IV as discussed above, any member of G1 would also be able to forge a valid signed response message M2 to be injected in G1.

10.7. Group OSCORE for Unicast Requests

If a request is intended to be sent over unicast as addressed to a single group member, it is NOT RECOMMENDED for the client to protect the request by using the group mode as defined in Section 8.1.

This does not include the case where the client sends a request over unicast to a proxy, to be forwarded to multiple intended recipients over multicast [I-D.ietf-core-groupcomm-bis]. In this case, the client MUST protect the request with the group mode, even though it is sent to the proxy over unicast (see Section 8).

If the client uses the group mode with its own Sender Key to protect a unicast request to a group member, an on-path adversary can, right then or later on, redirect that request to one/many different group member(s) over unicast, or to the whole OSCORE group over multicast. By doing so, the adversary can induce the target group member(s) to perform actions intended for one group member only. Note that the adversary can be external, i.e. (s)he does not need to also be a member of the OSCORE group.

This is due to the fact that the client is not able to indicate the single intended recipient in a way which is secure and possible to process for Group OSCORE on the server side. In particular, Group OSCORE does not protect network addressing information such as the IP address of the intended recipient server. It follows that the server(s) receiving the redirected request cannot assert whether that was the original intention of the client, and would thus simply assume so.

The impact of such an attack depends especially on the REST method of the request, i.e. the Inner CoAP Code of the OSCORE request message. In particular, safe methods such as GET and FETCH would trigger (several) unintended responses from the targeted server(s), while not resulting in destructive behavior. On the other hand, non safe methods such as PUT, POST and PATCH/iPATCH would result in the target server(s) taking active actions on their resources and possible cyber-physical environment, with the risk of destructive consequences and possible implications for safety.

A client can instead use the pairwise mode as defined in Section 9.2, in order to protect a request sent to a single group member by using pairwise keying material (see Section 2.3). This prevents the attack discussed above by construction, as only the intended server is able to derive the pairwise keying material used by the client to protect the request. A client supporting the pairwise mode SHOULD use it to protect requests sent to a single group member over unicast, instead of using the group mode. For an example where this is not fulfilled, see Section 6.2.1 in [I-D.tiloca-core-observe-multicast-notifications].

With particular reference to block-wise transfers [RFC7959], Section 2.3.6 of [I-D.ietf-core-groupcomm-bis] points out that, while an initial request including the CoAP Block2 option can be sent over multicast, any other request in a transfer has to occur over unicast, individually addressing the servers in the group.

Additional considerations are discussed in Appendix E.3, with respect to requests including a CoAP Echo Option [I-D.ietf-core-echo-request-tag] that has to be sent over unicast, as a challenge-response method for servers to achieve synchronization of clients' Sender Sequence Number.

10.8. End-to-end Protection

The same considerations from Section 12.1 of [RFC8613] hold for Group OSCORE.

Additionally, (D)TLS and Group OSCORE can be combined for protecting message exchanges occurring over unicast. However, it is not possible to combine (D)TLS and Group OSCORE for protecting message exchanges where messages are (also) sent over multicast.

10.9. Master Secret

Group OSCORE derives the Security Context using the same construction as OSCORE, and by using the Group Identifier of a group as the related ID Context. Hence, the same required properties of the

Security Context parameters discussed in Section 3.3 of [RFC8613] hold for this document.

With particular reference to the OSCORE Master Secret, it has to be kept secret among the members of the respective OSCORE group and the Group Manager responsible for that group. Also, the Master Secret must have a good amount of randomness, and the Group Manager can generate it offline using a good random number generator. This includes the case where the Group Manager rekeys the group by generating and distributing a new Master Secret. Randomness requirements for security are described in [RFC4086].

10.10. Replay Protection

As in OSCORE, also Group OSCORE relies on sender sequence numbers included in the COSE message field 'Partial IV' and used to build AEAD nonces.

Note that the Partial IV of an endpoint does not necessarily grow monotonically. For instance, upon exhaustion of the endpoint Sender Sequence Number, the Partial IV also gets exhausted. As discussed in Section 2.4.3, this results either in the endpoint being individually rekeyed and getting a new Sender ID, or in the establishment of a new Security Context in the group. Therefore, uniqueness of (key, nonce) pairs (see Section 10.2) is preserved also when a new Security Context is established.

As discussed in Section 6.1, an endpoint that has just joined a group is exposed to replay attack, as it is not aware of the Sender Sequence Numbers currently used by other group members. Appendix E describes how endpoints can synchronize with others' Sender Sequence Number.

Unless exchanges in a group rely only on unicast messages, Group OSCORE cannot be used with reliable transport. Thus, unless only unicast messages are sent in the group, it cannot be defined that only messages with sequence numbers that are equal to the previous sequence number + 1 are accepted.

The processing of response messages described in Section 2.3.1 of [I-D.ietf-core-groupcomm-bis] also ensures that a client accepts a single valid response to a given request from each replying server, unless CoAP observation is used.

10.11. Client Aliveness

As discussed in Section 12.5 of [RFC8613], a server may use the CoAP Echo Option [I-D.ietf-core-echo-request-tag] to verify the aliveness of the client that originated a received request. This would also allow the server to (re-)synchronize with the client's Sender Sequence Number, as well as to ensure that the request is fresh and has not been replayed or (purposely) delayed, if it is the first one received from that client after having joined the group or rebooted (see Appendix E.3).

10.12. Cryptographic Considerations

The same considerations from Section 12.6 of [RFC8613] about the maximum Sender Sequence Number hold for Group OSCORE.

As discussed in Section 2.4.2, an endpoint that experiences an exhaustion of its own Sender Sequence Numbers MUST NOT protect further messages including a Partial IV, until it has derived a new Sender Context. This prevents the endpoint to reuse the same AEAD nonces with the same Sender Key.

In order to renew its own Sender Context, the endpoint SHOULD inform the Group Manager, which can either renew the whole Security Context by means of group rekeying, or provide only that endpoint with a new Sender ID value. In either case, the endpoint derives a new Sender Context, and in particular a new Sender Key.

Additionally, the same considerations from Section 12.6 of [RFC8613] hold for Group OSCORE, about building the AEAD nonce and the secrecy of the Security Context parameters.

The EdDSA signature algorithm and the elliptic curve Ed25519 [RFC8032] are mandatory to implement. For endpoints that support the pairwise mode, the ECDH-SS + HKDF-256 algorithm specified in Section 6.3.1 of [I-D.ietf-cose-rfc8152bis-algs] and the X25519 curve [RFC7748] are also mandatory to implement.

Constrained IoT devices may alternatively represent Montgomery curves and (twisted) Edwards curves [RFC7748] in the short-Weierstrass form Wei25519, with which the algorithms ECDSA25519 and ECDH25519 can be used for signature operations and Diffie-Hellman secret calculation, respectively [I-D.ietf-lwig-curve-representations].

For many constrained IoT devices, it is problematic to support more than one signature algorithm or multiple whole cipher suites. As a consequence, some deployments using, for instance, ECDSA with NIST

P-256 may not support the mandatory signature algorithm but that should not be an issue for local deployments.

The derivation of pairwise keys defined in Section 2.3.1 is compatible with ECDSA and EdDSA asymmetric keys, but is not compatible with RSA asymmetric keys. The security of using the same key pair for Diffie-Hellman and for signing is demonstrated in [Degabriele].

10.13. Message Segmentation

The same considerations from Section 12.7 of [RFC8613] hold for Group OSCORE.

10.14. Privacy Considerations

Group OSCORE ensures end-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations. In particular, options are processed according to the same class U/I/E that they have for OSCORE. Therefore, the same privacy considerations from Section 12.8 of [RFC8613] hold for Group OSCORE.

Furthermore, the following privacy considerations hold, about the OSCORE option that may reveal information on the communicating endpoints.

- o The 'kid' parameter, which is intended to help a recipient endpoint to find the right Recipient Context, may reveal information about the Sender Endpoint. Since both requests and responses always include the 'kid' parameter, this may reveal information about both a client sending a group request and all the possibly replying servers sending their own individual response.
- o The 'kid context' parameter, which is intended to help a recipient endpoint to find the right Security Context, reveals information about the sender endpoint. In particular, it reveals that the sender endpoint is a member of a particular OSCORE group, whose current Group ID is indicated in the 'kid context' parameter.

When receiving a group request, each of the recipient endpoints can reply with a response that includes its Sender ID as 'kid' parameter. All these responses will be matchable with the request through the Token. Thus, even if these responses do not include a 'kid context' parameter, it becomes possible to understand that the responder endpoints are in the same group of the requester endpoint.

Furthermore, using the mechanisms described in Appendix E.3 to achieve sequence number synchronization with a client may reveal when a server device goes through a reboot. This can be mitigated by the server device storing the precise state of the replay window of each known client on a clean shutdown.

Finally, the mechanism described in Section 10.5 to prevent collisions of Group Identifiers from different Group Managers may reveal information about events in the respective OSCORE groups. In particular, a Group Identifier changes when the corresponding group is rekeyed. Thus, Group Managers might use the shared list of Group Identifiers to infer the rate and patterns of group membership changes triggering a group rekeying, e.g. due to newly joined members or evicted (compromised) members. In order to alleviate this privacy concern, it should be hidden from the Group Managers which exact Group Manager has currently assigned which Group Identifiers in its OSCORE groups.

11. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[This Document]" with the RFC number of this specification and delete this paragraph.

This document has the following actions for IANA.

11.1. OSCORE Flag Bits Registry

IANA is asked to add the following value entry to the "OSCORE Flag Bits" subregistry defined in Section 13.7 of [RFC8613] as part of the "CoRE Parameters" registry.

Bit Position	Name	Description	Reference
2	Group Flag	Set to 1 if the message is protected with the group mode of Group OSCORE	[This Document]

12. References

12.1. Normative References

[COSE.Algorithms]
IANA, "COSE Algorithms",
<<https://www.iana.org/assignments/cose/cose.xhtml#algorithms>>.

- [COSE.Key.Types]
IANA, "COSE Key Types",
<<https://www.iana.org/assignments/cose/cose.xhtml#key-type>>.
- [I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work in progress), September 2020.
- [I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-ietf-core-groupcomm-bis-02 (work in progress), November 2020.
- [I-D.ietf-cose-countersign]
Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", draft-ietf-cose-countersign-01 (work in progress), October 2020.
- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", draft-ietf-cose-rfc8152bis-algs-12 (work in progress), September 2020.
- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", draft-ietf-cose-rfc8152bis-struct-14 (work in progress), September 2020.
- [NIST-800-56A]
Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography - NIST Special Publication 800-56A, Revision 3", April 2018, <<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf>>.
- [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>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.

- [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>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [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>>.
- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

12.2. Informative References

- [Degabriele]
Degabriele, J., Lehmann, A., Paterson, K., Smart, N., and M. Stremler, "On the Joint Security of Encryption and Signature in EMV", December 2011, <<https://eprint.iacr.org/2011/615>>.
- [I-D.ietf-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-ietf-ace-key-groupcomm-10 (work in progress), November 2020.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-09 (work in progress), November 2020.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-35 (work in progress), June 2020.

- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-10 (work in progress), July 2020.
- [I-D.ietf-lwig-curve-representations]
Struik, R., "Alternative Elliptic Curve Representations", draft-ietf-lwig-curve-representations-12 (work in progress), August 2020.
- [I-D.ietf-lwig-security-protocol-comparison]
Mattsson, J., Palombini, F., and M. Vucinic, "Comparison of CoAP Security Protocols", draft-ietf-lwig-security-protocol-comparison-04 (work in progress), March 2020.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-38 (work in progress), May 2020.
- [I-D.mattsson-cfrg-det-sigs-with-noise]
Mattsson, J., Thormarker, E., and S. Ruohomaa, "Deterministic ECDSA and EdDSA Signatures with Additional Randomness", draft-mattsson-cfrg-det-sigs-with-noise-02 (work in progress), March 2020.
- [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-core-observe-multicast-notifications]
Tiloca, M., Hoeglund, R., Amsuess, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", draft-tiloca-core-observe-multicast-notifications-04 (work in progress), November 2020.
- [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>>.
- [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>>.
- [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>>.

Appendix A. Assumptions and Security Objectives

This section presents a set of assumptions and security objectives for the approach described in this document. The rest of this section refers to three types of groups:

- o Application group, i.e. a set of CoAP endpoints that share a common pool of resources.
- o Security group, as defined in Section 1.1 of this specification. There can be a one-to-one or a one-to-many relation between security groups and application groups, and vice versa.
- o CoAP group, as defined in [I-D.ietf-core-groupcomm-bis] i.e. a set of CoAP endpoints, where each endpoint is configured to receive CoAP multicast requests that are sent to the group's associated IP multicast address and UDP port. An endpoint may be a member of multiple CoAP groups. There can be a one-to-one or a one-to-many relation between application groups and CoAP groups. Note that a device sending a CoAP request to a CoAP group is not necessarily itself a member of that group: it is a member only if it also has a CoAP server endpoint listening to requests for this CoAP group, sent to the associated IP multicast address and port. In order to provide secure group communication, all members of a CoAP group as

well as all further endpoints configured only as clients sending CoAP (multicast) requests to the CoAP group have to be member of a security group. There can be a one-to-one or a one-to-many relation between security groups and CoAP groups, and vice versa.

A.1. Assumptions

The following assumptions are assumed to be already addressed and are out of the scope of this document.

- o Multicast communication topology: this document considers both 1-to-N (one sender and multiple recipients) and M-to-N (multiple senders and multiple recipients) 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.

In a 1-to-N communication model, only a single client transmits data to the CoAP group, in the form of request messages; in an M-to-N communication model (where M and N do not necessarily have the same value), M clients transmit data to the CoAP group. According to [I-D.ietf-core-groupcomm-bis], any possible proxy entity is supposed to know about the clients and to not perform aggregation of response messages from multiple servers. Also, every client expects and is able to handle multiple response messages associated to a same request sent to the CoAP group.

- o Group size: security solutions for group communication should be able to adequately support different and possibly large security groups. The group size is the current number of members in a security group. In the use cases mentioned in this document, the number of clients (normally the controlling devices) is expected to be much smaller than the number of servers (i.e. the controlled devices). A security solution for group communication that supports 1 to 50 clients 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. Security groups larger than that should be divided into smaller independent groups.
- o Communication with the Group Manager: an endpoint must use a secure dedicated channel when communicating with the Group Manager, also when not registered as a member of the security group.
- o Provisioning and management of Security Contexts: a Security Context must be established among the members of the security

group. A secure mechanism must be used to generate, revoke and (re-)distribute keying material, communication policies and security parameters in the security group. The actual provisioning and management of the Security Context is out of the scope of this document.

- o Multicast data security ciphersuite: all members of a security group 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.
- o Backward security: a new device joining the security group should not have access to any old Security Contexts used before its joining. This ensures that a new member of the security group is not able to decrypt confidential data sent before it has joined the security 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 in the security group upon a new member's joining has to be defined as part of the group key management scheme.
- o Forward security: entities that leave the security group should not have access to any future Security Contexts or message exchanged within the security group after their leaving. This ensures that a former member of the security group is not able to decrypt confidential data sent within the security group anymore. Also, it ensures that a former member is not able to send protected messages to the security group anymore. The actual mechanism to update the Security Context and renew the group keying material in the security group upon a 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:

- o Data replay protection: group request messages or response messages replayed within the security group must be detected.
- o Data confidentiality: messages sent within the security group shall be encrypted.
- o Group-level data confidentiality: the group mode provides 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 security group, but not by an external adversary or other external entities.

- o Pairwise data confidentiality: the pairwise mode especially provides pairwise data confidentiality, since messages are encrypted using pairwise keying material shared between any two group members, hence they can be decrypted only by the intended single recipient.
- o Source message authentication: messages sent within the security group shall be authenticated. That is, it is essential to ensure that a message is originated by a member of the security group in the first place, and in particular by a specific, identifiable member of the security group.
- o Message integrity: messages sent within the security group shall be integrity protected. That is, it is essential to ensure that a message has not been tampered with, either by a group member, or by an external adversary or other external entities which are not members of the security group.
- o Message ordering: it must be possible to determine the ordering of messages coming from a single sender. In accordance with OSCORE [RFC8613], this results in providing absolute freshness of responses that are not notifications, as well as relative freshness of group requests and notification responses. It is not required to determine ordering of messages from different senders.

Appendix B. List of Use Cases

Group Communication for CoAP [I-D.ietf-core-groupcomm-bis] 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 [I-D.ietf-core-groupcomm-bis] to understand the non-security related details. This section discusses a number of use cases that benefit from secure group communication, and refers to the three types of groups from Appendix A. Specific security requirements for these use cases are discussed in Appendix A.

- o Lighting control: consider a building equipped with IP-connected lighting devices, switches, and border routers. The lighting devices acting as servers are organized into application groups and CoAP groups, according to their physical location in the building. For instance, lighting devices in a room or corridor can be configured as members of a single application group and corresponding CoAP group. Those lighting devices together with the switches acting as clients in the same room or corridor can be

configured as members of the corresponding security group. Switches are then used to control the lighting devices by sending on/off/dimming commands to all lighting devices in the CoAP 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 with a role in the lighting application 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 set of connected lights, ensuring that the light preset (e.g. dimming level or color) of a large set 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 set 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 set of lighting devices. Especially in professional lighting scenarios, the roles of client and server are configured by the lighting commissioner, and devices strictly follow those roles.

- o 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 application groups and CoAP 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 application group and corresponding CoAP 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.
- o Software and firmware updates: software and firmware updates often comprise quite a large amount of data. This can overload a Low-power and Lossy Network (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 set 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.

- o Parameter and configuration update: by means of multicast communication, it is possible to update the settings of a set 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.
- o Commissioning of Low-power and Lossy Network (LLN) 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 application group and corresponding CoAP 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.
- o 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 latter 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 Multicast Protocol for Low-Power and Lossy Networks (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.

For each group, the Group Prefix is constant over time and 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. 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 each time new keying material, together with a new Gid, is distributed to the group in order to establish a new Security Context (see Section 3.1).

As an example, a 3-byte Gid 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 Common Context 61532 times in the group, its Gid will assume value '0xb1f05c'.

Using an immutable Group Prefix for a group assumes that enough time elapses before all possible Group Epoch values are used, since the Group Manager does not reassign the same Gid to the same group. Thus, the expected highest rate for addition/removal of group members and consequent group rekeying should be taken into account for a proper dimensioning of the Group Epoch size.

As discussed in Section 10.5, if endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible that Group Identifiers of different groups coincide at some point in time. In this case, a recipient has to handle coinciding Group Identifiers, and has to try using different Security Contexts to process an incoming message, until the right one is found and the message is correctly verified. Therefore, it is favourable that Group Identifiers from different Group Managers have a size that result in a small probability of collision. How small this probability should be is up to system designers.

Appendix D. Set-up of New Endpoints

An endpoint joins a group by explicitly interacting with the responsible Group Manager. When becoming members of a group, endpoints are not required to know how many and what endpoints are in the same group.

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 the scope of this document.

The Group Manager must verify that the joining endpoint is authorized to join 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. Further details about the authorization of joining endpoints are out of scope.

In case of successful authorization check, the Group Manager generates a Sender ID assigned to the joining endpoint, before proceeding with the rest of the join process. That is, the Group Manager provides the joining endpoint with the keying material and parameters to initialize the Security Context (see Section 2). The actual provisioning of keying material and parameters to the joining endpoint is out of the scope of this document.

It is RECOMMENDED that the join process adopts the approach described in [I-D.ietf-ace-key-groupcomm-oscure] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

Appendix E. Examples of Synchronization Approaches

This section describes three possible approaches that can be considered by server endpoints to synchronize with Sender Sequence Numbers of client endpoints sending group requests.

The Group Manager MAY indicate which of such approaches are used in the group, as part of the group communication policies signalled to candidate group members upon their group joining.

If a server has recently lost the mutable Security Context, e.g. due to a reboot, the server has also to establish an updated Security Context before resuming to send protected messages to the group (see Section 2.4.1). Since this results in deriving a new Sender Key for its Sender Context, the server does not reuse the same pair (key, nonce), even when using the Partial IV of (old re-injected) requests to build the AEAD nonce for protecting the corresponding responses.

E.1. Best-Effort Synchronization

Upon receiving a group request from a client, a server does not take any action to synchronize with the Sender Sequence Number of that client. This provides no assurance at all as to message freshness, which can be acceptable in non-critical use cases.

With the notable exception of Observe notifications and responses following a group rekeying, it is optional for the server to use its Sender Sequence Number as Partial IV when protecting a response. Instead, for efficiency reasons, the server may rather use the request's Partial IV when protecting a response to that request.

E.2. Baseline Synchronization

Upon receiving a group request from a given client for the first time, a server initializes the last-seen Sender Sequence Number associated to that client in its corresponding Recipient Context. The server may also drop the group request without delivering it to the application. This method provides a reference point to identify if future group requests from the same client are fresher than the last one received.

A replay time interval exists, between when a possibly replayed or delayed message is originally transmitted by a given client and the first authentic fresh message from that same client is received. This can be acceptable for use cases where servers admit such a trade-off between performance and assurance of message freshness.

With the notable exception of Observe notifications and responses following a group rekeying, it is optional for the server to use its Sender Sequence Number as Partial IV when protecting a response. Instead, for efficiency reasons, the server may rather use the request's Partial IV when protecting a response to that request.

E.3. Challenge-Response Synchronization

A server performs a challenge-response exchange with a client, by using the Echo Option for CoAP described in Section 2 of [I-D.ietf-core-echo-request-tag] and according to Appendix B.1.2 of [RFC8613].

That is, upon receiving a group request from a particular client for the first time, the server processes the message as described in this specification, but, even if valid, does not deliver it to the application. Instead, the server replies to the client with an OSCORE protected 4.01 (Unauthorized) response message, including only the Echo Option and no diagnostic payload. The server MUST NOT set the Echo Option to a value which is both predictable and reusable. Since this response is protected with the Security Context used in the group, the client will consider the response valid upon successfully decrypting and verifying it.

The server stores the Echo Option value included therein, together with the pair (gid,kid), where 'gid' is the Group Identifier of the

OSCORE group and 'kid' is the Sender ID of the client in the group, as specified in the 'kid context' and 'kid' fields of the OSCORE Option of the group request, respectively. After a group rekeying has been completed and a new Security Context has been established in the group, which results also in a new Group Identifier (see Section 3.1), the server MUST delete all the stored Echo values associated to members of that group.

Upon receiving a 4.01 (Unauthorized) response that includes an Echo Option and originates from a verified group member, a client sends a request as a unicast message addressed to the same server, echoing the Echo Option value. The client MUST NOT send the request including the Echo Option over multicast.

In particular, the client 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 client to not retain previously sent group requests for full retransmission, unless the application explicitly requires otherwise. In either case, the client uses a fresh Sender Sequence Number value from its own Sender Context. If the client stores group requests for possible retransmission with the Echo Option, it should not store a given request for longer than a pre-configured time interval. Note that the unicast request echoing the Echo Option is correctly treated and processed as a message, since the 'kid context' field including the Group Identifier of the OSCORE group is still present in the OSCORE Option as part of the COSE object (see Section 4).

Upon receiving the unicast request including the Echo Option, the server performs the following verifications.

- o If the server does not store an Echo Option value for the pair (gid,kid), it considers: i) the time t_1 when it has established the Security Context used to protect the received request; and ii) the time t_2 when the request has been received. Since a valid request cannot be older than the Security Context used to protect it, the server verifies that $(t_2 - t_1)$ is less than the largest amount of time acceptable to consider the request fresh.
- o If the server stores an Echo Option value for the pair (gid,kid) associated to that same client in the same group, the server verifies that the option value equals that same stored value previously sent to that client.

If the verifications above fail, the server MUST NOT process the request further and MAY send a 4.01 (Unauthorized) response including an Echo Option.

In case of positive verification, the request is further processed and verified. Finally, the server updates the Recipient Context associated to that client, by setting the Replay Window according to the Sender Sequence Number from the unicast request conveying the Echo Option. The server 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.

A server should not deliver group requests from a given client to the application until one valid request from that same client has been verified as fresh, as conveying an echoed Echo Option [I-D.ietf-core-echo-request-tag]. Also, a server may perform the challenge-response described above at any time, if synchronization with Sender Sequence Numbers of clients is (believed to be) lost, for instance after a device reboot. A client has to be always ready to perform the challenge-response based on the Echo Option in case a server starts it.

It is the role of the server application to define under what circumstances Sender Sequence Numbers lose synchronization. This can include experiencing a "large enough" gap $D = (SN2 - SN1)$, between the Sender Sequence Number $SN1$ of the latest accepted group request from a client and the Sender Sequence Number $SN2$ of a group request just received from that client. However, a client may send several unicast requests to different group members as protected with the pairwise mode (see Section 9.2), which may result in the server experiencing the gap D in a relatively short time. This would induce the server to perform more challenge-response exchanges than actually needed.

To ameliorate this, the server may rather rely on a trade-off between the Sender Sequence Number gap D and a time gap $T = (t2 - t1)$, where $t1$ is the time when the latest group request from a client was accepted and $t2$ is the time when the latest group request from that client has been received, respectively. Then, the server can start a challenge-response when experiencing a time gap T larger than a given, pre-configured threshold. Also, the server can start a challenge-response when experiencing a Sender Sequence Number gap D greater than a different threshold, computed as a monotonically increasing function of the currently experienced time gap T .

The challenge-response approach described in this appendix 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.

Note that endpoints configured as silent servers are not able to perform the challenge-response described above, as they do not store a Sender Context to secure the 4.01 (Unauthorized) response to the client. Therefore, silent servers should adopt alternative approaches to achieve and maintain synchronization with sender sequence numbers of clients.

Since requests including the Echo Option are sent over unicast, a server can be a victim of the attack discussed in Section 10.7, when such requests are protected with the group mode of Group OSCORE, as described in Section 8.1.

Instead, protecting requests with the Echo Option by using the pairwise mode of Group OSCORE as described in Section 9.2 prevents the attack in Section 10.7. In fact, only the exact server involved in the Echo exchange is able to derive the correct pairwise key used by the client to protect the request including the Echo Option.

In either case, an internal on-path adversary would not be able to mix up the Echo Option value of two different unicast requests, sent by a same client to any two different servers in the group. In fact, if the group mode was used, this would require the adversary to forge the client's counter signature in both such requests. As a consequence, each of the two servers remains able to selectively accept a request with the Echo Option only if it is waiting for that exact integrity-protected Echo Option value, and is thus the intended recipient.

Appendix F. No Verification of Signatures in Group Mode

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 messages protected with the group mode. 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 a message protected with the group mode, 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 messages protected with the

group mode. However, it is recognized that there may be situations where it is not always required. The consequence of not doing the signature validation in messages protected with the group mode 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 would have evidence that the 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.

Appendix G. Example Values with COSE Capabilities

The table below provides examples of values for Counter Signature Parameters in the Common Context (see Section 2.1.3), for different values of Counter Signature Algorithm.

Counter Signature Algorithm	Example Values for Counter Signature Parameters
(-8) // EdDSA	[1], [1, 6] // 1: OKP ; 1: OKP, 6: Ed25519
(-8) // EdDSA	[1], [1, 6] // 1: OKP ; 1: OKP, 7: Ed448
(-7) // ES256	[2], [2, 1] // 2: EC2 ; 2: EC2, 1: P-256
(-35) // ES384	[2], [2, 2] // 2: EC2 ; 2: EC2, 2: P-384
(-36) // ES512	[2], [2, 3] // 2: EC2 ; 2: EC2, 3: P-512
(-37) // PS256	[], [3] // empty ; 3: RSA
(-38) // PS384	[], [3] // empty ; 3: RSA
(-39) // PS512	[], [3] // empty ; 3: RSA

Figure 4: Examples of Counter Signature Parameters

The table below provides examples of values for Secret Derivation Parameters in the Common Context (see Section 2.1.5), for different values of Secret Derivation Algorithm.

Secret Derivation Algorithm	Example Values for Secret Derivation Parameters
(-27) // ECDH-SS // + HKDF-256	[1], [1, 6] // 1: OKP ; 1: OKP, 4: X25519
(-27) // ECDH-SS // + HKDF-256	[1], [1, 6] // 1: OKP ; 1: OKP, 5: X448
(-27) // ECDH-SS // + HKDF-256	[2], [2, 1] // 2: EC2 ; 2: EC2, 1: P-256
(-27) // ECDH-SS // + HKDF-256	[2], [2, 2] // 2: EC2 ; 2: EC2, 2: P-384
(-27) // ECDH-SS // + HKDF-256	[2], [2, 3] // 2: EC2 ; 2: EC2, 3: P-512

Figure 5: Examples of Secret Derivation Parameters

The table below provides examples of values for the 'par_countersign_key' element of the 'algorithms' array used in the two external_aad structures (see Section 4.3.1 and Section 4.3.2), for different values of Counter Signature Algorithm.

Counter Signature Algorithm	Example Values for 'par_countersign_key'
(-8) // EdDSA	[1, 6] // 1: OKP , 6: Ed25519
(-8) // EdDSA	[1, 6] // 1: OKP , 7: Ed448
(-7) // ES256	[2, 1] // 2: EC2 , 1: P-256
(-35) // ES384	[2, 2] // 2: EC2 , 2: P-384
(-36) // ES512	[2, 3] // 2: EC2 , 3: P-512
(-37) // PS256	[3] // 3: RSA
(-38) // PS384	[3] // 3: RSA
(-39) // PS512	[3] // 3: RSA

Figure 6: Examples of 'par_countersign_key'

Appendix H. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

H.1. Version -09 to -10

- o Removed 'Counter Signature Key Parameters' from the Common Context.

- o New parameters in the Common Context covering the DH secret derivation.
- o New counter signature header parameter from draft-ietf-cose-countersign.
- o Stronger policies non non-recycling of Sender IDs and Gid.
- o The Sender Sequence Number is reset when establishing a new Security Context.
- o Added 'request_kid_context' in the aad_array.
- o The server can respond with 5.03 if the client's public key is not available.
- o The observer client stores an invariant identifier of the group.
- o Relaxed storing of original 'kid' for observer clients.
- o Both client and server store the 'kid_context' of the original observation request.
- o The server uses a fresh PIV if protecting the response with a Security Context different from the one used to protect the request.
- o Clarifications on MTI algorithms and curves.
- o Removed optimized requests.
- o Overall clarifications and editorial revision.

H.2. Version -08 to -09

- o Pairwise keys are discarded after group rekeying.
- o Signature mode renamed to group mode.
- o The parameters for countersignatures use the updated COSE registries. Newly defined IANA registries have been removed.
- o Pairwise Flag bit renamed as Group Flag bit, set to 1 in group mode and set to 0 in pairwise mode.
- o Dedicated section on updating the Security Context.

- o By default, sender sequence numbers and replay windows are not reset upon group rekeying.
- o An endpoint implementing only a silent server does not support the pairwise mode.
- o Separate section on general message reception.
- o Pairwise mode moved to the document body.
- o Considerations on using the pairwise mode in non-multicast settings.
- o Optimized requests are moved as an appendix.
- o Normative support for the signature and pairwise mode.
- o Revised methods for synchronization with clients' sender sequence number.
- o Appendix with example values of parameters for countersignatures.
- o Clarifications and editorial improvements.

H.3. Version -07 to -08

- o Clarified relation between pairwise mode and group communication (Section 1).
- o Improved definition of "silent server" (Section 1.1).
- o Clarified when a Recipient Context is needed (Section 2).
- o Signature checkers as entities supported by the Group Manager (Section 2.3).
- o Clarified that the Group Manager is under exclusive control of Gid and Sender ID values in a group, with Sender ID values under each Gid value (Section 2.3).
- o Mitigation policies in case of recycled 'kid' values (Section 2.4).
- o More generic exhaustion (not necessarily wrap-around) of sender sequence numbers (Sections 2.5 and 10.11).
- o Pairwise key considerations, as to group rekeying and Sender Sequence Numbers (Section 3).

- o Added reference to static-static Diffie-Hellman shared secret (Section 3).
- o Note for implementation about the external_aad for signing (Section 4.3.2).
- o Retransmission by the application for group requests over multicast as Non-Confirmable (Section 7).
- o A server MUST use its own Partial IV in a response, if protecting it with a different context than the one used for the request (Section 7.3).
- o Security considerations: encryption of pairwise mode as alternative to group-level security (Section 10.1).
- o Security considerations: added approach to reduce the chance of global collisions of Gid values from different Group Managers (Section 10.5).
- o Security considerations: added implications for block-wise transfers when using the signature mode for requests over unicast (Section 10.7).
- o Security considerations: (multiple) supported signature algorithms (Section 10.13).
- o Security considerations: added privacy considerations on the approach for reducing global collisions of Gid values (Section 10.15).
- o Updates to the methods for synchronizing with clients' sequence number (Appendix E).
- o Simplified text on discovery services supporting the pairwise mode (Appendix G.1).
- o Editorial improvements.

H.4. Version -06 to -07

- o Updated abstract and introduction.
- o Clarifications of what pertains a group rekeying.
- o Derivation of pairwise keying material.

- o Content re-organization for COSE Object and OSCORE header compression.
- o Defined the Pairwise Flag bit for the OSCORE option.
- o Supporting CoAP Observe for group requests and responses.
- o Considerations on message protection across switching to new keying material.
- o New optimized mode based on pairwise keying material.
- o More considerations on replay protection and Security Contexts upon key renewal.
- o Security considerations on Group OSCORE for unicast requests, also as affecting the usage of the Echo option.
- o Clarification on different types of groups considered (application/security/CoAP).
- o New pairwise mode, using pairwise keying material for both requests and responses.

H.5. Version -05 to -06

- o Group IDs mandated to be unique under the same Group Manager.
- o Clarifications on parameter update upon group rekeying.
- o Updated external_aad structures.
- o Dynamic derivation of Recipient Contexts made optional and application specific.
- o Optional 4.00 response for failed signature verification on the server.
- o Removed client handling of duplicated responses to multicast requests.
- o Additional considerations on public key retrieval and group rekeying.
- o Added Group Manager responsibility on validating public keys.
- o Updates IANA registries.

- o Reference to RFC 8613.
- o Editorial improvements.

H.6. Version -04 to -05

- o Added references to draft-dijk-core-groupcomm-bis.
- o New parameter Counter Signature Key Parameters (Section 2).
- o Clarification about Recipient Contexts (Section 2).
- o Two different external_aad for encrypting and signing (Section 3.1).
- o Updated response verification to handle Observe notifications (Section 6.4).
- o Extended Security Considerations (Section 8).
- o New "Counter Signature Key Parameters" IANA Registry (Section 9.2).

H.7. Version -03 to -04

- o Added the new "Counter Signature Parameters" in the Common Context (see Section 2).
- o Added recommendation on using "deterministic ECDSA" if ECDSA is used as counter signature algorithm (see Section 2).
- o Clarified possible asynchronous retrieval of keying material from the Group Manager, in order to process incoming messages (see Section 2).
- o Structured Section 3 into subsections.
- o Added the new 'par_countersign' to the aad_array of the external_aad (see Section 3.1).
- o Clarified non reliability of 'kid' as identity indicator for a group member (see Section 2.1).
- o Described possible provisioning of new Sender ID in case of Partial IV wrap-around (see Section 2.2).
- o The former signature bit in the Flag Byte of the OSCORE option value is reverted to reserved (see Section 4.1).

- o Updated examples of compressed COSE object, now with the sixth less significant bit in the Flag Byte of the OSCORE option value set to 0 (see Section 4.3).
- o Relaxed statements on sending error messages (see Section 6).
- o Added explicit step on computing the counter signature for outgoing messages (see Sections 6.1 and 6.3).
- o Handling of just created Recipient Contexts in case of unsuccessful message verification (see Sections 6.2 and 6.4).
- o Handling of replied/repeated responses on the client (see Section 6.4).
- o New IANA Registry "Counter Signature Parameters" (see Section 9.1).

H.8. Version -02 to -03

- o Revised structure and phrasing for improved readability and better alignment with draft-ietf-core-object-security.
- o Added discussion on wrap-Around of Partial IVs (see Section 2.2).
- o Separate sections for the COSE Object (Section 3) and the OSCORE Header Compression (Section 4).
- o The countersignature is now appended to the encrypted payload of the OSCORE message, rather than included in the OSCORE Option (see Section 4).
- o Extended scope of Section 5, now titled " Message Binding, Sequence Numbers, Freshness and Replay Protection".
- o Clarifications about Non-Confirmable messages in Section 5.1 "Synchronization of Sender Sequence Numbers".
- o Clarifications about error handling in Section 6 "Message Processing".
- o Compacted list of responsibilities of the Group Manager in Section 7.
- o Revised and extended security considerations in Section 8.
- o Added IANA considerations for the OSCORE Flag Bits Registry in Section 9.

- o Revised Appendix D, now giving a short high-level description of a new endpoint set-up.

H.9. Version -01 to -02

- o Terminology has been made more aligned with RFC7252 and draft-ietf-core-object-security: i) "client" and "server" replace the old "multicaster" and "listener", respectively; ii) "silent server" replaces the old "pure listener".
- o Section 2 has been updated to have the Group Identifier stored in the 'ID Context' parameter defined in draft-ietf-core-object-security.
- o Section 3 has been updated with the new format of the Additional Authenticated Data.
- o Major rewriting of Section 4 to better highlight the differences with the message processing in draft-ietf-core-object-security.
- o Added Sections 7.2 and 7.3 discussing security considerations about uniqueness of (key, nonce) and collision of group identifiers, respectively.
- o Minor updates to Appendix A.1 about assumptions on multicast communication topology and group size.
- o Updated Appendix C on format of group identifiers, with practical implications of possible collisions of group identifiers.
- o Updated Appendix D.2, adding a pointer to draft-palombini-ace-key-groupcomm about retrieval of nodes' public keys through the Group Manager.
- o Minor updates to Appendix E.3 about Challenge-Response synchronization of sequence numbers based on the Echo option from draft-ietf-core-echo-request-tag.

H.10. Version -00 to -01

- o Section 1.1 has been updated with the definition of group as "security group".
- o Section 2 has been updated with:
 - * Clarifications on establishment/derivation of Security Contexts.

- * A table summarizing the the additional context elements compared to OSCORE.
- o Section 3 has been updated with:
 - * Examples of request and response messages.
 - * Use of CounterSignature0 rather than CounterSignature.
 - * Additional Authenticated Data including also the signature algorithm, while not including the Group Identifier any longer.
- o Added Section 6, listing the responsibilities of the Group Manager.
- o Added Appendix A (former section), including assumptions and security objectives.
- o Appendix B has been updated with more details on the use cases.
- o Added Appendix C, providing an example of Group Identifier format.
- o Appendix D has been updated to be aligned with draft-palombini-ace-key-groupcomm.

Acknowledgments

The authors sincerely thank Christian Amsuess, Stefan Beck, Rolf Blom, Carsten Bormann, Esko Dijk, Klaus Hartke, Rikard Hoeglund, Richard Kelsey, John Mattsson, Dave Robin, Jim Schaad, Ludwig Seitz, Peter van der Stok and Erik Thormarker for their feedback and comments.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; the H2020 project SIFIS-Home (Grant agreement 952652); the SSF project SEC4Factory under the grant RIT17-0032; and the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE 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 Working Group
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2022

M. Tiloca
RISE AB
G. Selander
F. Palombini
J. Mattsson
Ericsson AB
J. Park
Universitaet Duisburg-Essen
7 March 2022

Group OSCORE - Secure Group Communication for CoAP
draft-ietf-core-oscore-groupcomm-14

Abstract

This document defines Group Object Security for Constrained RESTful Environments (Group OSCORE), providing end-to-end security of CoAP messages exchanged between members of a group, e.g., sent over IP multicast. In particular, the described approach defines how OSCORE is used in a group communication setting to provide source authentication for CoAP group requests, sent by a client to multiple servers, and for protection of the corresponding CoAP responses. Group OSCORE also defines a pairwise mode where each member of the group can efficiently derive a symmetric pairwise key with any other member of the group for pairwise OSCORE communication.

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
1.1. Terminology	6
2. Security Context	8
2.1. Common Context	11
2.1.1. AEAD Algorithm	11
2.1.2. ID Context	11
2.1.3. Group Manager Authentication Credential	12
2.1.4. Signature Encryption Algorithm	12
2.1.5. Signature Algorithm	12
2.1.6. Group Encryption Key	12
2.1.7. Pairwise Key Agreement Algorithm	13
2.2. Sender Context and Recipient Context	13
2.3. Authentication Credentials	14
2.4. Pairwise Keys	16
2.4.1. Derivation of Pairwise Keys	16
2.4.2. ECDH with Montgomery Coordinates	18
2.4.3. Usage of Sequence Numbers	19
2.4.4. Security Context for Pairwise Mode	20
2.5. Update of Security Context	20
2.5.1. Loss of Mutable Security Context	21
2.5.2. Exhaustion of Sender Sequence Number	22
2.5.3. Retrieving New Security Context Parameters	23
3. The Group Manager	25
3.1. Support for Additional Entities	26
3.2. Management of Group Keying Material	27
3.2.1. Recycling of Identifiers	30
3.3. Responsibilities of the Group Manager	32
4. The COSE Object	34
4.1. Countersignature	34
4.1.1. Keystream Derivation	35
4.1.2. Clarifications on Using a Countersignature	36
4.2. The 'kid' and 'kid context' parameters	36
4.3. external_aad	36
5. OSCORE Header Compression	39
5.1. Examples of Compressed COSE Objects	40
5.1.1. Examples in Group Mode	40
5.1.2. Examples in Pairwise Mode	41

6.	Message Binding, Sequence Numbers, Freshness and Replay Protection	42
6.1.	Supporting Observe	42
6.2.	Update of Replay Window	42
6.3.	Message Freshness	43
7.	Message Reception	43
8.	Message Processing in Group Mode	44
8.1.	Protecting the Request	46
8.1.1.	Supporting Observe	46
8.2.	Verifying the Request	47
8.2.1.	Supporting Observe	49
8.3.	Protecting the Response	49
8.3.1.	Supporting Observe	50
8.4.	Verifying the Response	51
8.4.1.	Supporting Observe	53
8.5.	External Signature Checkers	54
9.	Message Processing in Pairwise Mode	55
9.1.	Pre-Conditions	56
9.2.	Main Differences from OSCORE	56
9.3.	Protecting the Request	57
9.4.	Verifying the Request	57
9.5.	Protecting the Response	57
9.6.	Verifying the Response	58
10.	Challenge-Response Synchronization	59
11.	Implementation Compliance	62
12.	Security Considerations	63
12.1.	Security of the Group Mode	64
12.2.	Security of the Pairwise Mode	66
12.3.	Uniqueness of (key, nonce)	67
12.4.	Management of Group Keying Material	67
12.5.	Update of Security Context and Key Rotation	68
12.5.1.	Late Update on the Sender	68
12.5.2.	Late Update on the Recipient	69
12.6.	Collision of Group Identifiers	69
12.7.	Cross-group Message Injection	70
12.7.1.	Attack Description	70
12.7.2.	Attack Prevention in Group Mode	71
12.8.	Prevention of Group Cloning Attack	72
12.9.	Group OSCORE for Unicast Requests	73
12.10.	End-to-end Protection	74
12.11.	Master Secret	74
12.12.	Replay Protection	75
12.13.	Message Freshness	75
12.14.	Client Aliveness	75
12.15.	Cryptographic Considerations	76
12.16.	Message Segmentation	77
12.17.	Privacy Considerations	78
13.	IANA Considerations	79

13.1. OSCORE Flag Bits Registry	79
14. References	79
14.1. Normative References	79
14.2. Informative References	81
Appendix A. Assumptions and Security Objectives	84
A.1. Assumptions	85
A.2. Security Objectives	86
Appendix B. List of Use Cases	87
Appendix C. Example of Group Identifier Format	90
Appendix D. Set-up of New Endpoints	91
Appendix E. Document Updates	91
E.1. Version -13 to -14	91
E.2. Version -12 to -13	92
E.3. Version -11 to -12	92
E.4. Version -10 to -11	93
E.5. Version -09 to -10	94
E.6. Version -08 to -09	95
E.7. Version -07 to -08	95
E.8. Version -06 to -07	97
E.9. Version -05 to -06	97
E.10. Version -04 to -05	98
E.11. Version -03 to -04	98
E.12. Version -02 to -03	99
E.13. Version -01 to -02	100
E.14. Version -00 to -01	101
Acknowledgments	102
Authors' Addresses	102

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 [I-D.ietf-core-groupcomm-bis] addresses use cases where deployed devices benefit from a group communication model, for example to reduce latencies, improve performance, and reduce bandwidth utilization. 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). Group communication for CoAP [I-D.ietf-core-groupcomm-bis] mainly uses UDP/IP multicast as the underlying data transport.

Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] describes a security protocol based on the exchange of protected CoAP messages. OSCORE builds on CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-rfc8152bis-struct] [I-D.ietf-cose-rfc8152bis-algs] and

provides end-to-end encryption, integrity, replay protection and binding of response to request between a sender and a recipient, independent of the transport layer also in the presence of intermediaries. To this end, a CoAP message is protected by including its payload (if any), certain options, and header fields in a COSE object, which replaces the authenticated and encrypted fields in the protected message.

This document defines Group OSCORE, a security protocol for Group communication for CoAP [I-D.ietf-core-groupcomm-bis], providing the same end-to-end security properties as OSCORE in the case where CoAP requests have multiple recipients. In particular, the described approach defines how OSCORE is used in a group communication setting to provide source authentication for CoAP group requests, sent by a client to multiple servers, and for protection of the corresponding CoAP responses. Group OSCORE also defines a pairwise mode where each member of the group can efficiently derive a symmetric pairwise key with any other member of the group for pairwise OSCORE communication. Just like OSCORE, Group OSCORE is independent of the transport layer and works wherever CoAP does.

As with OSCORE, it is possible to combine Group OSCORE with communication security on other layers. One example is the use of transport layer security, such as DTLS [RFC6347][I-D.ietf-tls-dtls13], between one client and one proxy (and vice versa), or between one proxy and one server (and vice versa). This prevents observers from accessing addressing information conveyed in CoAP options that would not be protected by Group OSCORE, but would be protected by DTLS. These options include Uri-Host, Uri-Port and Proxy-Uri. Note that DTLS does not define how to secure messages sent over IP multicast.

Group OSCORE defines two modes of operation, that can be used independently or together:

- * In the group mode, Group OSCORE requests and responses are digitally signed with the private key of the sender and the signature is embedded in the protected CoAP message. The group mode supports all COSE signature algorithms as well as signature verification by intermediaries. This mode is defined in Section 8.

- * In the pairwise mode, two group members exchange OSCORE requests and responses (typically) over unicast, and the messages are protected with symmetric keys. These symmetric keys are derived from Diffie-Hellman shared secrets, calculated with the asymmetric keys of the sender and recipient, allowing for shorter integrity tags and therefore lower message overhead. This mode is defined in Section 9.

Both modes provide source authentication of CoAP messages. The application decides what mode to use, potentially on a per-message basis. Such decisions can be based, for instance, on pre-configured policies or dynamic assessing of the target recipient and/or resource, among other things. One important case is when requests are protected with the group mode, and responses with the pairwise mode. Since such responses convey shorter integrity tags instead of bigger, full-fledged signatures, this significantly reduces the message overhead in case of many responses to one request.

A special deployment of Group OSCORE is to use pairwise mode only. For example, consider the case of a constrained-node network [RFC7228] with a large number of CoAP endpoints and the objective to establish secure communication between any pair of endpoints with a small provisioning effort and message overhead. Since the total number of security associations that needs to be established grows with the square of the number of endpoints, it is desirable to restrict the amount of secret keying material provided to each endpoint. Moreover, a key establishment protocol would need to be executed for each security association. One solution to this is to deploy Group OSCORE, with the endpoints being part of a group, and use the pairwise mode. This solution assumes a trusted third party called Group Manager (see Section 3). However, it has the benefit of providing a single shared secret, while distributing only the public keys of group members or a subset of those. After that, a CoAP endpoint can locally derive the OSCORE Security Context for the other endpoint in the group, and protect CoAP communications with very low overhead [I-D.ietf-lwig-security-protocol-comparison].

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", "client", "server", "sender" and "recipient"; group communication for CoAP

[I-D.ietf-core-groupcomm-bis]; CBOR [RFC8949]; COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] and related countersignatures [I-D.ietf-cose-countersign].

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" and "Master Secret", defined in [RFC8613].

Terminology for constrained environments, such as "constrained device" and "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].
- * Authentication credential: set of information associated with an entity, including that entity's public key and parameters associated with the public key. Examples of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claims Sets (CCSs) [RFC8392], X.509 certificates [RFC7925] and C509 certificates [I-D.ietf-cose-cbor-encoded-cert]. Further details about authentication credentials are provided in Section 2.3.
- * Group: a set of endpoints that share group keying material and security parameters (Common Context, see Section 2). That is, unless otherwise specified, the term group used in this document refers to a "security group" (see Section 2.1 of [I-D.ietf-core-groupcomm-bis]), not to be confused with "CoAP group" or "application group".
- * Group Manager: entity responsible for a group. Each endpoint in a group communicates securely with the respective Group Manager, which is neither required to be an actual group member nor to take part in the group communication. The full list of responsibilities of the Group Manager is provided in Section 3.3.

- * **Silent server:** member of a group that never sends protected responses in reply to requests. For CoAP group communications, requests are normally sent without necessarily expecting a response. A silent server may send unprotected responses, as error responses reporting an OSCORE error. Note that an endpoint can implement both a silent server and a client, i.e., the two roles are independent. An endpoint acting only as a silent server performs only Group OSCORE processing on incoming requests. Silent servers maintain less keying material and in particular do not have a Sender Context for the group. Since silent servers do not have a Sender ID, they cannot support the pairwise mode.
- * **Group Identifier (Gid):** identifier assigned to the group, unique within the set of groups of a given Group Manager.
- * **Birth Gid:** with respect to a group member, the Gid obtained by that group member upon (re-)joining the group.
- * **Group request:** CoAP request message sent by a client in the group to all servers in that group.
- * **Key Generation Number:** an integer value identifying the current version of the keying material used in a group.
- * **Source authentication:** evidence that a received message in the group originated from a specific identified group member. This also provides assurance that the message was not tampered with by anyone, be it a different legitimate group member or an endpoint which is not a group member.

2. Security Context

As per the terminology in Section 1.1, this document refers to a group as a set of endpoints sharing keying material and security parameters for executing the Group OSCORE protocol. Each endpoint of a group is aware of whether the group uses the group mode, or the pairwise mode, or both. Then, an endpoint can use any mode it supports if also used in the group.

All members of a group maintain a Security Context as defined in Section 3 of [RFC8613] and extended as defined in this section. How the Security Context is established by the group members is out of scope for this document, but if there is more than one Security Context applicable to a message, then the endpoints MUST be able to tell which Security Context was latest established.

The default setting for how to manage information about the group, including the Security Context, is described in terms of a Group Manager (see Section 3). In particular, the Group Manager indicates whether the group uses the group mode, the pairwise mode, or both of them, as part of the group data provided to candidate group members when joining the group.

The remainder of this section provides further details about the Security Context of Group OSCORE. In particular, each endpoint which is member of a group maintains a Security Context as defined in Section 3 of [RFC8613], extended as follows (see Figure 1).

- * One Common Context, shared by all the endpoints in the group. Several new parameters are included in the Common Context.

If a Group Manager is used for maintaining the group, the Common Context is extended with the authentication credential of the Group Manager, including the Group Manager's public key. When processing messages, the authentication credential of the Group Manager is included in the external additional authenticated data (see Section 4.3).

If the group uses the group mode, the Common context is extended with the following new parameters.

- Signature Encryption Algorithm and Signature Algorithm. These relate to the encryption/decryption operations and to the computation/verification of countersignatures, respectively, when a message is protected with the group mode (see Section 8).
- Group Encryption Key, used to perform encryption/decryption of countersignatures, when a message is protected with the group mode (see Section 8).

If the group uses the pairwise mode, the Common Context is extended with a Pairwise Key Agreement Algorithm used for agreement on a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.4.1) to protect messages with the pairwise mode (see Section 9).

- * One Sender Context, extended with the endpoint's private key and authentication credential including the endpoint's public key.

The private key is used to sign messages protected with the group mode, or for deriving pairwise keys in pairwise mode (see Section 2.4). The authentication credential is used for deriving pairwise keys in pairwise mode, and is included in the external additional authenticated data when processing outgoing messages (see Section 9).

If the endpoint supports the pairwise mode, the Sender Context is also extended with the Pairwise Sender Keys associated with the other endpoints (see Section 2.4).

The Sender Context is omitted if the endpoint is configured exclusively as silent server.

- * One Recipient Context for each other endpoint from which messages are received. It is not necessary to maintain Recipient Contexts associated with endpoints from which messages are not (expected to be) received. The Recipient Context is extended with the authentication credential of the other endpoint, including that endpoint's public key.

The public key is used to verify the signature of messages protected with the group mode from the other endpoint and for deriving the pairwise keys in pairwise mode (see Section 2.4). The authentication credential is used for deriving pairwise keys in pairwise mode, and is included in the external additional authenticated data when processing incoming messages from the other endpoint (see Section 9).

If the endpoint supports the pairwise mode, then the Recipient Context is also extended with the Pairwise Recipient Key associated with the other endpoint (see Section 2.4).

Context Component	New Information Elements
Common Context	Group Manager Authentication Credential * Signature Encryption Algorithm * Signature Algorithm * Group Encryption Key ^ Pairwise Key Agreement Algorithm
Sender Context	Endpoint's own private key Endpoint's own authentication credential ^ Pairwise Sender Keys for the other endpoints
Each Recipient Context	Other endpoint's authentication credential ^ Pairwise Recipient Key for the other endpoint

Figure 1: Additions to the OSCORE Security Context. The optional elements labeled with * (with ^) are present only if the group uses the group mode (the pairwise mode).

2.1. Common Context

The Common Context may be acquired from the Group Manager (see Section 3). The following sections define how the Common Context is extended, compared to [RFC8613].

2.1.1. AEAD Algorithm

AEAD Algorithm identifies the COSE AEAD algorithm to use for encryption, when messages are protected using the pairwise mode (see Section 9). This algorithm MUST provide integrity protection. This parameter is immutable once the Common Context is established, and it is not relevant if the group uses only the group mode.

2.1.2. ID Context

The ID Context parameter (see Sections 3.1 and 3.3 of [RFC8613]) in the Common Context SHALL contain the Group Identifier (Gid) of the group. The choice of the Gid format is application specific. An example of specific formatting of the Gid is given in Appendix C. The application needs to specify how to handle potential collisions between Gids (see Section 12.6).

2.1.3. Group Manager Authentication Credential

Group Manager Authentication Credential specifies the authentication credential of the Group Manager, including the Group Manager's public key. This is included in the external additional authenticated data when processing messages (see Section 4.3).

Each group member MUST obtain the authentication credential of the Group Manager with a valid proof-of-possession of the corresponding private key, for instance from the Group Manager itself when joining the group. Further details on the provisioning of the Group Manager's authentication credential to the group members are out of the scope of this document.

2.1.4. Signature Encryption Algorithm

Signature Encryption Algorithm identifies the algorithm to use for encryption, when messages are protected using the group mode (see Section 8). This algorithm MAY provide integrity protection. This parameter is immutable once the Common Context is established.

This algorithm is not used to encrypt the countersignature in messages protected using the group mode, for which the method defined in Section 4.1 is used.

2.1.5. Signature Algorithm

Signature Algorithm identifies the digital signature algorithm used to compute a countersignature on the COSE object (see Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign]), when messages are protected using the group mode (see Section 8). This parameter is immutable once the Common Context is established.

2.1.6. Group Encryption Key

Group Encryption Key specifies the encryption key for deriving a keystream to encrypt/decrypt a countersignature, when a message is protected with the group mode (see Section 8).

The Group Encryption Key is derived as defined for Sender/Recipient Keys in Section 3.2.1 of [RFC8613], with the following differences.

- * The 'id' element of the 'info' array is the empty byte string.
- * The 'alg_aead' element of the 'info' array takes the value of Signature Encryption Algorithm from the Common Context (see Section 2.1.5).

- * The 'type' element of the 'info' array is "Group Encryption Key". The label is an ASCII string and does not include a trailing NUL byte.
- * L and the 'L' element of the 'info' array are the size of the key for the Signature Encryption Algorithm from the Common Context (see Section 2.1.5), in bytes.

2.1.7. Pairwise Key Agreement Algorithm

Pairwise Key Agreement Algorithm identifies the elliptic curve Diffie-Hellman algorithm used to derive a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.4.1) to protect messages with the pairwise mode (see Section 9). This parameter is immutable once the Common Context is established.

2.2. Sender Context and Recipient Context

OSCORE specifies the derivation of Sender Context and Recipient Context, specifically of Sender/Recipient Keys and Common IV, from a set of input parameters (see Section 3.2 of [RFC8613]).

The derivation of Sender/Recipient Keys and Common IV defined in OSCORE applies also to Group OSCORE, with the following extensions compared to Section 3.2.1 of [RFC8613].

- * If the group uses (also) the group mode, the 'alg_aead' element of the 'info' array takes the value of Signature Encryption Algorithm from the Common Context (see Section 2.1.5).
- * If the group uses only the pairwise mode, the 'alg_aead' element of the 'info' array takes the value of AEAD Algorithm from the Common Context (see Section 2.1.1).

The Sender ID SHALL be unique for each endpoint in a group with a certain tuple (Master Secret, Master Salt, Group Identifier), see Section 3.3 of [RFC8613].

For Group OSCORE, the Sender Context and Recipient Context additionally contain asymmetric keys, as described previously in Section 2. The private key of the sender and the authentication credential including the corresponding public key can, for example, be generated by the endpoint or provisioned during manufacturing.

With the exception of the authentication credential of the sender endpoint and the possibly associated pairwise keys, a receiver endpoint can derive a complete Security Context from a received Group

OSCORE message and the Common Context. The authentication credentials in the Recipient Contexts can be retrieved from the Group Manager (see Section 3) upon joining the group. An authentication credential can alternatively be acquired from the Group Manager at a later time, for example the first time a message is received from a particular endpoint in the group (see Section 8.2 and Section 8.4).

For severely constrained devices, it may be not feasible to simultaneously handle the ongoing processing of a recently received message in parallel with the retrieval of the sender endpoint's authentication credential. Such devices can be configured to drop a received message for which there is no (complete) Recipient Context, and retrieve the sender endpoint's authentication credential in order to have it available to verify subsequent messages from that endpoint.

An endpoint admits a maximum amount of Recipient Contexts for a same Security Context, e.g., due to memory limitations. After reaching that limit, the creation of a new Recipient Context results in an overflow. When this happens, the endpoint has to delete a current Recipient Context to install the new one. It is up to the application to define policies for selecting the current Recipient Context to delete. If the new Recipient Context has been installed after the endpoint has experienced the overflow above, then the Recipient Context is initialized with an invalid Replay Window, and accordingly requires the endpoint to take appropriate actions (see Section 2.5.1.2).

2.3. Authentication Credentials

In a group, the following MUST hold for the authentication credential of each endpoint as well as for the authentication credential of the Group Manager.

- * All authentication credentials MUST be encoded according to the same format used in the group. The used format MUST provide the public key as well as the comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable).
- * All authentication credentials and the public key specified therein MUST be for the public key algorithm used in the group and aligned with the possible associated parameters used in the group, e.g., the used elliptic curve (when applicable).

If the group uses (also) the group mode, the public key algorithm is the Signature Algorithm used in the group. If the group uses only the pairwise mode, the public key algorithm is the Pairwise Key Agreement Algorithm used in the group.

If the authentication credentials are X.509 certificates [RFC7925] or C509 certificates [I-D.ietf-cose-cbor-encoded-cert], the public key algorithm is fully described by the "algorithm" field of the "SubjectPublicKeyInfo" structure, and by the "subjectPublicKeyAlgorithm" element, respectively.

If authentication credentials are CBOR Web Tokens (CWTs) or CWT Claims Sets (CCSs) [RFC8392], the public key algorithm is fully described by a COSE key type and its "kty" and "crv" parameters.

Authentication credentials are used to derive pairwise keys (see Section 2.4.1) and are included in the external additional authenticated data when processing messages (see Section 4.3). In both these cases, an endpoint in a group MUST treat authentication credentials as opaque data, i.e., by considering the same binary representation made available to other endpoints in the group, possibly through a designated trusted source (e.g., the Group Manager).

For example, an X.509 certificate is provided as its direct binary serialization. If C509 certificates or CWTs are used as authentication credentials, each is provided as the binary serialization of a (possibly tagged) CBOR array. If CCSs are used as authentication credentials, each is provided as the binary serialization of a CBOR map.

If authentication credentials are CWTs, then the untagged CWT associated with an entity is stored in the Security Context and used as authentication credential for that entity.

If authentication credentials are X.509 / C509 certificates or CWTs and the authentication credential associated with an entity is provided within a chain or a bag, then only the end-entity certificate or end-entity untagged CWT is stored in the Security Context and used as authentication credential for that entity.

Storing whole authentication credentials rather than only a subset of those may result in a non-negligible storage overhead. On the other hand, it also ensures that authentication credentials are correctly used in a simple, flexible and non-error-prone way, also taking into account future credential formats as entirely new or extending existing ones. In particular, it is ensured that:

- * When used to derive pairwise keys and when included in the external additional authenticated data, authentication credentials can also specify possible metadata and parameters related to the included public key. Besides the public key algorithm, these comprise other relevant pieces of information such as key usage, expiration time, issuer and subject.
- * All endpoints using another endpoint's authentication credential use exactly the same binary serialization, as obtained and distributed by the credential provider (e.g., the Group Manager) and as originally crafted by the credential issuer. In turn, this does not require to define and maintain canonical subsets of authentication credentials and their corresponding encoding, and spares endpoints from error-prone re-encoding operations.

Depending on the particular deployment and the intended group size, limiting the storage overhead of endpoints in a group can be an incentive for system/network administrators to prefer using a compact format of authentication credentials in the first place.

2.4. Pairwise Keys

Certain signature schemes, such as EdDSA and ECDSA, support a secure combined signature and encryption scheme. This section specifies the derivation of "pairwise keys", for use in the pairwise mode defined in Section 9.

Group OSCORE keys used for both signature and encryption MUST be used only for purposes related to Group OSCORE. These include the processing of messages with Group OSCORE, as well as performing proof-of-possession of private keys, e.g., upon joining a group through the Group Manager (see Section 3).

2.4.1. Derivation of Pairwise Keys

Using the Group OSCORE Security Context (see Section 2), a group member can derive AEAD keys, to protect point-to-point communication between itself and any other endpoint X in the group by means of the AEAD Algorithm from the Common Context (see Section 2.1.1). The key derivation of these so-called pairwise keys follows the same construction as in Section 3.2.1 of [RFC8613]:

Pairwise Sender Key = HKDF(Sender Key, IKM-Sender, info, L)
Pairwise Recipient Key = HKDF(Recipient Key, IKM-Recipient, info, L)

with

IKM-Sender = Sender Auth Cred | Recipient Auth Cred | Shared Secret
IKM-Recipient = Recipient Auth Cred | Sender Auth Cred | Shared Secret

where:

- * The Pairwise Sender Key is the AEAD key for processing outgoing messages addressed to endpoint X.
- * The Pairwise Recipient Key is the AEAD key for processing incoming messages from endpoint X.
- * HKDF is the OSCORE HKDF algorithm [RFC8613] from the Common Context.
- * The Sender Key from the Sender Context is used as salt in the HKDF, when deriving the Pairwise Sender Key.
- * The Recipient Key from the Recipient Context associated with endpoint X is used as salt in the HKDF, when deriving the Pairwise Recipient Key.
- * Sender Auth Cred is the endpoint's own authentication credential from the Sender Context.
- * Recipient Auth Cred is the endpoint X's authentication credential from the Recipient Context associated with the endpoint X.
- * The Shared Secret is computed as a cofactor Diffie-Hellman shared secret, see Section 5.7.1.2 of [NIST-800-56A], using the Pairwise Key Agreement Algorithm. The endpoint uses its private key from the Sender Context and the other endpoint's public key included in Recipient Auth Cred. Note the requirement of validation of public keys in Section 12.15. For X25519 and X448, the procedure is described in Section 5 of [RFC7748] using public keys mapped to Montgomery coordinates, see Section 2.4.2.
- * IKM-Sender is the Input Keying Material (IKM) used in the HKDF for the derivation of the Pairwise Sender Key. IKM-Sender is the byte string concatenation of Sender Auth Cred, Recipient Auth Cred and the Shared Secret. The authentication credentials Sender Auth Cred and Recipient Auth Cred are binary encoded as defined in Section 2.3.

- * IKM-Recipient is the Input Keying Material (IKM) used in the HKDF for the derivation of the Pairwise Recipient Key. IKM-Recipient is the byte string concatenation of Recipient Auth Cred, Sender Auth Cred and the Shared Secret. The authentication credentials Recipient Auth Cred and Sender Auth Cred are binary encoded as defined in Section 2.3.
- * info and L are as defined in Section 3.2.1 of [RFC8613]. That is:
 - The 'alg_aead' element of the 'info' array takes the value of AEAD Algorithm from the Common Context (see Section 2.1.1).
 - L and the 'L' element of the 'info' array are the size of the key for the AEAD Algorithm from the Common Context (see Section 2.1.1), in bytes.

If EdDSA asymmetric keys are used, the Edward coordinates are mapped to Montgomery coordinates using the maps defined in Sections 4.1 and 4.2 of [RFC7748], before using the X25519 and X448 functions defined in Section 5 of [RFC7748]. For further details, see Section 2.4.2. ECC asymmetric keys in Montgomery or Weierstrass form are used directly in the key agreement algorithm without coordinate mapping.

After establishing a partially or completely new Security Context (see Section 2.5 and Section 3.2), the old pairwise keys MUST be deleted. Since new Sender/Recipient Keys are derived from the new group keying material (see Section 2.2), every group member MUST use the new Sender/Recipient Keys when deriving new pairwise keys.

As long as any two group members preserve the same asymmetric keys, their Diffie-Hellman shared secret does not change across updates of the group keying material.

2.4.2. ECDH with Montgomery Coordinates

2.4.2.1. Curve25519

The y-coordinate of the other endpoint's Ed25519 public key is decoded as specified in Section 5.1.3 of [RFC8032]. The Curve25519 u-coordinate is recovered as $u = (1 + y) / (1 - y) \pmod{p}$ following the map in Section 4.1 of [RFC7748]. Note that the mapping is not defined for $y = 1$, and that $y = -1$ maps to $u = 0$ which corresponds to the neutral group element and thus will result in a degenerate shared secret. Therefore implementations MUST abort if the y-coordinate of the other endpoint's Ed25519 public key is 1 or -1 (mod p).

The private signing key byte strings (= the lower 32 bytes used for generating the public key, see step 1 of Section 5.1.5 of [RFC8032]) are decoded the same way for signing in Ed25519 and scalar multiplication in X25519. Hence, to compute the shared secret the endpoint applies the X25519 function to the Ed25519 private signing key byte string and the encoded u-coordinate byte string as specified in Section 5 of [RFC7748].

2.4.2.2. Curve448

The y-coordinate of the other endpoint's Ed448 public key is decoded as specified in Section 5.2.3. of [RFC8032]. The Curve448 u-coordinate is recovered as $u = y^2 * (d * y^2 - 1) / (y^2 - 1) \pmod{p}$ following the map from "edwards448" in Section 4.2 of [RFC7748], and also using the relation $x^2 = (y^2 - 1)/(d * y^2 - 1)$ from the curve equation. Note that the mapping is not defined for $y = 1$ or -1 . Therefore implementations MUST abort if the y-coordinate of the peer endpoint's Ed448 public key is 1 or $-1 \pmod{p}$.

The private signing key byte strings (= the lower 57 bytes used for generating the public key, see step 1 of Section 5.2.5 of [RFC8032]) are decoded the same way for signing in Ed448 and scalar multiplication in X448. Hence, to compute the shared secret the endpoint applies the X448 function to the Ed448 private signing key byte string and the encoded u-coordinate byte string as specified in Section 5 of [RFC7748].

2.4.3. Usage of Sequence Numbers

When using any of its Pairwise Sender Keys, a sender endpoint including the 'Partial IV' parameter in the protected message MUST use the current fresh value of the Sender Sequence Number from its Sender Context (see Section 2.2). That is, the same Sender Sequence Number space is used for all outgoing messages protected with Group OSCORE, thus limiting both storage and complexity.

On the other hand, when combining group and pairwise communication modes, this may result in the Partial IV values moving forward more often. This can happen when a client engages in frequent or long sequences of one-to-one exchanges with servers in the group, by sending requests over unicast. In turn, this contributes to a sooner exhaustion of the Sender Sequence Number space of the client, which would then require to take actions for deriving a new Sender Context before resuming communications in the group (see Section 2.5.2).

2.4.4. Security Context for Pairwise Mode

If the pairwise mode is supported, the Security Context additionally includes Pairwise Key Agreement Algorithm and the pairwise keys, as described at the beginning of Section 2.

The pairwise keys as well as the shared secrets used in their derivation (see Section 2.4.1) may be stored in memory or recomputed every time they are needed. The shared secret changes only when a public/private key pair used for its derivation changes, which results in the pairwise keys also changing. Additionally, the pairwise keys change if the Sender ID changes or if a new Security Context is established for the group (see Section 2.5.3). In order to optimize protocol performance, an endpoint may store the derived pairwise keys for easy retrieval.

In the pairwise mode, the Sender Context includes the Pairwise Sender Keys to use with the other endpoints (see Figure 1). In order to identify the right key to use, the Pairwise Sender Key for endpoint X may be associated with the Recipient ID of endpoint X, as defined in the Recipient Context (i.e., the Sender ID from the point of view of endpoint X). In this way, the Recipient ID can be used to lookup for the right Pairwise Sender Key. This association may be implemented in different ways, e.g., by storing the pair (Recipient ID, Pairwise Sender Key) or linking a Pairwise Sender Key to a Recipient Context.

2.5. Update of Security Context

It is RECOMMENDED that the immutable part of the Security Context is stored in non-volatile memory, or that it can otherwise be reliably accessed throughout the operation of the group, e.g., after a device reboots. However, also immutable parts of the Security Context may need to be updated, for example due to scheduled key renewal, new or re-joining members in the group, or the fact that the endpoint changes Sender ID (see Section 2.5.3).

On the other hand, the mutable parts of the Security Context are updated by the endpoint when executing the security protocol, but may nevertheless become outdated, e.g., due to loss of the mutable Security Context (see Section 2.5.1) or exhaustion of Sender Sequence Numbers (see Section 2.5.2).

If it is not feasible or practically possible to store and maintain up-to-date the mutable part in non-volatile memory (e.g., due to limited number of write operations), the endpoint MUST be able to detect a loss of the mutable Security Context and MUST accordingly take the actions defined in Section 2.5.1.

2.5.1. Loss of Mutable Security Context

An endpoint may lose its mutable Security Context, e.g., due to a reboot (see Section 2.5.1.1) or to an overflow of Recipient Contexts (see Section 2.5.1.2).

In such a case, the endpoint needs to prevent the re-use of a nonce with the same AEAD key, and to handle incoming replayed messages.

2.5.1.1. Reboot and Total Loss

In case a loss of the Sender Context and/or of the Recipient Contexts is detected (e.g., following a reboot), the endpoint MUST NOT protect further messages using this Security Context to avoid reusing an AEAD nonce with the same AEAD key.

In particular, before resuming its operations in the group, the endpoint MUST retrieve new Security Context parameters from the Group Manager (see Section 2.5.3) and use them to derive a new Sender Context (see Section 2.2). Since this includes a newly derived Sender Key, a server will not reuse the same pair (key, nonce), even when using the Partial IV of (old re-injected) requests to build the AEAD nonce for protecting the corresponding responses.

From then on, the endpoint MUST use the latest installed Sender Context to protect outgoing messages. Also, newly created Recipient Contexts will have a Replay Window which is initialized as valid.

If not able to establish an updated Sender Context, e.g., because of lack of connectivity with the Group Manager, the endpoint MUST NOT protect further messages using the current Security Context and MUST NOT accept incoming messages from other group members, as currently unable to detect possible replays.

An adversary may leverage the above to perform a Denial of Service attack and prevent some group members from communicating altogether. That is, the adversary can first block the communication path between the Group Manager and some individual group members. This can be achieved, for instance, by injecting fake responses to DNS queries for the Group Manager hostname, or by removing a network link used for routing traffic towards the Group Manager. Then, the adversary can induce a reboot for some endpoints in the group, e.g., by triggering a short power outage. After that, such endpoints that have lost their Sender Context and/or Recipient Contexts following the reboot would not be able to obtain new Security Context parameters from the Group Manager, as specified above. Thus, they would not be able to further communicate in the group until connectivity with the Group Manager is restored.

2.5.1.2. Overflow of Recipient Contexts

After reaching the maximum amount of Recipient Contexts, an endpoint will experience an overflow when installing a new Recipient Context, as it requires to first delete an existing one (see Section 2.2).

Every time this happens, the Replay Window of the new Recipient Context is initialized as not valid. Therefore, the endpoint **MUST** take the following actions, before accepting request messages from the client associated with the new Recipient Context.

If it is not configured as silent server, the endpoint **MUST** either:

- * Retrieve new Security Context parameters from the Group Manager and derive a new Sender Context, as defined in Section 2.5.1.1; or
- * When receiving a first request to process with the new Recipient Context, use the approach specified in Section 10 and based on the Echo Option for CoAP [RFC9175], if supported. In particular, the endpoint **MUST** use its Partial IV when generating the AEAD nonce and **MUST** include the Partial IV in the response message conveying the Echo Option. If the endpoint supports the CoAP Echo Option, it is **RECOMMENDED** to take this approach.

If it is configured exclusively as silent server, the endpoint **MUST** wait for the next group rekeying to occur, in order to derive a new Security Context and re-initialize the Replay Window of each Recipient Contexts as valid.

2.5.2. Exhaustion of Sender Sequence Number

An endpoint can eventually exhaust the Sender Sequence Number, which is incremented for each new outgoing message including a Partial IV. This is the case for group requests, Observe notifications [RFC7641] and, optionally, any other response.

Implementations **MUST** be able to detect an exhaustion of Sender Sequence Number, after the endpoint has consumed the largest usable value. If an implementation's integers support wrapping addition, the implementation **MUST** treat Sender Sequence Number as exhausted when a wrap-around is detected.

Upon exhausting the Sender Sequence Numbers, the endpoint **MUST NOT** use this Security Context to protect further messages including a Partial IV.

The endpoint SHOULD inform the Group Manager, retrieve new Security Context parameters from the Group Manager (see Section 2.5.3), and use them to derive a new Sender Context (see Section 2.2).

From then on, the endpoint MUST use its latest installed Sender Context to protect outgoing messages.

2.5.3. Retrieving New Security Context Parameters

The Group Manager can assist an endpoint with an incomplete Sender Context to retrieve missing data of the Security Context and thereby become fully operational in the group again. The two main options for the Group Manager are described in this section: i) assignment of a new Sender ID to the endpoint (see Section 2.5.3.1); and ii) establishment of a new Security Context for the group (see Section 2.5.3.2). The update of the Replay Window in each of the Recipient Contexts is discussed in Section 6.2.

As group membership changes, or as group members get new Sender IDs (see Section 2.5.3.1) so do the relevant Recipient IDs that the other endpoints need to keep track of. As a consequence, group members may end up retaining stale Recipient Contexts, that are no longer useful to verify incoming secure messages.

The Recipient ID ('kid') SHOULD NOT be considered as a persistent and reliable identifier of a group member. Such an indication can be achieved only by using that member's public key, when verifying countersignatures of received messages (in group mode), or when verifying messages integrity-protected with pairwise keying material derived from authentication credentials and associated asymmetric keys (in pairwise mode).

Furthermore, applications MAY define policies to: i) delete (long-)unused Recipient Contexts and reduce the impact on storage space; as well as ii) check with the Group Manager that an authentication credential with the public key included therein is currently the one associated with a 'kid' value, after a number of consecutive failed verifications.

2.5.3.1. New Sender ID for the Endpoint

The Group Manager may assign a new Sender ID to an endpoint, while leaving the Gid, Master Secret and Master Salt unchanged in the group. In this case, the Group Manager MUST assign a Sender ID that has not been used in the group since the latest time when the current Gid value was assigned to the group (see Section 3.2).

Having retrieved the new Sender ID, and potentially other missing data of the immutable Security Context, the endpoint can derive a new Sender Context (see Section 2.2). When doing so, the endpoint resets the Sender Sequence Number in its Sender Context to 0, and derives a new Sender Key. This is in turn used to possibly derive new Pairwise Sender Keys.

From then on, the endpoint MUST use its latest installed Sender Context to protect outgoing messages.

The assignment of a new Sender ID may be the result of different processes. The endpoint may request a new Sender ID, e.g., because of exhaustion of Sender Sequence Numbers (see Section 2.5.2). An endpoint may request to re-join the group, e.g., because of losing its mutable Security Context (see Section 2.5.1), and is provided with a new Sender ID together with the latest immutable Security Context.

For the other group members, the Recipient Context corresponding to the old Sender ID becomes stale (see Section 3.2).

2.5.3.2. New Security Context for the Group

The Group Manager may establish a new Security Context for the group (see Section 3.2). The Group Manager does not necessarily establish a new Security Context for the group if one member has an outdated Security Context (see Section 2.5.3.1), unless that was already planned or required for other reasons.

All the group members need to acquire new Security Context parameters from the Group Manager. Once having acquired new Security Context parameters, each group member performs the following actions.

- * From then on, it MUST NOT use the current Security Context to start processing new messages for the considered group.
- * It completes any ongoing message processing for the considered group.
- * It derives and install a new Security Context. In particular:
 - It re-derives the keying material stored in its Sender Context and Recipient Contexts (see Section 2.2). The Master Salt used for the re-derivations is the updated Master Salt parameter if provided by the Group Manager, or the empty byte string otherwise.

- It resets its Sender Sequence Number in its Sender Context to 0.
- It re-initializes the Replay Window of each Recipient Context.
- For each ongoing observation where it is an observer client and that it wants to keep active, it resets to 0 the Notification Number of each associated server (see Section 6.1).

From then on, it can resume processing new messages for the considered group. In particular:

- * It MUST use its latest installed Sender Context to protect outgoing messages.
- * It SHOULD use its latest installed Recipient Contexts to process incoming messages, unless application policies admit to temporarily retain and use the old, recent, Security Context (see Section 12.5.1).

The distribution of a new Gid and Master Secret may result in temporarily misaligned Security Contexts among group members. In particular, this may result in a group member not being able to process messages received right after a new Gid and Master Secret have been distributed. A discussion on practical consequences and possible ways to address them, as well as on how to handle the old Security Context, is provided in Section 12.5.

3. The Group Manager

As with OSCORE, endpoints communicating with Group OSCORE need to establish the relevant Security Context. Group OSCORE endpoints need to acquire OSCORE input parameters, information about the group(s) and about other endpoints in the group(s). This document is based on the existence of an entity called Group Manager and responsible for the group, but it does not mandate how the Group Manager interacts with the group members. The list of responsibilities of the Group Manager is compiled in Section 3.3.

A possible Group Manager to use is specified in [I-D.ietf-ace-key-groupcomm-oscore], where the join process is based on the ACE framework for authentication and authorization in constrained environments [I-D.ietf-ace-oauth-authz].

The Group Manager assigns an integer Key Generation Number to each of its groups, identifying the current version of the keying material used in that group. The first Key Generation Number assigned to every group MUST be 0. Separately for each group, the value of the

Key Generation Number increases strictly monotonically, each time the Group Manager distributes new keying material to that group (see Section 3.2). That is, if the current Key Generation Number for a group is X, then X+1 will denote the keying material distributed and used in that group immediately after the current one.

The Group Manager assigns unique Group Identifiers (Gids) to the groups under its control. Also, for each group, the Group Manager assigns unique Sender IDs (and thus Recipient IDs) to the respective group members. According to a hierarchical approach, the Gid value assigned to a group is associated with a dedicated space for the values of Sender ID and Recipient ID of the members of that group. When an endpoint (re-)joins a group, it is provided also with the current Gid to use in the group.

The Group Manager maintains records of the authentication credentials of endpoints in a group, and provides information about the group and its members to other group members and to external entities with selected roles (see Section 3.1). Upon endpoints' joining, the Group Manager collects such authentication credentials and MUST verify proof-of-possession of the respective private key.

An endpoint acquires group data such as the Gid and OSCORE input parameters including its own Sender ID from the Group Manager, and provides information about its authentication credential to the Group Manager, for example upon joining the group.

Furthermore, when joining the group or later on as a group member, an endpoint can retrieve from the Group Manager the authentication credential of the Group Manager as well as the authentication credential and other information associated with other members of the group, with which it can derive the corresponding Recipient Context. Together with the requested authentication credentials, the Group Manager MUST provide the Sender ID of the associated group members and the current Key Generation Number in the group. An application can configure a group member to asynchronously retrieve information about Recipient Contexts, e.g., by Observing [RFC7641] a resource at the Group Manager to get updates on the group membership.

3.1. Support for Additional Entities

The Group Manager MAY serve additional entities acting as signature checkers, e.g., intermediary gateways. These entities do not join a group as members, but can retrieve authentication credentials of group members and other selected group data from the Group Manager, in order to solely verify countersignatures of messages protected in group mode (see Section 8.5).

In order to verify countersignatures of messages in a group, a signature checker needs to retrieve the following information about that group from the Group Manager.

- * The current ID Context (Gid) used in the group.
- * The authentication credentials of the group members and the authentication credential of the Group Manager.

If the signature checker is provided with a CWT for a given entity, then the authentication credential associated with that entity that the signature checker stores and uses is the untagged CWT.

If the signature checker is provided with a chain or a bag of X.509 / C509 certificates or of CWTs for a given entity, then the authentication credential associated with that entity that the signature checker stores and uses is just the end-entity certificate or end-entity untagged CWT.

- * The current Group Encryption Key (see Section 2.1.6).
- * The identifiers of the algorithms used in the group (see Section 2), i.e.: i) Signature Encryption Algorithm and Signature Algorithm; and ii) AEAD Algorithm and Pairwise Key Agreement Algorithm, if the group uses also the pairwise mode.

A signature checker MUST be authorized before it can retrieve such information. To this end, the same method mentioned above based on the ACE framework [I-D.ietf-ace-oauth-authz] can be used.

3.2. Management of Group Keying Material

In order to establish a new Security Context for a group, the Group Manager MUST generate and assign to the group a new Group Identifier (Gid) and a new value for the Master Secret parameter. When doing so, a new value for the Master Salt parameter MAY also be generated and assigned to the group. When establishing the new Security Context, the Group Manager should preserve the current value of the Sender ID of each group member.

The specific group key management scheme used to distribute new keying material is out of the scope of this document. A simple group key management scheme is defined in [I-D.ietf-ace-key-groupcomm-oscure]. When possible, the delivery of rekeying messages should use a reliable transport, in order to reduce chances of group members missing a rekeying instance.

The set of group members should not be assumed as fixed, i.e., the group membership is subject to changes, possibly on a frequent basis.

The Group Manager MUST rekey the group when one or more endpoints leave the group. An endpoint may leave the group at own initiative, or may be evicted from the group by the Group Manager, e.g., in case an endpoint is compromised, or is suspected to be compromised. In either case, rekeying the group excludes such endpoints from future communications in the group, and thus preserves forward security. If a network node is compromised or suspected to be compromised, the Group Manager MUST evict from the group all the endpoints hosted by that node that are member of the group and rekey the group accordingly.

If required by the application, the Group Manager MUST rekey the group also before one or more new joining endpoints are added to the group, thus preserving backward security.

The establishment of the new Security Context for the group takes the following steps.

1. The Group Manager MUST increment the Key Generation Number for the group by 1.
2. The Group Manager MUST build a set of stale Sender IDs including:
 - * The Sender IDs that, during the current Gid, were both assigned to an endpoint and subsequently relinquished (see Section 2.5.3.1).
 - * The current Sender IDs of the group members that the upcoming group rekeying aims to exclude from future group communications, if any.
3. The Group Manager rekeys the group, by distributing:
 - * The new keying material, i.e., the new Master Secret, the new Gid and (optionally) the new Master Salt.
 - * The new Key Generation Number from step 1.
 - * The set of stale Sender IDs from step 2.

Further information may be distributed, depending on the specific group key management scheme used in the group.

When receiving the new group keying material, a group member considers the received stale Sender IDs and performs the following actions.

- * The group member MUST remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group.
- * The group member MUST delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

After that, the group member installs the new keying material and derives the corresponding new Security Context.

A group member might miss one group rekeying or more consecutive instances. As a result, the group member will retain old group keying material with Key Generation Number GEN_OLD. Eventually, the group member can notice the discrepancy, e.g., by repeatedly failing to verify incoming messages, or by explicitly querying the Group Manager for the current Key Generation Number. Once the group member gains knowledge of having missed a group rekeying, it MUST delete the old keying material it stores.

Then, the group member proceeds according to the following steps.

1. The group member retrieves from the Group Manager the current group keying material, together with the current Key Generation Number GEN_NEW. The group member MUST NOT install the obtained group keying material yet.
2. The group member asks the Group Manager for the set of stale Sender IDs.
3. If no exact indication can be obtained from the Group Manager, the group member MUST remove all the authentication credentials from its list of group members' authentication credentials used in the group and MUST delete all its Recipient Contexts used in the group.

Otherwise, the group member MUST remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group, and MUST delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

4. The group member installs the current group keying material, and derives the corresponding new Security Context.

Alternatively, the group member can re-join the group. In such a case, the group member MUST take one of the following two actions.

- * The group member performs steps 2 and 3 above. Then, the group member re-joins the group.
- * The group member re-joins the group with the same roles it currently has in the group, and, during the re-joining process, it asks the Group Manager for the authentication credentials of all the current group members.

Then, given Z the set of authentication credentials received from the Group Manager, the group member removes every authentication credential which is not in Z from its list of group members' authentication credentials used in the group, and deletes each of its Recipient Contexts used in the group that does not include any of the authentication credentials in Z.

By removing authentication credentials and deleting Recipient Contexts associated with stale Sender IDs, it is ensured that a recipient endpoint storing the latest group keying material does not store the authentication credentials of sender endpoints that are not current group members. This in turn allows group members to rely on stored authentication credentials to confidently assert the group membership of sender endpoints, when receiving incoming messages protected in group mode (see Section 8).

3.2.1. Recycling of Identifiers

This section specifies how the Group Manager handles and possibly reassigns Gid values and Sender ID values in a group.

3.2.1.1. Recycling of Group Identifiers

Since the Gid value changes every time a group is rekeyed, it can happen that, after several rekeying instances, the whole space of Gid values has been used for the group in question. When this happens, the Group Manager has no available Gid values to use that have never been assigned to the group during the group's lifetime.

The occurrence of such an event and how long it would take to occur depend on the format and encoding of Gid values used in the group (see, e.g., Appendix C), as well as on the frequency of rekeying instances yielding a change of Gid value. Independently for each group under its control, the Group Manager can take one of the two following approaches.

- * The Group Manager does not reassign Gid values. That is, once the whole space of Gid values has been used for a group, the Group Manager terminates the group and may re-establish a new group.

- * While the Gid value changes every time a group is rekeyed, the Group Manager can reassign Gid values previously used during a group's lifetime. By doing so, the group can continue to exist even once the whole space of Gid values has been used.

The Group Manager MAY support and use this approach. In such a case, the Group Manager MUST take additional actions when handling Gid values and rekeying the group, as specified below.

When a node (re-)joins the group and it is provided with the current Gid to use in the group, the Group Manager considers such a Gid as the Birth Gid of that endpoint for that group. For each group member, the Group Manager MUST store the latest corresponding Birth Gid until that member leaves the group. In case the endpoint has in fact re-joined the group, the newly determined Birth Gid overwrites the one currently stored.

When establishing a new Security Context for the group, the Group Manager takes the additional following step between steps 1 and 2 of Section 3.2.

A. The Group Manager MUST check if the new Gid to be distributed is equal to the Birth Gid of any of the current group members. If any of such "elder members" is found in the group, then:

- The Group Manager MUST evict the elder members from the group. That is, the Group Manager MUST terminate their membership and MUST rekey the group in such a way that the new keying material is not provided to those evicted elder members.

This ensures that an Observe notification [RFC7641] can never successfully match against the Observe requests of two different observations. In fact, the excluded elder members would eventually re-join the group, thus terminating any of their ongoing (long-lasting) observations (see Section 6.1). Therefore, it is ensured by construction that no observer client can have two different ongoing observations such that the two respective Observe requests were protected using the same Partial IV, Gid and Sender ID.

- Until a further following group rekeying, the Group Manager MUST store the list of those latest-evicted elder members. If any of those endpoints re-joins the group before a further following group rekeying occurs, the Group Manager MUST NOT rekey the group upon their re-joining. When one of those endpoints re-joins the group, the Group Manager can rely, e.g., on the ongoing secure communication association to recognize the endpoint as included in the stored list.

3.2.1.2. Recycling of Sender IDs

From the moment when a Gid is assigned to a group until the moment a new Gid is assigned to that same group, the Group Manager MUST NOT reassign a Sender ID within the group. This prevents to reuse a Sender ID ('kid') with the same Gid, Master Secret and Master Salt. Within this restriction, the Group Manager can assign a Sender ID used under an old Gid value (including under a same, recycled Gid value), thus avoiding Sender ID values to irrecoverably grow in size.

Even when an endpoint joining a group is recognized as a current member of that group, e.g., through the ongoing secure communication association, the Group Manager MUST assign a new Sender ID different than the one currently used by the endpoint in the group, unless the group is rekeyed first and a new Gid value is established.

3.2.1.3. Relation between Identifiers and Keying Material

Figure 2 overviews the different identifiers and keying material components, considering their relation and possible reuse across group rekeying.

Components changed in lockstep
upon a group rekeying

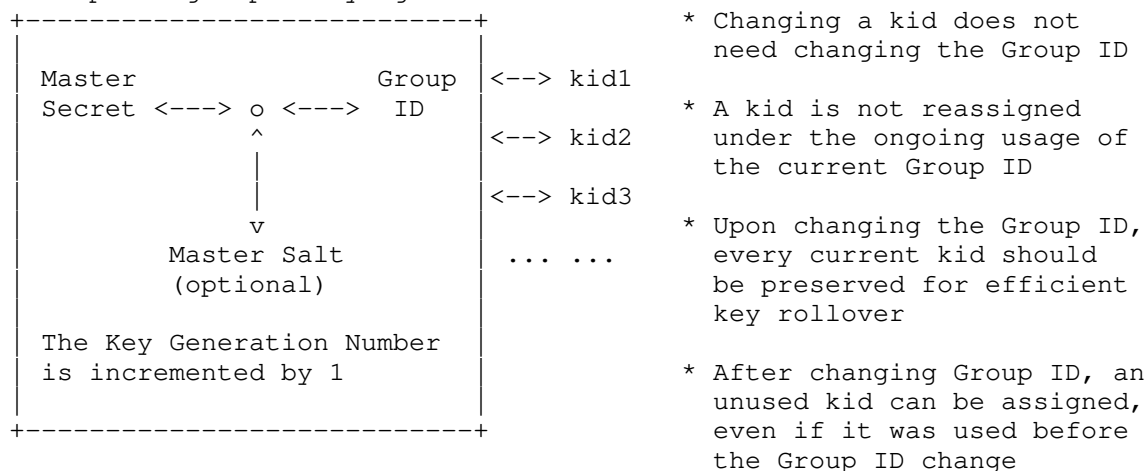


Figure 2: Relations among keying material components.

3.3. Responsibilities of the Group Manager

The Group Manager is responsible for performing the following tasks:

1. Creating and managing OSCORE groups. This includes the assignment of a Gid to every newly created group, ensuring uniqueness of Gids within the set of its OSCORE groups and, optionally, the secure recycling of Gids.
2. Defining policies for authorizing the joining of its OSCORE groups.
3. Handling the join process to add new endpoints as group members.
4. Establishing the Common Context part of the Security Context, and providing it to authorized group members during the join process, together with the corresponding Sender Context.
5. Updating the Key Generation Number and the Gid of its OSCORE groups, upon renewing the respective Security Context.
6. Generating and managing Sender IDs within its OSCORE groups, as well as assigning and providing them to new endpoints during the join process, or to current group members upon request of renewal or re-joining. This includes ensuring that:
 - * Each Sender ID is unique within each of the OSCORE groups;
 - * Each Sender ID is not reassigned within the same group since the latest time when the current Gid value was assigned to the group. That is, the Sender ID is not reassigned even to a current group member re-joining the same group, without a rekeying happening first.
7. Defining communication policies for each of its OSCORE groups, and signaling them to new endpoints during the join process.
8. Renewing the Security Context of an OSCORE group upon membership change, by revoking and renewing common security parameters and keying material (rekeying).
9. 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.
10. Assisting a group member that has missed a group rekeying instance to understand which authentication credentials and Recipient Contexts to delete, as associated with former group members.

11. Acting as key repository, in order to handle the authentication credentials of the members of its OSCORE groups, and providing such authentication credentials to other members of the same group upon request. The actual storage of authentication credentials may be entrusted to a separate secure storage device or service.
12. Validating that the format and parameters of authentication credentials of group members are consistent with the public key algorithm and related parameters used in the respective OSCORE group.

The Group Manager specified in [I-D.ietf-ace-key-groupcomm-oscore] provides these functionalities.

4. The COSE Object

Building on Section 5 of [RFC8613], this section defines how to use COSE [I-D.ietf-cose-rfc8152bis-struct] to wrap and protect data in the original message. OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Associated Data (AEAD) algorithm. Unless otherwise specified, the following modifications apply for both the group mode and the pairwise mode of Group OSCORE.

4.1. Countersignature

When protecting a message in group mode, the 'unprotected' field MUST additionally include the following parameter:

- * COSE_CounterSignature0: its value is set to the encrypted countersignature of the COSE object, namely ENC_SIGNATURE. That is:
 - The countersignature of the COSE object, namely SIGNATURE, is computed by the sender as described in Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign], by using its private key and according to the Signature Algorithm in the Security Context.

In particular, the Countersign_structure contains the context text string "CounterSignature0", the external_aad as defined in Section 4.3 of this document, and the ciphertext of the COSE object as payload.

- The encrypted countersignature, namely ENC_SIGNATURE, is computed as

ENC_SIGNATURE = SIGNATURE XOR KEYSTREAM

where KEYSTREAM is derived as per Section 4.1.1.

4.1.1. Keystream Derivation

The following defines how an endpoint derives the keystream KEYSTREAM, used to encrypt/decrypt the countersignature of an outgoing/incoming message M protected in group mode.

The keystream SHALL be derived as follows, by using the HKDF Algorithm from the Common Context (see Section 3.2 of [RFC8613]), which consists of composing the HKDF-Extract and HKDF-Expand steps [RFC5869].

KEYSTREAM = HKDF(salt, IKM, info, L)

The input parameters of HKDF are as follows.

- * salt takes as value the Partial IV (PIV) used to protect M. Note that, if M is a response, salt takes as value either: i) the fresh Partial IV generated by the server and included in the response; or ii) the same Partial IV of the request generated by the client and not included in the response.
- * IKM is the Group Encryption Key from the Common Context (see Section 2.1.6).
- * info is the serialization of a CBOR array consisting of (the notation follows [RFC8610]):

```
info = [  
  id : bstr,  
  id_context : bstr,  
  type : bool,  
  L: uint  
]
```

where:

- * id is the Sender ID of the endpoint that generated PIV.
- * id_context is the ID Context (Gid) used when protecting M.

Note that, in case of group rekeying, a server might use a different Gid when protecting a response, compared to the Gid that it used to verify (that the client used to protect) the request, see Section 8.3.

- * type is the CBOR simple value "true" (0xf5) if M is a request, or the CBOR simple value "false" (0xf4) otherwise.
- * L is the size of the countersignature, as per Signature Algorithm from the Common Context (see Section 2.1.5), in bytes.

4.1.2. Clarifications on Using a Countersignature

Note that the literature commonly refers to a countersignature as a signature computed by an entity A over a document already protected by a different entity B.

However, the COSE_Countersignature0 structure belongs to the set of abbreviated countersignatures defined in Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign], which were designed primarily to deal with the problem of encrypted group messaging, but where it is required to know who originated the message.

Since the parameters for computing or verifying the abbreviated countersignature generated by A are provided by the same context used to describe the security processing performed by B and to be countersigned, these structures are applicable also when the two entities A and B are actually the same one, like the sender of a Group OSCORE message protected in group mode.

4.2. The 'kid' and 'kid context' parameters

The value of the 'kid' parameter in the 'unprotected' field of response messages MUST be set to the Sender ID of the endpoint transmitting the message, if the request was protected in group mode. That is, unlike in [RFC8613], the 'kid' parameter is always present in responses to a request that was protected in group mode.

The value of the 'kid context' parameter in the 'unprotected' field of requests messages MUST be set to the ID Context, i.e., the Group Identifier value (Gid) of the group. That is, unlike in [RFC8613], the 'kid context' parameter is always present in requests.

4.3. external_aad

The external_aad of the Additional Authenticated Data (AAD) is different compared to OSCORE [RFC8613], and is defined in this section.

The same `external_aad` structure is used in group mode and pairwise mode for authenticated encryption/decryption (see Section 5.3 of [I-D.ietf-cose-rfc8152bis-struct]), as well as in group mode for computing and verifying the countersignature (see Section 4.4 of [I-D.ietf-cose-rfc8152bis-struct]).

In particular, the `external_aad` includes also the Signature Algorithm, the Signature Encryption Algorithm, the Pairwise Key Agreement Algorithm, the value of the 'kid context' in the COSE object of the request, the OSCORE option of the protected message, the sender's authentication credential, and the Group Manager's authentication credential.

The `external_aad` SHALL be a CBOR array wrapped in a `bstr` object as defined below, following the notation of [RFC8610]:

```
external_aad = bstr .cbor aad_array

aad_array = [
  oscore_version : uint,
  algorithms : [alg_aead : int / tstr / null,
               alg_signature_enc : int / tstr / null,
               alg_signature : int / tstr / null,
               alg_pairwise_key_agreement : int / tstr / null],
  request_kid : bstr,
  request_piv : bstr,
  options : bstr,
  request_kid_context : bstr,
  OSCORE_option: bstr,
  sender_cred: bstr,
  gm_cred: bstr / null
]
```

Figure 3: `external_aad`

Compared with Section 5.4 of [RFC8613], the `aad_array` has the following differences.

- * The 'algorithms' array is extended as follows.

The parameter 'alg_aead' MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the pairwise mode, regardless whether the endpoint supports the pairwise mode or not. Otherwise, this parameter MUST encode the value of AEAD Algorithm from the Common Context (see Section 2.1.1), as per Section 5.4 of [RFC8613].

Furthermore, the 'algorithms' array additionally includes:

- 'alg_signature_enc', which specifies Signature Encryption Algorithm from the Common Context (see Section 2.1.5). This parameter MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the group mode, regardless whether the endpoint supports the group mode or not. Otherwise, this parameter MUST encode the value of Signature Encryption Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this AEAD algorithm.
 - 'alg_signature', which specifies Signature Algorithm from the Common Context (see Section 2.1.5). This parameter MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the group mode, regardless whether the endpoint supports the group mode or not. Otherwise, this parameter MUST encode the value of Signature Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this signature algorithm.
 - 'alg_pairwise_key_agreement', which specifies Pairwise Key Agreement Algorithm from the Common Context (see Section 2.1.5). This parameter MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the pairwise mode, regardless whether the endpoint supports the pairwise mode or not. Otherwise, this parameter MUST encode the value of Pairwise Key Agreement Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this HKDF algorithm.
- * The new element 'request_kid_context' contains the value of the 'kid context' in the COSE object of the request (see Section 4.2).

In case Observe [RFC7641] is used, this enables endpoints to safely keep an observation active beyond a possible change of Gid (i.e., of ID Context), following a group rekeying (see Section 3.2). In fact, it ensures that every notification cryptographically matches with only one observation request, rather than with multiple ones that were protected with different keying material but share the same 'request_kid' and 'request_piv' values.

- * The new element 'OSCORE_option', containing the value of the OSCORE Option present in the protected message, encoded as a binary string. This prevents the attack described in Section 12.7 when using the group mode, as further explained in Section 12.7.2.

Note for implementation: this construction requires the OSCORE option of the message to be generated and finalized before computing the ciphertext of the COSE_Encrypt0 object (when using the group mode or the pairwise mode) and before calculating the countersignature (when using the group mode). Also, the aad_array needs to be large enough to contain the largest possible OSCORE option.

- * The new element 'sender_cred', containing the sender's authentication credential. This parameter MUST be set to a CBOR byte string, which encodes the sender's authentication credential in its original binary representation made available to other endpoints in the group (see Section 2.3).
- * The new element 'gm_cred', containing the Group Manager's authentication credential. If no Group Manager maintains the group, this parameter MUST encode the CBOR simple value "null" (0xf6). Otherwise, this parameter MUST be set to a CBOR byte string, which encodes the Group Manager's authentication credential in its original binary representation made available to other endpoints in the group (see Section 2.3). This prevents the attack described in Section 12.8.

5. OSCORE Header Compression

The OSCORE header compression defined in Section 6 of [RFC8613] is used, with the following differences.

- * The payload of the OSCORE message SHALL encode the ciphertext of the COSE_Encrypt0 object. In the group mode, the ciphertext above is concatenated with the value of the COSE_CounterSignature0 of the COSE object, computed as described in Section 4.1.
- * This document defines the usage of the sixth least significant bit, called "Group Flag", in the first byte of the OSCORE option containing the OSCORE flag bits. This flag bit is specified in Section 13.1.
- * The Group Flag MUST be set to 1 if the OSCORE message is protected using the group mode (see Section 8).
- * The Group Flag MUST be set to 0 if the OSCORE message is protected using the pairwise mode (see Section 9). The Group Flag MUST also be set to 0 for ordinary OSCORE messages processed according to [RFC8613].

5.1. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples of Group OSCORE used in group mode (see Section 5.1.1) or in pairwise mode (see Section 5.1.2).

The examples assume that the COSE_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the examples do not include the full CoAP unprotected message or the full Security Context, 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 5 and divided into two parts, since the object is transported in two CoAP fields: OSCORE option and payload.

The examples assume that the plaintext (see Section 5.3 of [RFC8613]) is 6 bytes long, and that the AEAD tag is 8 bytes long, hence resulting in a ciphertext which is 14 bytes long. When using the group mode, the COSE_CounterSignature0 byte string as described in Section 4 is assumed to be 64 bytes long.

5.1.1. Examples in Group Mode

- * Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c.

- * Before compression (96 bytes):

```
[
  h'',
  { 4:h'25', 6:h'05', 10:h'44616c', 11:h'de9e ... f1' },
  h'aea0155667924dff8a24e4cb35b9'
]
```

- * After compression (85 bytes):

Flag byte: 0b00111001 = 0x39 (1 byte)

Option Value: 0x39 05 03 44 61 6c 25 (7 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 de9e ... f1
(14 bytes + size of the encrypted countersignature)

- * Response with ciphertext = 0x60b035059d9ef5667c5a0710823b, kid = 0x52 and no Partial IV.

* Before compression (88 bytes):

```
[
  h'',
  { 4:h'52', 11:h'cale ... b3' },
  h'60b035059d9ef5667c5a0710823b'
]
```

* After compression (80 bytes):

Flag byte: 0b00101000 = 0x28 (1 byte)

Option Value: 0x28 52 (2 bytes)

Payload: 0x60b035059d9ef5667c5a0710823b cale ... b3
(14 bytes + size of the encrypted countersignature)

5.1.2. Examples in Pairwise Mode

* Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c.

* Before compression (29 bytes):

```
[
  h'',
  { 4:h'25', 6:h'05', 10:h'44616c' },
  h'aea0155667924dff8a24e4cb35b9'
]
```

* After compression (21 bytes):

Flag byte: 0b00011001 = 0x19 (1 byte)

Option Value: 0x19 05 03 44 61 6c 25 (7 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

* Response with ciphertext = 0x60b035059d9ef5667c5a0710823b and no Partial IV.

* Before compression (18 bytes):

```
[
  h'',
  {},
  h'60b035059d9ef5667c5a0710823b'
]
```

* After compression (14 bytes):

Flag byte: 0b00000000 = 0x00 (1 byte)

Option Value: 0x (0 bytes)

Payload: 0x60b035059d9ef5667c5a0710823b (14 bytes)

6. Message Binding, Sequence Numbers, Freshness and Replay Protection

The requirements and properties described in Section 7 of [RFC8613] also apply to Group OSCORE. In particular, Group OSCORE provides message binding of responses to requests, which enables absolute freshness of responses that are not notifications, relative freshness of requests and notification responses, and replay protection of requests. In addition, the following holds for Group OSCORE.

6.1. Supporting Observe

When Observe [RFC7641] is used, a client maintains for each ongoing observation one Notification Number for each different server. Then, separately for each server, the client uses the associated Notification Number to perform ordering and replay protection of notifications received from that server (see Section 8.4.1).

Group OSCORE allows to preserve an observation active indefinitely, even in case the group is rekeyed, with consequent change of ID Context, or in case the observer client obtains a new Sender ID.

As defined in Section 8 when discussing support for Observe, this is achieved by the client and server(s) storing the 'kid' and 'kid context' used in the original Observe request, throughout the whole duration of the observation.

Upon leaving the group or before re-joining the group, a group member MUST terminate all the ongoing observations that it has started in the group as observer client.

6.2. Update of Replay Window

Sender Sequence Numbers seen by a server as Partial IV values in request messages can spontaneously increase at a fast pace, for example when a client exchanges unicast messages with other servers using the Group OSCORE Security Context. As in OSCORE [RFC8613], a server always needs to accept such increases and accordingly updates the Replay Window in each of its Recipient Contexts.

As discussed in Section 2.5.1, a newly created Recipient Context would have an invalid Replay Window, if its installation has required to delete another Recipient Context. Hence, the server is not able to verify if a request from the client associated with the new Recipient Context is a replay. When this happens, the server **MUST** validate the Replay Window of the new Recipient Context, before accepting messages from the associated client (see Section 2.5.1).

Furthermore, when the Group Manager establishes a new Security Context for the group (see Section 2.5.3.2), every server re-initializes the Replay Window in each of its Recipient Contexts.

6.3. Message Freshness

When receiving a request from a client for the first time, the server is not synchronized with the client's Sender Sequence Number, i.e., it is not able to verify if that request is fresh. This applies to a server that has just joined the group, with respect to already present clients, and recurs as new clients are added as group members.

During its operations in the group, the server may also lose synchronization with a client's Sender Sequence Number. This can happen, for instance, if the server has rebooted or has deleted its previously synchronized version of the Recipient Context for that client (see Section 2.5.1).

If the application requires message freshness, e.g., according to time- or event-based policies, the server has to (re-)synchronize with a client's Sender Sequence Number before delivering request messages from that client to the application. To this end, the server can use the approach in Section 10 based on the Echo Option for CoAP [RFC9175], as a variant of the approach defined in Appendix B.1.2 of [RFC8613] applicable to Group OSCORE.

7. Message Reception

Upon receiving a protected message, a recipient endpoint retrieves a Security Context as in [RFC8613]. An endpoint **MUST** be able to distinguish between a Security Context to process OSCORE messages as in [RFC8613] and a Group OSCORE Security Context to process Group OSCORE messages as defined in this document.

To this end, an endpoint can take into account the different structure of the Security Context defined in Section 2, for example based on the presence of Signature Algorithm and/or Pairwise Key Agreement Algorithm in the Common Context. Alternatively implementations can use an additional parameter in the Security Context, to explicitly signal that it is intended for processing Group OSCORE messages.

If either of the following conditions holds, a recipient endpoint MUST discard the incoming protected message:

- * The Group Flag is set to 0, and the recipient endpoint retrieves a Security Context which is both valid to process the message and also associated with an OSCORE group, but the endpoint does not support the pairwise mode.
- * The Group Flag is set to 1, and the recipient endpoint retrieves a Security Context which is both valid to process the message and also associated with an OSCORE group, but the endpoint does not support the group mode.
- * The Group Flag is set to 1, and the recipient endpoint can not retrieve a Security Context which is both valid to process the message and also associated with an OSCORE group.

As per Section 6.1 of [RFC8613], this holds also when retrieving a Security Context which is valid but not associated with an OSCORE group. Future specifications may define how to process incoming messages protected with a Security Contexts as in [RFC8613], when the Group Flag bit is set to 1.

Otherwise, if a Security Context associated with an OSCORE group and valid to process the message is retrieved, the recipient endpoint processes the message with Group OSCORE, using the group mode (see Section 8) if the Group Flag is set to 1, or the pairwise mode (see Section 9) if the Group Flag is set to 0.

Note that, if the Group Flag is set to 0, and the recipient endpoint retrieves a Security Context which is valid to process the message but is not associated with an OSCORE group, then the message is processed according to [RFC8613].

8. Message Processing in Group Mode

When using the group mode, messages are protected and processed as specified in [RFC8613], with the modifications described in this section. The security objectives of the group mode are discussed in Appendix A.2.

The Group Manager indicates that the group uses (also) the group mode, as part of the group data provided to candidate group members when joining the group.

During all the steps of the message processing, an endpoint MUST use the same Security Context for the considered group. That is, an endpoint MUST NOT install a new Security Context for that group (see Section 2.5.3.2) until the message processing is completed.

The group mode MUST be used to protect group requests intended for multiple recipients or for the whole group. This includes both requests directly addressed to multiple recipients, e.g., sent by the client over multicast, as well as requests sent by the client over unicast to a proxy, that forwards them to the intended recipients over multicast [I-D.ietf-core-groupcomm-bis]. For encryption and decryption operations, the Signature Encryption Algorithm from the Common Context is used.

As per [RFC7252][I-D.ietf-core-groupcomm-bis], group requests sent over multicast MUST be Non-confirmable, and thus are not retransmitted by the CoAP messaging layer. Instead, applications should store such outgoing messages for a predefined, sufficient amount of time, in order to correctly perform potential retransmissions at the application layer. According to Section 5.2.3 of [RFC7252], responses to Non-confirmable group requests SHOULD also be Non-confirmable, but endpoints MUST be prepared to receive Confirmable responses in reply to a Non-confirmable group request. Confirmable group requests are acknowledged when sent over non-multicast transports, as specified in [RFC7252].

Furthermore, endpoints in the group locally perform error handling and processing of invalid messages according to the same principles adopted in [RFC8613]. However, a recipient MUST stop processing and reject any message which is malformed and does not follow the format specified in Section 4 of this document, or which is not cryptographically validated in a successful way.

In either case, it is RECOMMENDED that a server does not send back any error message in reply to a received request, if any of the two following conditions holds:

- * The server is not able to identify the received request as a group request, i.e., as sent to all servers in the group.
- * The server identifies the received request as a group request.

This prevents servers from replying with multiple error messages to a client sending a group request, so avoiding the risk of flooding and possibly congesting the network.

8.1. Protecting the Request

A client transmits a secure group request as described in Section 8.1 of [RFC8613], with the following modifications.

- * In step 2, the Additional Authenticated Data is modified as described in Section 4 of this document.
- * In step 4, the encryption of the COSE object is modified as described in Section 4 of this document. The encoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, the Group Flag MUST be set to 1. The Signature Encryption Algorithm from the Common Context MUST be used.
- * In step 5, the countersignature is computed and the format of the OSCORE message is modified as described in Section 4 and Section 5 of this document. In particular the payload of the OSCORE message includes also the encrypted countersignature (see Section 4.1).

8.1.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds for each newly started observation.

- * If the client intends to keep the observation active beyond a possible change of Sender ID, the client MUST store the value of the 'kid' parameter from the original Observe request, and retain it for the whole duration of the observation. Even in case the client is individually rekeyed and receives a new Sender ID from the Group Manager (see Section 2.5.3.1), the client MUST NOT update the stored value associated with a particular Observe request.
- * If the client intends to keep the observation active beyond a possible change of ID Context following a group rekeying (see Section 3.2), then the following applies.
 - The client MUST store the value of the 'kid context' parameter from the original Observe request, and retain it for the whole duration of the observation. Upon establishing a new Security Context with a new Gid as ID Context (see Section 2.5.3.2), the client MUST NOT update the stored value associated with a particular Observe request.

- The client MUST store an invariant identifier of the group, which is immutable even in case the Security Context of the group is re-established. For example, this invariant identifier can be the "group name" in [I-D.ietf-ace-key-groupcomm-oscore], where it is used for joining the group and retrieving the current group keying material from the Group Manager.

After a group rekeying, such an invariant information makes it simpler for the observer client to retrieve the current group keying material from the Group Manager, in case the client has missed both the rekeying messages and the first observe notification protected with the new Security Context (see Section 8.3.1).

8.2. Verifying the Request

Upon receiving a secure group request with the Group Flag set to 1, following the procedure in Section 7, a server proceeds as described in Section 8.2 of [RFC8613], with the following modifications.

- * In step 2, the decoding of the compressed COSE object follows Section 5 of this document. In particular:
 - If the server discards the request due to not retrieving a Security Context associated with the OSCORE group, the server MAY respond with a 4.01 (Unauthorized) error message. When doing so, the server MAY set an Outer Max-Age option with value zero, and MAY include a descriptive string as diagnostic payload.
 - If the received 'kid context' matches an existing ID Context (Gid) but the received 'kid' does not match any Recipient ID in this Security Context, then the server MAY create a new Recipient Context for this Recipient ID and initialize it according to Section 3 of [RFC8613], and also retrieve the authentication credential associated with the Recipient ID to be stored in the new Recipient Context. Such a configuration is application specific. If the application does not specify dynamic derivation of new Recipient Contexts, then the server SHALL stop processing the request.
- * In step 4, the Additional Authenticated Data is modified as described in Section 4 of this document.
- * In step 6, the server also verifies the countersignature, by using the public key from the client's authentication credential stored in the associated Recipient Context. In particular:

- If the server does not have the public key of the client yet, the server MUST stop processing the request and MAY respond with a 5.03 (Service Unavailable) response. The response MAY include a Max-Age Option, indicating to the client the number of seconds after which to retry. If the Max-Age Option is not present, a retry time of 60 seconds will be assumed by the client, as default value defined in Section 5.10.5 of [RFC7252].
- The server MUST perform signature verification before decrypting the COSE object, as defined below. Implementations that cannot perform the two steps in this order MUST ensure that no access to the plaintext is possible before a successful signature verification and MUST prevent any possible leak of time-related information that can yield side-channel attacks.
- The server retrieves the encrypted countersignature ENC_SIGNATURE from the message payload, and computes the original countersignature SIGNATURE as

SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

where KEYSTREAM is derived as per Section 4.1.1.

The server verifies the original countersignature SIGNATURE.

- If the signature verification fails, the server SHALL stop processing the request, SHALL NOT update the Replay Window, and MAY respond with a 4.00 (Bad Request) response. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain a string, which, if present, MUST be "Decryption failed" as if the decryption had failed.
 - When decrypting the COSE object using the Recipient Key, the Signature Encryption Algorithm from the Common Context MUST be used.
- * Additionally, if the used Recipient Context was created upon receiving this group request and the message is not verified successfully, the server MAY delete that Recipient Context. Such a configuration, which is specified by the application, mitigates attacks that aim at overloading the server's storage.

A server SHOULD NOT process a request if the received Recipient ID ('kid') is equal to its own Sender ID in its own Sender Context. For an example where this is not fulfilled, see Section 7.2.1 of [I-D.ietf-core-observe-multicast-notifications].

8.2.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds for each newly started observation.

- * The server MUST store the value of the 'kid' parameter from the original Observe request, and retain it for the whole duration of the observation. The server MUST NOT update the stored value of a 'kid' parameter associated with a particular Observe request, even in case the observer client is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.5.3.1).
- * The server MUST store the value of the 'kid context' parameter from the original Observe request, and retain it for the whole duration of the observation, beyond a possible change of ID Context following a group rekeying (see Section 3.2). That is, upon establishing a new Security Context with a new Gid as ID Context (see Section 2.5.3.2), the server MUST NOT update the stored value associated with the ongoing observation.

8.3. Protecting the Response

If a server generates a CoAP message in response to a Group OSCORE request, then the server SHALL follow the description in Section 8.3 of [RFC8613], with the modifications described in this section.

Note that the server always protects a response with the Sender Context from its latest Security Context, and that establishing a new Security Context resets the Sender Sequence Number to 0 (see Section 3.2).

- * In step 2, the Additional Authenticated Data is modified as described in Section 4 of this document.
- * In step 3, if the server is using a different Security Context for the response compared to what was used to verify the request (see Section 3.2), then the server MUST include its Sender Sequence Number as Partial IV in the response and use it to build the AEAD nonce to protect the response. This prevents the AEAD nonce from the request from being reused.
- * In step 4, the encryption of the COSE object is modified as described in Section 4 of this document. The encoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, the Group Flag MUST be set to 1. The Signature Encryption Algorithm from the Common Context MUST be used.

If the server is using a different ID Context (Gid) for the response compared to what was used to verify the request (see Section 3.2), then the new ID Context MUST be included in the 'kid context' parameter of the response.

The server can obtain a new Sender ID from the Group Manager, when individually rekeyed (see Section 2.5.3.1) or when re-joining the group. In such a case, the server can help the client to synchronize, by including the 'kid' parameter in a response protected in group mode, even when the request was protected in pairwise mode (see Section 9.3).

That is, when responding to a request protected in pairwise mode, the server SHOULD include the 'kid' parameter in a response protected in group mode, if it is replying to that client for the first time since the assignment of its new Sender ID.

- * In step 5, the countersignature is computed and the format of the OSCORE message is modified as described in Section 4 and Section 5 of this document. In particular the payload of the OSCORE message includes also the encrypted countersignature (see Section 4.1).

8.3.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds when protecting notifications for an ongoing observation.

- * The server MUST use the stored value of the 'kid' parameter from the original Observe request (see Section 8.2.1), as value for the 'request_kid' parameter in the external_aad structure (see Section 4.3).
- * The server MUST use the stored value of the 'kid context' parameter from the original Observe request (see Section 8.2.1), as value for the 'request_kid_context' parameter in the external_aad structure (see Section 4.3).

Furthermore, the server may have ongoing observations started by Observe requests protected with an old Security Context. After completing the establishment of a new Security Context, the server MUST protect the following notifications with the Sender Context of the new Security Context.

For each ongoing observation, the server can help the client to synchronize, by including also the 'kid context' parameter in notifications following a group rekeying, with value set to the ID Context (Gid) of the new Security Context.

If there is a known upper limit to the duration of a group rekeying, the server SHOULD include the 'kid context' parameter during that time. Otherwise, the server SHOULD include it until the Max-Age has expired for the last notification sent before the installation of the new Security Context.

8.4. Verifying the Response

Upon receiving a secure response message with the Group Flag set to 1, following the procedure in Section 7, the client proceeds as described in Section 8.4 of [RFC8613], with the following modifications.

Note that a client may receive a response protected with a Security Context different from the one used to protect the corresponding request, and that, upon the establishment of a new Security Context, the client re-initializes its Replay Windows in its Recipient Contexts (see Section 3.2).

- * In step 2, the decoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, a 'kid' may not be present, if the response is a reply to a request protected in pairwise mode. In such a case, the client assumes the response 'kid' to be the Recipient ID for the server to which the request protected in pairwise mode was intended for.

If the response 'kid context' matches an existing ID Context (Gid) but the received/assumed 'kid' does not match any Recipient ID in this Security Context, then the client MAY create a new Recipient Context for this Recipient ID and initialize it according to Section 3 of [RFC8613], and also retrieve the authentication credential associated with the Recipient ID to be stored in the new Recipient Context. If the application does not specify dynamic derivation of new Recipient Contexts, then the client SHALL stop processing the response.

- * In step 3, the Additional Authenticated Data is modified as described in Section 4 of this document.
- * In step 5, the client also verifies the countersignature, by using the public key from the server's authentication credential stored in the associated Recipient Context. In particular:

- The client MUST perform signature verification before decrypting the COSE object, as defined below. Implementations that cannot perform the two steps in this order MUST ensure that no access to the plaintext is possible before a successful signature verification and MUST prevent any possible leak of time-related information that can yield side-channel attacks.
- The client retrieves the encrypted countersignature ENC_SIGNATURE from the message payload, and computes the original countersignature SIGNATURE as

SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

where KEYSTREAM is derived as per Section 4.1.1.

The client verifies the original countersignature SIGNATURE.

- If the verification of the countersignature fails, the server SHALL stop processing the response, and SHALL NOT update the Notification Number associated with the server if the response is an Observe notification [RFC7641].
- After a successful verification of the countersignature, the client performs also the following actions if the response is not an Observe notification.
 - o In case the request was protected in pairwise mode and the 'kid' parameter is present in the response, the client checks whether this received 'kid' is equal to the expected 'kid', i.e., the known Recipient ID for the server to which the request was intended for.
 - o In case the request was protected in pairwise mode and the 'kid' parameter is not present in the response, the client checks whether the server that has sent the response is the same one to which the request was intended for. This can be done by checking that the public key used to verify the countersignature of the response is equal to the public key included in the authentication credential Recipient Auth Cred, which was taken as input to derive the Pairwise Sender Key used for protecting the request (see Section 2.4.1).

In either case, if the client determines that the response has come from a different server than the expected one, then the client SHALL discard the response and SHALL NOT deliver it to the application. Otherwise, the client hereafter considers the received 'kid' as the current Recipient ID for the server.

- When decrypting the COSE object using the Recipient Key, the Signature Encryption Algorithm from the Common Context MUST be used.
- * Additionally, if the used Recipient Context was created upon receiving this response and the message is not verified successfully, the client MAY delete that Recipient Context. Such a configuration, which is specified by the application, mitigates attacks that aim at overloading the client's storage.

8.4.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds when verifying notifications for an ongoing observation.

- * The client MUST use the stored value of the 'kid' parameter from the original Observe request (see Section 8.1.1), as value for the 'request_kid' parameter in the external_aad structure (see Section 4.3).
- * The client MUST use the stored value of the 'kid context' parameter from the original Observe request (see Section 8.1.1), as value for the 'request_kid_context' parameter in the external_aad structure (see Section 4.3).

This ensures that the client can correctly verify notifications, even in case it is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.5.3.1), as well as when it installs a new Security Context with a new ID Context (Gid) following a group rekeying (see Section 3.2).

- * The ordering and the replay protection of notifications received from a server are performed as per Sections 4.1.3.5.2 and 7.4.1 of [RFC8613], by using the Notification Number associated with that server for the observation in question. In addition, the client performs the following actions for each ongoing observation.
 - When receiving the first valid notification from a server, the client MUST store the current kid "kid1" of that server for the observation in question. If the 'kid' field is included in the OSCORE option of the notification, its value specifies "kid1". If the Observe request was protected in pairwise mode (see Section 9.3), the 'kid' field may not be present in the OSCORE option of the notification (see Section 4.2). In this case, the client assumes "kid1" to be the Recipient ID for the server to which the Observe request was intended for.

- When receiving another valid notification from the same server - which can be identified and recognized through the same public key used to verify the countersignature and included in the server's authentication credential - the client determines the current kid "kid2" of the server as above for "kid1", and MUST check whether "kid2" is equal to the stored "kid1". If "kid1" and "kid2" are different, the client MUST cancel or re-register the observation in question.

Note that, if "kid2" is different from "kid1" and the 'kid' field is omitted from the notification - which is possible if the Observe request was protected in pairwise mode - then the client will compute a wrong keystream to decrypt the countersignature (i.e., by using "kid1" rather than "kid2" in the 'id' field of the 'info' array in Section 4.1.1), thus subsequently failing to verify the countersignature and discarding the notification.

This ensures that the client remains able to correctly perform the ordering and replay protection of notifications, even in case a server legitimately starts using a new Sender ID, as received from the Group Manager when individually rekeyed (see Section 2.5.3.1) or when re-joining the group.

8.5. External Signature Checkers

When receiving a message protected in group mode, a signature checker (see Section 3.1) proceeds as follows.

- * The signature checker retrieves the encrypted countersignature ENC_SIGNATURE from the message payload, and computes the original countersignature SIGNATURE as

$$\text{SIGNATURE} = \text{ENC_SIGNATURE} \text{ XOR } \text{KEYSTREAM}$$

where KEYSTREAM is derived as per Section 4.1.1.

- * The signature checker verifies the original countersignature SIGNATURE, by using the public key of the sender endpoint as included in that endpoint's authentication credential. The signature checker determines the right authentication credential based on the ID Context (Gid) and the Sender ID of the sender endpoint.

Note that the following applies when attempting to verify the countersignature of a response message.

- * The response may not include a Partial IV and/or an ID Context. In such a case, the signature checker considers the same values from the corresponding request, i.e., the request matching with the response by CoAP Token value.
- * The response may not include a Sender ID. This can happen when the response protected in group mode matches a request protected in pairwise mode (see Section 9.1), with a case in point provided by [I-D.amsuess-core-cachable-oscore]. In such a case, the signature checker needs to use other means (e.g., source addressing information of the server endpoint) to identify the correct authentication credential including the public key to use for verifying the countersignature of the response.

The particular actions following a successful or unsuccessful verification of the countersignature are application specific and out of the scope of this document.

9. Message Processing in Pairwise Mode

When using the pairwise mode of Group OSCORE, messages are protected and processed as in [RFC8613], with the modifications described in this section. The security objectives of the pairwise mode are discussed in Appendix A.2.

The pairwise mode takes advantage of an existing Security Context for the group mode to establish a Security Context shared exclusively with any other member. In order to use the pairwise mode in a group that uses also the group mode, the signature scheme of the group mode MUST support a combined signature and encryption scheme. This can be, for example, signature using ECDSA, and encryption using AES-CCM with a key derived with ECDH. For encryption and decryption operations, the AEAD Algorithm from the Common Context is used (see Section 2.1.1).

The pairwise mode does not support the use of additional entities acting as verifiers of source authentication and integrity of group messages, such as intermediary gateways (see Section 3).

An endpoint implementing only a silent server does not support the pairwise mode.

If the signature algorithm used in the group supports ECDH (e.g., ECDSA, EdDSA), the pairwise mode MUST be supported by endpoints that use the CoAP Echo Option [RFC9175] and/or block-wise transfers [RFC7959], for instance for responses after the first block-wise request, which possibly targets all servers in the group and includes the CoAP Block2 option (see Section 3.8 of

[I-D.ietf-core-groupcomm-bis])). This prevents the attack described in Section 12.9, which leverages requests sent over unicast to a single group member and protected with the group mode.

Senders cannot use the pairwise mode to protect a message intended for multiple recipients. In fact, the pairwise mode is defined only between two endpoints and the keying material is thus only available to one recipient.

However, a sender can use the pairwise mode to protect a message sent to (but not intended for) multiple recipients, if interested in a response from only one of them. For instance, this is useful to support the address discovery service defined in Section 9.1, when a single 'kid' value is indicated in the payload of a request sent to multiple recipients, e.g., over multicast.

The Group Manager indicates that the group uses (also) the pairwise mode, as part of the group data provided to candidate group members when joining the group.

9.1. Pre-Conditions

In order to protect an outgoing message in pairwise mode, the sender needs to know the authentication credential and the Recipient ID for the recipient endpoint, as stored in the Recipient Context associated with that endpoint (see Section 2.4.4).

Furthermore, the sender needs to know the individual address of the recipient endpoint. This information may not be known at any given point in time. For instance, right after having joined the group, a client may know the authentication credential and Recipient ID for a given server, but not the addressing information required to reach it with an individual, one-to-one request.

To make addressing information of individual endpoints available, servers in the group MAY expose a resource to which a client can send a group request targeting a set of servers, identified by their 'kid' values specified in the request payload. The specified set may be empty, hence identifying all the servers in the group. Further details of such an interface are out of scope for this document.

9.2. Main Differences from OSCORE

The pairwise mode protects messages between two members of a group, essentially following [RFC8613], but with the following notable differences.

- * The 'kid' and 'kid context' parameters of the COSE object are used as defined in Section 4.2 of this document.
- * The external_aad defined in Section 4.3 of this document is used for the encryption process.
- * The Pairwise Sender/Recipient Keys used as Sender/Recipient keys are derived as defined in Section 2.4 of this document.

9.3. Protecting the Request

When using the pairwise mode, the request is protected as defined in Section 8.1 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following difference also applies.

- * If Observe [RFC7641] is supported, what is defined in Section 8.1.1 of this document holds.

9.4. Verifying the Request

Upon receiving a request with the Group Flag set to 0, following the procedure in Section 7, the server MUST process it as defined in Section 8.2 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following differences also apply.

- * If the server discards the request due to not retrieving a Security Context associated with the OSCORE group or to not supporting the pairwise mode, the server MAY respond with a 4.01 (Unauthorized) error message or a 4.02 (Bad Option) error message, respectively. When doing so, the server MAY set an Outer Max-Age option with value zero, and MAY include a descriptive string as diagnostic payload.
- * If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.4.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.
- * If Observe [RFC7641] is supported, what is defined in Section 8.2.1 of this document holds.

9.5. Protecting the Response

When using the pairwise mode, a response is protected as defined in Section 8.3 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following differences also apply.

- * If the server is using a different Security Context for the response compared to what was used to verify the request (see Section 3.2), then the server MUST include its Sender Sequence Number as Partial IV in the response and use it to build the AEAD nonce to protect the response. This prevents the AEAD nonce from the request from being reused.
- * If the server is using a different ID Context (Gid) for the response compared to what was used to verify the request (see Section 3.2), then the new ID Context MUST be included in the 'kid context' parameter of the response.
- * The server can obtain a new Sender ID from the Group Manager, when individually rekeyed (see Section 2.5.3.1) or when re-joining the group. In such a case, the server can help the client to synchronize, by including the 'kid' parameter in a response protected in pairwise mode, even when the request was also protected in pairwise mode.

That is, when responding to a request protected in pairwise mode, the server SHOULD include the 'kid' parameter in a response protected in pairwise mode, if it is replying to that client for the first time since the assignment of its new Sender ID.

- * If Observe [RFC7641] is supported, what is defined in Section 8.3.1 of this document holds.

9.6. Verifying the Response

Upon receiving a response with the Group Flag set to 0, following the procedure in Section 7, the client MUST process it as defined in Section 8.4 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following differences also apply.

- * The client may receive a response protected with a Security Context different from the one used to protect the corresponding request. Also, upon the establishment of a new Security Context, the client re-initializes its Replay Windows in its Recipient Contexts (see Section 2.2).
- * The same as described in Section 8.4 holds with respect to handling the 'kid' parameter of the response, when received as a reply to a request protected in pairwise mode. The client can also in this case check whether the replying server is the expected one, by relying on the server's public key. However, since the response is protected in pairwise mode, the public key is not used for verifying a countersignature as in Section 8.4. Instead, the expected server's authentication credential - namely

Recipient Auth Cred and including the server's public key - was taken as input to derive the Pairwise Recipient Key used to decrypt and verify the response (see Section 2.4.1).

- * If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.4.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.
- * If Observe [RFC7641] is supported, what is defined in Section 8.4.1 of this document holds. The client can also in this case identify a server to be the same one across a change of Sender ID, by relying on the server's public key. As to the expected server's authentication credential, the same holds as specified above for non-notification responses.

10. Challenge-Response Synchronization

This section describes how a server endpoint can synchronize with Sender Sequence Numbers of client endpoints in the group. Similarly to what is defined in Appendix B.1.2 of [RFC8613], the server performs a challenge-response exchange with a client, by using the Echo Option for CoAP specified in Section 2 of [RFC9175].

Upon receiving a request from a particular client for the first time, the server processes the message as described in this document, but, even if valid, does not deliver it to the application. Instead, the server replies to the client with an OSCORE protected 4.01 (Unauthorized) response message, including only the Echo Option and no diagnostic payload. The Echo option value SHOULD NOT be reused; when it is reused, it MUST be highly unlikely to have been recently used with this client. Since this response is protected with the Security Context used in the group, the client will consider the response valid upon successfully decrypting and verifying it.

The server stores the Echo Option value included in the response together with the pair (gid,kid), where 'gid' is the Group Identifier of the OSCORE group and 'kid' is the Sender ID of the client in the group. These are specified in the 'kid context' and 'kid' fields of the OSCORE Option of the request, respectively. After a group rekeying has been completed and a new Security Context has been established in the group, which results also in a new Group Identifier (see Section 3.2), the server MUST delete all the stored Echo values associated with members of the group.

Upon receiving a 4.01 (Unauthorized) response that includes an Echo Option and originates from a verified group member, the client sends a request as a unicast message addressed to the same server, echoing the Echo Option value. The client MUST NOT send the request including the Echo Option over multicast.

If the group uses also the group mode and the used Signature Algorithm supports ECDH (e.g., ECDSA, EdDSA), the client MUST use the pairwise mode to protect the request, as per Section 9.3. Note that, as defined in Section 9, endpoints that are members of such a group and that use the Echo Option MUST support the pairwise mode.

The client does not necessarily resend the same group request, but can instead send a more recent one, if the application permits it. This allows the client to not retain previously sent group requests for full retransmission, unless the application explicitly requires otherwise. In either case, the client uses a fresh Sender Sequence Number value from its own Sender Context. If the client stores group requests for possible retransmission with the Echo Option, it should not store a given request for longer than a preconfigured time interval. Note that the unicast request echoing the Echo Option is correctly treated and processed, since the 'kid context' field including the Group Identifier of the OSCORE group is still present in the OSCORE Option as part of the COSE object (see Section 4).

Upon receiving the unicast request including the Echo Option, the server performs the following verifications.

- * If the server does not store an Echo Option value for the pair (gid,kid), it considers: i) the time t1 when it has established the Security Context used to protect the received request; and ii) the time t2 when the request has been received. Since a valid request cannot be older than the Security Context used to protect it, the server verifies that (t2 - t1) is less than the largest amount of time acceptable to consider the request fresh.
- * If the server stores an Echo Option value for the pair (gid,kid) associated with that same client in the same group, the server verifies that the option value equals that same stored value previously sent to that client.

If the verifications above fail, the server MUST NOT process the request further and MAY send a 4.01 (Unauthorized) response including an Echo Option, hence performing a new challenge-response exchange.

If the verifications above are successful, the server proceeds as follows. In case the Replay Window in the Recipient Context associated with the client has not been set yet, the server updates

the Replay Window to mark the current Sender Sequence Number from the latest received request as seen (but all newer ones as new), and delivers the message as fresh to the application. Otherwise, the server discards the verification result and treats the message as fresh or as a replay, according to the existing Replay Window.

A server should not deliver requests from a given client to the application until one valid request from that same client has been verified as fresh, as conveying an echoed Echo Option. A server may perform the challenge-response described above at any time, if synchronization with Sender Sequence Numbers of clients is lost, e.g., after a device reboot. A client has to be ready to perform the challenge-response based on the Echo Option if a server starts it.

It is the role of the server application to define under what circumstances Sender Sequence Numbers lose synchronization. This can include experiencing a "large enough" gap $D = (SN2 - SN1)$, between the Sender Sequence Number $SN1$ of the latest accepted group request from a client and the Sender Sequence Number $SN2$ of a group request just received from that client. However, a client may send several unicast requests to different group members as protected with the pairwise mode, which may result in the server experiencing the gap D in a relatively short time. This would induce the server to perform more challenge-response exchanges than actually needed.

In order to ameliorate this, the server may rely on a trade-off between the Sender Sequence Number gap D and a time gap $T = (t2 - t1)$, where $t1$ is the time when the latest group request from a client was accepted and $t2$ is the time when the latest group request from that client has been received, respectively. Then, the server can start a challenge-response when experiencing a time gap T larger than a given, preconfigured threshold. Also, the server can start a challenge-response when experiencing a Sender Sequence Number gap D greater than a different threshold, computed as a monotonically increasing function of the currently experienced time gap T .

The challenge-response approach described in this section 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.

Endpoints configured as silent servers are not able to perform the challenge-response described above, as they do not store a Sender Context to secure the 4.01 (Unauthorized) response to the client. Thus, silent servers should adopt alternative approaches to achieve and maintain synchronization with Sender Sequence Numbers of clients.

Since requests including the Echo Option are sent over unicast, a server can be victim of the attack discussed in Section 12.9, in case such requests are protected with the group mode. Instead, protecting those requests with the pairwise mode prevents the attack above. In fact, only the exact server involved in the challenge-response exchange is able to derive the pairwise key used by the client to protect the request including the Echo Option.

In either case, an internal on-path adversary would not be able to mix up the Echo Option value of two different unicast requests, sent by a same client to any two different servers in the group. In fact, even if the group mode was used, this would require the adversary to forge the countersignature of both requests. As a consequence, each of the two servers remains able to selectively accept a request with the Echo Option only if it is waiting for that exact integrity-protected Echo Option value, and is thus the intended recipient.

11. Implementation Compliance

Like in [RFC8613], HKDF SHA-256 is the mandatory to implement HKDF.

An endpoint may support only the group mode, or only the pairwise mode, or both.

For endpoints that support the group mode, the following applies.

- * For endpoints that use authenticated encryption, the AEAD algorithm AES-CCM-16-64-128 defined in Section 4.2 of [I-D.ietf-cose-rfc8152bis-algs] is mandatory to implement as Signature Encryption Algorithm (see Section 2.1.4).
- * For many constrained IoT devices it is problematic to support more than one signature algorithm. Existing devices can be expected to support either EdDSA or ECDSA. In order to enable as much interoperability as we can reasonably achieve, the following applies with respect to the Signature Algorithm (see Section 2.1.5).

Less constrained endpoints SHOULD implement both: the EdDSA signature algorithm together with the elliptic curve Ed25519 [RFC8032]; and the ECDSA signature algorithm together with the elliptic curve P-256.

Constrained endpoints SHOULD implement: the EdDSA signature algorithm together with the elliptic curve Ed25519 [RFC8032]; or the ECDSA signature algorithm together with the elliptic curve P-256.

- * Endpoints that implement the ECDSA signature algorithm MAY use "deterministic ECDSA" as specified in [RFC6979]. Pure deterministic elliptic-curve signature algorithms such as deterministic ECDSA and EdDSA have the advantage of not requiring access to a source of high-quality randomness. However, these signature algorithms have been shown vulnerable to some side-channel and fault injection attacks due to their determinism, which can result in extracting a device's private key. As suggested in Section 2.1.1 of [I-D.ietf-cose-rfc8152bis-algs], this can be addressed by combining both randomness and determinism [I-D.mattsson-cfrg-det-sigs-with-noise].

For endpoints that support the pairwise mode, the following applies.

- * The AEAD algorithm AES-CCM-16-64-128 defined in Section 4.2 of [I-D.ietf-cose-rfc8152bis-algs] is mandatory to implement as AEAD Algorithm (see Section 2.1.1).
- * The ECDH-SS + HKDF-256 algorithm specified in Section 6.3.1 of [I-D.ietf-cose-rfc8152bis-algs] is mandatory to implement as Pairwise Key Agreement Algorithm (see Section 2.1.7).
- * In order to enable as much interoperability as we can reasonably achieve in the presence of constrained devices (see above), the following applies.

Less constrained endpoints SHOULD implement both the X25519 curve [RFC7748] and the P-256 curve as ECDH curves.

Constrained endpoints SHOULD implement the X25519 curve [RFC7748] or the P-256 curve as ECDH curve.

Constrained IoT devices may alternatively represent Montgomery curves and (twisted) Edwards curves [RFC7748] in the short-Weierstrass form Wei25519, with which the algorithms ECDSA25519 and ECDH25519 can be used for signature operations and Diffie-Hellman secret calculation, respectively [I-D.ietf-lwig-curve-representations].

12. Security Considerations

The same threat model discussed for OSCORE in Appendix D.1 of [RFC8613] holds for Group OSCORE. In addition, when using the group mode, source authentication of messages is explicitly ensured by means of countersignatures, as discussed in Section 12.1.

Note that, even if an endpoint is authorized to be a group member and to take part in group communications, there is a risk that it behaves inappropriately. For instance, it can forward the content of

messages in the group 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), which results in a practically limited risk and enables a prompt detection/reaction in case of misbehaving.

The same considerations on supporting Proxy operations discussed for OSCORE in Appendix D.2 of [RFC8613] hold for Group OSCORE.

The same considerations on protected message fields for OSCORE discussed in Appendix D.3 of [RFC8613] hold for Group OSCORE.

The same considerations on uniqueness of (key, nonce) pairs for OSCORE discussed in Appendix D.4 of [RFC8613] hold for Group OSCORE. This is further discussed in Section 12.3 of this document.

The same considerations on unprotected message fields for OSCORE discussed in Appendix D.5 of [RFC8613] hold for Group OSCORE, with the following differences. First, the 'kid context' of request messages is part of the Additional Authenticated Data, thus safely enabling to keep observations active beyond a possible change of ID Context (Gid), following a group rekeying (see Section 4.3). Second, the countersignature included in a Group OSCORE message protected in group mode is computed also over the value of the OSCORE option, which is also part of the Additional Authenticated Data used in the signing process. This is further discussed in Section 12.7 of this document.

As discussed in Section 6.2.3 of [I-D.ietf-core-groupcomm-bis], Group OSCORE addresses security attacks against CoAP listed in Sections 11.2-11.6 of [RFC7252], especially when run over IP multicast.

The rest of this section first discusses security aspects to be taken into account when using Group OSCORE. Then it goes through aspects covered in the security considerations of OSCORE (see Section 12 of [RFC8613]), and discusses how they hold when Group OSCORE is used.

12.1. Security of the Group Mode

The group mode defined in Section 8 relies on commonly shared group keying material to protect communication within a group. Using the group mode has the implications discussed below. The following refers to group members as the endpoints in the group storing the latest version of the group keying material.

- * 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.
- * If the used encryption algorithm provides integrity protection, then it also ensures group authentication and proof of group membership, but not source authentication. That is, it ensures that a message sent to a group has been sent by a member of that group, but not necessarily by the alleged sender. In fact, any group member is able to derive the Sender Key used by the actual sender endpoint, and thus can compute a valid authentication tag. Therefore, the message content could originate from any of the current group members.

Furthermore, if the used encryption algorithm does not provide integrity protection, then it does not ensure any level of message authentication or proof of group membership.

On the other hand, proof of group membership is always ensured by construction through the strict management of the group keying material (see Section 3.2). That is, the group is rekeyed in case of members' leaving, and the current group members are informed of former group members. Thus, a current group member storing the latest group keying material does not store the authentication credential of any former group member.

This allows a recipient endpoint to rely on the stored authentication credentials and public keys included therein, in order to always confidently assert the group membership of a sender endpoint when processing an incoming message, i.e., to assert that the sender endpoint was a group member when it signed the message. In turn, this prevents a former group member to possibly re-sign and inject in the group a stored message that was protected with old keying material.

- * Source authentication of messages sent to a group is ensured through a countersignature, which is computed by the sender using its own private key and then appended to the message payload. Also, the countersignature is encrypted by using a keystream derived from the group keying material (see Section 4.1). This ensures group privacy, i.e., an attacker cannot track an endpoint over two groups by linking messages between the two groups, unless being also a member of those groups.

The security properties of the group mode are summarized below.

1. Asymmetric source authentication, by means of a countersignature.

2. Symmetric group authentication, by means of an authentication tag (only for encryption algorithms providing integrity protection).
3. Symmetric group confidentiality, by means of symmetric encryption.
4. Proof of group membership, by strictly managing the group keying material, as well as by means of integrity tags when using an encryption algorithm that provides also integrity protection.
5. Group privacy, by encrypting the countersignature.

The group mode fulfills the security properties above while also displaying the following benefits. First, the use of an encryption algorithm that does not provide integrity protection results in a minimal communication overhead, by limiting the message payload to the ciphertext and the encrypted countersignature. Second, it is possible to deploy semi-trusted entities such as signature checkers (see Section 3.1), which can break property 5, but cannot break properties 1, 2 and 3.

12.2. Security of the Pairwise Mode

The pairwise mode defined in Section 9 protects messages by using pairwise symmetric keys, derived from the static-static Diffie-Hellman shared secret computed from the asymmetric keys of the sender and recipient endpoint (see Section 2.4).

The used encryption algorithm MUST provide integrity protection. Therefore, the pairwise mode ensures both pairwise data-confidentiality and source authentication of messages, without using countersignatures. Furthermore, the recipient endpoint achieves proof of group membership for the sender endpoint, since only current group members have the required keying material to derive a valid Pairwise Sender/Recipient Key.

The long-term storing of the Diffie-Hellman shared secret is a potential security issue. In fact, if the shared secret of two group members is leaked, a third group member can exploit it to impersonate any of those two group members, by deriving and using their pairwise key. The possibility of such leakage should be contemplated, as more likely to happen than the leakage of a private key, which could be rather protected at a significantly higher level than generic memory, e.g., by using a Trusted Platform Module. Therefore, there is a trade-off between the maximum amount of time a same shared secret is stored and the frequency of its re-computing.

12.3. Uniqueness of (key, nonce)

The proof for uniqueness of (key, nonce) pairs in Appendix D.4 of [RFC8613] is also valid in group communication scenarios. That is, given an OSCORE group:

- * Uniqueness of Sender IDs within the group is enforced by the Group Manager. In fact, from the moment when a Gid is assigned to a group until the moment a new Gid is assigned to that same group, the Group Manager does not reassign a Sender ID within the group (see Section 3.2).
- * The case A in Appendix D.4 of [RFC8613] concerns all group requests and responses including a Partial IV (e.g., Observe notifications). In this case, same considerations from [RFC8613] apply here as well.
- * The case B in Appendix D.4 of [RFC8613] concerns responses not including a Partial IV (e.g., single response to a group request). In this case, same considerations from [RFC8613] apply here as well.

As a consequence, each message encrypted/decrypted with the same Sender Key is processed by using a different (ID_PIV, PIV) pair. This means that nonces used by any fixed encrypting endpoint are unique. Thus, each message is processed with a different (key, nonce) pair.

12.4. Management of Group Keying Material

The approach described in this document should take into account the risk of compromise of group members. In particular, this document specifies that a key management scheme for secure revocation and renewal of Security Contexts and group keying material **MUST** be adopted.

[I-D.ietf-ace-key-groupcomm-oscore] specifies a simple rekeying scheme for renewing the Security Context in a group.

Alternative rekeying schemes which are more scalable with the group size may be needed in dynamic, large groups where endpoints can join and leave at any time, in order to limit the impact on performance due to the Security Context and keying material update.

12.5. Update of Security Context and Key Rotation

A group member can receive a message shortly after the group has been rekeyed, and new security parameters and keying material have been distributed by the Group Manager.

This may result in a client using an old Security Context to protect a request, and a server using a different new Security Context to protect a corresponding response. As a consequence, clients may receive a response protected with a Security Context different from the one used to protect the corresponding request.

In particular, a server may first get a request protected with the old Security Context, then install the new Security Context, and only after that produce a response to send back to the client. In such a case, as specified in Section 8.3, the server **MUST** protect the potential response using the new Security Context. Specifically, the server **MUST** include its Sender Sequence Number as Partial IV in the response and use it to build the AEAD nonce to protect the response. This prevents the AEAD nonce from the request from being reused with the new Security Context.

The client will process that response using the new Security Context, provided that it has installed the new security parameters and keying material before the message processing.

In case block-wise transfer [RFC7959] is used, the same considerations from Section 10.3 of [I-D.ietf-ace-key-groupcomm] hold.

Furthermore, as described below, a group rekeying may temporarily result in misaligned Security Contexts between the sender and recipient of a same message.

12.5.1. Late Update on the Sender

In this case, the sender protects a message using the old Security Context, i.e., before having installed the new Security Context. However, the recipient receives the message after having installed the new Security Context, and is thus unable to correctly process it.

A possible way to ameliorate this issue is to preserve the old, recent, Security Context for a maximum amount of time defined by the application. By doing so, the recipient can still try to process the received message using the old retained Security Context as a second attempt. This makes particular sense when the recipient is a client, that would hence be able to process incoming responses protected with the old, recent, Security Context used to protect the associated

group request. Instead, a recipient server would better and more simply discard an incoming group request which is not successfully processed with the new Security Context.

This tolerance preserves the processing of secure messages throughout a long-lasting key rotation, as group rekeying processes may likely take a long time to complete, especially in large groups. On the other hand, a former (compromised) group member can abusively take advantage of this, and send messages protected with the old retained Security Context. Therefore, a conservative application policy should not admit the retention of old Security Contexts.

12.5.2. Late Update on the Recipient

In this case, the sender protects a message using the new Security Context, but the recipient receives that message before having installed the new Security Context. Therefore, the recipient would not be able to correctly process the message and hence discards it.

If the recipient installs the new Security Context shortly after that and the sender endpoint retransmits the message, the former will still be able to receive and correctly process the message.

In any case, the recipient should actively ask the Group Manager for an updated Security Context according to an application-defined policy, for instance after a given number of unsuccessfully decrypted incoming messages.

12.6. Collision of Group Identifiers

In case endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible for Group Identifiers of different groups to coincide.

This does not impair the security of the AEAD algorithm. In fact, as long as the Master Secret is different for different groups and this condition holds over time, AEAD keys are different among different groups.

In case multiple groups use the same IP multicast address, the entity assigning that address may help limiting the chances to experience such collisions of Group Identifiers. In particular, it may allow the Group Managers of those groups using the same IP multicast address to share their respective list of assigned Group Identifiers currently in use.

12.7. Cross-group Message Injection

A same endpoint is allowed to and would likely use the same pair (private key, authentication credential) in multiple OSCORE groups, possibly administered by different Group Managers.

When a sender endpoint sends a message protected in pairwise mode to a recipient endpoint in an OSCORE group, a malicious group member may attempt to inject the message to a different OSCORE group also including the same endpoints (see Section 12.7.1).

This practically relies on altering the content of the OSCORE option, and having the same MAC in the ciphertext still correctly validating, which has a success probability depending on the size of the MAC.

As discussed in Section 12.7.2, the attack is practically infeasible if the message is protected in group mode, thanks to the countersignature also bound to the OSCORE option through the Additional Authenticated Data used in the signing process (see Section 4.3).

12.7.1. Attack Description

Let us consider:

- * Two OSCORE groups G1 and G2, with ID Context (Group ID) Gid1 and Gid2, respectively. Both G1 and G2 use the AEAD cipher AES-CCM-16-64-128, i.e., the MAC of the ciphertext is 8 bytes in size.
- * A sender endpoint X which is member of both G1 and G2, and uses the same pair (private key, authentication credential) in both groups. The endpoint X has Sender ID Sid1 in G1 and Sender ID Sid2 in G2. The pairs (Sid1, Gid1) and (Sid2, Gid2) identify the same authentication credential of X in G1 and G2, respectively.
- * A recipient endpoint Y which is member of both G1 and G2, and uses the same pair (private key, authentication credential) in both groups. The endpoint Y has Sender ID Sid3 in G1 and Sender ID Sid4 in G2. The pairs (Sid3, Gid1) and (Sid4, Gid2) identify the same authentication credential of Y in G1 and G2, respectively.
- * A malicious endpoint Z is also member of both G1 and G2. Hence, Z is able to derive the Sender Keys used by X in G1 and G2.

When X sends a message M1 addressed to Y in G1 and protected in pairwise mode, Z can intercept M1, and attempt to forge a valid message M2 to be injected in G2, making it appear as still sent by X to Y and valid to be accepted.

More in detail, Z intercepts and stops message M1, and forges a message M2 by changing the value of the OSCORE option from M1 as follows: the 'kid context' is set to G2 (rather than G1); and the 'kid' is set to Sid2 (rather than Sid1). Then, Z injects message M2 as addressed to Y in G2.

Upon receiving M2, there is a probability equal to 2^{-64} that Y successfully verifies the same unchanged MAC by using the Pairwise Recipient Key associated with X in G2.

Note that Z does not know the pairwise keys of X and Y, since it does not know and is not able to compute their shared Diffie-Hellman secret. Therefore, Z is not able to check offline if a performed forgery is actually valid, before sending the forged message to G2.

12.7.2. Attack Prevention in Group Mode

When a Group OSCORE message is protected with the group mode, the countersignature is computed also over the value of the OSCORE option, which is part of the Additional Authenticated Data used in the signing process (see Section 4.3).

That is, other than over the ciphertext, the countersignature is computed over: the ID Context (Gid) and the Partial IV, which are always present in group requests; as well as the Sender ID of the message originator, which is always present in group requests as well as in responses to requests protected in group mode.

Since the signing process takes as input also the ciphertext of the COSE_Encrypt0 object, the countersignature is bound not only to the intended OSCORE group, hence to the triplet (Master Secret, Master Salt, ID Context), but also to a specific Sender ID in that group and to its specific symmetric key used for AEAD encryption, hence to the quartet (Master Secret, Master Salt, ID Context, Sender ID).

This makes it practically infeasible to perform the attack described in Section 12.7.1, since it would require the adversary to additionally forge a valid countersignature that replaces the original one in the forged message M2.

If, hypothetically, the countersignature did not cover the OSCORE option:

- * The attack described in Section 12.7.1 would still be possible against response messages protected in group mode, since the same unchanged countersignature from message M1 would be also valid in message M2.

- * A simplification would also be possible in performing the attack, since Z is able to derive the Sender/Recipient Keys of X and Y in G1 and G2. That is, Z can also set a convenient Partial IV in the response, until the same unchanged MAC is successfully verified by using G2 as 'request_kid_context', Sid2 as 'request_kid', and the symmetric key associated with X in G2.

Since the Partial IV is 5 bytes in size, this requires 2^{40} operations to test all the Partial IVs, which can be done in real-time. The probability that a single given message M1 can be used to forge a response M2 for a given request would be equal to 2^{-24} , since there are more MAC values (8 bytes in size) than Partial IV values (5 bytes in size).

Note that, by changing the Partial IV as discussed above, any member of G1 would also be able to forge a valid signed response message M2 to be injected in the same group G1.

12.8. Prevention of Group Cloning Attack

Both when using the group mode and the pairwise mode, the message protection covers also the Group Manager's authentication credential. This is included in the Additional Authenticated Data used in the signing process and/or in the integrity-protected encryption process (see Section 4.3).

By doing so, an endpoint X member of a group G1 cannot perform the following attack.

1. X sets up a group G2 where it acts as Group Manager.
2. X makes G2 a "clone" of G1, i.e., G1 and G2 use the same algorithms and have the same Master Secret, Master Salt and ID Context.
3. X collects a message M sent to G1 and injects it in G2.
4. Members of G2 accept M and believe it to be originated in G2.

The attack above is effectively prevented, since message M is protected by including the authentication credential of G1's Group Manager in the Additional Authenticated Data. Therefore, members of G2 do not successfully verify and decrypt M, since they correctly use the authentication credential of X as Group Manager of G2 when attempting to.

12.9. Group OSCORE for Unicast Requests

If a request is intended to be sent over unicast as addressed to a single group member, it is NOT RECOMMENDED for the client to protect the request by using the group mode as defined in Section 8.1.

This does not include the case where the client sends a request over unicast to a proxy, to be forwarded to multiple intended recipients over multicast [I-D.ietf-core-groupcomm-bis]. In this case, the client MUST protect the request with the group mode, even though it is sent to the proxy over unicast (see Section 8).

If the client uses the group mode with its own Sender Key to protect a unicast request to a group member, an on-path adversary can, right then or later on, redirect that request to one/many different group member(s) over unicast, or to the whole OSCORE group over multicast. By doing so, the adversary can induce the target group member(s) to perform actions intended for one group member only. Note that the adversary can be external, i.e., (s)he does not need to also be a member of the OSCORE group.

This is due to the fact that the client is not able to indicate the single intended recipient in a way which is secure and possible to process for Group OSCORE on the server side. In particular, Group OSCORE does not protect network addressing information such as the IP address of the intended recipient server. It follows that the server(s) receiving the redirected request cannot assert whether that was the original intention of the client, and would thus simply assume so.

The impact of such an attack depends especially on the REST method of the request, i.e., the Inner CoAP Code of the OSCORE request message. In particular, safe methods such as GET and FETCH would trigger (several) unintended responses from the targeted server(s), while not resulting in destructive behavior. On the other hand, non safe methods such as PUT, POST and PATCH/iPATCH would result in the target server(s) taking active actions on their resources and possible cyber-physical environment, with the risk of destructive consequences and possible implications for safety.

A client can instead use the pairwise mode as defined in Section 9.3, in order to protect a request sent to a single group member by using pairwise keying material (see Section 2.4). This prevents the attack discussed above by construction, as only the intended server is able to derive the pairwise keying material used by the client to protect the request. A client supporting the pairwise mode SHOULD use it to protect requests sent to a single group member over unicast, instead of using the group mode. For an example where this is not fulfilled, see Section 7.2.1 of [I-D.ietf-core-observe-multicast-notifications].

With particular reference to block-wise transfers [RFC7959], Section 3.8 of [I-D.ietf-core-groupcomm-bis] points out that, while an initial request including the CoAP Block2 option can be sent over multicast, any other request in a transfer has to occur over unicast, individually addressing the servers in the group.

Additional considerations are discussed in Section 10, with respect to requests including a CoAP Echo Option [RFC9175] that have to be sent over unicast, as a challenge-response method for servers to achieve synchronization of clients' Sender Sequence Number.

12.10. End-to-end Protection

The same considerations from Section 12.1 of [RFC8613] hold for Group OSCORE.

Additionally, (D)TLS and Group OSCORE can be combined for protecting message exchanges occurring over unicast. However, it is not possible to combine (D)TLS and Group OSCORE for protecting message exchanges where messages are (also) sent over multicast.

12.11. Master Secret

Group OSCORE derives the Security Context using the same construction as OSCORE, and by using the Group Identifier of a group as the related ID Context. Hence, the same required properties of the Security Context parameters discussed in Section 3.3 of [RFC8613] hold for this document.

With particular reference to the OSCORE Master Secret, it has to be kept secret among the members of the respective OSCORE group and the Group Manager responsible for that group. Also, the Master Secret must have a good amount of randomness, and the Group Manager can generate it offline using a good random number generator. This includes the case where the Group Manager rekeys the group by generating and distributing a new Master Secret. Randomness requirements for security are described in [RFC4086].

12.12. Replay Protection

As in OSCORE [RFC8613], also Group OSCORE relies on Sender Sequence Numbers included in the COSE message field 'Partial IV' and used to build AEAD nonces.

Note that the Partial IV of an endpoint does not necessarily grow monotonically. For instance, upon exhaustion of the endpoint Sender Sequence Number, the Partial IV also gets exhausted. As discussed in Section 2.5.3, this results either in the endpoint being individually rekeyed and getting a new Sender ID, or in the establishment of a new Security Context in the group. Therefore, uniqueness of (key, nonce) pairs (see Section 12.3) is preserved also when a new Security Context is established.

Since one-to-many communication such as multicast usually involves unreliable transports, the simplification of the Replay Window to a size of 1 suggested in Section 7.4 of [RFC8613] is not viable with Group OSCORE, unless exchanges in the group rely only on unicast messages.

As discussed in Section 6.2, a Replay Window may be initialized as not valid, following the loss of mutable Security Context Section 2.5.1. In particular, Section 2.5.1.1 and Section 2.5.1.2 define measures that endpoints need to take in such a situation, before resuming to accept incoming messages from other group members.

12.13. Message Freshness

As discussed in Section 6.3, a server may not be able to assert whether an incoming request is fresh, in case it does not have or has lost synchronization with the client's Sender Sequence Number.

If freshness is relevant for the application, the server may (re-)synchronize with the client's Sender Sequence Number at any time, by using the approach described in Section 10 and based on the CoAP Echo Option [RFC9175], as a variant of the approach defined in Appendix B.1.2 of [RFC8613] applicable to Group OSCORE.

12.14. Client Aliveness

Building on Section 12.5 of [RFC8613], a server may use the CoAP Echo Option [RFC9175] to verify the aliveness of the client that originated a received request, by using the approach described in Section 10 of this document.

12.15. Cryptographic Considerations

The same considerations from Section 12.6 of [RFC8613] about the maximum Sender Sequence Number hold for Group OSCORE.

As discussed in Section 2.5.2, an endpoint that experiences an exhaustion of its own Sender Sequence Numbers MUST NOT protect further messages including a Partial IV, until it has derived a new Sender Context. This prevents the endpoint to reuse the same AEAD nonces with the same Sender Key.

In order to renew its own Sender Context, the endpoint SHOULD inform the Group Manager, which can either renew the whole Security Context by means of group rekeying, or provide only that endpoint with a new Sender ID value. In either case, the endpoint derives a new Sender Context, and in particular a new Sender Key.

Additionally, the same considerations from Section 12.6 of [RFC8613] hold for Group OSCORE, about building the AEAD nonce and the secrecy of the Security Context parameters.

The group mode uses the "encrypt-then-sign" construction, i.e., the countersignature is computed over the COSE_Encrypt0 object (see Section 4.1). This is motivated by enabling additional entities acting as signature checkers (see Section 3.1), which do not join a group as members but are allowed to verify countersignatures of messages protected in group mode without being able to decrypt them (see Section 8.5).

If the encryption algorithm used in group mode provides integrity protection, countersignatures of COSE_Encrypt0 with short authentication tags do not provide the security properties associated with the same algorithm used in COSE_Sign (see Section 6 of [I-D.ietf-cose-countersign]). To provide 128-bit security against collision attacks, the tag length MUST be at least 256-bits. A countersignature of a COSE_Encrypt0 with AES-CCM-16-64-128 provides at most 32 bits of integrity protection.

The derivation of pairwise keys defined in Section 2.4.1 is compatible with ECDSA and EdDSA asymmetric keys, but is not compatible with RSA asymmetric keys.

For the public key translation from Ed25519 (Ed448) to X25519 (X448) specified in Section 2.4.1, variable time methods can be used since the translation operates on public information. Any byte string of appropriate length is accepted as a public key for X25519 (X448) in [RFC7748]. It is therefore not necessary for security to validate the translated public key (assuming the translation was successful).

The security of using the same key pair for Diffie-Hellman and for signing (by considering the ECDH procedure in Section 2.4 as a Key Encapsulation Mechanism (KEM)) is demonstrated in [Degabriele] and [Thormarker].

Applications using ECDH (except X25519 and X448) based KEM in Section 2.4 are assumed to verify that a peer endpoint's public key is on the expected curve and that the shared secret is not the point at infinity. The KEM in [Degabriele] checks that the shared secret is different from the point at infinity, as does the procedure in Section 5.7.1.2 of [NIST-800-56A] which is referenced in Section 2.4.

Extending Theorem 2 of [Degabriele], [Thormarker] shows that the same key pair can be used with X25519 and Ed25519 (X448 and Ed448) for the KEM specified in Section 2.4. By symmetry in the KEM used in this document, both endpoints can consider themselves to have the recipient role in the KEM - as discussed in Section 7 of [Thormarker] - and rely on the mentioned proofs for the security of their key pairs.

Theorem 3 in [Degabriele] shows that the same key pair can be used for an ECDH based KEM and ECDSA. The KEM uses a different KDF than in Section 2.4, but the proof only depends on that the KDF has certain required properties, which are the typical assumptions about HKDF, e.g., that output keys are pseudorandom. In order to comply with the assumptions of Theorem 3, received public keys MUST be successfully validated, see Section 5.6.2.3.4 of [NIST-800-56A]. The validation MAY be performed by a trusted Group Manager. For [Degabriele] to apply as it is written, public keys need to be in the expected subgroup. For this we rely on cofactor DH, Section 5.7.1.2 of [NIST-800-56A] which is referenced in Section 2.4.

HashEdDSA variants of Ed25519 and Ed448 are not used by COSE, see Section 2.2 of [I-D.ietf-cose-rfc8152bis-algs], and are not covered by the analysis in [Thormarker]. Hence, they MUST NOT be used with the public keys used to derive pairwise keys as specified in this document.

12.16. Message Segmentation

The same considerations from Section 12.7 of [RFC8613] hold for Group OSCORE.

12.17. Privacy Considerations

Group OSCORE ensures end-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations. In particular, options are processed according to the same class U/I/E that they have for OSCORE. Therefore, the same privacy considerations from Section 12.8 of [RFC8613] hold for Group OSCORE, with the following addition.

- * When protecting a message in group mode, the countersignature is encrypted by using a keystream derived from the group keying material (see Section 4.1 and Section 4.1.1). This ensures group privacy. That is, an attacker cannot track an endpoint over two groups by linking messages between the two groups, unless being also a member of those groups.

Furthermore, the following privacy considerations hold about the OSCORE option, which may reveal information on the communicating endpoints.

- * The 'kid' parameter, which is intended to help a recipient endpoint to find the right Recipient Context, may reveal information about the Sender Endpoint. When both a request and the corresponding responses include the 'kid' parameter, this may reveal information about both a client sending a request and all the possibly replying servers sending their own individual response.
- * The 'kid context' parameter, which is intended to help a recipient endpoint to find the right Security Context, reveals information about the sender endpoint. In particular, it reveals that the sender endpoint is a member of a particular OSCORE group, whose current Group ID is indicated in the 'kid context' parameter.

When receiving a group request, each of the recipient endpoints can reply with a response that includes its Sender ID as 'kid' parameter. All these responses will be matchable with the request through the Token. Thus, even if these responses do not include a 'kid context' parameter, it becomes possible to understand that the responder endpoints are in the same group of the requester endpoint.

Furthermore, using the approach described in Section 10 to achieve Sender Sequence Number synchronization with a client may reveal when a server device goes through a reboot. This can be mitigated by the server device storing the precise state of the Replay Window of each known client on a clean shutdown.

Finally, the approach described in Section 12.6 to prevent collisions of Group Identifiers from different Group Managers may reveal information about events in the respective OSCORE groups. In particular, a Group Identifier changes when the corresponding group is rekeyed. Thus, Group Managers might use the shared list of Group Identifiers to infer the rate and patterns of group membership changes triggering a group rekeying, e.g., due to newly joined members or evicted (compromised) members. In order to alleviate this privacy concern, it should be hidden from the Group Managers which exact Group Manager has currently assigned which Group Identifiers in its OSCORE groups.

13. IANA Considerations

Note to RFC Editor: Please replace "[This Document]" with the RFC number of this document and delete this paragraph.

This document has the following actions for IANA.

13.1. OSCORE Flag Bits Registry

IANA is asked to add the following value entry to the "OSCORE Flag Bits" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Bit Position	Name	Description	Reference
2	Group Flag	For using a Group OSCORE Security Context, set to 1 if the message is protected with the group mode	[This Document]

14. References

14.1. Normative References

[I-D.ietf-core-groupcomm-bis]

Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-06, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-06.txt>>.

[I-D.ietf-cose-countersign]

Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", Work in Progress,

Internet-Draft, draft-ietf-cose-countersign-05, 23 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-countersign-05.txt>>.

[I-D.ietf-cose-rfc8152bis-algs]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.

[I-D.ietf-cose-rfc8152bis-struct]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.

[NIST-800-56A]

Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography - NIST Special Publication 800-56A, Revision 3", April 2018, <<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf>>.

[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>>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <<https://www.rfc-editor.org/info/rfc6979>>.

[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>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [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>>.
- [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>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.
- [RFC9175] Amsüss, C., Preuß Mattsson, J., and G. Selander, "Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing", RFC 9175, DOI 10.17487/RFC9175, February 2022, <<https://www.rfc-editor.org/info/rfc9175>>.

14.2. Informative References

- [Degabriele]
Degabriele, J.P., Lehmann, A., Paterson, K.G., Smart, N.P., and M. Streffler, "On the Joint Security of Encryption and Signature in EMV", December 2011, <<https://eprint.iacr.org/2011/615>>.

[I-D.amsuess-core-cachable-oscore]

Amsüss, C. and M. Tiloca, "Cacheable OSCORE", Work in Progress, Internet-Draft, draft-amsuess-core-cachable-oscore-04, 6 March 2022, <<https://www.ietf.org/archive/id/draft-amsuess-core-cachable-oscore-04.txt>>.

[I-D.ietf-ace-key-groupcomm]

Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-15, 23 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-15.txt>>.

[I-D.ietf-ace-key-groupcomm-oscore]

Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-13, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-13.txt>>.

[I-D.ietf-ace-oauth-authz]

Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-46, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>>.

[I-D.ietf-core-observe-multicast-notifications]

Tiloca, M., Höglund, R., Amsüss, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-ietf-core-observe-multicast-notifications-03, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-observe-multicast-notifications-03.txt>>.

[I-D.ietf-cose-cbor-encoded-cert]

Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuheid, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-03, 10 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-cose-cbor-encoded-cert-03.txt>>.

[I-D.ietf-lwig-curve-representations]

Struik, R., "Alternative Elliptic Curve Representations", Work in Progress, Internet-Draft, draft-ietf-lwig-curve-

representations-23, 21 January 2022,
<<https://www.ietf.org/archive/id/draft-ietf-lwig-curve-representations-23.txt>>.

[I-D.ietf-lwig-security-protocol-comparison]
Mattsson, J. P., Palombini, F., and M. Vucinic,
"Comparison of CoAP Security Protocols", Work in Progress,
Internet-Draft, draft-ietf-lwig-security-protocol-
comparison-05, 2 November 2020,
<<https://www.ietf.org/archive/id/draft-ietf-lwig-security-protocol-comparison-05.txt>>.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The
Datagram Transport Layer Security (DTLS) Protocol Version
1.3", Work in Progress, Internet-Draft, draft-ietf-tls-
dtls13-43, 30 April 2021, <<https://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-43.txt>>.

[I-D.mattsson-cfrg-det-sigs-with-noise]
Mattsson, J. P., Thormarker, E., and S. Ruohomaa,
"Deterministic ECDSA and EdDSA Signatures with Additional
Randomness", Work in Progress, Internet-Draft, draft-
mattsson-cfrg-det-sigs-with-noise-04, 15 February 2022,
<<https://www.ietf.org/archive/id/draft-mattsson-cfrg-det-sigs-with-noise-04.txt>>.

[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>>.

[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>>.

[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>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [Thormarker]
Thormarker, E., "On using the same key pair for Ed25519 and an X25519 based KEM", April 2021, <<https://eprint.iacr.org/2021/509>>.

Appendix A. Assumptions and Security Objectives

This section presents a set of assumptions and security objectives for the approach described in this document. The rest of this section refers to three types of groups:

- * Application group, i.e., a set of CoAP endpoints that share a common pool of resources.
- * Security group, as defined in Section 1.1 of this document. There can be a one-to-one or a one-to-many relation between security groups and application groups, and vice versa.
- * CoAP group, i.e., a set of CoAP endpoints where each endpoint is configured to receive one-to-many CoAP requests, e.g., sent to the group's associated IP multicast address and UDP port as defined in [I-D.ietf-core-groupcomm-bis]. An endpoint may be a member of multiple CoAP groups. There can be a one-to-one or a one-to-many relation between application groups and CoAP groups. Note that a

device sending a CoAP request to a CoAP group is not necessarily itself a member of that group: it is a member only if it also has a CoAP server endpoint listening to requests for this CoAP group, sent to the associated IP multicast address and port. In order to provide secure group communication, all members of a CoAP group as well as all further endpoints configured only as clients sending CoAP (multicast) requests to the CoAP group have to be member of a security group. There can be a one-to-one or a one-to-many relation between security groups and CoAP groups, and vice versa.

A.1. Assumptions

The following points 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 sender and multiple recipients) and M-to-N (multiple senders and multiple recipients) 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.

In a 1-to-N communication model, only a single client transmits data to the CoAP group, in the form of request messages; in an M-to-N communication model (where M and N do not necessarily have the same value), M clients transmit data to the CoAP group. According to [I-D.ietf-core-groupcomm-bis], any possible proxy entity is supposed to know about the clients. Also, every client expects and is able to handle multiple response messages associated with a same request sent to the CoAP group.

- * Group size: security solutions for group communication should be able to adequately support different and possibly large security groups. The group size is the current number of members in a security group. In the use cases mentioned in this document, the number of clients (normally the controlling devices) is expected to be much smaller than the number of servers (i.e., the controlled devices). A security solution for group communication that supports 1 to 50 clients 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. Security groups larger than that should be divided into smaller independent groups. One should not assume that the set of members of a security group remains fixed. That is, the group membership is subject to changes, possibly on a frequent basis.

- * Communication with the Group Manager: an endpoint must use a secure dedicated channel when communicating with the Group Manager, also when not registered as a member of the security group.
- * Provisioning and management of Security Contexts: a Security Context must be established among the members of the security group. A secure mechanism must be used to generate, revoke and (re-)distribute keying material, communication policies and security parameters in the security group. The actual provisioning and management of the Security Context is out of the scope of this document.
- * Multicast data security ciphersuite: all members of a security group must use the same 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 security group should not have access to any old Security Contexts used before its joining. This ensures that a new member of the security group is not able to decrypt confidential data sent before it has joined the security 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 in the security group upon a new member's joining has to be defined as part of the group key management scheme.
- * Forward security: entities that leave the security group should not have access to any future Security Contexts or message exchanged within the security group after their leaving. This ensures that a former member of the security group is not able to decrypt confidential data sent within the security group anymore. Also, it ensures that a former member is not able to send protected messages to the security group anymore. The actual mechanism to update the Security Context and renew the group keying material in the security group upon a 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: group request messages or response messages replayed within the security group must be detected.

- * Data confidentiality: messages sent within the security group shall be encrypted.
- * Group-level data confidentiality: the group mode provides 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 security group, but not by an external adversary or other external entities.
- * Pairwise data confidentiality: the pairwise mode especially provides pairwise data confidentiality, since messages are encrypted using pairwise keying material shared between any two group members, hence they can be decrypted only by the intended single recipient.
- * Source message authentication: messages sent within the security group shall be authenticated. That is, it is essential to ensure that a message is originated by a member of the security group in the first place, and in particular by a specific, identifiable member of the security group.
- * Message integrity: messages sent within the security group shall be integrity protected. That is, it is essential to ensure that a message has not been tampered with, either by a group member, or by an external adversary or other external entities which are not members of the security group.
- * Message ordering: it must be possible to determine the ordering of messages coming from a single sender. In accordance with OSCORE [RFC8613], this results in providing absolute freshness of responses that are not notifications, as well as relative freshness of group requests and notification responses. It is not required to determine ordering of messages from different senders.

Appendix B. List of Use Cases

Group Communication for CoAP [I-D.ietf-core-groupcomm-bis] 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 [I-D.ietf-core-groupcomm-bis] to understand the non-security related details. This section discusses a number of use cases that benefit from secure group communication, and refers to the three types of groups from Appendix A. 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 lighting devices acting as servers are organized into application groups and CoAP groups, according to their physical location in the building. For instance, lighting devices in a room or corridor can be configured as members of a single application group and corresponding CoAP group. Those lighting devices together with the switches acting as clients in the same room or corridor can be configured as members of the corresponding security group. Switches are then used to control the lighting devices by sending on/off/dimming commands to all lighting devices in the CoAP 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 with a role in the lighting application 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 set of connected lights, ensuring that the light preset (e.g., dimming level or color) of a large set 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 set 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 set of lighting devices. Especially in professional lighting scenarios, the roles of client and server are configured by the lighting commissioner, and devices strictly follow those roles.
- * **Integrated building control:** enabling Building Automation and Control Systems (BACSSs) to control multiple heating, ventilation and air-conditioning units to predefined presets. Controlled units can be organized into application groups and CoAP 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 application group and corresponding CoAP 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 Low-power and Lossy Network (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 set 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 set 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 Low-power and Lossy Network (LLN) 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 application group and corresponding CoAP 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 latter

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-tolerant multicast algorithm, such as Multicast Protocol for Low-Power and Lossy Networks (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.

For each group, the Group Prefix is constant over time and is uniquely defined in the set of all the groups associated with the same Group Manager. The choice of the Group Prefix for a given group's Security Context is application specific. 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 each time new keying material, together with a new Gid, is distributed to the group in order to establish a new Security Context (see Section 3.2).

As an example, a 3-byte Gid 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 Common Context 61532 times in the group, its Gid will assume value '0xb1f05c'.

Using an immutable Group Prefix for a group assumes that enough time elapses before all possible Group Epoch values are used, i.e., before the Group Manager terminates the group or starts reassigning Gid values to the group (see Section 3.2). Thus, the expected highest rate for addition/removal of group members and consequent group rekeying should be taken into account for a proper dimensioning of the Group Epoch size.

As discussed in Section 12.6, if endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible that Group Identifiers of different groups coincide at some point in time. In this case, a recipient has to handle coinciding Group Identifiers, and has to try using different Security Contexts to process an incoming message, until the right one is found and the message is correctly verified. Therefore, it is favorable that Group Identifiers from different Group Managers have a size that result in a small probability of collision. How small this probability should be is up to system designers.

Appendix D. Set-up of New Endpoints

An endpoint joins a group by explicitly interacting with the responsible Group Manager. When becoming members of a group, endpoints are not required to know how many and what endpoints are in the same group.

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 the scope of this document.

The Group Manager must verify that the joining endpoint is authorized to join 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. Further details about the authorization of joining endpoints are out of scope.

In case of successful authorization check, the Group Manager generates a Sender ID assigned to the joining endpoint, before proceeding with the rest of the join process. That is, the Group Manager provides the joining endpoint with the keying material and parameters to initialize the Security Context, including its own authentication credential (see Section 2). The actual provisioning of keying material and parameters to the joining endpoint is out of the scope of this document.

As mentioned in Section 3, the Group Manager and the join process can be as specified in [I-D.ietf-ace-key-groupcomm-oscore].

Appendix E. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

E.1. Version -13 to -14

- * Replaced "node" with "endpoint" where appropriate.
- * Replaced "owning" with "storing" (of keying material).
- * Distinction between "authentication credential" and "public key".
- * Considerations on storing whole authentication credentials.
- * Considerations on Denial of Service.
- * Recycling of Group IDs by tracking the "Birth Gid" of each group member is now optional to support and use for the Group Manager.

- * Fine-grained suppression of error responses.
- * Changed section title "Mandatory-to-Implement Compliance Requirements" to "Implementation Compliance".
- * "Challenge-Response Synchronization" moved to the document body.
- * RFC 7641 and draft-ietf-core-echo-request-tag as normative references.
- * Clarifications and editorial improvements.

E.2. Version -12 to -13

- * Fixes in the derivation of the Group Encryption Key.
- * Added Mandatory-to-Implement compliance requirements.
- * Changed UCCS to CCS.

E.3. Version -11 to -12

- * No mode of operation is mandatory to support.
- * Revised parameters of the Security Context, COSE object and external_aad.
- * Revised management of keying material for the Group Manager.
- * Informing of former members when rekeying the group.
- * Admit encryption-only algorithms in group mode.
- * Encrypted countersignature through a keystream.
- * Added public key of the Group Manager as key material and protected data.
- * Clarifications about message processing, especially notifications.
- * Guidance for message processing of external signature checkers.
- * Updated derivation of pairwise keys, with more security considerations.
- * Termination of ongoing observations as client, upon leaving or before re-joining the group.

- * Recycling Group IDs by tracking the "Birth Gid" of each group member.
- * Expanded security and privacy considerations about the group mode.
- * Removed appendices on skipping signature verification and on COSE capabilities.
- * Fixes and editorial improvements.

E.4. Version -10 to -11

- * Loss of Recipient Contexts due to their overflow.
- * Added diagram on keying material components and their relation.
- * Distinction between anti-replay and freshness.
- * Preservation of Sender IDs over rekeying.
- * Clearer cause-effect about reset of SSN.
- * The GM provides public keys of group members with associated Sender IDs.
- * Removed 'par_countersign_key' from the external_aad.
- * One single format for the external_aad, both for encryption and signing.
- * Presence of 'kid' in responses to requests protected with the pairwise mode.
- * Inclusion of 'kid_context' in notifications following a group rekeying.
- * Pairwise mode presented with OSCORE as baseline.
- * Revised examples with signature values.
- * Decoupled growth of clients' Sender Sequence Numbers and loss of synchronization for server.
- * Sender IDs not recycled in the group under the same Gid.
- * Processing and description of the Group Flag bit in the OSCORE option.

- * Usage of the pairwise mode for multicast requests.
- * Clarifications on synchronization using the Echo option.
- * General format of context parameters and external_aad elements, supporting future registered COSE algorithms (new Appendix).
- * Fixes and editorial improvements.

E.5. Version -09 to -10

- * Removed 'Counter Signature Key Parameters' from the Common Context.
- * New parameters in the Common Context covering the DH secret derivation.
- * New countersignature header parameter from draft-ietf-cose-countersign.
- * Stronger policies non non-recycling of Sender IDs and Gid.
- * The Sender Sequence Number is reset when establishing a new Security Context.
- * Added 'request_kid_context' in the aad_array.
- * The server can respond with 5.03 if the client's public key is not available.
- * The observer client stores an invariant identifier of the group.
- * Relaxed storing of original 'kid' for observer clients.
- * Both client and server store the 'kid_context' of the original observation request.
- * The server uses a fresh PIV if protecting the response with a Security Context different from the one used to protect the request.
- * Clarifications on MTI algorithms and curves.
- * Removed optimized requests.
- * Overall clarifications and editorial revision.

E.6. Version -08 to -09

- * Pairwise keys are discarded after group rekeying.
- * Signature mode renamed to group mode.
- * The parameters for countersignatures use the updated COSE registries. Newly defined IANA registries have been removed.
- * Pairwise Flag bit renamed as Group Flag bit, set to 1 in group mode and set to 0 in pairwise mode.
- * Dedicated section on updating the Security Context.
- * By default, sender sequence numbers and replay windows are not reset upon group rekeying.
- * An endpoint implementing only a silent server does not support the pairwise mode.
- * Separate section on general message reception.
- * Pairwise mode moved to the document body.
- * Considerations on using the pairwise mode in non-multicast settings.
- * Optimized requests are moved as an appendix.
- * Normative support for the signature and pairwise mode.
- * Revised methods for synchronization with clients' sender sequence number.
- * Appendix with example values of parameters for countersignatures.
- * Clarifications and editorial improvements.

E.7. Version -07 to -08

- * Clarified relation between pairwise mode and group communication (Section 1).
- * Improved definition of "silent server" (Section 1.1).
- * Clarified when a Recipient Context is needed (Section 2).

- * Signature checkers as entities supported by the Group Manager (Section 2.3).
- * Clarified that the Group Manager is under exclusive control of Gid and Sender ID values in a group, with Sender ID values under each Gid value (Section 2.3).
- * Mitigation policies in case of recycled 'kid' values (Section 2.4).
- * More generic exhaustion (not necessarily wrap-around) of sender sequence numbers (Sections 2.5 and 10.11).
- * Pairwise key considerations, as to group rekeying and Sender Sequence Numbers (Section 3).
- * Added reference to static-static Diffie-Hellman shared secret (Section 3).
- * Note for implementation about the external_aad for signing (Section 4.3.2).
- * Retransmission by the application for group requests over multicast as Non-confirmable (Section 7).
- * A server MUST use its own Partial IV in a response, if protecting it with a different context than the one used for the request (Section 7.3).
- * Security considerations: encryption of pairwise mode as alternative to group-level security (Section 10.1).
- * Security considerations: added approach to reduce the chance of global collisions of Gid values from different Group Managers (Section 10.5).
- * Security considerations: added implications for block-wise transfers when using the signature mode for requests over unicast (Section 10.7).
- * Security considerations: (multiple) supported signature algorithms (Section 10.13).
- * Security considerations: added privacy considerations on the approach for reducing global collisions of Gid values (Section 10.15).

- * Updates to the methods for synchronizing with clients' sequence number (Appendix E).
- * Simplified text on discovery services supporting the pairwise mode (Appendix G.1).
- * Editorial improvements.

E.8. Version -06 to -07

- * Updated abstract and introduction.
- * Clarifications of what pertains a group rekeying.
- * Derivation of pairwise keying material.
- * Content re-organization for COSE Object and OSCORE header compression.
- * Defined the Pairwise Flag bit for the OSCORE option.
- * Supporting CoAP Observe for group requests and responses.
- * Considerations on message protection across switching to new keying material.
- * New optimized mode based on pairwise keying material.
- * More considerations on replay protection and Security Contexts upon key renewal.
- * Security considerations on Group OSCORE for unicast requests, also as affecting the usage of the Echo option.
- * Clarification on different types of groups considered (application/security/CoAP).
- * New pairwise mode, using pairwise keying material for both requests and responses.

E.9. Version -05 to -06

- * Group IDs mandated to be unique under the same Group Manager.
- * Clarifications on parameter update upon group rekeying.
- * Updated external_aad structures.

- * Dynamic derivation of Recipient Contexts made optional and application specific.
- * Optional 4.00 response for failed signature verification on the server.
- * Removed client handling of duplicated responses to multicast requests.
- * Additional considerations on public key retrieval and group rekeying.
- * Added Group Manager responsibility on validating public keys.
- * Updates IANA registries.
- * Reference to RFC 8613.
- * Editorial improvements.

E.10. Version -04 to -05

- * Added references to draft-dijk-core-groupcomm-bis.
- * New parameter Counter Signature Key Parameters (Section 2).
- * Clarification about Recipient Contexts (Section 2).
- * Two different external_aad for encrypting and signing (Section 3.1).
- * Updated response verification to handle Observe notifications (Section 6.4).
- * Extended Security Considerations (Section 8).
- * New "Counter Signature Key Parameters" IANA Registry (Section 9.2).

E.11. Version -03 to -04

- * Added the new "Counter Signature Parameters" in the Common Context (see Section 2).
- * Added recommendation on using "deterministic ECDSA" if ECDSA is used as countersignature algorithm (see Section 2).

- * Clarified possible asynchronous retrieval of keying material from the Group Manager, in order to process incoming messages (see Section 2).
- * Structured Section 3 into subsections.
- * Added the new 'par_countersign' to the aad_array of the external_aad (see Section 3.1).
- * Clarified non reliability of 'kid' as identity identifier for a group member (see Section 2.1).
- * Described possible provisioning of new Sender ID in case of Partial IV wrap-around (see Section 2.2).
- * The former signature bit in the Flag Byte of the OSCORE option value is reverted to reserved (see Section 4.1).
- * Updated examples of compressed COSE object, now with the sixth less significant bit in the Flag Byte of the OSCORE option value set to 0 (see Section 4.3).
- * Relaxed statements on sending error messages (see Section 6).
- * Added explicit step on computing the countersignature for outgoing messages (see Sections 6.1 and 6.3).
- * Handling of just created Recipient Contexts in case of unsuccessful message verification (see Sections 6.2 and 6.4).
- * Handling of replied/repeated responses on the client (see Section 6.4).
- * New IANA Registry "Counter Signature Parameters" (see Section 9.1).

E.12. Version -02 to -03

- * Revised structure and phrasing for improved readability and better alignment with draft-ietf-core-object-security.
- * Added discussion on wrap-Around of Partial IVs (see Section 2.2).
- * Separate sections for the COSE Object (Section 3) and the OSCORE Header Compression (Section 4).

- * The countersignature is now appended to the encrypted payload of the OSCORE message, rather than included in the OSCORE Option (see Section 4).
- * Extended scope of Section 5, now titled " Message Binding, Sequence Numbers, Freshness and Replay Protection".
- * Clarifications about Non-confirmable messages in Section 5.1 "Synchronization of Sender Sequence Numbers".
- * Clarifications about error handling in Section 6 "Message Processing".
- * Compacted list of responsibilities of the Group Manager in Section 7.
- * Revised and extended security considerations in Section 8.
- * Added IANA considerations for the OSCORE Flag Bits Registry in Section 9.
- * Revised Appendix D, now giving a short high-level description of a new endpoint set-up.

E.13. Version -01 to -02

- * Terminology has been made more aligned with RFC7252 and draft-ietf-core-object-security: i) "client" and "server" replace the old "multicaster" and "listener", respectively; ii) "silent server" replaces the old "pure listener".
- * Section 2 has been updated to have the Group Identifier stored in the 'ID Context' parameter defined in draft-ietf-core-object-security.
- * Section 3 has been updated with the new format of the Additional Authenticated Data.
- * Major rewriting of Section 4 to better highlight the differences with the message processing in draft-ietf-core-object-security.
- * Added Sections 7.2 and 7.3 discussing security considerations about uniqueness of (key, nonce) and collision of group identifiers, respectively.
- * Minor updates to Appendix A.1 about assumptions on multicast communication topology and group size.

- * Updated Appendix C on format of group identifiers, with practical implications of possible collisions of group identifiers.
- * Updated Appendix D.2, adding a pointer to draft-palombini-ace-key-groupcomm about retrieval of nodes' public keys through the Group Manager.
- * Minor updates to Appendix E.3 about Challenge-Response synchronization of sequence numbers based on the Echo option from draft-ietf-core-echo-request-tag.

E.14. Version -00 to -01

- * Section 1.1 has been updated with the definition of group as "security group".
- * Section 2 has been updated with:
 - Clarifications on establishment/derivation of Security Contexts.
 - A table summarizing the the additional context elements compared to OSCORE.
- * Section 3 has been updated with:
 - Examples of request and response messages.
 - Use of CounterSignature0 rather than CounterSignature.
 - Additional Authenticated Data including also the signature algorithm, while not including the Group Identifier any longer.
- * Added Section 6, listing the responsibilities of the Group Manager.
- * Added Appendix A (former section), including assumptions and security objectives.
- * Appendix B has been updated with more details on the use cases.
- * Added Appendix C, providing an example of Group Identifier format.
- * Appendix D has been updated to be aligned with draft-palombini-ace-key-groupcomm.

Acknowledgments

The authors sincerely thank Christian Amsuess, Stefan Beck, Rolf Blom, Carsten Bormann, Esko Dijk, Martin Gunnarsson, Klaus Hartke, Rikard Hoeglund, Richard Kelsey, Dave Robin, Jim Schaad, Ludwig Seitz, Peter van der Stok and Erik Thormarker for their feedback and comments.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; the H2020 project SIFIS-Home (Grant agreement 952652); the SSF project SEC4Factory under the grant RIT17-0032; and the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

Göran Selander
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: francesca.palombini@ericsson.com

John Preuss Mattsson
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: john.mattsson@ericsson.com

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
45127 Essen
Germany
Email: ji-ye.park@uni-due.de

CoRE
Internet-Draft
Intended status: Standards Track
Expires: 6 May 2021

C. Amsüss, Ed.
Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
2 November 2020

CoRE Resource Directory
draft-ietf-core-resource-directory-26

Abstract

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

Note to Readers

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

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

Status of This Memo

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 6 May 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
2. Terminology	4
3. Architecture and Use Cases	6
3.1. Principles	7
3.2. Architecture	7
3.3. RD Content Model	8
3.4. Link-local addresses and zone identifiers	12
3.5. Use Case: Cellular M2M	12
3.6. Use Case: Home and Building Automation	13
3.7. Use Case: Link Catalogues	14
4. RD discovery and other interface-independent components	14
4.1. Finding a Resource Directory	15
4.1.1. Resource Directory Address Option (RDAO)	17
4.1.2. Using DNS-SD to discover a Resource Directory	19
4.2. Payload Content Formats	19
4.3. URI Discovery	19
5. Registration	22
5.1. Simple Registration	27
5.2. Third-party registration	29
5.3. Operations on the Registration Resource	30

5.3.1.	Registration Update	30
5.3.2.	Registration Removal	33
5.3.3.	Further operations	34
6.	RD Lookup	34
6.1.	Resource lookup	35
6.2.	Lookup filtering	36
6.3.	Resource lookup examples	38
6.4.	Endpoint lookup	40
7.	Security policies	41
7.1.	Endpoint name	42
7.1.1.	Random endpoint names	42
7.2.	Entered resources	42
7.3.	Link confidentiality	43
7.4.	Segmentation	43
7.5.	First-Come-First-Remembered: A default policy	44
8.	Security Considerations	45
8.1.	Discovery	46
8.2.	Endpoint Identification and Authentication	46
8.3.	Access Control	47
8.4.	Denial of Service Attacks	47
9.	IANA Considerations	48
9.1.	Resource Types	48
9.2.	IPv6 ND Resource Directory Address Option	48
9.3.	RD Parameter Registry	48
9.3.1.	Full description of the "Endpoint Type" RD Parameter	51
9.4.	"Endpoint Type" (et=) RD Parameter values	51
9.5.	Multicast Address Registration	52
9.6.	Well-Known URIs	52
9.7.	Service Names and Transport Protocol Port Number Registry	52
10.	Examples	53
10.1.	Lighting Installation	53
10.1.1.	Installation Characteristics	53
10.1.2.	RD entries	54
10.2.	OMA Lightweight M2M (LwM2M)	57
11.	Acknowledgments	58
12.	Changelog	58
13.	References	72
13.1.	Normative References	72
13.2.	Informative References	73
Appendix A.	Groups Registration and Lookup	76
Appendix B.	Web links and the Resource Directory	78
B.1.	A simple example	78
B.1.1.	Resolving the URIs	79
B.1.2.	Interpreting attributes and relations	79
B.2.	A slightly more complex example	79
B.3.	Enter the Resource Directory	80

B.4. A note on differences between link-format and Link header fields	82
Appendix C. Limited Link Format	83
Authors' Addresses	83

1. Introduction

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

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

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

2. Terminology

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

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC3986], [RFC8288] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

resolve against

The expression "a URI-reference is *_resolved against_* a base URI" is used to describe the process of [RFC3986] Section 5.2.

Noteworthy corner cases are that if the URI-reference is a (full) URI and resolved against any base URI, that gives the original full URI, and that resolving an empty URI reference gives the base URI without any fragment identifier.

Resource Directory (RD)

A web entity that stores information about web resources and implements the REST interfaces defined in this specification for discovery, for the creation, the maintenance and the removal of registrations, and for lookup of the registered resources.

Sector

In the context of an RD, a sector is a logical grouping of endpoints.

The abbreviation "d=" is used for the sector in query parameters for compatibility with deployed implementations.

Endpoint

Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the RD. An endpoint is identified by its endpoint name, which is included during registration, and has a unique name within the associated sector of the registration.

Registration Base URI

The Base URI of a Registration is a URI that typically gives scheme and authority information about an Endpoint. The Registration Base URI is provided at registration time, and is used by the RD to resolve relative references of the registration into URIs.

Target

The target of a link is the destination address (URI) of the link. It is sometimes identified with "href=", or displayed as "<target>". Relative targets need resolving with respect to the Base URI (section 5.2 of [RFC3986]).

This use of the term Target is consistent with [RFC8288]'s use of the term.

Context

The context of a link is the source address (URI) of the link, and describes which resource is linked to the target. A link's context is made explicit in serialized links as the "anchor=" attribute.

This use of the term Context is consistent with [RFC8288]'s use of the term.

Directory Resource

A resource in the RD containing registration resources.

Registration Resource

A resource in the RD that contains information about an Endpoint and its links.

Commissioning Tool

Commissioning Tool (CT) is a device that assists during installation events by assigning values to parameters, naming endpoints and groups, or adapting the installation to the needs of the applications.

Registrant-ep

Registrant-ep is the endpoint that is registered into the RD. The registrant-ep can register itself, or a CT registers the registrant-ep.

RDAO

Resource Directory Address Option. A new IPv6 Neighbor Discovery option defined for announcing an RD's address.

3. Architecture and Use Cases

3.1. Principles

The RD is primarily a tool to make discovery operations more efficient than querying `/.well-known/core` on all connected devices, or across boundaries that would limit those operations.

It provides information about resources hosted by other devices that could otherwise only be obtained by directly querying the `/.well-known/core` resource on these other devices, either by a unicast request or a multicast request.

Information SHOULD only be stored in the RD if it can be obtained by querying the described device's `/.well-known/core` resource directly.

Data in the RD can only be provided by the device which hosts those data or a dedicated Commissioning Tool (CT). These CTs act on behalf of endpoints too constrained, or generally unable, to present that information themselves. No other client can modify data in the RD. Changes to the information in the RD do not propagate automatically back to the web servers from where the information originated.

3.2. Architecture

The RD architecture is illustrated in Figure 1. An RD is used as a repository of registrations describing resources hosted on other web servers, also called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port. A physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain RD registrations, and for endpoints to lookup resources from the RD. An RD can be logically segmented by the use of Sectors.

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Registrations in the RD are soft state and need to be periodically refreshed.

An endpoint uses specific interfaces to register, update and remove a registration. It is also possible for an RD to fetch Web Links from endpoints and add their contents to its registrations.

At the first registration of an endpoint, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of registrations.

A lookup interface for discovering any of the Web Links stored in the RD is provided using the CoRE Link Format.

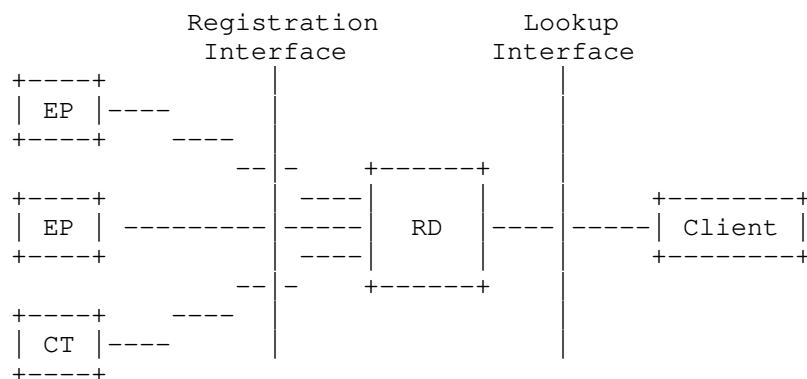


Figure 1: The RD architecture.

A Registrant-EP MAY keep concurrent registrations to more than one RD at the same time if explicitly configured to do so, but that is not expected to be supported by typical EP implementations. Any such registrations are independent of each other. The usual expectation when multiple discovery mechanisms or addresses are configured is that they constitute a fall-back path for a single registration.

3.3. RD Content Model

The Entity-Relationship (ER) models shown in Figure 2 and Figure 3 model the contents of `/.well-known/core` and the RD respectively, with entity-relationship diagrams [ER]. Entities (rectangles) are used for concepts that exist independently. Attributes (ovals) are used for concepts that exist only in connection with a related entity. Relations (diamonds) give a semantic meaning to the relation between entities. Numbers specify the cardinality of the relations.

Some of the attribute values are URIs. Those values are always full URIs and never relative references in the information model. They can, however, be expressed as relative references in serializations, and often are.

These models provide an abstract view of the information expressed in link-format documents and an RD. They cover the concepts, but not necessarily all details of an RD's operation; they are meant to give an overview, and not be a template for implementations.

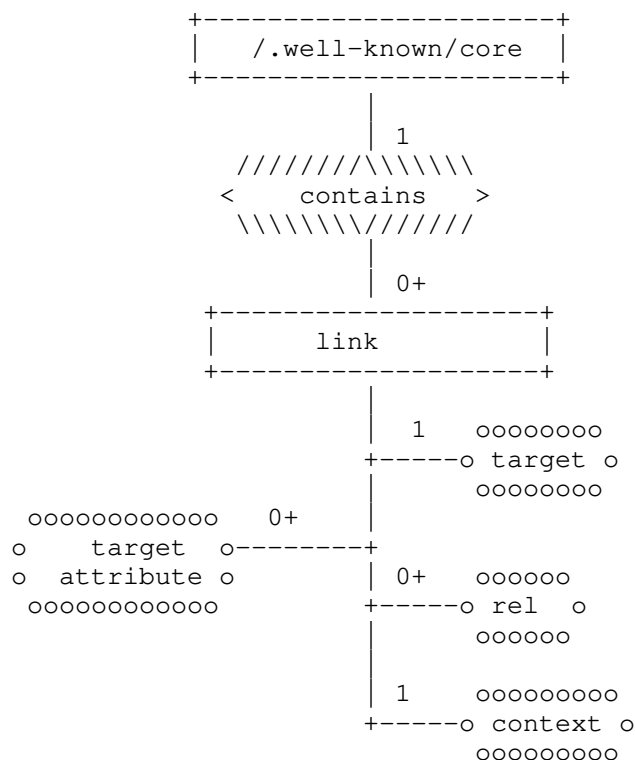


Figure 2: ER Model of the content of `/.well-known/core`

The model shown in Figure 2 models the contents of `/.well-known/core` which contains:

- * a set of links belonging to the hosting web server

The web server is free to choose links it deems appropriate to be exposed in its `"/.well-known/core"`. Typically, the links describe resources that are served by the host, but the set can also contain links to resources on other servers (see examples in [RFC6690] page 14). The set does not necessarily contain links to all resources served by the host.

A link has the following attributes (see [RFC8288]):

- * Zero or more link relations: They describe relations between the link context and the link target.

In link-format serialization, they are expressed as space-separated values in the "rel" attribute, and default to "hosts".

- * A link context URI: It defines the source of the relation, e.g. `_who_ "hosts" something`.

In link-format serialization, it is expressed in the "anchor" attribute. It defaults to that document's URI.

- * A link target URI: It defines the destination of the relation (e.g. `_what_ is hosted`), and is the topic of all target attributes.

In link-format serialization, it is expressed between angular brackets, and sometimes called the "href".

- * Other target attributes (e.g. resource type (rt), interface (if), or content format (ct)). These provide additional information about the target URI.

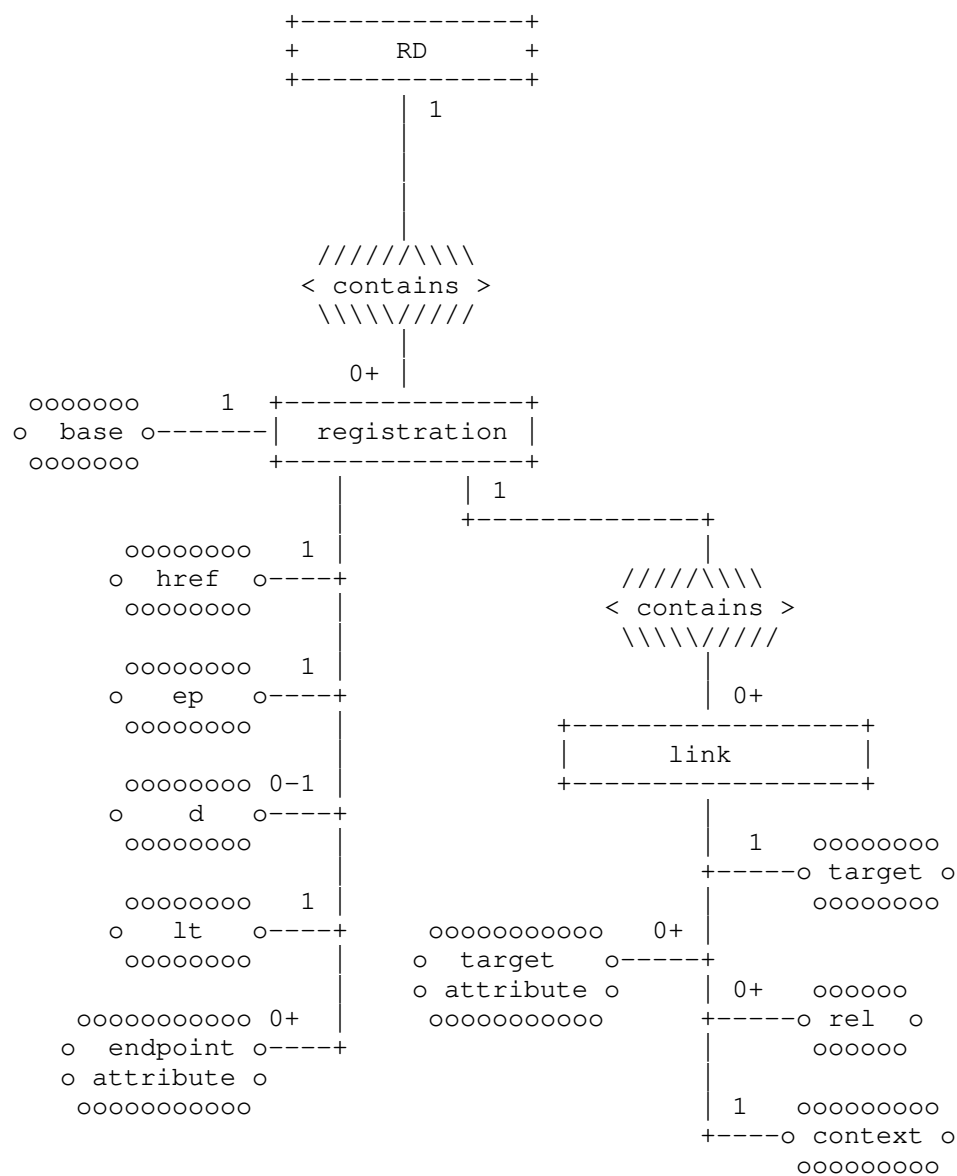


Figure 3: ER Model of the content of the RD

The model shown in Figure 3 models the contents of the RD which contains in addition to /.well-known/core:

* 0 to n Registrations of endpoints,

A registration is associated with one endpoint. A registration defines a set of links as defined for `/.well-known/core`. A Registration has six types of attributes:

- * an endpoint name ("ep", a Unicode string) unique within a sector
- * a Registration Base URI ("base", a URI typically describing the `scheme://authority` part)
- * a lifetime ("lt"),
- * a registration resource location inside the RD ("href"),
- * optionally a sector ("d", a Unicode string)
- * optional additional endpoint attributes (from Section 9.3)

The cardinality of "base" is currently 1; future documents are invited to extend the RD specification to support multiple values (e.g. `[I-D.silverajan-core-coap-protocol-negotiation]`). Its value is used as a Base URI when resolving URIs in the links contained in the endpoint.

Links are modelled as they are in Figure 2.

3.4. Link-local addresses and zone identifiers

Registration Base URIs can contain link-local IP addresses. To be usable across hosts, those cannot be serialized to contain zone identifiers (see [\[RFC6874\]](#) Section 1).

Link-local addresses can only be used on a single link (therefore RD servers cannot announce them when queried on a different link), and lookup clients using them need to keep track of which interface they got them from.

Therefore, it is advisable in many scenarios to use addresses with larger scope if available.

3.5. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded wireless interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. The ambition in such systems is to build them from reusable components. These speed up

development and deployment, and enable shared use of machines across different applications. One crucial component of such systems is the discovery of resources (and thus the endpoints they are hosted on) capable of providing required information at a given time or acting on instructions from the end users.

Imagine a scenario where endpoints installed on vehicles enable tracking of the position of these vehicles for fleet management purposes and allow monitoring of environment parameters. During the boot-up process endpoints register with an RD, which is hosted by the mobile operator or somewhere in the cloud. Periodically, these endpoints update their registration and may modify resources they offer.

When endpoints are not always connected, for example because they enter a sleep mode, a remote server is usually used to provide proxy access to the endpoints. Mobile apps or web applications for environment monitoring contact the RD, look up the endpoints capable of providing information about the environment using an appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server), and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look up for EPs deployed on the vehicles the application is responsible for.

3.6. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of IoT web services. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices. Both can use the common RD infrastructure to establish device interactions efficiently, but can pick security policies suitable for their needs.

Two phases can be discerned for a network servicing the system: (1) installation and (2) operation. During the operational phase, the network is connected to the Internet with a Border Router (e.g. a 6LoWPAN Border Router (6LBR), see {{RFC6775}}) and the nodes connected to the network can use the Internet services that are provided by the Internet Provider or the network administrator. During the installation phase, the network is completely stand-alone, no Border Router is connected, and the network only supports the IP communication between the connected nodes. The installation phase is

usually followed by the operational phase. As an RD's operations work without hard dependencies on names or addresses, it can be used for discovery across both phases.

3.7. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge beforehand of who is going to consume the data. An RD can be used to hold links about resources and services hosted anywhere to make them discoverable by a general class of applications.

For example, environmental and weather sensors that generate data for public consumption may provide data to an intermediary server, or broker. Sensor data are published to the intermediary upon changes or at regular intervals. Descriptions of the sensors that resolve to links to sensor data may be published to an RD. Applications wishing to consume the data can use RD Lookup to discover and resolve links to the desired resources and endpoints. The RD service need not be coupled with the data intermediary service. Mapping of RDs to data intermediaries may be many-to-many.

Metadata in web link formats like [RFC6690] which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links, need to be supported by RDs. External catalogues that are represented in other formats may be converted to common web linking formats for storage and access by RDs. Since it is common practice for these to be encoded in URNs [RFC8141], simple and lossless structural transforms should generally be sufficient to store external metadata in RDs.

The additional features of an RD allow sectors to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Application groups with multicast addresses may be defined to support efficient data transport.

4. RD discovery and other interface-independent components

This and the following sections define the required set of REST interfaces between an RD, endpoints and lookup clients. Although the examples throughout these sections assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. The multicast discovery and simple registration operations are exceptions to that, as they rely on mechanisms unavailable in HTTP. In all definitions in these sections, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces.

All operations on the contents of the RD MUST be atomic and idempotent.

For several operations, interface templates are given in list form; those describe the operation participants, request codes, URIs, content formats and outcomes. Sections of those templates contain normative content about Interaction, Method, URI Template and URI Template Variables as well as the details of the Success condition. The additional sections on options like Content-Format and on Failure codes give typical cases that an implementation of the RD should deal with. Those serve to illustrate the typical responses to readers who are not yet familiar with all the details of CoAP based interfaces; they do not limit what a server may respond under atypical circumstances.

REST clients (registrant-EPs and CTs during registration and maintenance, lookup clients, RD servers during simple registrations) must be prepared to receive any unsuccessful code and act upon it according to its definition, options and/or payload to the best of their capabilities, falling back to failing the operation if recovery is not possible. In particular, they SHOULD retry the request upon 5.03 (Service Unavailable; 503 in HTTP) according to the Max-Age (Retry-After in HTTP) option, and SHOULD fall back to link-format when receiving 4.15 (Unsupported Content-Format; 415 in HTTP).

An RD MAY make the information submitted to it available to further directories (subject to security policies on link confidentiality), if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

4.1. Finding a Resource Directory

A (re-)starting device may want to find one or more RDs before it can discover their URIs. Dependent on the operational conditions, one or more of the techniques below apply.

The device may be pre-configured to exercise specific mechanisms for finding the RD:

1. It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close; each target network environment in which some of these preconfigured nodes are to be brought up is then configured with a route for this anycast address that leads to an appropriate RD. (Instead of using an anycast address, a multicast address can also be preconfigured. The RD servers then need to configure one of their interfaces with this multicast address.)
2. It may be configured with a DNS name for the RD and use DNS to return the IP address of the RD; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.
3. It may be configured to use a service discovery mechanism such as DNS-SD, as outlined in Section 4.1.2.

For cases where the device is not specifically configured with a way to find an RD, the network may want to provide a suitable default.

1. The IPv6 Neighbor Discovery option RDAO Section 4.1.1 can do that.
2. When DHCP is in use, this could be provided via a DHCP option (no such option is defined at the time of writing).

Finally, if neither the device nor the network offers any specific configuration, the device may want to employ heuristics to find a suitable RD.

The present specification does not fully define these heuristics, but suggests a number of candidates:

1. In a 6LoWPAN, just assume the Border Router (6LBR) can act as an RD (using the ABRO option to find that [RFC6775]). Confirmation can be obtained by sending a unicast to "coap://[6LBR]/.well-known/core?rt=core.rd*".
2. In a network that supports multicast well, discovering the RD using a multicast query for /.well-known/core as specified in CoRE Link Format [RFC6690]: Sending a Multicast GET to "coap://[MCD1]/.well-known/core?rt=core.rd*". RDs within the multicast scope will answer the query.

When answering a multicast request directed at a link-local group, the RD may want to respond from a routable address; this makes it easier for registrants to use one of their own routable addresses for

registration. When [RFC6724] is used for source address selection, this can be achieved by applying the changes of its Section 10.4, picking public addresses in its Section 5 Rule 7, and superseding rule 8 with preferring the source address's precedence.

As some of the RD addresses obtained by the methods listed here are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

The following RD discovery mechanisms are recommended:

- * In managed networks with border routers that need stand-alone operation, the RDAO option is recommended (e.g. operational phase described in Section 3.6).
- * In managed networks without border router (no Internet services available), the use of a preconfigured anycast address is recommended (e.g. installation phase described in Section 3.6).
- * In networks managed using DNS-SD, the use of DNS-SD for discovery as described in Section 4.1.2 is recommended.

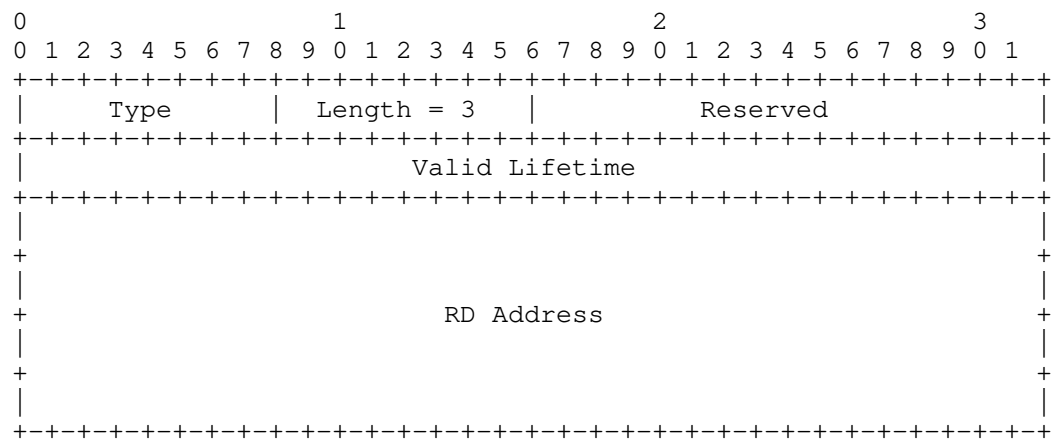
The use of multicast discovery in mesh networks is NOT RECOMMENDED.

4.1.1. Resource Directory Address Option (RDAO)

The Resource Directory Address Option (RDAO) carries information about the address of the RD in RAs (Router Advertisements) of IPv6 Neighbor Discovery (ND), similar to how RDNSS options [RFC8106] are sent. This information is needed when endpoints cannot discover the RD with a link-local or realm-local scope multicast address, for instance because the endpoint and the RD are separated by a Border Router (6LBR). In many circumstances the availability of DHCP cannot be guaranteed either during commissioning of the network. The presence and the use of the RD is essential during commissioning.

It is possible to send multiple RDAO options in one message, indicating as many RD addresses.

The RDAO format is:



Fields:

- Type: TBD38
- Length: 8-bit unsigned integer. The length of the option in units of 8 bytes. Always 3.
- Reserved: This field is unused. It MUST be initialized to zero by the sender and MUST be ignored by the receiver.
- Valid Lifetime: 32-bit unsigned integer. The length of time in seconds (relative to the time the packet is received) that this RD address is valid. A value of all zero bits (0x0) indicates that this RD address is not valid anymore.
- RD Address: IPv6 address of the RD.

Figure 4: Resource Directory Address Option

4.1.2. Using DNS-SD to discover a Resource Directory

An RD can advertise its presence in DNS-SD [RFC6763] using the service name "_core-rd._udp" (for CoAP), "_core-rd-dtls._udp" (for CoAP over DTLS), "_core-rd._tcp" (for CoAP over TCP) or "_core-rd-tls._tcp" (for CoAP over TLS) defined in this document. (For the WebSocket transports of CoAP, no service is defined as DNS-SD is typically unavailable in environments where CoAP over WebSockets is used).

The selection of the service indicates the protocol used, and the SRV record points the client to a host name and port to use as a starting point for the URI discovery steps of Section 4.3.

This section is a simplified concrete application of the more generic mechanism specified in [I-D.ietf-core-rd-dns-sd].

4.2. Payload Content Formats

RDs implementing this specification MUST support the application/link-format content format (ct=40).

RDs implementing this specification MAY support additional content formats.

Any additional content format supported by an RD implementing this specification SHOULD be able to express all the information expressible in link-format. It MAY be able to express information that is inexpressible in link-format, but those expressions SHOULD be avoided where possible.

4.3. URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. This section defines discovery of the RD and its URIs using the well-known interface of the CoRE Link Format [RFC6690] after having discovered a host as described in Section 4.1.

Discovery of the RD registration URI is performed by sending either a multicast or unicast GET request to "/.well-known/core" and including a Resource Type (rt) parameter [RFC6690] with the value "core.rd" in the query string. Likewise, a Resource Type parameter value of "core.rd-lookup*" is used to discover the URIs for RD Lookup operations, core.rd* is used to discover all URIs for RD operations. Upon success, the response will contain a payload with a link format entry for each RD function discovered, indicating the URI of the RD function returned and the corresponding Resource Type. When

performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network (see Section 9.5).

An RD MAY provide hints about the content-formats it supports in the links it exposes or registers, using the "ct" target attribute, as shown in the example below. Clients MAY use these hints to select alternate content-formats for interaction with the RD.

HTTP does not support multicast and consequently only unicast discovery can be supported at the using the HTTP `"/.well-known/core"` resource.

RDs implementing this specification MUST support query filtering for the `rt` parameter as defined in [RFC6690].

While the link targets in this discovery step are often expressed in path-absolute form, this is not a requirement. Clients of the RD SHOULD therefore accept URIs of all schemes they support, both as URIs and relative references, and not limit the set of discovered URIs to those hosted at the address used for URI discovery.

With security policies where the client requires the RD to be authorized to act as an RD, that authorization may be limited to resources on which the authorized RD advertises the adequate resource types. Clients that have obtained links they can not rely on yet can repeat the URI discovery step at the `/.well-known/core` resource of the indicated host to obtain the resource type information from an authorized source.

The URI Discovery operation can yield multiple URIs of a given resource type. The client of the RD can use any of the discovered addresses initially.

The discovery request interface is specified as follows (this is exactly the Well-Known Interface of [RFC6690] Section 4, with the additional requirement that the server MUST support query filtering):

Interaction: EP, CT or Client -> RD

Method: GET

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

URI Template Variables: `rt` := Resource Type. SHOULD contain one of the values `"core.rd"`, `"core.rd-lookup*"`, `"core.rd-lookup-res"`, `"core.rd-lookup-ep"`, or `"core.rd*"`

Accept: absent, application/link-format or any other media type representing web links

The following response is expected on this interface:

Success: 2.05 "Content" or 200 "OK" with an application/link-format or other web link payload containing one or more matching entries for the RD resource.

The following example shows an endpoint discovering an RD using this interface, thus learning that the directory resource location, in this example, is /rd, and that the content-format delivered by the server hosting the resource is application/link-format (ct=40). Note that it is up to the RD to choose its RD locations.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content

```
</rd>;rt="core.rd";ct=40,  
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct=40,  
</rd-lookup/res>;rt="core.rd-lookup-res";ct=40
```

Figure 5: Example discovery exchange

The following example shows the way of indicating that a client may request alternate content-formats. The Content-Format code attribute "ct" MAY include a space-separated sequence of Content-Format codes as specified in Section 7.2.1 of [RFC7252], indicating that multiple content-formats are available. The example below shows the required Content-Format 40 (application/link-format) indicated as well as a CBOR and JSON representation from [I-D.ietf-core-links-json] (which have no numeric values assigned yet, so they are shown as TBD64 and TBD504 as in that draft). The RD resource locations /rd, and /rd-lookup are example values. The server in this example also indicates that it is capable of providing observation on resource lookups.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content

```
</rd>;rt="core.rd";ct="40 65225",  
</rd-lookup/res>;rt="core.rd-lookup-res";ct="40 TBD64 TBD504";obs,  
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct="40 TBD64 TBD504"
```

Figure 6: Example discovery exchange indicating additional content-formats

From a management and maintenance perspective, it is necessary to identify the components that constitute the RD server. The identification refers to information about for example client-server incompatibilities, supported features, required updates and other aspects. The URI discovery address, as 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"
```

Figure 7: Example exchange of obtaining implementation information, using the relation type currently proposed in the work-in-progress document

Note that depending on the particular server's architecture, such a link could be anchored at the RD server's root, 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. Registration

After discovering the location of an RD, a registrant-ep or CT MAY register the resources of the registrant-ep using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690] or other representations of web links, along with query parameters indicating the name of the endpoint, and optionally the sector, lifetime and base URI of the registration. It is expected that other specifications will define further parameters (see Section 9.3). The RD then creates a new registration resource in the RD and returns its location. The receiving endpoint MUST use that location when refreshing registrations using this interface. Registration resources in the RD are kept active for the period indicated by the lifetime parameter. The creating endpoint is responsible for refreshing the registration resource within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameters ep and d (sector) does not create multiple registration resources.

The following rules apply for a registration request targeting a given (ep, d) value pair:

- * When the (ep, d) value pair of the registration-request is different from any existing registration, a new registration is generated.
- * When the (ep, d) value pair of the registration-request is equal to an existing registration, the content and parameters of the existing registration are replaced with the content of the registration request. Like the later changes to registration resources, security policies (Section 7) usually require such requests to come from the same device.

The posted link-format document can (and typically does) contain relative references both in its link targets and in its anchors, or contain empty anchors. The RD server needs to resolve these references in order to faithfully represent them in lookups. They are resolved against the base URI of the registration, which is provided either explicitly in the "base" parameter or constructed implicitly from the requester's URI as constructed from its network address and scheme.

For media types to which Appendix C applies (i.e. documents in application/link-format), the RD only needs to accept representations in Limited Link Format as described there. Its behavior with representations outside that subset is implementation defined.

The registration request interface is specified as follows:

Interaction: EP or CT -> RD

Method: POST

URI Template: {+rd}{?ep,d,lt,base,extra-attrs*}

URI Template Variables: rd := RD registration URI (mandatory).
This is the location of the RD, as obtained from discovery.

ep := Endpoint name (mostly mandatory).
The endpoint name is an identifier that MUST be unique within a sector. As the endpoint name is a Unicode string, it is encoded in UTF-8 (and possibly pct-encoded) during variable expansion (see [RFC6570] Section 3.2.1). The endpoint name MUST NOT contain any character in the inclusive ranges 0-31 or 127-159. The maximum length of this parameter is 63 UTF-8 encoded bytes. If the RD is configured to recognize the endpoint to be authorized to use exactly one endpoint name, the

RD assigns that name. In that case, giving the endpoint name becomes optional for the client; if the client gives any other endpoint name, it is not authorized to perform the registration.

`d` := Sector (optional). The sector to which this endpoint belongs. When this parameter is not present, the RD MAY associate the endpoint with a configured default sector (possibly based on the endpoint's authorization) or leave it empty. The sector is encoded like the `ep` parameter, and is limited to 63 UTF-8 encoded bytes as well.

`lt` := Lifetime (optional). Lifetime of the registration in seconds. Range of 1-4294967295. If no lifetime is included in the initial registration, a default value of 90000 (25 hours) SHOULD be assumed.

`base` := Base URI (optional). This parameter sets the base URI of the registration, under which the relative links in the payload are to be interpreted. The specified URI typically does not have a path component of its own, and MUST be suitable as a base URI to resolve any relative references given in the registration. The parameter is therefore usually of the shape "scheme://authority" for HTTP and CoAP URIs. The URI SHOULD NOT have a query or fragment component as any non-empty relative part in a reference would remove those parts from the resulting URI.

In the absence of this parameter the scheme of the protocol, source address and source port of the registration request are assumed. The Base URI is consecutively constructed by concatenating the used protocol's scheme with the characters "://", the requester's source address as an address literal and ":" followed by its port (if it was not the protocol's default one) in analogy to [RFC7252] Section 6.5.

This parameter is mandatory when the directory is filled by a third party such as an commissioning tool.

If the registrant-`ep` uses an ephemeral port to register with, it MUST include the base parameter in the registration to provide a valid network path.

A registrant that cannot be reached by potential lookup clients at the address it registers from (e.g. because it is behind some form of Network Address Translation (NAT)) MUST provide a reachable base address with its registration.

If the Base URI contains a link-local IP literal, it MUST NOT contain a Zone Identifier, and MUST be local to the link on which the registration request is received.

Endpoints that register with a base that contains a path component cannot meaningfully use [RFC6690] Link Format due to its prevalence of the Origin concept in relative reference resolution. Those applications should use different representations of links to which Appendix C is not applicable (e.g. [I-D.hartke-t2trg-coral]).

extra-attrs := Additional registration attributes (optional). The endpoint can pass any parameter registered at Section 9.3 to the directory. If the RD is aware of the parameter's specified semantics, it processes it accordingly. Otherwise, it MUST store the unknown key and its value(s) as an endpoint attribute for further lookup.

Content-Format: application/link-format or any other indicated media type representing web links

The following response is expected on this interface:

Success: 2.01 "Created" or 201 "Created". The Location-Path option or Location header field MUST be included in the response. This location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration resource. The registration resource location thus returned is for the purpose of updating the lifetime of the registration and for maintaining the content of the registered links, including updating and deleting links.

A registration with an already registered ep and d value pair responds with the same success code and location as the original registration; the set of links registered with the endpoint is replaced with the links from the payload.

The location MUST NOT have a query or fragment component, as that could conflict with query parameters during the Registration Update operation. Therefore, the Location-Query option MUST NOT be present in a successful response.

If the registration fails, including request timeouts, or if delays from Service Unavailable responses with Max-Age or Retry-After accumulate to exceed the registrant's configured timeouts, it SHOULD pick another registration URI from the "URI Discovery" step and if there is only one or the list is exhausted, pick other choices from the "Finding a Resource Directory" step. Care has to be taken to

consider the freshness of results obtained earlier, e.g. of the result of a `"/.well-known/core"` response, the lifetime of an RDAO option and of DNS responses. Any rate limits and persistent errors from the "Finding a Resource Directory" step must be considered for the whole registration time, not only for a single operation.

The following example shows a registrant-ep with the name "node1" registering two resources to an RD using this interface. The location `"/rd"` is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=node1
Content-Format: 40
Payload:
</sensors/temp>;rt="temperature-c";if="sensor",
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel="describedby"

Res: 2.01 Created
Location-Path: /rd/4521
```

Figure 8: Example registration payload

An RD may optionally support HTTP. Here is an example of almost the same registration operation above, when done using HTTP.

```
Req:
POST /rd?ep=node1&base=http://[2001:db8:1::1] HTTP/1.1
Host: rd.example.com
Content-Type: application/link-format

</sensors/temp>;rt="temperature-c";if="sensor",
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel="describedby"

Res:
HTTP/1.1 201 Created
Location: /rd/4521
```

Figure 9: Example registration payload as expressed using HTTP

5.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to an RD as described in Section 5. Instead, simple endpoints can implement the Simple Registration approach described in this section. An RD implementing this specification **MUST** implement Simple Registration. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the registrant-ep makes available the hosted resources that it wants to be discovered, as links on its `"/.well-known/core"` interface as specified in [RFC6690]. The links in that document are subject to the same limitations as the payload of a registration (with respect to Appendix C).

- * The registrant-ep finds one or more addresses of the directory server as described in Section 4.1.
- * The registrant-ep sends (and regularly refreshes with) a POST request to the `"/.well-known/rd"` URI of the directory server of choice. The body of the POST request is empty, and triggers the resource directory server to perform GET requests at the requesting registrant-ep's `/.well-known/core` to obtain the link-format payload to register.

The registrant-ep includes the same registration parameters in the POST request as it would per Section 5. The registration base URI of the registration is taken from the registrant-ep's network address (as is default with regular registrations).

Example request from registrant-EP to RD (unanswered until the next step):

```
Req: POST /.well-known/rd?lt=6000&ep=node1
(No payload)
```

Figure 10: First half example exchange of a simple registration

- * The RD queries the registrant-ep's discovery resource to determine the success of the operation. It **SHOULD** keep a cache of the discovery resource and not query it again as long as it is fresh.

Example request from the RD to the registrant-EP:

```
Req: GET /.well-known/core
Accept: 40
```

```
Res: 2.05 Content
Content-Format: 40
Payload:
</sen/temp>
```

Figure 11: Example exchange of the RD querying the simple endpoint

With this response, the RD would answer the previous step's request:

```
Res: 2.04 Changed
```

Figure 12: Second half example exchange of a simple registration

The sequence of fetching the registration content before sending a successful response was chosen to make responses reliable, and the point about caching was chosen to still allow very constrained registrants. Registrants MUST be able to serve a GET request to `"/.well-known/core"` after having requested registration. Constrained devices MAY regard the initial request as temporarily failed when they need RAM occupied by their own request to serve the RD's GET, and retry later when the RD already has a cached representation of their discovery resources. Then, the RD can reply immediately and the registrant can receive the response.

The simple registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: `/.well-known/rd{?ep,d,lt,extra-attrs*}`

URI Template Variables are as they are for registration in Section 5. The base attribute is not accepted to keep the registration interface simple; that rules out registration over CoAP-over-TCP or HTTP that would need to specify one. For some time during this document's development, the URI template `"/.well-known/core{?ep,...}"` has been in use instead.

The following response is expected on this interface:

```
Success: 2.04 "Changed".
```

For the second interaction triggered by the above, the registrant-ep takes the role of server and the RD the role of client. (Note that this is exactly the Well-Known Interface of [RFC6690] Section 4):

Interaction: RD -> EP

Method: GET

URI Template: /.well-known/core

The following response is expected on this interface:

Success: 2.05 "Content".

When the RD is in a position to successfully execute this second interaction and other network participants that can reach it are not, it SHOULD verify that the apparent registrant-ep intends to register with the given registration parameters before revealing the obtained discovery information to lookup clients. An easy way to do that is to verify the simple registration request's sender address using the Echo option as described in [I-D.ietf-core-echo-request-tag] Section 2.4.

The RD MUST delete registrations created by simple registration after the expiration of their lifetime. Additional operations on the registration resource cannot be executed because no registration location is returned.

5.2. Third-party registration

For some applications, even Simple Registration may be too taxing for some very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the RD can be filled by a third party device, called a Commissioning Tool (CT). The commissioning tool can fill the RD from a database or other means. For that purpose scheme, IP address and port of the URI of the registered device is the value of the "base" parameter of the registration described in Section 5.

It should be noted that the value of the "base" parameter applies to all the links of the registration and has consequences for the anchor value of the individual links as exemplified in Appendix B. An eventual (currently non-existing) "base" attribute of the link is not affected by the value of "base" parameter in the registration.

5.3. Operations on the Registration Resource

This section describes how the registering endpoint can maintain the registrations that it created. The registering endpoint can be the registrant-ep or the CT. The registrations are resources of the RD.

An endpoint should not use this interface for registrations that it did not create. This is usually enforced by security policies, which in general require equivalent credentials for creation of and operations on a registration.

After the initial registration, the registering endpoint retains the returned location of the Registration Resource for further operations, including refreshing the registration in order to extend the lifetime and "keep-alive" the registration. When the lifetime of the registration has expired, the RD SHOULD NOT respond to discovery queries concerning this endpoint. The RD SHOULD continue to provide access to the Registration Resource after a registration time-out occurs in order to enable the registering endpoint to eventually refresh the registration. The RD MAY eventually remove the registration resource for the purpose of garbage collection. If the Registration Resource is removed, the corresponding endpoint will need to be re-registered.

The Registration Resource may also be used cancel the registration using DELETE, and to perform further operations beyond the scope of this specification.

The operations on the Registration Resource are described below.

5.3.1. Registration Update

The update interface is used by the registering endpoint to refresh or update its registration with an RD. To use the interface, the registering endpoint sends a POST request to the registration resource returned by the initial registration operation.

An update MAY update registration parameters like lifetime, base URI or others. Parameters that are not being changed should not be included in an update. Adding parameters that have not changed increases the size of the message but does not have any other implications. Parameters are included as query parameters in an update operation as in Section 5.

A registration update resets the timeout of the registration to the (possibly updated) lifetime of the registration, independent of whether a "lt" parameter was given.

If the base URI of the registration is changed in an update, relative references submitted in the original registration or later updates are resolved anew against the new base.

The registration update operation only describes the use of POST with an empty payload. Future standards might describe the semantics of using content formats and payloads with the POST method to update the links of a registration (see Section 5.3.3).

The update registration request interface is specified as follows:

Interaction: EP or CT -> RD

Method: POST

URI Template: {+location}{?lt,base,extra-attrs*}

URI Template Variables: location := This is the Location returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 1-4294967295. If no lifetime is included, the previous last lifetime set on a previous update or the original registration (falling back to 90000) SHOULD be used.

base := Base URI (optional). This parameter updates the Base URI established in the original registration to a new value, and is subject to the same restrictions as in the registration. If the parameter is set in an update, it is stored by the RD as the new Base URI under which to interpret the relative links present in the payload of the original registration. If the parameter is not set in the request but was set before, the previous Base URI value is kept unmodified. If the parameter is not set in the request and was not set before either, the source address and source port of the update request are stored as the Base URI.

extra-attrs := Additional registration attributes (optional). As with the registration, the RD processes them if it knows their semantics. Otherwise, unknown attributes are stored as endpoint attributes, overriding any previously stored endpoint attributes of the same key.

Note that this default behavior does not allow removing an endpoint attribute in an update. For attributes whose functionality depends on the endpoints' ability to remove them in an update, it can make sense to define a value whose

presence is equivalent to the absence of a value. As an alternative, an extension can define different updating rules for their attributes. That necessitates either discovery of whether the RD is aware of that extension, or tolerating the default behavior.

Content-Format: none (no payload)

The following responses are expected on this interface:

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

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

If the registration update fails in any way, including "Not Found" and request timeouts, or if the time indicated in a Service Unavailable Max-Age/Retry-After exceeds the remaining lifetime, the registering endpoint SHOULD attempt registration again.

The following example shows how the registering endpoint resets the timeout on its registration resource at an RD using this interface with the example location value: /rd/4521.

Req: POST /rd/4521

Res: 2.04 Changed

Figure 13: Example update of a registration

The following example shows the registering endpoint updating its registration resource at an RD using this interface with the example location value: /rd/4521. The initial registration by the registering endpoint set the following values:

- * endpoint name (ep)=endpoint1
- * lifetime (lt)=500
- * Base URI (base)=coap://local-proxy-old.example.com:5683
- * payload of Figure 8

The initial state of the RD is reflected in the following request:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.05 Content
Payload:
<coap://local-proxy-old.example.com:5683/sensors/temp>;
  rt="temperature-c";if="sensor";
  anchor="coap://local-proxy-old.example.com:5683/",
<http://www.example.com/sensors/temp>;
  anchor="coap://local-proxy-old.example.com:5683/sensors/temp";
  rel="describedby"
```

Figure 14: Example lookup before a change to the base address

The following example shows the registering endpoint changing the Base URI to "coaps://new.example.com:5684":

```
Req: POST /rd/4521?base=coaps://new.example.com:5684

Res: 2.04 Changed
```

Figure 15: Example registration update that changes the base address

The consecutive query returns:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.05 Content
Payload:
<coap://new.example.com:5684/sensors/temp>;
  rt="temperature-c";if="sensor";
  anchor="coap://new.example.com:5684/",
<http://www.example.com/sensors/temp>;
  anchor="coap://new.example.com:5684/sensors/temp";
  rel="describedby"
```

Figure 16: Example lookup after a change to the base address

5.3.2. Registration Removal

Although RD registrations have soft state and will eventually timeout after their lifetime, the registering endpoint SHOULD explicitly remove an entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP or CT -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables: location := This is the Location returned by the RD as a result of a successful earlier registration.

The following responses are expected on this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may already have been removed).

The following examples shows successful removal of the endpoint from the RD with example location value /rd/4521.

Req: DELETE /rd/4521

Res: 2.02 Deleted

Figure 17: Example of a registration removal

5.3.3. Further operations

Additional operations on the registration can be specified in future documents, for example:

- * Send iPATCH (or PATCH) updates ([RFC8132]) to add, remove or change the links of a registration.
- * Use GET to read the currently stored set of links in a registration resource.

Those operations are out of scope of this document, and will require media types suitable for modifying sets of links.

6. RD Lookup

To discover the resources registered with the RD, a lookup interface must be provided. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. JSON or CBOR link format [I-D.ietf-core-links-json]) or using more advanced interfaces (e.g. supporting context or semantic based lookup) on different resources that are discovered independently.

RD Lookup allows lookups for endpoints and resources using attributes defined in this document and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, an endpoint lookup MUST return a list of endpoints and a resource lookup MUST return a list of links to resources.

The lookup type is selected by a URI endpoint, which is indicated by a Resource Type as per Table 1 below:

Lookup Type	Resource Type	Mandatory
Resource	core.rd-lookup-res	Mandatory
Endpoint	core.rd-lookup-ep	Mandatory

Table 1: Lookup Types

6.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD by the registrant. The links and link parameters returned by the lookup are equal to the originally submitted ones, except that the target and anchor references are fully resolved.

Links that did not have an anchor attribute are therefore returned with the base URI of the registration as the anchor. Links of which href or anchor was submitted as a (full) URI are returned with these attributes unmodified.

The above rules allow the client to interpret the response as links without any further knowledge of the storage conventions of the RD. The RD MAY replace the registration base URIs with a configured intermediate proxy, e.g. in the case of an HTTP lookup interface for CoAP endpoints.

If the base URI of a registration contains a link-local address, the RD MUST NOT show its links unless the lookup was made from the link on which the registered endpoint can be reached. The RD MUST NOT include zone identifiers in the resolved URIs.

6.2. Lookup filtering

Using the Accept Option, the requester can control whether the returned list is returned in CoRE Link Format ("application/link-format", default) or in alternate content-formats (e.g. from [I-D.ietf-core-links-json]).

Multiple search criteria MAY be included in a lookup. All included criteria MUST match for a link to be returned. The RD MUST support matching with multiple search criteria.

A link matches a search criterion if it has an attribute of the same name and the same value, allowing for a trailing "*" wildcard operator as in Section 4.1 of [RFC6690]. Attributes that are defined as "relation-types" (in the link-format ABNF) match if the search value matches any of their values (see Section 4.1 of [RFC6690]; e.g. "?if=tag:example.net,2020:sensor" matches ";if=example.regname tag:example.net,2020:sensor;"). A resource link also matches a search criterion if its endpoint would match the criterion, and vice versa, an endpoint link matches a search criterion if any of its resource links matches it.

Note that "href" is a valid search criterion and matches target references. Like all search criteria, on a resource lookup it can match the target reference of the resource link itself, but also the registration resource of the endpoint that registered it. Queries for resource link targets MUST be in URI form (i.e. not relative references) and are matched against a resolved link target. Queries for endpoints SHOULD be expressed in path-absolute form if possible and MUST be expressed in URI form otherwise; the RD SHOULD recognize either. The "anchor" attribute is usable for resource lookups, and, if queried, MUST be in URI form as well.

Additional query parameters "page" and "count" are used to obtain lookup results in specified increments using pagination, where count specifies how many links to return and page specifies which subset of links organized in sequential pages, each containing 'count' links, starting with link zero and page zero. Thus, specifying count of 10 and page of 0 will return the first 10 links in the result set (links 0-9). Count = 10 and page = 1 will return the next 'page' containing links 10-19, and so on.

Endpoints that are interested in a lookup result repeatedly or continuously can use mechanisms like ETag caching, resource observation ([RFC7641]), or any future mechanism that might allow more efficient observations of collections. These are advertised, detected and used according to their own specifications and can be used with the lookup interface as with any other resource.

When resource observation is used, every time the set of matching links changes, or the content of a matching link changes, the RD sends a notification with the matching link set. The notification contains the successful current response to the given request, especially with respect to representing zero matching links (see "Success" item below).

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: {+type-lookup-location}{?page,count,search*}

URI Template Variables: type-lookup-location := RD Lookup URI for a given lookup type (mandatory). The address is discovered as described in Section 4.3.

search := Search criteria for limiting the number of results (optional).

The search criteria are an associative array, expressed in a form-style query as per the URI template (see [RFC6570] Sections 2.4.2 and 3.2.8)

page := Page (optional). Parameter cannot be used without the count parameter. Results are returned from result set in pages that contain 'count' links starting from index (page * count). Page numbering starts with zero.

count := Count (optional). Number of results is limited to this parameter value. If the page parameter is also present, the response MUST only include 'count' links starting with the (page * count) link in the result set from the query. If the count parameter is not present, then the response MUST return all matching links in the result set. Link numbering starts with zero.

Accept: absent, application/link-format or any other indicated media type representing web links

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-

format" or other web link payload containing matching entries for the lookup. The payload can contain zero links (which is an empty payload in [RFC6690] link format, but could also be "[]" in JSON based formats), indicating that no entities matched the request.

6.3. Resource lookup examples

The examples in this section assume the existence of CoAP hosts with a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples.

The following example shows a client performing a resource lookup with the example resource look-up locations discovered in Figure 5:

```
Req: GET /rd-lookup/res?rt=tag:example.org,2020:temperature
```

```
Res: 2.05 Content
```

```
<coap://[2001:db8:3::123]:61616/temp>;  
  rt="tag:example.org,2020:temperature";  
  anchor="coap://[2001:db8:3::123]:61616"
```

Figure 18: Example a resource lookup

A client that wants to be notified of new resources as they show up can use observation:

```
Req: GET /rd-lookup/res?rt=tag:example.org,2020:light
```

```
Observe: 0
```

```
Res: 2.05 Content
```

```
Observe: 23
```

```
Payload: empty
```

(at a later point in time)

```
Res: 2.05 Content
```

```
Observe: 24
```

```
Payload:
```

```
<coap://[2001:db8:3::124]/west>;rt="tag:example.org,2020:light";  
  anchor="coap://[2001:db8:3::124]",  
<coap://[2001:db8:3::124]/south>;rt="tag:example.org,2020:light";  
  anchor="coap://[2001:db8:3::124]",  
<coap://[2001:db8:3::124]/east>;rt="tag:example.org,2020:light";  
  anchor="coap://[2001:db8:3::124]"
```

Figure 19: Example an observing resource lookup

The following example shows a client performing a paginated resource lookup

```
Req: GET /rd-lookup/res?page=0&count=5
```

```
Res: 2.05 Content
```

```
<coap://[2001:db8:3::123]:61616/res/0>;ct=60;  
  anchor="coap://[2001:db8:3::123]:61616",  
<coap://[2001:db8:3::123]:61616/res/1>;ct=60;  
  anchor="coap://[2001:db8:3::123]:61616",  
<coap://[2001:db8:3::123]:61616/res/2>;ct=60;  
  anchor="coap://[2001:db8:3::123]:61616",  
<coap://[2001:db8:3::123]:61616/res/3>;ct=60;  
  anchor="coap://[2001:db8:3::123]:61616",  
<coap://[2001:db8:3::123]:61616/res/4>;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>;ct=60;  
  anchor="coap://[2001:db8:3::123]:61616",  
<coap://[2001:db8:3::123]:61616/res/6>;ct=60;  
  anchor="coap://[2001:db8:3::123]:61616",  
<coap://[2001:db8:3::123]:61616/res/7>;ct=60;  
  anchor="coap://[2001:db8:3::123]:61616",  
<coap://[2001:db8:3::123]:61616/res/8>;ct=60;  
  anchor="coap://[2001:db8:3::123]:61616",  
<coap://[2001:db8:3::123]:61616/res/9>;ct=60;  
  anchor="coap://[2001:db8:3::123]:61616"
```

Figure 20: Examples of paginated resource lookup

The following example shows a client performing a lookup of all resources of all endpoints of a given endpoint type. It assumes that two endpoints (with endpoint names "sensor1" and "sensor2") have previously registered with their respective addresses "coap://sensor1.example.com" and "coap://sensor2.example.com", and posted the very payload of the 6th response of section 5 of [RFC6690].

It demonstrates how absolute link targets stay unmodified, while relative ones are resolved:

```

Req: GET /rd-lookup/res?et=tag:example.com,2020:platform

<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"

```

Figure 21: Example of resource lookup from multiple endpoints

6.4. Endpoint lookup

The endpoint lookup returns links to and information about registration resources, which themselves can only be manipulated by the registering endpoint.

Endpoint registration resources are annotated with their endpoint names (ep), sectors (d, if present) and registration base URI (base; reports the registrant-ep's address if no explicit base was given) as well as a constant resource type (rt="core.rd-ep"); the lifetime (lt) is not reported. Additional endpoint attributes are added as target attributes to their endpoint link unless their specification says otherwise.

Links to endpoints SHOULD be presented in path-absolute form or, if required, as (full) URIs. (This avoids the RFC6690 ambiguities.)

Base addresses that contain link-local addresses MUST NOT include zone identifiers, and such registrations MUST NOT be shown unless the lookup was made from the same link from which the registration was made.

While Endpoint Lookup does expose the registration resources, the RD does not need to make them accessible to clients. Clients SHOULD NOT attempt to dereference or manipulate them.

An RD can report registrations in lookup whose URI scheme and authority differ from the lookup resource's. Lookup clients MUST be prepared to see arbitrary URIs as registration resources in the results and treat them as opaque identifiers; the precise semantics of such links are left to future specifications.

The following example shows a client performing an endpoint lookup limited to endpoints of endpoint type
"tag:example.com,2020:platform":

Req: GET /rd-lookup/ep?et=tag:example.com,2020:platform

Res: 2.05 Content

```
</rd/1234>;base="coap://[2001:db8:3::127]:61616";ep="node5";  
  et="tag:example.com,2020:platform";ct="40";rt="core.rd-ep",  
</rd/4521>;base="coap://[2001:db8:3::129]:61616";ep="node7";  
  et="tag:example.com,2020:platform";ct="40";d="floor-3";  
  rt="core.rd-ep"
```

Figure 22: Examples of endpoint lookup

7. Security policies

The security policies that are applicable to an RD strongly depend on the application, and are not set out normatively here.

This section provides a list of aspects that applications should consider when describing their use of the RD, without claiming to cover all cases. It is using terminology of [I-D.ietf-ace-oauth-authz], in which the RD acts as the Resource Server (RS), and both registrant-eps and lookup clients act as Clients (C) with support from an Authorization Server (AS), without the intention of ruling out other (e.g. certificate / public-key infrastructure (PKI) based) schemes.

Any, all or none of the below can apply to an application. Which are relevant depends on its protection objectives.

Security policies are set by configuration of the RD, or by choice of the implementation. Lookup clients (and, where relevant, endpoints) can only trust an RD to uphold them if it is authenticated, and authorized to serve as an RD according to the application's requirements.

7.1. Endpoint name

Whenever an RD needs to provide trustworthy results to clients doing endpoint lookup, or resource lookup with filtering on the endpoint name, the RD must ensure that the registrant is authorized to use the given endpoint name. This applies both to registration and later to operations on the registration resource. It is immaterial whether the client is the registrant-ep itself or a CT is doing the registration: The RD cannot tell the difference, and CTs may use authorization credentials authorizing only operations on that particular endpoint name, or a wider range of endpoint names.

It is up to the concrete security policy to describe how endpoint name and sector are transported when certificates are used. For example, it may describe how SubjectAltName dNSName entries are mapped to endpoint and domain names.

7.1.1. Random endpoint names

Conversely, in applications where the RD does not check the endpoint name, the authorized registering endpoint can generate a random number (or string) that identifies the endpoint. The RD should then remember unique properties of the registrant, associate them with the registration for as long as its registration resource is active (which may be longer than the registration's lifetime), and require the same properties for operations on the registration resource.

Registrants that are prepared to pick a different identifier when their initial attempt (or attempts, in the unlikely case of two subsequent collisions) at registration is unauthorized should pick an identifier at least twice as long as the expected number of registrants; registrants without such a recovery options should pick significantly longer endpoint names (e.g. using UUID URNs [RFC4122]).

7.2. Entered resources

When lookup clients expect that certain types of links can only originate from certain endpoints, then the RD needs to apply filtering to the links an endpoint may register.

For example, if clients use an RD to find a server that provides firmware updates, then any registrant that wants to register (or update) links to firmware sources will need to provide suitable credentials to do so, independently of its endpoint name.

Note that the impact of having undesirable links in the RD depends on the application: if the client requires the firmware server to present credentials as a firmware server, a fraudulent link's impact

is limited to the client revealing its intention to obtain updates and slowing down the client until it finds a legitimate firmware server; if the client accepts any credentials from the server as long as they fit the provided URI, the impact is larger.

An RD may also require that links are only registered if the registrant is authorized to publish information about the anchor (or even target) of the link. One way to do this is to demand that the registrant present the same credentials as a client that they'd need to present if contacted as a server at the resources' URI, which may include using the address and port that are part of the URI. Such a restriction places severe practical limitations on the links that can be registered.

As above, the impact of undesirable links depends on the extent to which the lookup client relies on the RD. To avoid the limitations, RD applications should consider prescribing that lookup clients only use the discovered information as hints, and describe which pieces of information need to be verified because they impact the application's security. A straightforward way to verify such information is to request it again from an authorized server, typically the one that hosts the target resource. That similar to what happens in Section 4.3 when the URI discovery step is repeated.

7.3. Link confidentiality

When registrants publish information in the RD that is not available to any client that would query the registrant's /.well-known/core interface, or when lookups to that interface are subject to stricter firewalling than lookups to the RD, the RD may need to limit which lookup clients may access the information.

In this case, the endpoint (and not the lookup clients) needs to be careful to check the RD's authorization.

7.4. Segmentation

Within a single RD, different security policies can apply.

One example of this are multi-tenant deployments separated by the sector (d) parameter. Some sectors might apply limitations on the endpoint names available, while others use a random identifier approach to endpoint names and place limits on the entered links based on their attributes instead.

Care must be taken in such setups to determine the applicable access control measures to each operation. One easy way to do that is to mandate the use of the sector parameter on all operations, as no credentials are suitable for operations across sector borders anyway.

7.5. First-Come-First-Remembered: A default policy

The First-Come-First-Remembered policy is provided both as a reference example for a security policy definition, and as a policy that implementations may choose to use as default policy in absence of other configuration. It is designed to enable efficient discovery operations even in ad-hoc settings.

Under this policy, the RD accepts registrations for any endpoint name that is not assigned to an active registration resource, and only accepts registration updates from the same endpoint. The policy is minimal in that towards lookup clients it does not make any of the claims of Section 7.2 and Section 7.3, and its claims on Section 7.1 are limited to the lifetime of that endpoint's registration. It does, however, guarantee towards any endpoint that for the duration of its registration, its links will be discoverable on the RD.

When a registration or operation is attempted, the RD MUST determine the client's subject name or public key:

- * If the client's credentials indicate any subject name that is certified by any authority which the RD recognizes (which may be the system's trust anchor store), all those subject names are stored. With CWT or JWT based credentials (as common with ACE), the Subject (sub) claim is stored as a single name, if it exists. With X.509 certificates, the Common Name (CN) and the complete list of SubjectAltName entries are stored. In both cases, the authority that certified the claim is stored along with the subject, as the latter may only be locally unique.
- * Otherwise, if the client proves possession of a private key, the matching public key is stored. This applies both to raw public keys and to the public keys indicated in certificates that failed the above authority check.
- * If neither is present, a reference to the security session itself is stored. With (D)TLS, that is the connection itself, or the session resumption information if available. With OSCORE, that is the security context.

As part of the registration operation, that information is stored along with the registration resource.

The RD MUST accept all registrations whose registration resource is not already active, as long as they are made using a security layer supported by the RD.

Any operation on a registration resource, including registrations that lead to an existing registration resource, MUST be rejected by the RD unless all the stored information is found in the new request's credentials.

Note that even though subject names are compared in this policy, they are never directly compared to endpoint names, and an endpoint can not expect to "own" any particular endpoint name outside of an active registration - even if a certificate says so. It is an accepted shortcoming of this approach that the endpoint has no indication of whether the RD remembers it by its subject name or public key; recognition by subject happens on a best-effort base (given the RD may not recognize any authority). Clients MUST be prepared to pick a different endpoint name when rejected by the RD initially or after a change in their credentials; picking an endpoint name as per Section 7.1.1 is an easy option for that.

For this policy to be usable without configuration, clients should not set a sector name in their registrations. An RD can set a default sector name for registrations accepted under this policy, which is useful especially in a segmented setup where different policies apply to different sectors. The configuration of such a behavior, as well as any other configuration applicable to such an RD (i.e. the set of recognized authorities) is out of scope for this document.

8. Security Considerations

The security considerations as described in Section 5 of [RFC8288] and Section 6 of [RFC6690] apply. The `"/.well-known/core"` resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252].

Access that is limited or affects sensitive data SHOULD be protected, e.g. using (D)TLS or OSCORE ([RFC8613]; which aspects of the RD this affects depends on the security policies of the application (see Section 7)).

8.1. Discovery

Most steps in discovery of the RD, and possibly its resources, are not covered by CoAP's security mechanisms. This will not endanger the security properties of the registrations and lookup itself (where the client requires authorization of the RD if it expects any security properties of the operation), but may leak the client's intention to third parties, and allow them to slow down the process.

To mitigate that, clients can retain the RD's address, use secure discovery options like configured addresses, and send queries for RDs in a very general form ("?rt=core.rd*" rather than "?rt=core.rd-lookup-ep").

8.2. Endpoint Identification and Authentication

An Endpoint (name, sector) pair is unique within the set of endpoints registered by the RD. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint on an RD SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security.

Consider the following threat: two devices A and B are registered at a single server. Both devices have unique, per-device credentials for use with DTLS to make sure that only parties with authorization to access A or B can do so.

Now, imagine that a malicious device A wants to sabotage the device B. It uses its credentials during the DTLS exchange. Then, it specifies the endpoint name of device B as the name of its own endpoint in device A. If the server does not check whether the identifier provided in the DTLS handshake matches the identifier used at the CoAP layer then it may be inclined to use the endpoint name for looking up what information to provision to the malicious device.

Endpoint authorization needs to be checked on registration and registration resource operations independently of whether there are configured requirements on the credentials for a given endpoint name (and sector; Section 7.1) or whether arbitrary names are accepted (Section 7.1.1).

Simple registration could be used to circumvent address-based access control: An attacker would send a simple registration request with the victim's address as source address, and later look up the victim's /.well-known/core content in the RD. Mitigation for this is recommended in Section 5.1.

The Registration Resource path is visible to any client that is allowed endpoint lookup, and can be extracted by resource lookup clients as well. The same goes for registration attributes that are shown as target attributes or lookup attributes. The RD needs to consider this in the choice of Registration Resource paths, and administrators or endpoint in their choice of attributes.

8.3. Access Control

Access control SHOULD be performed separately for the RD registration and Lookup API paths, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the sector, endpoint or resource level.

The precise access controls necessary (and the consequences of failure to enforce them) depend on the protection objectives of the application and the security policies (Section 7) derived from them.

8.4. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly amplify and distribute a DoS attack as UDP does not require return routability check. Since RD lookup responses can be significantly larger than requests, RDs are prone to this.

[RFC7252] describes this at length in its Section 11.3, including some mitigation by using small block sizes in responses. The upcoming [I-D.ietf-core-echo-request-tag] updates that by describing a source address verification mechanism using the Echo option.

[If this document is published together with or after I-D.ietf-core-echo-request-tag, the above paragraph is replaced with the following:

[RFC7252] describes this at length in its Section 11.3, and [I-D.ietf-core-echo-request-tag] (which updates the former) recommends using the Echo option to verify the request's source address.

]

9. IANA Considerations

9.1. Resource Types

IANA is asked to enter the following values into the Resource Type (rt=) Link Target Attribute Values sub-registry of the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690]:

Value	Description	Reference
core.rd	Directory resource of an RD	RFCTHIS Section 4.3
core.rd-lookup-res	Resource lookup of an RD	RFCTHIS Section 4.3
core.rd-lookup-ep	Endpoint lookup of an RD	RFCTHIS Section 4.3
core.rd-ep	Endpoint resource of an RD	RFCTHIS Section 6

Table 2

9.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the sub-registry "IPv6 Neighbor Discovery Option Formats" of the "Internet Control Message Protocol version 6 (ICMPv6) Parameters" registry:

* Resource Directory Address Option (TBD38)

[The RFC editor is asked to replace TBD38 with the assigned number in the document; the value 38 is suggested.]

9.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include

* the human readable name of the parameter,

- * the short name as used in query parameters or target attributes,
- * indication of whether it can be passed as a query parameter at registration of endpoints, as a query parameter in lookups, or be expressed as a target attribute,
- * syntax and validity requirements if any,
- * a description,
- * and a link to reference documentation.

The query parameter MUST be both a valid URI query key [RFC3986] and a token as used in [RFC8288].

The description must give details on whether the parameter can be updated, and how it is to be processed in lookups.

The mechanisms around new RD parameters should be designed in such a way that they tolerate RD implementations that are unaware of the parameter and expose any parameter passed at registration or updates on in endpoint lookups. (For example, if a parameter used at registration were to be confidential, the registering endpoint should be instructed to only set that parameter if the RD advertises support for keeping it confidential at the discovery step.)

Initial entries in this sub-registry are as follows:

Full name	Short	Validity	Use	Description
Endpoint Name	ep	Unicode*	RLA	Name of the endpoint
Lifetime	lt	1-4294967295	R	Lifetime of the registration in seconds
Sector	d	Unicode*	RLA	Sector to which this endpoint belongs
Registration Base URI	base	URI	RLA	The scheme, address and port and path at which this server is available
Page	page	Integer	L	Used for pagination
Count	count	Integer	L	Used for pagination
Endpoint Type	et	Section 9.3.1	RLA	Semantic type of the endpoint (see Section 9.4)

Table 3: RD Parameters

(Short: Short name used in query parameters or target attributes.
 Validity: Unicode* = 63 Bytes of UTF-8 encoded Unicode, with no control characters as per Section 5. Use: R = used at registration, L = used at lookup, A = expressed in target attribute.)

The descriptions for the options defined in this document are only summarized here. To which registrations they apply and when they are to be shown is described in the respective sections of this document. All their reference documentation entries point to this document.

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC8126]. The evaluation should consider formal criteria, duplication of functionality (Is the new entry redundant with an existing one?), topical suitability (E.g. is the described property actually a property of the endpoint and not a property of a particular resource, in which case it should go into the payload of the registration and need not be registered?), and the potential for conflict with commonly used target attributes (For

example, "if" could be used as a parameter for conditional registration if it were not to be used in lookup or attributes, but would make a bad parameter for lookup, because a resource lookup with an "if" query parameter could ambiguously filter by the registered endpoint property or the [RFC6690] target attribute).

9.3.1. Full description of the "Endpoint Type" RD Parameter

An endpoint registering at an RD can describe itself with endpoint types, similar to how resources are described with Resource Types in [RFC6690]. An endpoint type is expressed as a string, which can be either a URI or one of the values defined in the Endpoint Type sub-registry. Endpoint types can be passed in the "et" query parameter as part of extra-attrs at the Registration step, are shown on endpoint lookups using the "et" target attribute, and can be filtered for using "et" as a search criterion in resource and endpoint lookup. Multiple endpoint types are given as separate query parameters or link attributes.

Note that Endpoint Type differs from Resource Type in that it uses multiple attributes rather than space separated values. As a result, RDs implementing this specification automatically support correct filtering in the lookup interfaces from the rules for unknown endpoint attributes.

9.4. "Endpoint Type" (et=) RD Parameter values

This specification establishes a new sub-registry under "CoRE Parameters" called '"Endpoint Type" (et=) RD Parameter values'. The registry properties (required policy, requirements, template) are identical to those of the Resource Type parameters in [RFC6690], in short:

The review policy is IETF Review for values starting with "core", and Specification Required for others.

The requirements to be enforced are:

- * The values MUST be related to the purpose described in Section 9.3.1.
- * The registered values MUST conform to the ABNF reg-rel-type definition of [RFC6690] and MUST NOT be a URI.
- * It is recommended to use the period "." character for segmentation.

The registry initially contains one value:

- * "core.rd-group": An application group as described in Appendix A.

9.5. Multicast Address Registration

IANA is asked to assign the following multicast addresses for use by CoAP nodes:

IPv4 - "all CoRE Resource Directories" address MCD2 (suggestion: 224.0.1.189), from the "IPv4 Multicast Address Space Registry". As the address is used for discovery that may span beyond a single network, it has come from the Internetwork Control Block (224.0.1.x) [RFC5771].

IPv6 - "all CoRE Resource Directories" address MCD1 (suggestions FF0X::FE), from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only.

[The RFC editor is asked to replace MCD1 and MCD2 with the assigned addresses throughout the document.]

9.6. Well-Known URIs

IANA is asked to permanently register the URI suffix "rd" in the "Well-Known URIs" registry. The change controller is the IETF, this document is the reference.

9.7. Service Names and Transport Protocol Port Number Registry

IANA is asked to enter four new items into the Service Names and Transport Protocol Port Number Registry:

- * Service name: "core-rd", Protocol: "udp", Description: "Resource Directory accessed using CoAP"
- * Service name "core-rd-dtls", Protocol: "udp", Description: "Resource Directory accessed using CoAP over DTLS"
- * Service name: "core-rd", Protocol: "tcp", Description: "Resource Directory accessed using CoAP over TCP"
- * Service name "core-rd-tls", Protocol: "tcp", Description: "Resource Directory accessed using CoAP over TLS"

All in common have this document as their reference.

10. Examples

Two examples are presented: a Lighting Installation example in Section 10.1 and a LwM2M example in Section 10.2.

10.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the RD with a CoAP interface to facilitate the installation and start-up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address as described in Appendix A. No conclusions must be drawn on the realization of actual installation or naming procedures, because the example only "emphasizes" some of the issues that may influence the use of the RD and does not pretend to be normative.

10.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one endpoint. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager.

At the moment of installation, the network under installation is not necessarily connected to the DNS infrastructure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and the sensor. The addresses shown in Table 4 below stand in for these in the following examples.

Name	IPv6 address
luminary1	2001:db8:4::1
luminary2	2001:db8:4::2
Presence sensor	2001:db8:4::3
RD	2001:db8:4::ff

Table 4: Addresses used in the examples

In Section 10.1.2 the use of RD during installation is presented.

10.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have `rt=tag:example.com,2020:light`, and the presence sensor has `rt=tag:example.com,2020:p-sensor`. The endpoints have names which are relevant to the light installation manager. In this case `luminary1`, `luminary2`, and the presence sensor are located in room 2-4-015, where `luminary1` is located at the window and `luminary2` and the presence sensor are located at the door. The endpoint names reflect this physical location. The middle, left and right lamps are accessed via path `/light/middle`, `/light/left`, and `/light/right` respectively. The identifiers relevant to the RD are shown in Table 5 below:

Name	endpoint	resource path	resource type
luminary1	lm_R2-4-015_wndw	/light/left	tag:example.com,2020:light
luminary1	lm_R2-4-015_wndw	/light/middle	tag:example.com,2020:light
luminary1	lm_R2-4-015_wndw	/light/right	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/left	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/middle	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/right	tag:example.com,2020:light
Presence sensor	ps_R2-4-015_door	/ps	tag:example.com,2020:p-sensor

Table 5: RD identifiers

It is assumed that the CT has performed RD discovery and has received a response like the one in the Section 4.3 example.

The CT inserts the endpoints of the luminaries and the sensor in the RD using the registration base URI parameter (base) to specify the interface address:

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=lm_R2-4-015_wndw&base=coap://[2001:db8:4::1]&d=R2-4-015
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/4521

Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=lm_R2-4-015_door&base=coap://[2001:db8:4::2]&d=R2-4-015
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/4522

Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=ps_R2-4-015_door&base=coap://[2001:db8:4::3]&d=R2-4-015
Payload:
</ps>;rt="tag:example.com,2020:p-sensor"

Res: 2.01 Created
Location-Path: /rd/4523
```

Figure 23: Example of registrations a CT enters into an RD

The sector name d=R2-4-015 has been added for an efficient lookup because filtering on "ep" name is more awkward. The same sector name is communicated to the two luminaries and the presence sensor by the CT.

The group is specified in the RD. The base parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, the resources supported by all group members are published.

```
Req: POST coap://[2001:db8:4::ff]/rd
?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::1]
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/501
```

Figure 24: Example of a multicast group a CT enters into an RD

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its sector and being configured to join any group containing lights, searches for candidate groups and joins them:

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
?d=R2-4-015&et=core.rd-group&rt=light

Res: 2.05 Content
</rd/501>;ep="grp_R2-4-015";et="core.rd-group";
base="coap://[ff05::1]";rt="core.rd-ep"
```

Figure 25: Example of a lookup exchange to find suitable multicast addresses

From the returned base parameter value, the luminary learns the multicast address of the multicast group.

The presence sensor can learn the presence of groups that support resources with `rt=tag:example.com,2020:light` in its own sector by sending the same request, as used by the luminary. The presence sensor learns the multicast address to use for sending messages to the luminaries.

10.2. OMA Lightweight M2M (LwM2M)

OMA LwM2M is a profile for device services based on CoAP, providing interfaces and operations for device management and device service enablement.

An LwM2M server is an instance of an LwM2M middleware service layer, containing an RD ([LwM2M] page 36f).

That RD only implements the registration interface, and no lookup is implemented. Instead, the LwM2M server provides access to the registered resources, in a similar way to a reverse proxy.

The location of the LwM2M Server and RD URI path is provided by the LwM2M Bootstrap process, so no dynamic discovery of the RD is used. LwM2M Servers and endpoints are not required to implement the /.well-known/core resource.

11. Acknowledgments

Oscar Novo, Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Jim Schaad, Mohit Sethi, Hauke Petersen, Hannes Tschofenig, Sampo Ukkola, Linyi Tian, Jan Newmarch, Matthias Kovatsch, Jaime Jimenez and Ted Lemon have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the RD concepts were originally developed.

12. Changelog

changes from -25 to -26

* Security policies:

- The First-Come-First-Remembered policy is added as an example and a potential default behavior.
- Clarify that the mapping between endpoint names and subject fields is up to a policy that defines reliance on names, and give an example.
- Random EP names: Point that multiple collisions are possible but unlikely.
- Add pointers to policies:
 - o RD replication: Point out that policies may limit that.
 - o Registration: Reword (ep, d) mapping to a previous registration's resource that could have been read as another endpoint taking over an existing registration.
- Clarify that the security policy is a property of the RD the any client may need to verify by checking the RD's authorization.
- Clarify how information from an untrusted RD can be verified

- Remove speculation about how in detail ACE scopes are obtained.
- * Security considerations:
 - Generalize to all current options for security layers usable with CoAP (OSCORE was missing as the text predated RFC8613)
 - Relax the previous SHOULD on secure access to SHOULD where protection is indicated by security policies (bringing the text in line with the -25 changes)
 - Point out that failure to follow the security considerations has implications depending on the protection objective described with the security policies
 - Shorten amplification mitigation
 - Add note about information in Registration Resource path.
 - Acknowledge that most host discovery operations are not secured; mention consequences and mitigation.
- * Abstract, introduction: removed "or disperse networks"
- * RD discovery:
 - Drop the previously stated assumption that RDAO and any DHCP options would only be used together with SLAAC and DHCP for address configuration, respectively.
 - Give concrete guidance for address selection based on RFC6724 when responding to multicasts
 - RDAO:
 - o Clarify that it is an option for RAs and not other ND messages.
 - o Change Lifetime from 16-bit minutes to 32-bit seconds and swap it with Reserved (aligning it with RDNSS which it shares other properties as well).
 - Point out that clients may need to check RD authorization already in last discovery step
- * Registration:

- Wording around "mostly mandatory" has been improved, conflicts clarified and sector default selection adjusted.
- * Simple registration: Rather than coopting POSTs to /.well-known/core, a new resource /.well-known/rd is registered. A historical note in the text documents the change.
- * Examples:
 - Use example URIs rather than unclear reg names (unless it's RFC6690 examples, which were kept for continuity)
 - The LwM2M example was reduced from an outdated explanation of the complete LwM2M model to a summary of how RD is used in there, with a reference to the current specification.
 - Luminary example: Explain example addresses
 - Luminary example: Drop reference to coap-group mechanism that's becoming obsolete, and thus also to RFC7390
 - Multicast addresses in the examples were changed from ff35:30:2001:db8::x to ff35:30:2001:db8:f1::8000:x; the 8000 is to follow RFC 3307, and the f1 is for consistency with all the other example addresses where 2001:db8::/32 is subnetted to 2001:db8:x::/48 by groups of internally consistent examples.
- * Use case text enhancements
 - Home and building automation: Tie in with RD
 - M2M: Move system design paragraph towards the topic of reusability.
- * Various editorial fixes in response to Gen-ART and IESG reviews.
- * Rename 'Full description of the "Endpoint Type" Registration Parameter' section to '... RD Parameter'
- * Error handling: Place a SHOULD around the likely cases, and make the previous "MUST to the best of their capabilities" a "must".
- * impl-info: Add note about the type being WIP
- * Interaction tables: list CTs as possible initiators where applicable

- * Registration update: Relax requirement to not send parameters needlessly
- * Terminology: Clarify that the CTs' installation events can occur multiple times.
- * Promote RFCs 7252, 7230 and 8288 to normative references
- * Moved Christian Amsuess to first author

changes from -24 to -25

- * Large rework of section 7 (Security policies)

Rather than prescribing which data in the RD is authenticated (and how), it now describes what applications built on an RD can choose to authenticate, show possibilities on how to do it and outline what it means for clients.

This addresses Russ' Genart review points on details in the text in a rather broad fashion. That is because the discussion on the topic inside the WG showed that that text on security has been driven more review-by-review than by an architectural plan of the authors and WG.

- * Add concrete suggestions (twice as long as registrant number with retries, or UUIDs without) for random endpoint names
- * Point out that simple registration can have faked origins, RECOMMEND mitigation when applicable and suggest the Echo mechanism to implement it.
- * Reference existing and upcoming specifications for DDOS mitigation in CoAP.
- * Explain the provenance of the example's multicast address.
- * Make "SHOULD" of not manipulating foreign registrations a "should" and explain how it is enforced
- * Clarify application of RFC6570 to search parameters
- * Syntactic fixes in examples
- * IANA:
 - Don't announce expected number of registrations (goes to write-up)

- Include syntax as part of a field's validity in entry requirements
- * Editorial changes
 - Align wording between abstract and introduction
 - Abbreviation normalization: "ER model", "RD"
 - RFC8174 boilerplate update
 - Minor clarity fixes
 - Markup and layouting

changes from -23 to -24

- * Discovery using DNS-SD added again
- * Minimum lifetime (lt) reduced from 60 to 1
- * References added
- * IANA considerations
 - added about .well-known/core resource
 - added DNS-SD service names
 - made RDAO option number a suggestion
 - added "reference" field to endpoint type registry
- * Lookup: mention that anchor is a legitimate lookup attribute
- * Terminology and example fixes
- * Layout fixes, esp. the use of non-ASCII characters in figures

changes from -22 to -23

- * Explain that updates can not remove attributes
- * Typo fixes

changes from -21 to -22

- * Request a dedicated IPv4 address from IANA (rather than sharing with All CoAP nodes)
- * Fix erroneous examples
- * Editorial changes
 - Add figure numbers to examples
 - Update RD parameters table to reflect changes of earlier versions in the text
 - Typos and minor wording

changes from -20 to -21

(Processing comments during WGLC)

- * Defer outdated description of using DNS-SD to find an RD to the defining document
- * Describe operational conditions in automation example
- * Recommend particular discovery mechanisms for some managed network scenarios

changes from -19 to -20

(Processing comments from the WG chair review)

- * Define the permissible characters in endpoint and sector names
- * Express requirements on NAT situations in more abstract terms
- * Shifted heading levels to have the interfaces on the same level
- * Group instructions for error handling into general section
- * Simple Registration: process reflowed into items list
- * Updated introduction to reflect state of CoRE in general, reference RFC7228 (defining "constrained") and use "IoT" term in addition to "M2M"
- * Update acknowledgements
- * Assorted editorial changes

- Unify examples style
- Terminology: RDAO defined and not only expanded
- Add CT to Figure 1
- Consistency in the use of the term "Content Format"

changes from -18 to -19

- * link-local addresses: allow but prescribe split-horizon fashion when used, disallow zone identifiers
- * Remove informative references to documents not mentioned any more

changes from -17 to -18

- * Rather than re-specifying link format (Modernized Link Format), describe a Limited Link Format that's the uncontested subset of Link Format
- * Acknowledging the -17 version as part of the draft
- * Move "Read endpoint links" operation to future specification like PATCH
- * Demote links-json to an informative reference, and removed them from exchange examples
- * Add note on unusability of link-local IP addresses, and describe mitigation.
- * Reshuffling of sections: Move additional operations and endpoint lookup back from appendix, and groups into one
- * Lookup interface tightened to not imply applicability for non link-format lookups (as those can have vastly different views on link cardinality)
- * Simple registration: Change sequence of GET and POST-response, ensuring unsuccessful registrations are reported as such, and suggest how devices that would have required the inverse behavior can still cope with it.
- * Abstract and introduction reworded to avoid the impression that resources are stored in full in the RD

- * Simplify the rules governing when a registration resource can or must be changed.
- * Drop a figure that has become useless due to the changes of and -13 and -17
- * Wording consistency fixes: Use "Registrations" and "target attributes"
- * Fix incorrect use of content negotiation in discovery interface description (Content-Format -> Accept)
- * State that the base attribute value is part of endpoint lookup even when implicit in the registration
- * Update references from RFC5988 to its update RFC8288
- * Remove appendix on protocol-negotiation (which had a note to be removed before publication)

changes from -16 to -17

(Note that -17 is published as a direct follow-up to -16, containing a single change to be discussed at IETF103)

- * Removed groups that are enumerations of registrations and have dedicated mechanism
- * Add groups that are enumerations of shared resources and are a special case of endpoint registrations

changes from -15 to -16

- * Recommend a common set of resources for members of a group
- * Clarified use of multicast group in lighting example
- * Add note on concurrent registrations from one EP being possible but not expected
- * Refresh web examples appendix to reflect current use of Modernized Link Format
- * Add examples of URIs where Modernized Link Format matters
- * Editorial changes

changes from -14 to -15

- * Rewrite of section "Security policies"
- * Clarify that the "base" parameter text applies both to relative references both in anchor and href
- * Renamed "Registree-EP" to Registrant-EP"
- * Talk of "relative references" and "URIs" rather than "relative" and "absolute" URIs. (The concept of "absolute URIs" of [RFC3986] is not needed in RD).
- * Fixed examples
- * Editorial changes

changes from -13 to -14

- * Rename "registration context" to "registration base URI" (and "con" to "base") and "domain" to "sector" (where the abbreviation "d" stays for compatibility reasons)
- * Introduced resource types core.rd-ep and core.rd-gp
- * Registration management moved to appendix A, including endpoint and group lookup
- * Minor editorial changes
 - PATCH/iPATCH is clearly deferred to another document
 - Recommend against query / fragment identifier in con=
 - Interface description lists are described as illustrative
 - Rewording of Simple Registration
- * Simple registration carries no error information and succeeds immediately (previously, sequence was unspecified)
- * Lookup: href are matched against resolved values (previously, this was unspecified)
- * Lookup: lt are not exposed any more
- * con/base: Paths are allowed
- * Registration resource locations can not have query or fragment parts

- * Default life time extended to 25 hours
 - * clarified registration update rules
 - * lt-value semantics for lookup clarified.
 - * added template for simple registration
- changes from -12 to -13
- * Added "all resource directory" nodes MC address
 - * Clarified observation behavior
 - * version identification
 - * example rt= and et= values
 - * domain from figure 2
 - * more explanatory text
 - * endpoints of a groups hosted by different RD
 - * resolve RFC6690-vs-8288 resolution ambiguities:
 - require registered links not to be relative when using anchor
 - return absolute URIs in resource lookup
- changes from -11 to -12
- * added Content Model section, including ER diagram
 - * removed domain lookup interface; domains are now plain attributes of groups and endpoints
 - * updated chapter "Finding a Resource Directory"; now distinguishes configuration-provided, network-provided and heuristic sources
 - * improved text on: atomicity, idempotency, lookup with multiple parameters, endpoint removal, simple registration
 - * updated LWM2M description
 - * clarified where relative references are resolved, and how context and anchor interact

- * new appendix on the interaction with RFCs 6690, 5988 and 3986
- * lookup interface: group and endpoint lookup return group and registration resources as link targets
- * lookup interface: search parameters work the same across all entities
- * removed all methods that modify links in an existing registration (POST with payload, PATCH and iPATCH)
- * removed plurality definition (was only needed for link modification)
- * enhanced IANA registry text
- * state that lookup resources can be observable
- * More examples and improved text

changes from -09 to -10

- * removed "ins" and "exp" link-format extensions.
- * removed all text concerning DNS-SD.
- * removed inconsistency in RDAO text.
- * suggestions taken over from various sources
- * replaced "Function Set" with "REST API", "base URI", "base path"
- * moved simple registration to registration section

changes from -08 to -09

- * clarified the "example use" of the base RD resource values /rd, /rd-lookup, and /rd-group.
- * changed "ins" ABNF notation.
- * various editorial improvements, including in examples
- * clarifications for RDAO

changes from -07 to -08

- * removed link target value returned from domain and group lookup types
- * Maximum length of domain parameter 63 bytes for consistency with group
- * removed option for simple POST of link data, don't require a .well-known/core resource to accept POST data and handle it in a special way; we already have /rd for that
- * add IPv6 ND Option for discovery of an RD
- * clarify group configuration section 6.1 that endpoints must be registered before including them in a group
- * removed all superfluous client-server diagrams
- * simplified lighting example
- * introduced Commissioning Tool
- * RD-Look-up text is extended.

changes from -06 to -07

- * added text in the discovery section to allow content format hints to be exposed in the discovery link attributes
- * editorial updates to section 9
- * update author information
- * minor text corrections

Changes from -05 to -06

- * added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

- * added Update Endpoint Links using PATCH
- * http access made explicit in interface specification
- * Added http examples

Changes from -03 to -04:

- * Added http response codes
- * Clarified endpoint name usage
- * Add application/link-format+cbor content-format

Changes from -02 to -03:

- * Added an example for lighting and DNS integration
- * Added an example for RD use in OMA LWM2M
- * Added Read Links operation for link inspection by endpoints
- * Expanded DNS-SD section
- * Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

- * Added a catalogue use case.
- * Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.
- * Additional examples section added for more complex use cases.
- * New DNS-SD mapping section.
- * Added text on endpoint identification and authentication.
- * Error code 4.04 added to Registration Update and Delete requests.
- * Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

- * Removed the ETag validation feature.
- * Place holder for the DNS-SD mapping section.
- * Explicitly disabled GET or POST on returned Location.
- * New registry for RD parameters.
- * Added support for the JSON Link Format.

- * Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

- * Updated the version and date.

Changes from -04 to -05:

- * Restricted Update to parameter updates.
- * Added pagination support for the Lookup interface.
- * Minor editing, bug fixes and reference updates.
- * Added group support.
- * Changed rt to et for the registration and update interface.

Changes from -03 to -04:

- * Added the ins= parameter back for the DNS-SD mapping.
- * Integrated the Simple Directory Discovery from Carsten.
- * Editorial improvements.
- * Fixed the use of ETags.
- * Fixed tickets 383 and 372

Changes from -02 to -03:

- * Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- * Updated REST interface descriptions to use RFC6570 URI Template format.
- * Introduced an improved RD Lookup design as its own function set.
- * Improved the security considerations section.
- * Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:

- * Added a terminology section.

- * Changed the inclusion of an ETag in registration or update to a MAY.
- * Added the concept of an RD Domain and a registration parameter for it.
- * Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- * Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

13. References

13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

13.2. Informative References

- [ER] Chen, P., "The entity-relationship model--toward a unified view of data", DOI 10.1145/320434.320440, ACM Transactions on Database Systems Vol. 1, pp. 9-36, March 1976, <<https://doi.org/10.1145/320434.320440>>.
- [I-D.bormann-t2trg-rel-impl]
Bormann, C., "impl-info: A link relation type for disclosing implementation information", Work in Progress, Internet-Draft, draft-bormann-t2trg-rel-impl-02, 27 September 2020, <<http://www.ietf.org/internet-drafts/draft-bormann-t2trg-rel-impl-02.txt>>.
- [I-D.hartke-t2trg-coral]
Hartke, K., "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-hartke-t2trg-coral-09, 8 July 2019, <<http://www.ietf.org/internet-drafts/draft-hartke-t2trg-coral-09.txt>>.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-35, 24 June 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-ace-oauth-authz-35.txt>>.

- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", Work in Progress, Internet-Draft, draft-ietf-core-echo-request-tag-10, 13 July 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-core-echo-request-tag-10.txt>>.
- [I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", Work in Progress, Internet-Draft, draft-ietf-core-links-json-10, 26 February 2018, <<http://www.ietf.org/internet-drafts/draft-ietf-core-links-json-10.txt>>.
- [I-D.ietf-core-rd-dns-sd]
Stok, P., Koster, M., and C. Amsuess, "CoRE Resource Directory: DNS-SD mapping", Work in Progress, Internet-Draft, draft-ietf-core-rd-dns-sd-05, 7 July 2019, <<http://www.ietf.org/internet-drafts/draft-ietf-core-rd-dns-sd-05.txt>>.
- [I-D.silverajan-core-coap-protocol-negotiation]
Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", Work in Progress, Internet-Draft, draft-silverajan-core-coap-protocol-negotiation-09, 2 July 2018, <<http://www.ietf.org/internet-drafts/draft-silverajan-core-coap-protocol-negotiation-09.txt>>.
- [LwM2M] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification: Transport Bindings (Candidate Version 1.1)", 12 June 2018, <https://openmobilealliance.org/RELEASE/LightweightM2M/V1_1-20180612-C/OMA-TS-LightweightM2M_Transport-V1_1-20180612-C.pdf>.
- [RFC3306] Haberman, B. and D. Thaler, "Unicast-Prefix-based IPv6 Multicast Addresses", RFC 3306, DOI 10.17487/RFC3306, August 2002, <<https://www.rfc-editor.org/info/rfc3306>>.
- [RFC3849] Huston, G., Lord, A., and P. Smith, "IPv6 Address Prefix Reserved for Documentation", RFC 3849, DOI 10.17487/RFC3849, July 2004, <<https://www.rfc-editor.org/info/rfc3849>>.

- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<https://www.rfc-editor.org/info/rfc4122>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC5771] Cotton, M., Vegoda, L., and D. Meyer, "IANA Guidelines for IPv4 Multicast Address Assignments", BCP 51, RFC 5771, DOI 10.17487/RFC5771, March 2010, <<https://www.rfc-editor.org/info/rfc5771>>.
- [RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <<https://www.rfc-editor.org/info/rfc6724>>.
- [RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <<https://www.rfc-editor.org/info/rfc6775>>.
- [RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <<https://www.rfc-editor.org/info/rfc6874>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 8106, DOI 10.17487/RFC8106, March 2017, <<https://www.rfc-editor.org/info/rfc8106>>.

- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <<https://www.rfc-editor.org/info/rfc8141>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

Appendix A. Groups Registration and Lookup

The RD-Groups usage pattern allows announcing application groups inside an RD.

Groups are represented by endpoint registrations. Their base address is a multicast address, and they SHOULD be entered with the endpoint type "core.rd-group". The endpoint name can also be referred to as a group name in this context.

The registration is inserted into the RD by a Commissioning Tool, which might also be known as a group manager here. It performs third party registration and registration updates.

The links it registers SHOULD be available on all members that join the group. Depending on the application, members that lack some resource MAY be permissible if requests to them fail gracefully.

The following example shows a CT registering a group with the name "lights" which provides two resources. The directory resource path /rd is an example RD location discovered in a request similar to Figure 5. The group address in the example is constructed from [RFC3849]'s reserved 2001:db8:: prefix as a unicast-prefix based site-local address (see [RFC3306]).

```
Req: POST coap://rd.example.com/rd?ep=lights&et=core.rd-group
      &base=coap://[ff35:30:2001:db8:f1::8000:1]
Content-Format: 40
Payload:
</light>;rt="tag:example.com,2020:light";
      if="tag:example.net,2020:actuator",
</color-temperature>;if="tag:example.net,2020:parameter";u="K"

Res: 2.01 Created
Location-Path: /rd/12
```

Figure 26: Example registration of a group

In this example, the group manager can easily permit devices that have no writable color-temperature to join, as they would still respond to brightness changing commands. Had the group instead contained a single resource that sets brightness and color temperature atomically, endpoints would need to support both properties.

The resources of a group can be looked up like any other resource, and the group registrations (along with any additional registration parameters) can be looked up using the endpoint lookup interface.

The following example shows a client performing an endpoint lookup for all groups.

```
Req: GET /rd-lookup/ep?et=core.rd-group

Res: 2.05 Content
Payload:
</rd/501>;ep="grp_R2-4-015";et="core.rd-group";
      base="coap://[ff05::1]",
</rd/12>;ep=lights&et=core.rd-group;
      base="coap://[ff35:30:2001:f1:db8::8000:1]";rt="core.rd-ep"
```

Figure 27: Example lookup of groups

The following example shows a client performing a lookup of all resources of all endpoints (groups) with et=core.rd-group.

```

Req: GET /rd-lookup/res?et=core.rd-group

<coap://[ff35:30:2001:db8:f1::8000:1]/light>;
  rt="tag:example.com,2020:light";
  if="tag:example.net,2020:actuator";
  anchor="coap://[ff35:30:2001:db8:f1::8000:1]",
<coap://[ff35:30:2001:db8:f1::8000:1]/color-temperature>;
  if="tag:example.net,2020:parameter";u="K";
  anchor="coap://[ff35:30:2001:db8:f1::8000:1]"

```

Figure 28: Example lookup of resources inside groups

Appendix B. Web links and the Resource Directory

Understanding the semantics of a link-format document and its URI references is a journey through different documents ([RFC3986] defining URIs, [RFC6690] defining link-format documents based on [RFC8288] which defines Link header fields, and [RFC7252] providing the transport). This appendix summarizes the mechanisms and semantics at play from an entry in `"/.well-known/core"` to a resource lookup.

This text is primarily aimed at people entering the field of Constrained Restful Environments from applications that previously did not use web mechanisms.

The explanation of the steps makes some shortcuts in the more confusing details of [RFC6690], which are justified as all examples being in Limited Link Format.

B.1. A simple example

Let's start this example with a very simple host, `"2001:db8:f0::1"`. A client that follows classical CoAP Discovery ([RFC7252] Section 7), sends the following multicast request to learn about neighbours supporting resources with resource-type `"temperature"`.

The client sends a link-local multicast:

```

GET coap://[ff02::fd]:5683/.well-known/core?rt=temperature

RES 2.05 Content
</temp>;rt=temperature;ct=0

```

Figure 29: Example of direct resource discovery

where the response is sent by the server, `"[2001:db8:f0::1]:5683"`.

While the client - on the practical or implementation side - can just go ahead and create a new request to "[2001:db8:f0::1]:5683" with Uri-Path: "temp", the full resolution steps for insertion into and retrieval from the RD without any shortcuts are:

B.1.1. Resolving the URIs

The client parses the single returned record. The link's target (sometimes called "href") is `"/temp"`, which is a relative URI that needs resolving. The base URI `<coap://[ff02::fd]:5683/.well-known/core>` is used to resolve the reference `/temp` against.

The Base URI of the requested resource can be composed from the options of the CoAP GET request by following the steps of [RFC7252] section 6.5 (with an addition at the end of 8.2) into `"coap://[2001:db8:f0::1]/.well-known/core"`.

Because `"/temp"` starts with a single slash, the record's target is resolved by replacing the path `"/.well-known/core"` from the Base URI (section 5.2 [RFC3986]) with the relative target URI `"/temp"` into `"coap://[2001:db8:f0::1]/temp"`.

B.1.2. Interpreting attributes and relations

Some more information but the record's target can be obtained from the payload: the resource type of the target is "temperature", and its content format is text/plain (ct=0).

A relation in a web link is a three-part statement that specifies a named relation between the so-called "context resource" and the target resource, like `"_This page_ has _its table of contents_ at _/toc.html_"`. In link format documents, there is an implicit "host relation" specified with default parameter: `rel="hosts"`.

In our example, the context resource of the link is the URI specified in the GET request `"coap://[2001:db8:f0::1]/.well-known/core"`. 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.'`

B.2. A slightly more complex example

Omitting the `"rt=temperature"` filter, the discovery query would have given some more records in the payload:

```
GET coap://[ff02::fd]:5683/.well-known/core

RES 2.05 Content
</temp>;rt=temperature;ct=0,
</light>;rt=light-lux;ct=0,
</t>;anchor="/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;anchor="/temp";
  rel="describedby"
```

Figure 30: Extended example of direct resource discovery

Parsing the third record, the client encounters the "anchor" parameter. It is a URI relative to the Base URI of the request and is thus resolved to "coap://[2001:db8:f0::1]/sensors/temp". That is the context resource of the link, so the "rel" statement is not about the target and the Base URI any more, but about the target and the resolved URI. Thus, the third record could be read as "coap://[2001:db8:f0::1]/sensors/temp" has an alternate representation at "coap://[2001:db8:f0::1]/t".

Following the same resolution steps, the fourth record can be read as "coap://[2001:db8:f0::1]/sensors/temp" is described by "http://www.example.com/sensors/t123".

B.3. Enter the Resource Directory

The RD tries to carry the semantics obtainable by classical CoAP discovery over to the resource lookup interface as faithfully as possible.

For the following queries, we will assume that the simple host has used Simple Registration to register at the RD that was announced to it, sending this request from its UDP port "[2001:db8:f0::1]:6553":

```
POST coap://[2001:db8:f01::ff]/.well-known/rd?ep=simple-host1
```

Figure 31: Example request starting a simple registration

The RD would have accepted the registration, and queried the simple host's "/.well-known/core" by itself. As a result, the host is registered as an endpoint in the RD with the name "simple-host1". The registration is active for 90000 seconds, and the endpoint registration Base URI is "coap://[2001:db8:f0::1]" following the resolution steps described in Appendix B.1.1. It should be remarked that the Base URI constructed that way always yields a URI of the form: scheme://authority without path suffix.

If the client now queries the RD as it would previously have issued a multicast request, it would go through the RD discovery steps by fetching "coap://[2001:db8:f0::ff]/.well-known/core?rt=core.rd-lookup-res", obtain "coap://[2001:db8:f0::ff]/rd-lookup/res" as the resource lookup endpoint, and 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]"
```

Figure 32: Example payload of a response to a resource lookup

This is not literally the same response that it would have received from a multicast request, but it contains the equivalent statement:

```
'"coap://[2001:db8:f0::1]" is hosting the resource
"coap://[2001:db8:f0::1]/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 does not matter in this application; if it did, the endpoint would have been more explicit. Actually, /.well-known/core does NOT host the resource but stores a URI reference to the resource.)

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"
```

Figure 33: Extended example payload of a response to a resource lookup

All the target and anchor references are already in absolute form there, which don't need to be resolved any further.

Had the simple host done an equivalent full registration with a base= parameter (e.g. "?ep=simple-host1&base=coap+tcp://simple-host1.example.com"), that context would have been used to resolve the relative anchor values instead, giving

```
<coap+tcp://simple-host1.example.com/temp>;rt=temperature;ct=0;  
  anchor="coap+tcp://simple-host1.example.com"
```

Figure 34: Example payload of a response to a resource lookup with a dedicated base URI

and analogous records.

B.4. A note on differences between link-format and Link header fields

While link-format and Link header fields look very similar and are based on the same model of typed links, there are some differences between [RFC6690] and [RFC8288], which are dealt with differently:

- * "Resolving the target against the anchor": [RFC6690] Section 2.1 states that the anchor of a link is used as the Base URI against which the term inside the angle brackets (the target) is resolved, falling back to the resource's URI with paths stripped off (its "Origin"). In contrast to that, [RFC8288] Section B.2 describes that the anchor is immaterial to the resolution of the target reference.

RFC6690, in the same section, also states that absent anchors set the context of the link to the target's URI with its path stripped off, while according to [RFC8288] Section 3.2, the context is the resource's base URI.

The rules introduced in Appendix C ensure that an RD does not need to deal with those differences when processing input data. Lookup results are required to be absolute references for the same reason.

- * There is no percent encoding in link-format documents.

A link-format document is a UTF-8 encoded string of Unicode characters and does not have percent encoding, while Link header fields are practically ASCII strings that use percent encoding for non-ASCII characters, stating the encoding explicitly when required.

For example, while a Link header field in a page about a Swedish city might read

Link: </temperature/Malm%C3%B6>;rel="live-environment-data"

a link-format document from the same source might describe the link as

</temperature/Malmö>;rel="live-environment-data"

Parsers and producers of link-format and header fields need to be aware of this difference.

Appendix C. Limited Link Format

The CoRE Link Format as described in [RFC6690] has been interpreted differently by implementers, and a strict implementation rules out some use cases of an RD (e.g. base values with path components).

This appendix describes a subset of link format documents called Limited Link Format. The rules herein are not very limiting in practice - all examples in RFC6690, and all deployments the authors are aware of already stick to them - but ease the implementation of RD servers.

It is applicable to representations in the application/link-format media type, and any other media types that inherit [RFC6690] Section 2.1.

A link format representation is in Limited Link format if, for each link in it, the following applies:

- * All URI references either follow the URI or the path-absolute ABNF rule of RFC3986 (i.e. target and anchor each either start with a scheme or with a single slash),
- * if the anchor reference starts with a scheme, the target reference starts with a scheme as well (i.e. relative references in target cannot be used when the anchor is a full URI), and
- * the application does not care whether links without an explicitly given anchor have the origin's "/" or "/.well-known/core" resource as their link context.

Authors' Addresses

Christian Amsüss (editor)
Hollandstr. 12/4
1020
Austria

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

Zach Shelby
ARM
150 Rose Orchard
San Jose, 95134
United States of America

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

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View, 94043
United States of America

Phone: +1-707-502-5136
Email: Michael.Koster@smarththings.com

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

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

Peter van der Stok
consultant

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

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

C. Amsüss, Ed.
Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
7 March 2021

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

Abstract

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

Note to Readers

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

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

Status of This Memo

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 8 September 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
2. Terminology	4
3. Architecture and Use Cases	6
3.1. Principles	6
3.2. Architecture	7
3.3. RD Content Model	8
3.4. Link-local addresses and zone identifiers	12
3.5. Use Case: Cellular M2M	12
3.6. Use Case: Home and Building Automation	13
3.7. Use Case: Link Catalogues	14
4. RD discovery and other interface-independent components	14
4.1. Finding a Resource Directory	15
4.1.1. Resource Directory Address Option (RDAO)	17
4.1.2. Using DNS-SD to discover a Resource Directory	19
4.2. Payload Content Formats	19
4.3. URI Discovery	19
5. Registration	22
5.1. Simple Registration	27
5.2. Third-party registration	29
5.3. Operations on the Registration Resource	30

5.3.1.	Registration Update	30
5.3.2.	Registration Removal	34
5.3.3.	Further operations	34
5.3.4.	Request freshness	35
6.	RD Lookup	37
6.1.	Resource lookup	37
6.2.	Lookup filtering	38
6.3.	Resource lookup examples	40
6.4.	Endpoint lookup	42
7.	Security policies	43
7.1.	Endpoint name	44
7.1.1.	Random endpoint names	44
7.2.	Entered resources	44
7.3.	Link confidentiality	45
7.4.	Segmentation	46
7.5.	First-Come-First-Remembered: A default policy	46
8.	Security Considerations	48
8.1.	Discovery	48
8.2.	Endpoint Identification and Authentication	48
8.3.	Access Control	49
8.4.	Denial of Service Attacks	49
8.5.	Skipping freshness checks	50
9.	IANA Considerations	50
9.1.	Resource Types	50
9.2.	IPv6 ND Resource Directory Address Option	51
9.3.	RD Parameter Registry	51
9.3.1.	Full description of the "Endpoint Type" RD Parameter	54
9.4.	"Endpoint Type" (et=) RD Parameter values	54
9.5.	Multicast Address Registration	55
9.6.	Well-Known URIs	55
9.7.	Service Names and Transport Protocol Port Number Registry	55
10.	Examples	56
10.1.	Lighting Installation	56
10.1.1.	Installation Characteristics	56
10.1.2.	RD entries	57
10.2.	OMA Lightweight M2M (LwM2M)	60
11.	Acknowledgments	61
12.	Changelog	61
13.	References	76
13.1.	Normative References	76
13.2.	Informative References	77
Appendix A.	Groups Registration and Lookup	80
Appendix B.	Web links and the Resource Directory	82
B.1.	A simple example	82
B.1.1.	Resolving the URIs	82
B.1.2.	Interpreting attributes and relations	83

B.2. A slightly more complex example	83
B.3. Enter the Resource Directory	84
B.4. A note on differences between link-format and Link header fields	86
Appendix C. Limited Link Format	86
Authors' Addresses	87

1. Introduction

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

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

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

2. Terminology

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

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC3986], [RFC8288] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

resolve against

The expression "a URI-reference is resolved against a base URI" is used to describe the process of [RFC3986] Section 5.2.

Noteworthy corner cases are that if the URI-reference is a (full) URI and resolved against any base URI, that gives the original full URI, and that resolving an empty URI reference gives the base URI without any fragment identifier.

Resource Directory (RD)

A web entity that stores information about web resources and implements the REST interfaces defined in this specification for discovery, for the creation, maintenance and removal of registrations, and for lookup of the registered resources.

Sector

In the context of an RD, a sector is a logical grouping of endpoints.

The abbreviation "d=" is used for the sector in query parameters for compatibility with deployed implementations.

Endpoint

Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the RD. An endpoint is identified by its endpoint name, which is included during registration, and has a unique name within the associated sector of the registration.

Registration Base URI

The Base URI of a Registration is a URI that typically gives scheme and authority information about an Endpoint. The Registration Base URI is provided at registration time, and is used by the RD to resolve relative references of the registration into URIs.

Target

The target of a link is the destination address (URI) of the link. It is sometimes identified with "href=", or displayed as "<target>". Relative targets need resolving with respect to the Base URI (section 5.2 of [RFC3986]).

This use of the term Target is consistent with [RFC8288]'s use of the term.

Context

The context of a link is the source address (URI) of the link, and describes which resource is linked to the target. A link's context is made explicit in serialized links as the "anchor=" attribute.

This use of the term Context is consistent with [RFC8288]'s use of the term.

Directory Resource

A resource in the RD containing registration resources.

Registration Resource

A resource in the RD that contains information about an Endpoint and its links.

Commissioning Tool

Commissioning Tool (CT) is a device that assists during installation events by assigning values to parameters, naming endpoints and groups, or adapting the installation to the needs of the applications.

Registrant-ep

Registrant-ep is the endpoint that is registered into the RD. The registrant-ep can register itself, or a CT registers the registrant-ep.

RDAO

Resource Directory Address Option. A new IPv6 Neighbor Discovery option defined for announcing an RD's address.

3. Architecture and Use Cases

3.1. Principles

The RD is primarily a tool to make discovery operations more efficient than querying /.well-known/core on all connected devices, or across boundaries that would limit those operations.

It provides information about resources hosted by other devices that could otherwise only be obtained by directly querying the /.well-known/core resource on these other devices, either by a unicast request or a multicast request.

Information SHOULD only be stored in the RD if it can be obtained by querying the described device's /.well-known/core resource directly.

Data in the RD can only be provided by the device which hosts those data or a dedicated Commissioning Tool (CT). These CTs act on behalf of endpoints too constrained, or generally unable, to present that information themselves. No other client can modify data in the RD. Changes to the information in the RD do not propagate automatically back to the web servers from where the information originated.

3.2. Architecture

The RD architecture is illustrated in Figure 1. An RD is used as a repository of registrations describing resources hosted on other web servers, also called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port. A physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain RD registrations, and for endpoints to lookup resources from the RD. An RD can be logically segmented by the use of Sectors.

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Registrations in the RD are soft state and need to be periodically refreshed.

An endpoint uses specific interfaces to register, update and remove a registration. It is also possible for an RD to fetch Web Links from endpoints and add their contents to its registrations.

At the first registration of an endpoint, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of registrations.

A lookup interface for discovering any of the Web Links stored in the RD is provided using the CoRE Link Format.

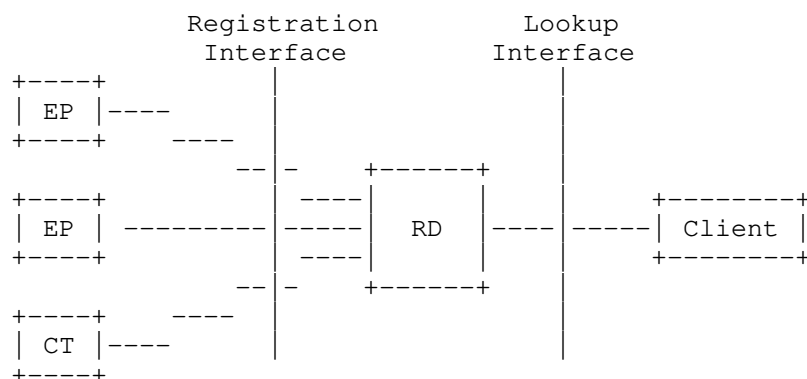


Figure 1: The RD architecture.

A Registrant-EP MAY keep concurrent registrations to more than one RD at the same time if explicitly configured to do so, but that is not expected to be supported by typical EP implementations. Any such registrations are independent of each other. The usual expectation when multiple discovery mechanisms or addresses are configured is that they constitute a fall-back path for a single registration.

3.3. RD Content Model

The Entity-Relationship (ER) models shown in Figure 2 and Figure 3 model the contents of `/.well-known/core` and the RD respectively, with entity-relationship diagrams [ER]. Entities (rectangles) are used for concepts that exist independently. Attributes (ovals) are used for concepts that exist only in connection with a related entity. Relations (diamonds) give a semantic meaning to the relation between entities. Numbers specify the cardinality of the relations.

Some of the attribute values are URIs. Those values are always full URIs and never relative references in the information model. They can, however, be expressed as relative references in serializations, and often are.

These models provide an abstract view of the information expressed in link-format documents and an RD. They cover the concepts, but not necessarily all details of an RD's operation; they are meant to give an overview, and not be a template for implementations.

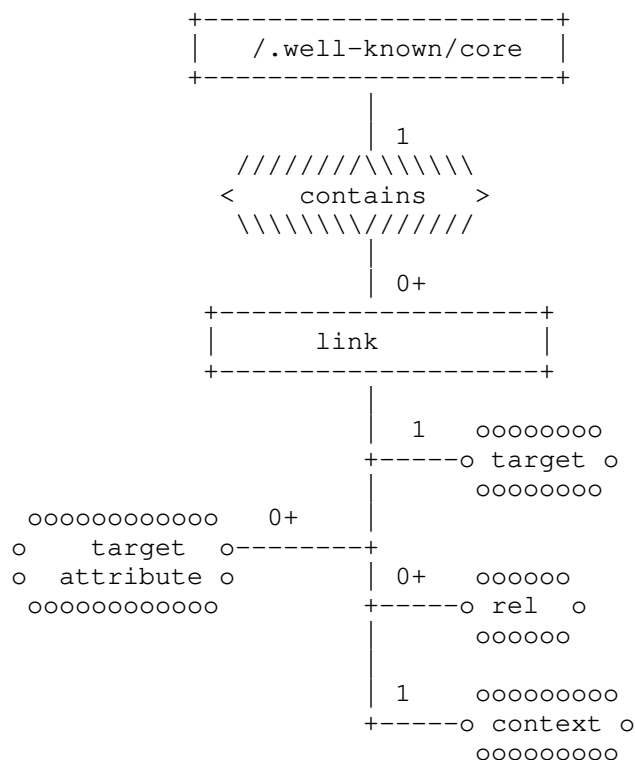


Figure 2: ER Model of the content of `/.well-known/core`

The model shown in Figure 2 models the contents of `/.well-known/core` which contains:

- * a set of links belonging to the hosting web server

The web server is free to choose links it deems appropriate to be exposed in its `"/.well-known/core"`. Typically, the links describe resources that are served by the host, but the set can also contain links to resources on other servers (see examples in [RFC6690] page 14). The set does not necessarily contain links to all resources served by the host.

A link has the following attributes (see [RFC8288]):

- * Zero or more link relations: They describe relations between the link context and the link target.

In link-format serialization, they are expressed as space-separated values in the "rel" attribute, and default to "hosts".

- * A link context URI: It defines the source of the relation, e.g. `_who_ "hosts" something`.

In link-format serialization, it is expressed in the "anchor" attribute and defaults to the Origin of the target (practically: the target with its path and later components removed)

- * A link target URI: It defines the destination of the relation (e.g. `_what_ is hosted`), and is the topic of all target attributes.

In link-format serialization, it is expressed between angular brackets, and sometimes called the "href".

- * Other target attributes (e.g. resource type (rt), interface (if), or content format (ct)). These provide additional information about the target URI.

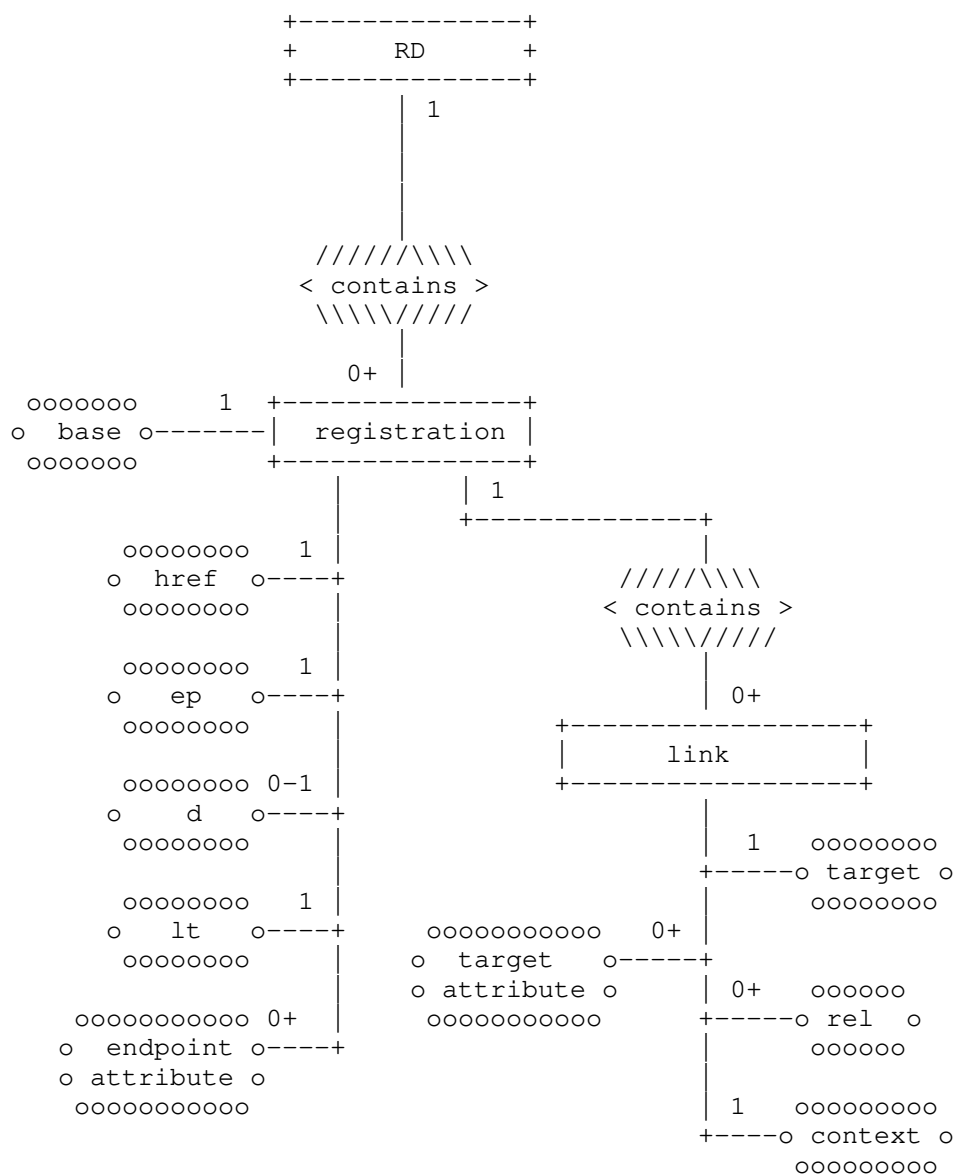


Figure 3: ER Model of the content of the RD

The model shown in Figure 3 models the contents of the RD which contains in addition to /.well-known/core:

* 0 to n Registrations of endpoints,

A registration is associated with one endpoint. A registration defines a set of links as defined for `/.well-known/core`. A Registration has six types of attributes:

- * an endpoint name ("ep", a Unicode string) unique within a sector
- * a Registration Base URI ("base", a URI typically describing the `scheme://authority` part)
- * a lifetime ("lt"),
- * a registration resource location inside the RD ("href"),
- * optionally a sector ("d", a Unicode string)
- * optional additional endpoint attributes (from Section 9.3)

The cardinality of "base" is currently 1; future documents are invited to extend the RD specification to support multiple values (e.g. `[I-D.silverajan-core-coap-protocol-negotiation]`). Its value is used as a Base URI when resolving URIs in the links contained in the endpoint.

Links are modelled as they are in Figure 2.

3.4. Link-local addresses and zone identifiers

Registration Base URIs can contain link-local IP addresses. To be usable across hosts, those cannot be serialized to contain zone identifiers (see [RFC6874] Section 1).

Link-local addresses can only be used on a single link (therefore RD servers cannot announce them when queried on a different link), and lookup clients using them need to keep track of which interface they got them from.

Therefore, it is advisable in many scenarios to use addresses with larger scope if available.

3.5. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded wireless interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. The ambition in such systems is to build them from reusable components. These speed up

development and deployment, and enable shared use of machines across different applications. One crucial component of such systems is the discovery of resources (and thus the endpoints they are hosted on) capable of providing required information at a given time or acting on instructions from the end users.

Imagine a scenario where endpoints installed on vehicles enable tracking of the position of these vehicles for fleet management purposes and allow monitoring of environment parameters. During the boot-up process endpoints register with an RD, which is hosted by the mobile operator or somewhere in the cloud. Periodically, these endpoints update their registration and may modify resources they offer.

When endpoints are not always connected, for example because they enter a sleep mode, a remote server is usually used to provide proxy access to the endpoints. Mobile apps or web applications for environment monitoring contact the RD, look up the endpoints capable of providing information about the environment using an appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server), and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look up for EPs deployed on the vehicles the application is responsible for.

3.6. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of IoT web services. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices. Both can use the common RD infrastructure to establish device interactions efficiently, but can pick security policies suitable for their needs.

Two phases can be discerned for a network servicing the system: (1) installation and (2) operation. During the operational phase, the network is connected to the Internet with a Border Router (e.g. a 6LoWPAN Border Router (6LBR), see [RFC6775]) and the nodes connected to the network can use the Internet services that are provided by the Internet Provider or the network administrator. During the installation phase, the network is completely stand-alone, no Border Router is connected, and the network only supports the IP communication between the connected nodes. The installation phase is

usually followed by the operational phase. As an RD's operations work without hard dependencies on names or addresses, it can be used for discovery across both phases.

3.7. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge beforehand of who is going to consume the data. An RD can be used to hold links about resources and services hosted anywhere to make them discoverable by a general class of applications.

For example, environmental and weather sensors that generate data for public consumption may provide data to an intermediary server, or broker. Sensor data are published to the intermediary upon changes or at regular intervals. Descriptions of the sensors that resolve to links to sensor data may be published to an RD. Applications wishing to consume the data can use RD Lookup to discover and resolve links to the desired resources and endpoints. The RD service need not be coupled with the data intermediary service. Mapping of RDs to data intermediaries may be many-to-many.

Metadata in web link formats like [RFC6690] which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links, need to be supported by RDs. External catalogues that are represented in other formats may be converted to common web linking formats for storage and access by RDs. Since it is common practice for these to be encoded in URNs [RFC8141], simple and lossless structural transforms should generally be sufficient to store external metadata in RDs.

The additional features of an RD allow sectors to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Application groups with multicast addresses may be defined to support efficient data transport.

4. RD discovery and other interface-independent components

This and the following sections define the required set of REST interfaces between an RD, endpoints and lookup clients. Although the examples throughout these sections assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. The multicast discovery and simple registration operations are exceptions to that, as they rely on mechanisms unavailable in HTTP. In all definitions in these sections, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces.

All operations on the contents of the RD MUST be atomic and idempotent.

For several operations, interface templates are given in list form; those describe the operation participants, request codes, URIs, content formats and outcomes. Sections of those templates contain normative content about Interaction, Method, URI Template and URI Template Variables as well as the details of the Success condition. The additional sections on options like Content-Format and on Failure codes give typical cases that an implementation of the RD should deal with. Those serve to illustrate the typical responses to readers who are not yet familiar with all the details of CoAP based interfaces; they do not limit what a server may respond under atypical circumstances.

REST clients (registrant-EPs and CTs during registration and maintenance, lookup clients, RD servers during simple registrations) must be prepared to receive any unsuccessful code and act upon it according to its definition, options and/or payload to the best of their capabilities, falling back to failing the operation if recovery is not possible. In particular, they SHOULD retry the request upon 5.03 (Service Unavailable; 503 in HTTP) according to the Max-Age (Retry-After in HTTP) option, and SHOULD fall back to link-format when receiving 4.15 (Unsupported Content-Format; 415 in HTTP).

An RD MAY make the information submitted to it available to further directories (subject to security policies on link confidentiality), if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

4.1. Finding a Resource Directory

A (re-)starting device may want to find one or more RDs before it can discover their URIs. Dependent on the operational conditions, one or more of the techniques below apply.

The device may be pre-configured to exercise specific mechanisms for finding the RD:

1. It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close; each target network environment in which some of these preconfigured nodes are to be brought up is then configured with a route for this anycast address that leads to an appropriate RD. (Instead of using an anycast address, a multicast address can also be preconfigured. The RD servers then need to configure one of their interfaces with this multicast address.)
2. It may be configured with a DNS name for the RD and use DNS to return the IP address of the RD; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.
3. It may be configured to use a service discovery mechanism such as DNS-SD, as outlined in Section 4.1.2.

For cases where the device is not specifically configured with a way to find an RD, the network may want to provide a suitable default.

1. The IPv6 Neighbor Discovery option RDAO Section 4.1.1 can do that.
2. When DHCP is in use, this could be provided via a DHCP option (no such option is defined at the time of writing).

Finally, if neither the device nor the network offers any specific configuration, the device may want to employ heuristics to find a suitable RD.

The present specification does not fully define these heuristics, but suggests a number of candidates:

1. In a 6LoWPAN, just assume the Border Router (6LBR) can act as an RD (using the ABRO option to find that [RFC6775]). Confirmation can be obtained by sending a unicast to "coap://[6LBR]/.well-known/core?rt=core.rd*".
2. In a network that supports multicast well, discovering the RD using a multicast query for /.well-known/core as specified in CoRE Link Format [RFC6690]: Sending a Multicast GET to "coap://[MCD1]/.well-known/core?rt=core.rd*". RDs within the multicast scope will answer the query.

When answering a multicast request directed at a link-local group, the RD may want to respond from a routable address; this makes it easier for registrants to use one of their own routable addresses for

registration. When [RFC6724] is used for source address selection, this can be achieved by applying the changes of its Section 10.4, picking public addresses in its Section 5 Rule 7, and superseding rule 8 with preferring the source address's precedence.

As some of the RD addresses obtained by the methods listed here are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

The following RD discovery mechanisms are recommended:

- * In managed networks with border routers that need stand-alone operation, the RDAO option is recommended (e.g. operational phase described in Section 3.6).
- * In managed networks without border router (no Internet services available), the use of a preconfigured anycast address is recommended (e.g. installation phase described in Section 3.6).
- * In networks managed using DNS-SD, the use of DNS-SD for discovery as described in Section 4.1.2 is recommended.

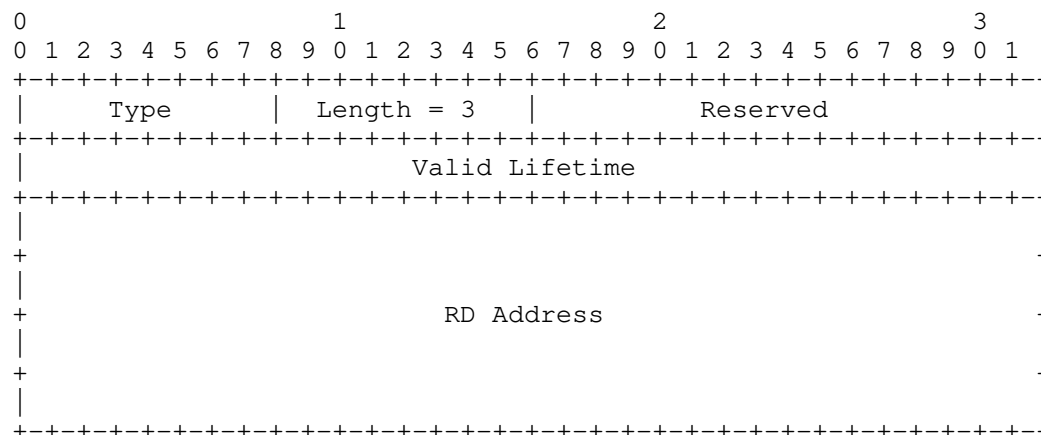
The use of multicast discovery in mesh networks is NOT RECOMMENDED.

4.1.1. Resource Directory Address Option (RDAO)

The Resource Directory Address Option (RDAO) carries information about the address of the RD in RAs (Router Advertisements) of IPv6 Neighbor Discovery (ND), similar to how RDNSS options [RFC8106] are sent. This information is needed when endpoints cannot discover the RD with a link-local or realm-local scope multicast address, for instance because the endpoint and the RD are separated by a Border Router (6LBR). In many circumstances the availability of DHCP cannot be guaranteed either during commissioning of the network. The presence and the use of the RD is essential during commissioning.

It is possible to send multiple RDAO options in one message, indicating as many RD addresses.

The RDAO format is:



Fields:

Type: TBD38

Length: 8-bit unsigned integer. The length of the option in units of 8 bytes. Always 3.

```
Reserved:      This field is unused.  It MUST be
                initialized to zero by the sender and
                MUST be ignored by the receiver.
```

Valid Lifetime: 32-bit unsigned integer. The length of time in seconds (relative to the time the packet is received) that this RD address is valid. A value of all zero bits (0x0) indicates that this RD address is not valid anymore.

RD Address: IPv6 address of the RD.

Figure 4: Resource Directory Address Option

4.1.2. Using DNS-SD to discover a Resource Directory

An RD can advertise its presence in DNS-SD [RFC6763] using the service name "_core-rd._udp" (for CoAP), "_core-rd-dtls._udp" (for CoAP over DTLS), "_core-rd._tcp" (for CoAP over TCP) or "_core-rd-tls._tcp" (for CoAP over TLS) defined in this document. (For the WebSocket transports of CoAP, no service is defined as DNS-SD is typically unavailable in environments where CoAP over WebSockets is used).

The selection of the service indicates the protocol used, and the SRV record points the client to a host name and port to use as a starting point for the URI discovery steps of Section 4.3.

This section is a simplified concrete application of the more generic mechanism specified in [I-D.ietf-core-rd-dns-sd].

4.2. Payload Content Formats

RDs implementing this specification MUST support the application/link-format content format (ct=40).

RDs implementing this specification MAY support additional content formats.

Any additional content format supported by an RD implementing this specification SHOULD be able to express all the information expressible in link-format. It MAY be able to express information that is inexpressible in link-format, but those expressions SHOULD be avoided where possible.

4.3. URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. This section defines discovery of the RD and its URIs using the well-known interface of the CoRE Link Format [RFC6690] after having discovered a host as described in Section 4.1.

Discovery of the RD registration URI is performed by sending either a multicast or unicast GET request to `"/.well-known/core"` and including a Resource Type (rt) parameter [RFC6690] with the value `"core.rd"` in the query string. Likewise, a Resource Type parameter value of `"core.rd-lookup"` is used to discover the URIs for RD Lookup operations, `core.rd*` is used to discover all URIs for RD operations. Upon success, the response will contain a payload with a link format entry for each RD function discovered, indicating the URI of the RD function returned and the corresponding Resource Type. When

performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network (see Section 9.5).

An RD MAY provide hints about the content-formats it supports in the links it exposes or registers, using the "ct" target attribute, as shown in the example below. Clients MAY use these hints to select alternate content-formats for interaction with the RD.

HTTP does not support multicast and consequently only unicast discovery can be supported at the using the HTTP `"/.well-known/core"` resource.

RDs implementing this specification MUST support query filtering for the `rt` parameter as defined in [RFC6690].

While the link targets in this discovery step are often expressed in path-absolute form, this is not a requirement. Clients of the RD SHOULD therefore accept URIs of all schemes they support, both as URIs and relative references, and not limit the set of discovered URIs to those hosted at the address used for URI discovery.

With security policies where the client requires the RD to be authorized to act as an RD, that authorization may be limited to resources on which the authorized RD advertises the adequate resource types. Clients that have obtained links they can not rely on yet can repeat the URI discovery step at the `/.well-known/core` resource of the indicated host to obtain the resource type information from an authorized source.

The URI Discovery operation can yield multiple URIs of a given resource type. The client of the RD can use any of the discovered addresses initially.

The discovery request interface is specified as follows (this is exactly the Well-Known Interface of [RFC6690] Section 4, with the additional requirement that the server MUST support query filtering):

Interaction: EP, CT or Client -> RD

Method: GET

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

URI Template Variables: `rt` := Resource Type. SHOULD contain one of the values `"core.rd"`, `"core.rd-lookup*"`, `"core.rd-lookup-res"`, `"core.rd-lookup-ep"`, or `"core.rd*"`

Accept: absent, application/link-format or any other media type representing web links

The following response is expected on this interface:

Success: 2.05 "Content" or 200 "OK" with an application/link-format or other web link payload containing one or more matching entries for the RD resource.

The following example shows an endpoint discovering an RD using this interface, thus learning that the directory resource location, in this example, is /rd, and that the content-format delivered by the server hosting the resource is application/link-format (ct=40). Note that it is up to the RD to choose its RD locations.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content

Payload:

```
</rd>;rt=core.rd;ct=40,  
</rd-lookup/ep>;rt=core.rd-lookup-ep;ct=40,  
</rd-lookup/res>;rt=core.rd-lookup-res;ct=40
```

Figure 5: Example discovery exchange

The following example shows the way of indicating that a client may request alternate content-formats. The Content-Format code attribute "ct" MAY include a space-separated sequence of Content-Format codes as specified in Section 7.2.1 of [RFC7252], indicating that multiple content-formats are available. The example below shows the required Content-Format 40 (application/link-format) indicated as well as a CBOR and JSON representation from [I-D.ietf-core-links-json] (which have no numeric values assigned yet, so they are shown as TBD64 and TBD504 as in that draft). The RD resource locations /rd, and /rd-lookup are example values. The server in this example also indicates that it is capable of providing observation on resource lookups.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content

Payload:

```
</rd>;rt=core.rd;ct="40 65225",  
</rd-lookup/res>;rt=core.rd-lookup-res;ct="40 TBD64 TBD504";obs,  
</rd-lookup/ep>;rt=core.rd-lookup-ep;ct="40 TBD64 TBD504"
```

Figure 6: Example discovery exchange indicating additional content-formats

For maintenance, management and debugging, it can be useful to identify the components that constitute the RD server. The identification can be used to find client-server incompatibilities, supported features, required updates and other aspects. The Well-Known interface described in Section 4 of [RFC6690] can be used to find such data.

It would typically be stored in an implementation information link (as described in [I-D.bormann-t2trg-rel-impl]):

```
Req: GET /.well-known/core?rel=impl-info
```

```
Res: 2.05 Content
```

```
Payload:
```

```
<http://software.example.com/shiny-resource-directory/1.0beta1>;  
rel=impl-info
```

Figure 7: Example exchange of obtaining implementation information, using the relation type currently proposed in the work-in-progress document

Note that depending on the particular server's architecture, such a link could be anchored at the RD server's root (as in this example), or at individual RD components. The latter is to be expected when different applications are run on the same server.

5. Registration

After discovering the location of an RD, a registrant-ep or CT MAY register the resources of the registrant-ep using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690] or other representations of web links, along with query parameters indicating the name of the endpoint, and optionally the sector, lifetime and base URI of the registration. It is expected that other specifications will define further parameters (see Section 9.3). The RD then creates a new registration resource in the RD and returns its location. The receiving endpoint MUST use that location when refreshing registrations using this interface. Registration resources in the RD are kept active for the period indicated by the lifetime parameter. The creating endpoint is responsible for refreshing the registration resource within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameters ep and d (sector) does not create multiple registration resources.

The following rules apply for a registration request targeting a given (ep, d) value pair:

- * When the (ep, d) value pair of the registration-request is different from any existing registration, a new registration is generated.
- * When the (ep, d) value pair of the registration-request is equal to an existing registration, the content and parameters of the existing registration are replaced with the content of the registration request. Like the later changes to registration resources, security policies (Section 7) usually require such requests to come from the same device.

The posted link-format document can (and typically does) contain relative references both in its link targets and in its anchors, or contain empty anchors. The RD server needs to resolve these references in order to faithfully represent them in lookups. They are resolved against the base URI of the registration, which is provided either explicitly in the "base" parameter or constructed implicitly from the requester's URI as constructed from its network address and scheme.

For media types to which Appendix C applies (i.e. documents in application/link-format), request bodies MUST be expressed in Limited Link Format.

The registration request interface is specified as follows:

Interaction: EP or CT -> RD

Method: POST

URI Template: {+rd}{?ep,d,lt,base,extra-attrs*}

URI Template Variables: rd := RD registration URI (mandatory).
This is the location of the RD, as obtained from discovery.

ep := Endpoint name (mostly mandatory).
The endpoint name is an identifier that MUST be unique within a sector.

As the endpoint name is a Unicode string, it is encoded in UTF-8 (and possibly pct-encoded) during variable expansion (see [RFC6570] Section 3.2.1). The endpoint name MUST NOT contain any character in the inclusive ranges 0-31 or 127-159.

The maximum length of this parameter is 63 UTF-8 encoded bytes.

If the RD is configured to recognize the endpoint to be authorized to use exactly one endpoint name, the RD assigns that name. In that case, giving the endpoint name becomes optional for the client; if the client gives any other endpoint name, it is not authorized to perform the registration.

`d` := Sector (optional). The sector to which this endpoint belongs. When this parameter is not present, the RD MAY associate the endpoint with a configured default sector (possibly based on the endpoint's authorization) or leave it empty.

The sector is encoded like the `ep` parameter, and is limited to 63 UTF-8 encoded bytes as well.

`lt` := Lifetime (optional). Lifetime of the registration in seconds. Range of 1-4294967295. If no lifetime is included in the initial registration, a default value of 90000 (25 hours) SHOULD be assumed.

`base` := Base URI (optional). This parameter sets the base URI of the registration, under which the relative links in the payload are to be interpreted. The specified URI typically does not have a path component of its own, and MUST be suitable as a base URI to resolve any relative references given in the registration. The parameter is therefore usually of the shape "scheme://authority" for HTTP and CoAP URIs. The URI SHOULD NOT have a query or fragment component as any non-empty relative part in a reference would remove those parts from the resulting URI.

In the absence of this parameter the scheme of the protocol, source address and source port of the registration request are assumed. The Base URI is consecutively constructed by concatenating the used protocol's scheme with the characters "://", the requester's source address as an address literal and ":" followed by its port (if it was not the protocol's default one) in analogy to [RFC7252] Section 6.5.

This parameter is mandatory when the directory is filled by a third party such as an commissioning tool.

If the registrant-ep uses an ephemeral port to register with, it MUST include the base parameter in the registration to provide a valid network path.

A registrant that cannot be reached by potential lookup clients

at the address it registers from (e.g. because it is behind some form of Network Address Translation (NAT)) MUST provide a reachable base address with its registration.

If the Base URI contains a link-local IP literal, it MUST NOT contain a Zone Identifier, and MUST be local to the link on which the registration request is received.

Endpoints that register with a base that contains a path component cannot efficiently express their registrations in Limited Link Format (Appendix C). Those applications should use different representations of links to which Appendix C is not applicable (e.g. [I-D.hartke-t2trg-coral]).

extra-attrs := Additional registration attributes (optional). The endpoint can pass any parameter registered at Section 9.3 to the directory. If the RD is aware of the parameter's specified semantics, it processes it accordingly. Otherwise, it MUST store the unknown key and its value(s) as an endpoint attribute for further lookup.

Content-Format: application/link-format or any other indicated media type representing web links

The following response is expected on this interface:

Success: 2.01 "Created" or 201 "Created". The Location-Path option or Location header field MUST be included in the response. This location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration resource. The registration resource location thus returned is for the purpose of updating the lifetime of the registration and for maintaining the content of the registered links, including updating and deleting links.

A registration with an already registered ep and d value pair responds with the same success code and location as the original registration; the set of links registered with the endpoint is replaced with the links from the payload.

The location MUST NOT have a query or fragment component, as that could conflict with query parameters during the Registration Update operation. Therefore, the Location-Query option MUST NOT be present in a successful response.

If the registration fails, including request timeouts, or if delays from Service Unavailable responses with Max-Age or Retry-After accumulate to exceed the registrant's configured timeouts, it SHOULD

pick another registration URI from the "URI Discovery" step and if there is only one or the list is exhausted, pick other choices from the "Finding a Resource Directory" step. Care has to be taken to consider the freshness of results obtained earlier, e.g. of the result of a `"/.well-known/core"` response, the lifetime of an RDAO option and of DNS responses. Any rate limits and persistent errors from the "Finding a Resource Directory" step must be considered for the whole registration time, not only for a single operation.

The following example shows a registrant-ep with the name "node1" registering two resources to an RD using this interface. The location `"/rd"` is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=node1
Content-Format: 40
Payload:
</sensors/temp>;rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel=describedby

Res: 2.01 Created
Location-Path: /rd/4521
```

Figure 8: Example registration payload

An RD may optionally support HTTP. Here is an example of almost the same registration operation above, when done using HTTP.

```
Req:
POST /rd?ep=node1&base=http://[2001:db8:1::1] HTTP/1.1
Host: rd.example.com
Content-Type: application/link-format

</sensors/temp>;rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel=describedby

Res:
HTTP/1.1 201 Created
Location: /rd/4521
```

Figure 9: Example registration payload as expressed using HTTP

5.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to an RD as described in Section 5. Instead, simple endpoints can implement the Simple Registration approach described in this section. An RD implementing this specification **MUST** implement Simple Registration. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the registrant-ep makes available the hosted resources that it wants to be discovered, as links on its `"/.well-known/core"` interface as specified in [RFC6690]. The links in that document are subject to the same limitations as the payload of a registration (with respect to Appendix C).

- * The registrant-ep finds one or more addresses of the directory server as described in Section 4.1.
- * The registrant-ep sends (and regularly refreshes with) a POST request to the `"/.well-known/rd"` URI of the directory server of choice. The body of the POST request is empty, and triggers the resource directory server to perform GET requests at the requesting registrant-ep's `/.well-known/core` to obtain the link-format payload to register.

The registrant-ep includes the same registration parameters in the POST request as it would with a regular registration per Section 5. The registration base URI of the registration is taken from the registrant-ep's network address (as is default with regular registrations).

Example request from registrant-EP to RD (unanswered until the next step):

```
Req: POST /.well-known/rd?lt=6000&ep=node1
(No payload)
```

Figure 10: First half example exchange of a simple registration

- * The RD queries the registrant-ep's discovery resource to determine the success of the operation. It **SHOULD** keep a cache of the discovery resource and not query it again as long as it is fresh.

Example request from the RD to the registrant-EP:

```
Req: GET /.well-known/core
Accept: 40
```

```
Res: 2.05 Content
Content-Format: 40
Payload:
</sen/temp>
```

Figure 11: Example exchange of the RD querying the simple endpoint

With this response, the RD would answer the previous step's request:

```
Res: 2.04 Changed
```

Figure 12: Second half example exchange of a simple registration

The sequence of fetching the registration content before sending a successful response was chosen to make responses reliable, and the point about caching was chosen to still allow very constrained registrants. Registrants MUST be able to serve a GET request to `"/.well-known/core"` after having requested registration. Constrained devices MAY regard the initial request as temporarily failed when they need RAM occupied by their own request to serve the RD's GET, and retry later when the RD already has a cached representation of their discovery resources. Then, the RD can reply immediately and the registrant can receive the response.

The simple registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: `/.well-known/rd{?ep,d,lt,extra-attrs*}`

URI Template Variables are as they are for registration in Section 5. The base attribute is not accepted to keep the registration interface simple; that rules out registration over CoAP-over-TCP or HTTP that would need to specify one. For some time during this document's development, the URI template `"/.well-known/core{?ep,...}"` has been in use instead.

The following response is expected on this interface:

```
Success: 2.04 "Changed".
```

For the second interaction triggered by the above, the registrant-ep takes the role of server and the RD the role of client. (Note that this is exactly the Well-Known Interface of [RFC6690] Section 4):

Interaction: RD -> EP

Method: GET

URI Template: /.well-known/core

The following response is expected on this interface:

Success: 2.05 "Content".

When the RD uses any authorization credentials to access the endpoint's discovery resource, or when it is deployed in a location where third parties might reach it but not the endpoint, it SHOULD verify that the apparent registrant-ep intends to register with the given registration parameters before revealing the obtained discovery information to lookup clients. An easy way to do that is to verify the simple registration request's sender address using the Echo option as described in [I-D.ietf-core-echo-request-tag] Section 2.4.

The RD MUST delete registrations created by simple registration after the expiration of their lifetime. Additional operations on the registration resource cannot be executed because no registration location is returned.

5.2. Third-party registration

For some applications, even Simple Registration may be too taxing for some very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the RD can be filled by a third party device, called a Commissioning Tool (CT). The commissioning tool can fill the RD from a database or other means. For that purpose scheme, IP address and port of the URI of the registered device is the value of the "base" parameter of the registration described in Section 5.

It should be noted that the value of the "base" parameter applies to all the links of the registration and has consequences for the anchor value of the individual links as exemplified in Appendix B. An eventual (currently non-existing) "base" attribute of the link is not affected by the value of "base" parameter in the registration.

5.3. Operations on the Registration Resource

This section describes how the registering endpoint can maintain the registrations that it created. The registering endpoint can be the registrant-ep or the CT. The registrations are resources of the RD.

An endpoint should not use this interface for registrations that it did not create. This is usually enforced by security policies, which in general require equivalent credentials for creation of and operations on a registration.

After the initial registration, the registering endpoint retains the returned location of the registration resource for further operations, including refreshing the registration in order to extend the lifetime and "keep-alive" the registration. When the lifetime of the registration has expired, the RD SHOULD NOT respond to discovery queries concerning this endpoint. The RD SHOULD continue to provide access to the registration resource after a registration time-out occurs in order to enable the registering endpoint to eventually refresh the registration. The RD MAY eventually remove the registration resource for the purpose of garbage collection. If the registration resource is removed, the corresponding endpoint will need to be re-registered.

The registration resource may also be used cancel the registration using DELETE, and to perform further operations beyond the scope of this specification.

Operations on the registration resource are sensitive to reordering; Section 5.3.4 describes how order is restored.

The operations on the registration resource are described below.

5.3.1. Registration Update

The update interface is used by the registering endpoint to refresh or update its registration with an RD. To use the interface, the registering endpoint sends a POST request to the registration resource returned by the initial registration operation.

An update MAY update registration parameters like lifetime, base URI or others. Parameters that are not being changed should not be included in an update. Adding parameters that have not changed increases the size of the message but does not have any other implications. Parameters are included as query parameters in an update operation as in Section 5.

A registration update resets the timeout of the registration to the (possibly updated) lifetime of the registration, independent of whether a "lt" parameter was given.

If the base URI of the registration is changed in an update, relative references submitted in the original registration or later updates are resolved anew against the new base.

The registration update operation only describes the use of POST with an empty payload. Future standards might describe the semantics of using content formats and payloads with the POST method to update the links of a registration (see Section 5.3.3).

The update registration request interface is specified as follows:

Interaction: EP or CT -> RD

Method: POST

URI Template: {+location}{?lt,base,extra-attrs*}

URI Template Variables: location := This is the Location returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 1-4294967295. If no lifetime is included, the previous last lifetime set on a previous update or the original registration (falling back to 90000) SHOULD be used.

base := Base URI (optional). This parameter updates the Base URI established in the original registration to a new value, and is subject to the same restrictions as in the registration.

If the parameter is set in an update, it is stored by the RD as the new Base URI under which to interpret the relative links present in the payload of the original registration.

If the parameter is not set in the request but was set before, the previous Base URI value is kept unmodified.

If the parameter is not set in the request and was not set before either, the source address and source port of the update request are stored as the Base URI.

extra-attrs := Additional registration

attributes (optional). As with the registration, the RD processes them if it knows their semantics. Otherwise, unknown attributes are stored as endpoint attributes, overriding any previously stored endpoint attributes of the same key.

Note that this default behavior does not allow removing an endpoint attribute in an update. For attributes whose functionality depends on the endpoints' ability to remove them in an update, it can make sense to define a value whose presence is equivalent to the absence of a value. As an alternative, an extension can define different updating rules for their attributes. That necessitates either discovery of whether the RD is aware of that extension, or tolerating the default behavior.

Content-Format: none (no payload)

The following responses are expected on this interface:

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

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

If the registration update fails in any way, including "Not Found" and request timeouts, or if the time indicated in a Service Unavailable Max-Age/Retry-After exceeds the remaining lifetime, the registering endpoint SHOULD attempt registration again.

The following example shows how the registering endpoint resets the timeout on its registration resource at an RD using this interface with the example location value: /rd/4521.

Req: POST /rd/4521

Res: 2.04 Changed

Figure 13: Example update of a registration

The following example shows the registering endpoint updating its registration resource at an RD using this interface with the example location value: /rd/4521. The initial registration by the registering endpoint set the following values:

- * endpoint name (ep)=endpoint1
- * lifetime (lt)=500

* Base URI (base)=coap://local-proxy-old.example.com

* payload of Figure 8

The initial state of the RD is reflected in the following request:

Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.05 Content

Payload:

```
<coap://local-proxy-old.example.com/sensors/temp>;
  rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="coap://local-proxy-old.example.com/sensors/temp";
  rel=describedby
```

Figure 14: Example lookup before a change to the base address

The following example shows the registering endpoint changing the Base URI to "coaps://new.example.com:5684":

Req: POST /rd/4521?base=coaps://new.example.com

Res: 2.04 Changed

Figure 15: Example registration update that changes the base address

The consecutive query returns:

Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.05 Content

Payload:

```
<coaps://new.example.com/sensors/temp>;
  rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="coaps://new.example.com/sensors/temp";
  rel=describedby
```

Figure 16: Example lookup after a change to the base address

5.3.2. Registration Removal

Although RD registrations have soft state and will eventually timeout after their lifetime, the registering endpoint SHOULD explicitly remove an entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP or CT -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables: location := This is the Location returned by the RD as a result of a successful earlier registration.

The following responses are expected on this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may already have been removed).

The following examples shows successful removal of the endpoint from the RD with example location value /rd/4521.

Req: DELETE /rd/4521

Res: 2.02 Deleted

Figure 17: Example of a registration removal

5.3.3. Further operations

Additional operations on the registration can be specified in future documents, for example:

- * Send iPATCH (or PATCH) updates ([RFC8132]) to add, remove or change the links of a registration.
- * Use GET to read the currently stored set of links in a registration resource.

Those operations are out of scope of this document, and will require media types suitable for modifying sets of links.

5.3.4. Request freshness

Some security mechanisms usable with an RD allow out of order request processing, or do not even mandate replay protection at all. The RD needs to ensure that operations on the registration resource are executed in an order that does not distort the client's intentions.

This ordering of operations is expressed in terms of freshness as defined in [I-D.ietf-core-echo-request-tag]. Requests that alter a resource's state need to be fresh relative to the latest request that altered that state in a conflicting way.

An RD SHOULD determine a request's freshness, and MUST use the Echo option if it requires request freshness and can not determine the it in any other way. An endpoint MUST support the use of the Echo option. (One reason why an RD would not require freshness is when no relevant registration properties are covered by its security policies.)

5.3.4.1. Efficient use of Echo by an RD

To keep latency and traffic added by the freshness requirements to a minimum, RDs should avoid naive (sufficient but inefficient) freshness criteria.

Some simple mechanisms the RD can employ are:

- * State counter. The RD can keep a monotonous counter that increments whenever a registration changes. For every registration resource, it stores the post-increment value of that resource's last change. Requests altering them need to have at least that value encoded in their Echo option, and are otherwise rejected with a 4.01 Unauthorized and the current counter value as the Echo value. If other applications on the same server use Echo as well, that encoding may include a prefix indicating that it pertains to the RD's counter.

The value associated with a resource needs to be kept across the removal of registrations if the same registration resource is to be reused.

The counter can be reset (and the values of removed resources forgotten) when all previous security associations are reset.

This is the "Persistent Counter" method of
[I-D.ietf-core-echo-request-tag] Appendix A.

- * Preemptive Echo values. The current state counter can be sent in an Echo option not only when requests are rejected with 4.01 Unauthorized, but also with successful responses. Thus, clients can be provided with Echo values sufficient for their next request on a regular basis.

While endpoints may discard received Echo values at leisure between requests, they are encouraged to retain these values for the next request to avoid additional round trips.

- * If the RD can ensure that only one security association has modifying access to any registration at any given time, and that security association provides order on the requests, that order is sufficient to show request freshness.

5.3.4.2. Examples of Echo usage

Figure 18 shows the interactions of an endpoint that has forgotten the server's latest Echo value and temporarily reduces its registration lifetime:

Req: POST /rd/4521?lt=7200

Res: 4.01 Unauthorized
Echo: 0x0123

(EP tries again immediately)

Req: POST /rd/4521?lt=7200
Echo: 0x0123

Res: 2.04 Changed
Echo: 0x0124

(Later the EP regains its confidence in its long-term reachability)

Req: POST /rd/4521?lt=90000
Echo: 0x0124

Res: 2.04 Changed
Echo: 0x0247

Figure 18: Example update of a registration

The other examples do not show Echo options for simplicity, and because they lack the context for any example values to have meaning.

6. RD Lookup

To discover the resources registered with the RD, a lookup interface must be provided. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. JSON or CBOR link format [I-D.ietf-core-links-json]) or using more advanced interfaces (e.g. supporting context or semantic based lookup) on different resources that are discovered independently.

RD Lookup allows lookups for endpoints and resources using attributes defined in this document and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, an endpoint lookup MUST return a list of endpoints and a resource lookup MUST return a list of links to resources.

The lookup type is selected by a URI endpoint, which is indicated by a Resource Type as per Table 1 below:

Lookup Type	Resource Type	Mandatory
Resource	core.rd-lookup-res	Mandatory
Endpoint	core.rd-lookup-ep	Mandatory

Table 1: Lookup Types

6.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD by the registrant. The links and link parameters returned by the lookup are equal to the originally submitted ones, except that the target reference is fully resolved, and that the anchor reference is fully resolved if it is present in the lookup result at all.

Links that did not have an anchor attribute in the registration are returned without an anchor attribute. Links of which href or anchor was submitted as a (full) URI are returned with the respective attribute unmodified.

The above rules allow the client to interpret the response as links without any further knowledge of the storage conventions of the RD. The RD MAY replace the registration base URIs with a configured intermediate proxy, e.g. in the case of an HTTP lookup interface for CoAP endpoints.

If the base URI of a registration contains a link-local address, the RD MUST NOT show its links unless the lookup was made from the link on which the registered endpoint can be reached. The RD MUST NOT include zone identifiers in the resolved URIs.

6.2. Lookup filtering

Using the Accept Option, the requester can control whether the returned list is returned in CoRE Link Format ("application/link-format", default) or in alternate content-formats (e.g. from [I-D.ietf-core-links-json]).

Multiple search criteria MAY be included in a lookup. All included criteria MUST match for a link to be returned. The RD MUST support matching with multiple search criteria.

A link matches a search criterion if it has an attribute of the same name and the same value, allowing for a trailing "*" wildcard operator as in Section 4.1 of [RFC6690]. Attributes that are defined as "relation-types" (in the link-format ABNF) match if the search value matches any of their values (see Section 4.1 of [RFC6690]; e.g. "?if=tag:example.net,2020:sensor" matches ";if=example.regname tag:example.net,2020:sensor;"). A resource link also matches a search criterion if its endpoint would match the criterion, and vice versa, an endpoint link matches a search criterion if any of its resource links matches it.

Note that "href" is a valid search criterion and matches target references. Like all search criteria, on a resource lookup it can match the target reference of the resource link itself, but also the registration resource of the endpoint that registered it. Queries for resource link targets MUST be in URI form (i.e. not relative references) and are matched against a resolved link target. Queries for endpoints SHOULD be expressed in path-absolute form if possible and MUST be expressed in URI form otherwise; the RD SHOULD recognize either. The "anchor" attribute is usable for resource lookups, and, if queried, MUST be in URI form as well.

Additional query parameters "page" and "count" are used to obtain lookup results in specified increments using pagination, where count specifies how many links to return and page specifies which subset of links organized in sequential pages, each containing 'count' links,

starting with link zero and page zero. Thus, specifying count of 10 and page of 0 will return the first 10 links in the result set (links 0-9). Count = 10 and page = 1 will return the next 'page' containing links 10-19, and so on. Unlike block-wise transfer of a complete result set, these parameters ensure that each chunk of results can be interpreted on its own. This simplifies the processing, but can result in duplicate or missed items when coinciding with changes from the registration interface.

Endpoints that are interested in a lookup result repeatedly or continuously can use mechanisms like ETag caching, resource observation ([RFC7641]), or any future mechanism that might allow more efficient observations of collections. These are advertised, detected and used according to their own specifications and can be used with the lookup interface as with any other resource.

When resource observation is used, every time the set of matching links changes, or the content of a matching link changes, the RD sends a notification with the matching link set. The notification contains the successful current response to the given request, especially with respect to representing zero matching links (see "Success" item below).

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: {+type-lookup-location}{?page,count,search*}

URI Template Variables: type-lookup-location := RD Lookup URI for a given lookup type (mandatory). The address is discovered as described in Section 4.3.

search := Search criteria for limiting the number of results (optional).

The search criteria are an associative array, expressed in a form-style query as per the URI template (see [RFC6570] Sections 2.4.2 and 3.2.8)

page := Page (optional). Parameter cannot be used without the count parameter. Results are returned from result set in pages that contain 'count' links starting from index (page * count). Page numbering starts with zero.

count := Count (optional). Number of

results is limited to this parameter value. If the page parameter is also present, the response MUST only include 'count' links starting with the (page * count) link in the result set from the query. If the count parameter is not present, then the response MUST return all matching links in the result set. Link numbering starts with zero.

Accept: absent, application/link-format or any other indicated media type representing web links

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format" or other web link payload containing matching entries for the lookup.

The payload can contain zero links (which is an empty payload in [RFC6690] link format, but could also be "[]" in JSON based formats), indicating that no entities matched the request.

6.3. Resource lookup examples

The examples in this section assume the existence of CoAP hosts with a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples.

The following example shows a client performing a resource lookup with the example resource look-up locations discovered in Figure 5:

Req: GET /rd-lookup/res?rt=tag:example.org,2020:temperature

Res: 2.05 Content

Payload:

```
<coap://[2001:db8:3::123]:61616/temp>;  
  rt="tag:example.org,2020:temperature"
```

Figure 19: Example a resource lookup

A client that wants to be notified of new resources as they show up can use observation:

```
Req: GET /rd-lookup/res?rt=tag:example.org,2020:light
Observe: 0
```

```
Res: 2.05 Content
Observe: 23
Payload: empty
```

(at a later point in time)

```
Res: 2.05 Content
Observe: 24
Payload:
<coap://[2001:db8:3::124]/west>;rt="tag:example.org,2020:light",
<coap://[2001:db8:3::124]/south>;rt="tag:example.org,2020:light",
<coap://[2001:db8:3::124]/east>;rt="tag:example.org,2020:light"
```

Figure 20: Example an observing resource lookup

The following example shows a client performing a paginated resource lookup

```
Req: GET /rd-lookup/res?page=0&count=5
```

```
Res: 2.05 Content
Payload:
<coap://[2001:db8:3::123]:61616/res/0>;ct=60,
<coap://[2001:db8:3::123]:61616/res/1>;ct=60,
<coap://[2001:db8:3::123]:61616/res/2>;ct=60,
<coap://[2001:db8:3::123]:61616/res/3>;ct=60,
<coap://[2001:db8:3::123]:61616/res/4>;ct=60
```

```
Req: GET /rd-lookup/res?page=1&count=5
```

```
Res: 2.05 Content
Payload:
<coap://[2001:db8:3::123]:61616/res/5>;ct=60,
<coap://[2001:db8:3::123]:61616/res/6>;ct=60,
<coap://[2001:db8:3::123]:61616/res/7>;ct=60,
<coap://[2001:db8:3::123]:61616/res/8>;ct=60,
<coap://[2001:db8:3::123]:61616/res/9>;ct=60
```

Figure 21: Examples of paginated resource lookup

The following example shows a client performing a lookup of all resources of all endpoints of a given endpoint type. It assumes that two endpoints (with endpoint names "sensor1" and "sensor2") have previously registered with their respective addresses "coap://sensor1.example.com" and "coap://sensor2.example.com", and posted the very payload of the 6th response of section 5 of [RFC6690].

It demonstrates how absolute link targets stay unmodified, while relative ones are resolved:

Req: GET /rd-lookup/res?et=tag:example.com,2020:platform

Res: 2.05 Content

Payload:

```
<coap://sensor1.example.com/sensors>;ct=40;title="Sensor Index",
<coap://sensor1.example.com/sensors/temp>;rt=temperature-c;if=sensor,
<coap://sensor1.example.com/sensors/light>;rt=light-lux;if=sensor,
<http://www.example.com/sensors/t123>;rel=describedby;
  anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor1.example.com/t>;rel=alternate;
  anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor2.example.com/sensors>;ct=40;title="Sensor Index",
<coap://sensor2.example.com/sensors/temp>;rt=temperature-c;if=sensor,
<coap://sensor2.example.com/sensors/light>;rt=light-lux;if=sensor,
<http://www.example.com/sensors/t123>;rel=describedby;
  anchor="coap://sensor2.example.com/sensors/temp",
<coap://sensor2.example.com/t>;rel=alternate;
  anchor="coap://sensor2.example.com/sensors/temp"
```

Figure 22: Example of resource lookup from multiple endpoints

6.4. Endpoint lookup

The endpoint lookup returns links to and information about registration resources, which themselves can only be manipulated by the registering endpoint.

Endpoint registration resources are annotated with their endpoint names (ep), sectors (d, if present) and registration base URI (base; reports the registrant-ep's address if no explicit base was given) as well as a constant resource type (rt="core.rd-ep"); the lifetime (lt) is not reported. Additional endpoint attributes are added as target attributes to their endpoint link unless their specification says otherwise.

Links to endpoints SHOULD be presented in path-absolute form or, if required, as (full) URIs. (This ensures that the output conforms to Limited Link Format as described in Appendix C.)

Base addresses that contain link-local addresses MUST NOT include zone identifiers, and such registrations MUST NOT be shown unless the lookup was made from the same link from which the registration was made.

While Endpoint Lookup does expose the registration resources, the RD does not need to make them accessible to clients. Clients SHOULD NOT attempt to dereference or manipulate them.

An RD can report registrations in lookup whose URI scheme and authority differ from the lookup resource's. Lookup clients MUST be prepared to see arbitrary URIs as registration resources in the results and treat them as opaque identifiers; the precise semantics of such links are left to future specifications.

The following example shows a client performing an endpoint lookup limited to endpoints of endpoint type

```
"tag:example.com,2020:platform":
```

```
Req: GET /rd-lookup/ep?et=tag:example.com,2020:platform
```

```
Res: 2.05 Content
```

```
Payload:
```

```
</rd/1234>;base="coap://[2001:db8:3::127]:61616";ep=node5;  
  et="tag:example.com,2020:platform";ct=40;rt=core.rd-ep,  
</rd/4521>;base="coap://[2001:db8:3::129]:61616";ep=node7;  
  et="tag:example.com,2020:platform";ct=40;d=floor-3;  
  rt=core.rd-ep
```

Figure 23: Examples of endpoint lookup

7. Security policies

The security policies that are applicable to an RD strongly depend on the application, and are not set out normatively here.

This section provides a list of aspects that applications should consider when describing their use of the RD, without claiming to cover all cases. It is using terminology of [I-D.ietf-ace-oauth-authz], in which the RD acts as the Resource Server (RS), and both registrant-eps and lookup clients act as Clients (C) with support from an Authorization Server (AS), without the intention of ruling out other (e.g. certificate / public-key infrastructure (PKI) based) schemes.

Any, all or none of the below can apply to an application. Which are relevant depends on its protection objectives.

Security policies are set by configuration of the RD, or by choice of the implementation. Lookup clients (and, where relevant, endpoints) can only trust an RD to uphold them if it is authenticated, and authorized to serve as an RD according to the application's requirements.

7.1. Endpoint name

Whenever an RD needs to provide trustworthy results to clients doing endpoint lookup, or resource lookup with filtering on the endpoint name, the RD must ensure that the registrant is authorized to use the given endpoint name. This applies both to registration and later to operations on the registration resource. It is immaterial whether the client is the registrant-ep itself or a CT is doing the registration: The RD cannot tell the difference, and CTs may use authorization credentials authorizing only operations on that particular endpoint name, or a wider range of endpoint names.

It is up to the concrete security policy to describe how endpoint name and sector are transported when certificates are used. For example, it may describe how SubjectAltName dNSName entries are mapped to endpoint and domain names.

7.1.1. Random endpoint names

Conversely, in applications where the RD does not check the endpoint name, the authorized registering endpoint can generate a random number (or string) that identifies the endpoint. The RD should then remember unique properties of the registrant, associate them with the registration for as long as its registration resource is active (which may be longer than the registration's lifetime), and require the same properties for operations on the registration resource.

Registrants that are prepared to pick a different identifier when their initial attempt (or attempts, in the unlikely case of two subsequent collisions) at registration is unauthorized should pick an identifier at least twice as long as the expected number of registrants; registrants without such a recovery options should pick significantly longer endpoint names (e.g. using UUID URNs [RFC4122]).

7.2. Entered resources

When lookup clients expect that certain types of links can only originate from certain endpoints, then the RD needs to apply filtering to the links an endpoint may register.

For example, if clients use an RD to find a server that provides firmware updates, then any registrant that wants to register (or update) links to firmware sources will need to provide suitable credentials to do so, independently of its endpoint name.

Note that the impact of having undesirable links in the RD depends on the application: if the client requires the firmware server to present credentials as a firmware server, a fraudulent link's impact is limited to the client revealing its intention to obtain updates and slowing down the client until it finds a legitimate firmware server; if the client accepts any credentials from the server as long as they fit the provided URI, the impact is larger.

An RD may also require that links are only registered if the registrant is authorized to publish information about the anchor (or even target) of the link. One way to do this is to demand that the registrant present the same credentials as a client that they'd need to present if contacted as a server at the resources' URI, which may include using the address and port that are part of the URI. Such a restriction places severe practical limitations on the links that can be registered.

As above, the impact of undesirable links depends on the extent to which the lookup client relies on the RD. To avoid the limitations, RD applications should consider prescribing that lookup clients only use the discovered information as hints, and describe which pieces of information need to be verified because they impact the application's security. A straightforward way to verify such information is to request it again from an authorized server, typically the one that hosts the target resource. That similar to what happens in Section 4.3 when the URI discovery step is repeated.

7.3. Link confidentiality

When registrants publish information in the RD that is not available to any client that would query the registrant's /.well-known/core interface, or when lookups to that interface are subject to stricter firewalling than lookups to the RD, the RD may need to limit which lookup clients may access the information.

In this case, the endpoint (and not the lookup clients) needs to be careful to check the RD's authorization. The RD needs to check any lookup client's authorization before revealing information directly (in resource lookup) or indirectly (when using it to satisfy a resource lookup search criterion).

7.4. Segmentation

Within a single RD, different security policies can apply.

One example of this are multi-tenant deployments separated by the sector (d) parameter. Some sectors might apply limitations on the endpoint names available, while others use a random identifier approach to endpoint names and place limits on the entered links based on their attributes instead.

Care must be taken in such setups to determine the applicable access control measures to each operation. One easy way to do that is to mandate the use of the sector parameter on all operations, as no credentials are suitable for operations across sector borders anyway.

7.5. First-Come-First-Remembered: A default policy

The First-Come-First-Remembered policy is provided both as a reference example for a security policy definition, and as a policy that implementations may choose to use as default policy in absence of other configuration. It is designed to enable efficient discovery operations even in ad-hoc settings.

Under this policy, the RD accepts registrations for any endpoint name that is not assigned to an active registration resource, and only accepts registration updates from the same endpoint. The policy is minimal in that towards lookup clients it does not make any of the claims of Section 7.2 and Section 7.3, and its claims on Section 7.1 are limited to the lifetime of that endpoint's registration. It does, however, guarantee towards any endpoint that for the duration of its registration, its links will be discoverable on the RD.

When a registration or operation is attempted, the RD MUST determine the client's subject name or public key:

- * If the client's credentials indicate any subject name that is certified by any authority which the RD recognizes (which may be the system's trust anchor store), all such subject names are stored. With CWT or JWT based credentials (as common with ACE), the Subject (sub) claim is stored as a single name, if it exists. With X.509 certificates, the Common Name (CN) and the complete list of SubjectAltName entries are stored. In both cases, the authority that certified the claim is stored along with the subject, as the latter may only be locally unique.

- * Otherwise, if the client proves possession of a private key, the matching public key is stored. This applies both to raw public keys and to the public keys indicated in certificates that failed the above authority check.
- * If neither is present, a reference to the security session itself is stored. With (D)TLS, that is the connection itself, or the session resumption information if available. With OSCORE, that is the security context.

As part of the registration operation, that information is stored along with the registration resource.

The RD MUST accept all registrations whose registration resource is not already active, as long as they are made using a security layer supported by the RD.

Any operation on a registration resource, including registrations that lead to an existing registration resource, MUST be rejected by the RD unless all the stored information is found in the new request's credentials.

Note that even though subject names are compared in this policy, they are never directly compared to endpoint names, and an endpoint can not expect to "own" any particular endpoint name outside of an active registration -- even if a certificate says so. It is an accepted shortcoming of this approach that the endpoint has no indication of whether the RD remembers it by its subject name or public key; recognition by subject happens on a best-effort base (given the RD may not recognize any authority). Clients MUST be prepared to pick a different endpoint name when rejected by the RD initially or after a change in their credentials; picking an endpoint name as per Section 7.1.1 is an easy option for that.

For this policy to be usable without configuration, clients should not set a sector name in their registrations. An RD can set a default sector name for registrations accepted under this policy, which is useful especially in a segmented setup where different policies apply to different sectors. The configuration of such a behavior, as well as any other configuration applicable to such an RD (i.e. the set of recognized authorities) is out of scope for this document.

8. Security Considerations

The security considerations as described in Section 5 of [RFC8288] and Section 6 of [RFC6690] apply. The `"/.well-known/core"` resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252].

Access that is limited or affects sensitive data SHOULD be protected, e.g. using (D)TLS or OSCORE ([RFC8613]; which aspects of the RD this affects depends on the security policies of the application (see Section 7).

8.1. Discovery

Most steps in discovery of the RD, and possibly its resources, are not covered by CoAP's security mechanisms. This will not endanger the security properties of the registrations and lookup itself (where the client requires authorization of the RD if it expects any security properties of the operation), but may leak the client's intention to third parties, and allow them to slow down the process.

To mitigate that, clients can retain the RD's address, use secure discovery options like configured addresses, and send queries for RDs in a very general form (`"?rt=core.rd"` rather than `"?rt=core.rd-lookup-ep"`).

8.2. Endpoint Identification and Authentication

An Endpoint (name, sector) pair is unique within the set of endpoints registered by the RD. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint on an RD SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security.

Consider the following threat: two devices A and B are registered at a single server. Both devices have unique, per-device credentials for use with DTLS to make sure that only parties with authorization to access A or B can do so.

Now, imagine that a malicious device A wants to sabotage the device B. It uses its credentials during the DTLS exchange. Then, it specifies the endpoint name of device B as the name of its own endpoint in device A. If the server does not check whether the identifier provided in the DTLS handshake matches the identifier used at the CoAP layer then it may be inclined to use the endpoint name for looking up what information to provision to the malicious device.

Endpoint authorization needs to be checked on registration and registration resource operations independently of whether there are configured requirements on the credentials for a given endpoint name (and sector; Section 7.1) or whether arbitrary names are accepted (Section 7.1.1).

Simple registration could be used to circumvent address-based access control: An attacker would send a simple registration request with the victim's address as source address, and later look up the victim's /.well-known/core content in the RD. Mitigation for this is recommended in Section 5.1.

The registration resource path is visible to any client that is allowed endpoint lookup, and can be extracted by resource lookup clients as well. The same goes for registration attributes that are shown as target attributes or lookup attributes. The RD needs to consider this in the choice of registration resource paths, and administrators or endpoint in their choice of attributes.

8.3. Access Control

Access control SHOULD be performed separately for the RD registration and Lookup API paths, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the sector, endpoint or resource level.

The precise access controls necessary (and the consequences of failure to enforce them) depend on the protection objectives of the application and the security policies (Section 7) derived from them.

8.4. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly amplify and distribute a DoS attack as UDP does not require return routability check. Since RD lookup responses can be significantly larger than requests, RDs are prone to this.

[RFC7252] describes this at length in its Section 11.3, including some mitigation by using small block sizes in responses. The upcoming [I-D.ietf-core-echo-request-tag] updates that by describing a source address verification mechanism using the Echo option.

[If this document is published together with or after I-D.ietf-core-echo-request-tag, the above paragraph is replaced with the following:

[RFC7252] describes this at length in its Section 11.3, and [I-D.ietf-core-echo-request-tag] (which updates the former) recommends using the Echo option to verify the request's source address.

]

8.5. Skipping freshness checks

When RD based applications are built in which request freshness checks are not performed, these concerns need to be balanced:

- * When alterations to registration attributes are reordered, an attacker may create any combination of attributes ever set, with the attack difficulty determined by the security layer's replay properties.

For example, if Figure 18 were conducted without freshness assurances, an attacker could later reset the lifetime back to 7200. Thus, the device is made unreachable to lookup clients.

- * When registration updates without query parameters (which just serve to restart the lifetime) can be reordered, an attacker can use intercepted messages to give the appearance of the device being alive to the RD.

This is unacceptable when the RD's security policy promises reachability of endpoints (e.g. when disappearing devices would trigger further investigation), but may be acceptable with other policies.

9. IANA Considerations

9.1. Resource Types

IANA is asked to enter the following values into the Resource Type (rt=) Link Target Attribute Values sub-registry of the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690]:

Value	Description	Reference
core.rd	Directory resource of an RD	RFCTHIS Section 4.3
core.rd-lookup-res	Resource lookup of an RD	RFCTHIS Section 4.3
core.rd-lookup-ep	Endpoint lookup of an RD	RFCTHIS Section 4.3
core.rd-ep	Endpoint resource of an RD	RFCTHIS Section 6

Table 2

9.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the sub-registry "IPv6 Neighbor Discovery Option Formats" of the "Internet Control Message Protocol version 6 (ICMPv6) Parameters" registry:

- * Resource Directory Address Option (TBD38)

[The RFC editor is asked to replace TBD38 with the assigned number in the document; the value 38 is suggested.]

9.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include

- * the human readable name of the parameter,
- * the short name as used in query parameters or target attributes,
- * indication of whether it can be passed as a query parameter at registration of endpoints, as a query parameter in lookups, or be expressed as a target attribute,
- * syntax and validity requirements if any,

- * a description,
- * and a link to reference documentation.

The query parameter MUST be both a valid URI query key [RFC3986] and a token as used in [RFC8288].

The description must give details on whether the parameter can be updated, and how it is to be processed in lookups.

The mechanisms around new RD parameters should be designed in such a way that they tolerate RD implementations that are unaware of the parameter and expose any parameter passed at registration or updates on in endpoint lookups. (For example, if a parameter used at registration were to be confidential, the registering endpoint should be instructed to only set that parameter if the RD advertises support for keeping it confidential at the discovery step.)

Initial entries in this sub-registry are as follows:

Full name	Short	Validity	Use	Description
Endpoint Name	ep	Unicode*	RLA	Name of the endpoint
Lifetime	lt	1-4294967295	R	Lifetime of the registration in seconds
Sector	d	Unicode*	RLA	Sector to which this endpoint belongs
Registration Base URI	base	URI	RLA	The scheme, address and port and path at which this server is available
Page	page	Integer	L	Used for pagination
Count	count	Integer	L	Used for pagination
Endpoint Type	et	Section 9.3.1	RLA	Semantic type of the endpoint (see Section 9.4)

Table 3: RD Parameters

(Short: Short name used in query parameters or target attributes.
 Validity: Unicode* = 63 Bytes of UTF-8 encoded Unicode, with no control characters as per Section 5. Use: R = used at registration, L = used at lookup, A = expressed in target attribute.)

The descriptions for the options defined in this document are only summarized here. To which registrations they apply and when they are to be shown is described in the respective sections of this document. All their reference documentation entries point to this document.

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC8126]. The evaluation should consider formal criteria, duplication of functionality (Is the new entry redundant with an existing one?), topical suitability (E.g. is the described property actually a property of the endpoint and not a property of a particular resource, in which case it should go into the payload of the registration and need not be registered?), and the potential for conflict with commonly used target attributes (For

example, "if" could be used as a parameter for conditional registration if it were not to be used in lookup or attributes, but would make a bad parameter for lookup, because a resource lookup with an "if" query parameter could ambiguously filter by the registered endpoint property or the [RFC6690] target attribute).

9.3.1. Full description of the "Endpoint Type" RD Parameter

An endpoint registering at an RD can describe itself with endpoint types, similar to how resources are described with Resource Types in [RFC6690]. An endpoint type is expressed as a string, which can be either a URI or one of the values defined in the Endpoint Type sub-registry. Endpoint types can be passed in the "et" query parameter as part of extra-attrs at the Registration step, are shown on endpoint lookups using the "et" target attribute, and can be filtered for using "et" as a search criterion in resource and endpoint lookup. Multiple endpoint types are given as separate query parameters or link attributes.

Note that Endpoint Type differs from Resource Type in that it uses multiple attributes rather than space separated values. As a result, RDs implementing this specification automatically support correct filtering in the lookup interfaces from the rules for unknown endpoint attributes.

9.4. "Endpoint Type" (et=) RD Parameter values

This specification establishes a new sub-registry under "CoRE Parameters" called '"Endpoint Type" (et=) RD Parameter values'. The registry properties (required policy, requirements, template) are identical to those of the Resource Type parameters in [RFC6690], in short:

The review policy is IETF Review for values starting with "core", and Specification Required for others.

The requirements to be enforced are:

- * The values MUST be related to the purpose described in Section 9.3.1.
- * The registered values MUST conform to the ABNF reg-rel-type definition of [RFC6690] and MUST NOT be a URI.
- * It is recommended to use the period "." character for segmentation.

The registry initially contains one value:

- * "core.rd-group": An application group as described in Appendix A.

9.5. Multicast Address Registration

IANA is asked to assign the following multicast addresses for use by CoAP nodes:

IPv4 -- "all CoRE Resource Directories" address MCD2 (suggestion: 224.0.1.189), from the "IPv4 Multicast Address Space Registry". As the address is used for discovery that may span beyond a single network, it has come from the Internetwork Control Block (224.0.1.x) [RFC5771].

IPv6 -- "all CoRE Resource Directories" address MCD1 (suggestions FF0X::FE), from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only.

[The RFC editor is asked to replace MCD1 and MCD2 with the assigned addresses throughout the document.]

9.6. Well-Known URIs

IANA is asked to permanently register the URI suffix "rd" in the "Well-Known URIs" registry. The change controller is the IETF, this document is the reference.

9.7. Service Names and Transport Protocol Port Number Registry

IANA is asked to enter four new items into the Service Names and Transport Protocol Port Number Registry:

- * Service name: "core-rd", Protocol: "udp", Description: "Resource Directory accessed using CoAP"
- * Service name "core-rd-dtls", Protocol: "udp", Description: "Resource Directory accessed using CoAP over DTLS"
- * Service name: "core-rd", Protocol: "tcp", Description: "Resource Directory accessed using CoAP over TCP"
- * Service name "core-rd-tls", Protocol: "tcp", Description: "Resource Directory accessed using CoAP over TLS"

All in common have this document as their reference.

10. Examples

Two examples are presented: a Lighting Installation example in Section 10.1 and a LwM2M example in Section 10.2.

10.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the RD with a CoAP interface to facilitate the installation and start-up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address as described in Appendix A. No conclusions must be drawn on the realization of actual installation or naming procedures, because the example only "emphasizes" some of the issues that may influence the use of the RD and does not pretend to be normative.

10.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one endpoint. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager.

At the moment of installation, the network under installation is not necessarily connected to the DNS infrastructure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and the sensor. The addresses shown in Table 4 below stand in for these in the following examples.

Name	IPv6 address
luminary1	2001:db8:4::1
luminary2	2001:db8:4::2
Presence sensor	2001:db8:4::3
RD	2001:db8:4::ff

Table 4: Addresses used in the examples

In Section 10.1.2 the use of RD during installation is presented.

10.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have `rt=tag:example.com,2020:light`, and the presence sensor has `rt=tag:example.com,2020:p-sensor`. The endpoints have names which are relevant to the light installation manager. In this case `luminary1`, `luminary2`, and the presence sensor are located in room 2-4-015, where `luminary1` is located at the window and `luminary2` and the presence sensor are located at the door. The endpoint names reflect this physical location. The middle, left and right lamps are accessed via path `/light/middle`, `/light/left`, and `/light/right` respectively. The identifiers relevant to the RD are shown in Table 5 below:

Name	endpoint	resource path	resource type
luminary1	lm_R2-4-015_wndw	/light/left	tag:example.com,2020:light
luminary1	lm_R2-4-015_wndw	/light/middle	tag:example.com,2020:light
luminary1	lm_R2-4-015_wndw	/light/right	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/left	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/middle	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/right	tag:example.com,2020:light
Presence sensor	ps_R2-4-015_door	/ps	tag:example.com,2020:p-sensor

Table 5: RD identifiers

It is assumed that the CT has performed RD discovery and has received a response like the one in the Section 4.3 example.

The CT inserts the endpoints of the luminaries and the sensor in the RD using the registration base URI parameter (base) to specify the interface address:

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=lm_R2-4-015_wndw&base=coap://[2001:db8:4::1]&d=R2-4-015
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/4521

Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=lm_R2-4-015_door&base=coap://[2001:db8:4::2]&d=R2-4-015
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/4522

Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=ps_R2-4-015_door&base=coap://[2001:db8:4::3]&d=R2-4-015
Payload:
</ps>;rt="tag:example.com,2020:p-sensor"

Res: 2.01 Created
Location-Path: /rd/4523
```

Figure 24: Example of registrations a CT enters into an RD

The sector name d=R2-4-015 has been added for an efficient lookup because filtering on "ep" name is more awkward. The same sector name is communicated to the two luminaries and the presence sensor by the CT.

The group is specified in the RD. The base parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, the resources supported by all group members are published.

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::1]
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/501
```

Figure 25: Example of a multicast group a CT enters into an RD

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its sector and being configured to join any group containing lights, searches for candidate groups and joins them:

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
      ?d=R2-4-015&et=core.rd-group&rt=light

Res: 2.05 Content
Payload:
</rd/501>;ep=grp_R2-4-015;et=core.rd-group;
      base="coap://[ff05::1]";rt=core.rd-ep
```

Figure 26: Example of a lookup exchange to find suitable multicast addresses

From the returned base parameter value, the luminary learns the multicast address of the multicast group.

The presence sensor can learn the presence of groups that support resources with `rt=tag:example.com,2020:light` in its own sector by sending the same request, as used by the luminary. The presence sensor learns the multicast address to use for sending messages to the luminaries.

10.2. OMA Lightweight M2M (LwM2M)

OMA LwM2M is a profile for device services based on CoAP, providing interfaces and operations for device management and device service enablement.

An LwM2M server is an instance of an LwM2M middleware service layer, containing an RD ([LwM2M] page 36f).

That RD only implements the registration interface, and no lookup is implemented. Instead, the LwM2M server provides access to the registered resources, in a similar way to a reverse proxy.

The location of the LwM2M Server and RD URI path is provided by the LwM2M Bootstrap process, so no dynamic discovery of the RD is used. LwM2M Servers and endpoints are not required to implement the /.well-known/core resource.

11. Acknowledgments

Oscar Novo, Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Jim Schaad, Mohit Sethi, Hauke Petersen, Hannes Tschofenig, Sampo Ukkola, Linyi Tian, Jan Newmarch, Matthias Kovatsch, Jaime Jimenez and Ted Lemon have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the RD concepts were originally developed.

12. Changelog

changes from -27 to -28

- * Security policies / link confidentiality: Point out the RD's obligations that follow from such a policy.
- * Simple registration: clarify term "regular registration" by introducing it along with the reference to Section 5
- * Wording fix in first-come-first-remembered
- * Wording fixes in RD definition
- * Capitalization: Consistently using "registration resource"

changes from -26 to -27

- * In general, this addresses the points that were pointed out in <https://mailarchive.ietf.org/arch/msg/core/xWLomwwhovkU-CPGNxnvs40BhaM/> as having "evolved from the review comments being discussed in the interim meetings", and the review comments from Esko Dijk that were largely entangled in these points.
- * Relaxation of the serialization rules for link-format

The interpretation of RFC6690 used in Appendix B.4 was shown to be faulty. Along with a correction, the common implementations of link-format were surveyed again and it was found that the only one

that employed the faulty interpretation can still safely be upgraded. These were removed from the set considered for Limited Link Format, making the set of valid Limited Link Format documents larger.

As a consequence, the prescribed serialization of RD output can be roughly halved in bytes.

There might be additional usage patterns that are possible with the new set of constraints, but there is insufficient implementation and deployment experience with them to warrant a change changes on that front at this point. The specification can later be extended compatibly to allow these cases and drop the requirement of Limited Link Format.

- * Add Request freshness subsection

It is now recommended (with security considerations on consequences of not doing it) to require ordering of RD operations.

The Echo mechanism (previously suggested in various places but never exclusively) is the one prescribed way of getting this ordering, making the echo-request-tag reference normative.

- * Improved expression about when an RD needs to verify simple registration.

The simple wording missed the authorization part, and did not emphasize that this is a per-deployment property.

- * Point out the non-atomic properties of paginated access.

- * Clarification around impl-info reference.

- * Inconsistencies and extraneous quotations removed from examples.

changes from -25 to -26

- * Security policies:

- The First-Come-First-Remembered policy is added as an example and a potential default behavior.
- Clarify that the mapping between endpoint names and subject fields is up to a policy that defines reliance on names, and give an example.

- Random EP names: Point that multiple collisions are possible but unlikely.
- Add pointers to policies:
 - o RD replication: Point out that policies may limit that.
 - o Registration: Reword (ep, d) mapping to a previous registration's resource that could have been read as another endpoint taking over an existing registration.
- Clarify that the security policy is a property of the RD the any client may need to verify by checking the RD's authorization.
- Clarify how information from an untrusted RD can be verified
- Remove speculation about how in detail ACE scopes are obtained.
- * Security considerations:
 - Generalize to all current options for security layers usable with CoAP (OSCORE was missing as the text predated RFC8613)
 - Relax the previous SHOULD on secure access to SHOULD where protection is indicated by security policies (bringing the text in line with the -25 changes)
 - Point out that failure to follow the security considerations has implications depending on the protection objective described with the security policies
 - Shorten amplification mitigation
 - Add note about information in Registration Resource path.
 - Acknowledge that most host discovery operations are not secured; mention consequences and mitigation.
- * Abstract, introduction: removed "or disperse networks"
- * RD discovery:
 - Drop the previously stated assumption that RDAO and any DHCP options would only be used together with SLAAC and DHCP for address configuration, respectively.

- Give concrete guidance for address selection based on RFC6724 when responding to multicasts
- RDAO:
 - o Clarify that it is an option for RAs and not other ND messages.
 - o Change Lifetime from 16-bit minutes to 32-bit seconds and swap it with Reserved (aligning it with RDNSS which it shares other properties as well).
- Point out that clients may need to check RD authorization already in last discovery step
- * Registration:
 - Wording around "mostly mandatory" has been improved, conflicts clarified and sector default selection adjusted.
- * Simple registration: Rather than coopting POSTs to /.well-known/core, a new resource /.well-known/rd is registered. A historical note in the text documents the change.
- * Examples:
 - Use example URIs rather than unclear reg names (unless it's RFC6690 examples, which were kept for continuity)
 - The LwM2M example was reduced from an outdated explanation of the complete LwM2M model to a summary of how RD is used in there, with a reference to the current specification.
 - Luminary example: Explain example addresses
 - Luminary example: Drop reference to coap-group mechanism that's becoming obsolete, and thus also to RFC7390
 - Multicast addresses in the examples were changed from ff35:30:2001:db8::x to ff35:30:2001:db8:f1::8000:x; the 8000 is to follow RFC 3307, and the f1 is for consistency with all the other example addresses where 2001:db8::/32 is subnetted to 2001:db8:x::/48 by groups of internally consistent examples.
- * Use case text enhancements
 - Home and building automation: Tie in with RD

- M2M: Move system design paragraph towards the topic of reusability.
- * Various editorial fixes in response to Gen-ART and IESG reviews.
- * Rename 'Full description of the "Endpoint Type" Registration Parameter' section to '... RD Parameter'
- * Error handling: Place a SHOULD around the likely cases, and make the previous "MUST to the best of their capabilities" a "must".
- * impl-info: Add note about the type being WIP
- * Interaction tables: list CTs as possible initiators where applicable
- * Registration update: Relax requirement to not send parameters needlessly
- * Terminology: Clarify that the CTs' installation events can occur multiple times.
- * Promote RFCs 7252, 7230 and 8288 to normative references
- * Moved Christian Amsuess to first author

changes from -24 to -25

- * Large rework of section 7 (Security policies)

Rather than prescribing which data in the RD is authenticated (and how), it now describes what applications built on an RD can choose to authenticate, show possibilities on how to do it and outline what it means for clients.

This addresses Russ' Genart review points on details in the text in a rather broad fashion. That is because the discussion on the topic inside the WG showed that that text on security has been driven more review-by-review than by an architectural plan of the authors and WG.

- * Add concrete suggestions (twice as long as registrant number with retries, or UUIDs without) for random endpoint names
- * Point out that simple registration can have faked origins, RECOMMEND mitigation when applicable and suggest the Echo mechanism to implement it.

- * Reference existing and upcoming specifications for DDOS mitigation in CoAP.
 - * Explain the provenance of the example's multicast address.
 - * Make "SHOULD" of not manipulating foreign registrations a "should" and explain how it is enforced
 - * Clarify application of RFC6570 to search parameters
 - * Syntactic fixes in examples
 - * IANA:
 - Don't announce expected number of registrations (goes to write-up)
 - Include syntax as part of a field's validity in entry requirements
 - * Editorial changes
 - Align wording between abstract and introduction
 - Abbreviation normalization: "ER model", "RD"
 - RFC8174 boilerplate update
 - Minor clarity fixes
 - Markup and layouting
- changes from -23 to -24
- * Discovery using DNS-SD added again
 - * Minimum lifetime (lt) reduced from 60 to 1
 - * References added
 - * IANA considerations
 - added about .well-known/core resource
 - added DNS-SD service names
 - made RDAO option number a suggestion

- added "reference" field to endpoint type registry
 - * Lookup: mention that anchor is a legitimate lookup attribute
 - * Terminology and example fixes
 - * Layout fixes, esp. the use of non-ASCII characters in figures
- changes from -22 to -23
- * Explain that updates can not remove attributes
 - * Typo fixes
- changes from -21 to -22
- * Request a dedicated IPv4 address from IANA (rather than sharing with All CoAP nodes)
 - * Fix erroneous examples
 - * Editorial changes
- Add figure numbers to examples
 - Update RD parameters table to reflect changes of earlier versions in the text
 - Typos and minor wording
- changes from -20 to -21
- (Processing comments during WGLC)
- * Defer outdated description of using DNS-SD to find an RD to the defining document
 - * Describe operational conditions in automation example
 - * Recommend particular discovery mechanisms for some managed network scenarios
- changes from -19 to -20
- (Processing comments from the WG chair review)
- * Define the permissible characters in endpoint and sector names

- * Express requirements on NAT situations in more abstract terms
- * Shifted heading levels to have the interfaces on the same level
- * Group instructions for error handling into general section
- * Simple Registration: process reflowed into items list
- * Updated introduction to reflect state of CoRE in general, reference RFC7228 (defining "constrained") and use "IoT" term in addition to "M2M"
- * Update acknowledgements
- * Assorted editorial changes
 - Unify examples style
 - Terminology: RDAO defined and not only expanded
 - Add CT to Figure 1
 - Consistency in the use of the term "Content Format"

changes from -18 to -19

- * link-local addresses: allow but prescribe split-horizon fashion when used, disallow zone identifiers
- * Remove informative references to documents not mentioned any more

changes from -17 to -18

- * Rather than re-specifying link format (Modernized Link Format), describe a Limited Link Format that's the uncontested subset of Link Format
- * Acknowledging the -17 version as part of the draft
- * Move "Read endpoint links" operation to future specification like PATCH
- * Demote links-json to an informative reference, and removed them from exchange examples
- * Add note on unusability of link-local IP addresses, and describe mitigation.

- * Reshuffling of sections: Move additional operations and endpoint lookup back from appendix, and groups into one
- * Lookup interface tightened to not imply applicability for non link-format lookups (as those can have vastly different views on link cardinality)
- * Simple registration: Change sequence of GET and POST-response, ensuring unsuccessful registrations are reported as such, and suggest how devices that would have required the inverse behavior can still cope with it.
- * Abstract and introduction reworded to avoid the impression that resources are stored in full in the RD
- * Simplify the rules governing when a registration resource can or must be changed.
- * Drop a figure that has become useless due to the changes of and -13 and -17
- * Wording consistency fixes: Use "Registrations" and "target attributes"
- * Fix incorrect use of content negotiation in discovery interface description (Content-Format -> Accept)
- * State that the base attribute value is part of endpoint lookup even when implicit in the registration
- * Update references from RFC5988 to its update RFC8288
- * Remove appendix on protocol-negotiation (which had a note to be removed before publication)

changes from -16 to -17

(Note that -17 is published as a direct follow-up to -16, containing a single change to be discussed at IETF103)

- * Removed groups that are enumerations of registrations and have dedicated mechanism
- * Add groups that are enumerations of shared resources and are a special case of endpoint registrations

changes from -15 to -16

- * Recommend a common set of resources for members of a group
- * Clarified use of multicast group in lighting example
- * Add note on concurrent registrations from one EP being possible but not expected
- * Refresh web examples appendix to reflect current use of Modernized Link Format
- * Add examples of URIs where Modernized Link Format matters
- * Editorial changes

changes from -14 to -15

- * Rewrite of section "Security policies"
- * Clarify that the "base" parameter text applies both to relative references both in anchor and href
- * Renamed "Registree-EP" to Registrant-EP"
- * Talk of "relative references" and "URIs" rather than "relative" and "absolute" URIs. (The concept of "absolute URIs" of [RFC3986] is not needed in RD).
- * Fixed examples
- * Editorial changes

changes from -13 to -14

- * Rename "registration context" to "registration base URI" (and "con" to "base") and "domain" to "sector" (where the abbreviation "d" stays for compatibility reasons)
- * Introduced resource types core.rd-ep and core.rd-gp
- * Registration management moved to appendix A, including endpoint and group lookup
- * Minor editorial changes
 - PATCH/iPATCH is clearly deferred to another document
 - Recommend against query / fragment identifier in con=

- Interface description lists are described as illustrative
- Rewording of Simple Registration
- * Simple registration carries no error information and succeeds immediately (previously, sequence was unspecified)
- * Lookup: href are matched against resolved values (previously, this was unspecified)
- * Lookup: lt are not exposed any more
- * con/base: Paths are allowed
- * Registration resource locations can not have query or fragment parts
- * Default life time extended to 25 hours
- * clarified registration update rules
- * lt-value semantics for lookup clarified.
- * added template for simple registration

changes from -12 to -13

- * Added "all resource directory" nodes MC address
- * Clarified observation behavior
- * version identification
- * example rt= and et= values
- * domain from figure 2
- * more explanatory text
- * endpoints of a groups hosted by different RD
- * resolve RFC6690-vs-8288 resolution ambiguities:
 - require registered links not to be relative when using anchor
 - return absolute URIs in resource lookup

changes from -11 to -12

- * added Content Model section, including ER diagram
- * removed domain lookup interface; domains are now plain attributes of groups and endpoints
- * updated chapter "Finding a Resource Directory"; now distinguishes configuration-provided, network-provided and heuristic sources
- * improved text on: atomicity, idempotency, lookup with multiple parameters, endpoint removal, simple registration
- * updated LWM2M description
- * clarified where relative references are resolved, and how context and anchor interact
- * new appendix on the interaction with RFCs 6690, 5988 and 3986
- * lookup interface: group and endpoint lookup return group and registration resources as link targets
- * lookup interface: search parameters work the same across all entities
- * removed all methods that modify links in an existing registration (POST with payload, PATCH and iPATCH)
- * removed plurality definition (was only needed for link modification)
- * enhanced IANA registry text
- * state that lookup resources can be observable
- * More examples and improved text

changes from -09 to -10

- * removed "ins" and "exp" link-format extensions.
- * removed all text concerning DNS-SD.
- * removed inconsistency in RDAO text.
- * suggestions taken over from various sources
- * replaced "Function Set" with "REST API", "base URI", "base path"

- * moved simple registration to registration section

changes from -08 to -09

- * clarified the "example use" of the base RD resource values /rd, /rd-lookup, and /rd-group.
- * changed "ins" ABNF notation.
- * various editorial improvements, including in examples
- * clarifications for RDAO

changes from -07 to -08

- * removed link target value returned from domain and group lookup types
- * Maximum length of domain parameter 63 bytes for consistency with group
- * removed option for simple POST of link data, don't require a .well-known/core resource to accept POST data and handle it in a special way; we already have /rd for that
- * add IPv6 ND Option for discovery of an RD
- * clarify group configuration section 6.1 that endpoints must be registered before including them in a group
- * removed all superfluous client-server diagrams
- * simplified lighting example
- * introduced Commissioning Tool
- * RD-Look-up text is extended.

changes from -06 to -07

- * added text in the discovery section to allow content format hints to be exposed in the discovery link attributes
- * editorial updates to section 9
- * update author information
- * minor text corrections

Changes from -05 to -06

- * added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

- * added Update Endpoint Links using PATCH
- * http access made explicit in interface specification
- * Added http examples

Changes from -03 to -04:

- * Added http response codes
- * Clarified endpoint name usage
- * Add application/link-format+cbor content-format

Changes from -02 to -03:

- * Added an example for lighting and DNS integration
- * Added an example for RD use in OMA LWM2M
- * Added Read Links operation for link inspection by endpoints
- * Expanded DNS-SD section
- * Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

- * Added a catalogue use case.
- * Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.
- * Additional examples section added for more complex use cases.
- * New DNS-SD mapping section.
- * Added text on endpoint identification and authentication.
- * Error code 4.04 added to Registration Update and Delete requests.

- * Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

- * Removed the ETag validation feature.
- * Place holder for the DNS-SD mapping section.
- * Explicitly disabled GET or POST on returned Location.
- * New registry for RD parameters.
- * Added support for the JSON Link Format.
- * Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

- * Updated the version and date.

Changes from -04 to -05:

- * Restricted Update to parameter updates.
- * Added pagination support for the Lookup interface.
- * Minor editing, bug fixes and reference updates.
- * Added group support.
- * Changed rt to et for the registration and update interface.

Changes from -03 to -04:

- * Added the ins= parameter back for the DNS-SD mapping.
- * Integrated the Simple Directory Discovery from Carsten.
- * Editorial improvements.
- * Fixed the use of ETags.
- * Fixed tickets 383 and 372

Changes from -02 to -03:

- * Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- * Updated REST interface descriptions to use RFC6570 URI Template format.
- * Introduced an improved RD Lookup design as its own function set.
- * Improved the security considerations section.
- * Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:

- * Added a terminology section.
- * Changed the inclusion of an ETag in registration or update to a MAY.
- * Added the concept of an RD Domain and a registration parameter for it.
- * Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- * Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

13. References

13.1. Normative References

- [I-D.ietf-core-echo-request-tag]
Amsüss, C., Mattsson, J. P., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", Work in Progress, Internet-Draft, draft-ietf-core-echo-request-tag-12, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-echo-request-tag-12.txt>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

13.2. Informative References

- [ER] Chen, P., "The entity-relationship model--toward a unified view of data", DOI 10.1145/320434.320440, ACM Transactions on Database Systems Vol. 1, pp. 9-36, March 1976, <<https://doi.org/10.1145/320434.320440>>.

[I-D.bormann-t2trg-rel-impl]

Bormann, C., "impl-info: A link relation type for disclosing implementation information", Work in Progress, Internet-Draft, draft-bormann-t2trg-rel-impl-02, 27 September 2020, <<https://www.ietf.org/archive/id/draft-bormann-t2trg-rel-impl-02.txt>>.

[I-D.hartke-t2trg-coral]

Hartke, K., "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-hartke-t2trg-coral-09, 8 July 2019, <<https://www.ietf.org/archive/id/draft-hartke-t2trg-coral-09.txt>>.

[I-D.ietf-ace-oauth-authz]

Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-37, 4 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-37.txt>>.

[I-D.ietf-core-links-json]

LI, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", Work in Progress, Internet-Draft, draft-ietf-core-links-json-10, 26 February 2018, <<https://www.ietf.org/archive/id/draft-ietf-core-links-json-10.txt>>.

[I-D.ietf-core-rd-dns-sd]

Stok, P. V. D., Koster, M., and C. Amsüss, "CoRE Resource Directory: DNS-SD mapping", Work in Progress, Internet-Draft, draft-ietf-core-rd-dns-sd-05, 7 July 2019, <<https://www.ietf.org/archive/id/draft-ietf-core-rd-dns-sd-05.txt>>.

[I-D.silverajan-core-coap-protocol-negotiation]

Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", Work in Progress, Internet-Draft, draft-silverajan-core-coap-protocol-negotiation-09, 2 July 2018, <<https://www.ietf.org/archive/id/draft-silverajan-core-coap-protocol-negotiation-09.txt>>.

[LwM2M]

Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification: Transport Bindings (Candidate Version 1.1)", 12 June 2018,

- https://openmobilealliance.org/RELEASE/LightweightM2M/V1_1-20180612-C/OMA-TS-LightweightM2M_Transport-V1_1-20180612-C.pdf.
- [RFC3306] Haberman, B. and D. Thaler, "Unicast-Prefix-based IPv6 Multicast Addresses", RFC 3306, DOI 10.17487/RFC3306, August 2002, <https://www.rfc-editor.org/info/rfc3306>.
- [RFC3849] Huston, G., Lord, A., and P. Smith, "IPv6 Address Prefix Reserved for Documentation", RFC 3849, DOI 10.17487/RFC3849, July 2004, <https://www.rfc-editor.org/info/rfc3849>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.
- [RFC5771] Cotton, M., Vegoda, L., and D. Meyer, "IANA Guidelines for IPv4 Multicast Address Assignments", BCP 51, RFC 5771, DOI 10.17487/RFC5771, March 2010, <https://www.rfc-editor.org/info/rfc5771>.
- [RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/info/rfc6724>.
- [RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <https://www.rfc-editor.org/info/rfc6775>.
- [RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <https://www.rfc-editor.org/info/rfc6874>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 8106, DOI 10.17487/RFC8106, March 2017, <<https://www.rfc-editor.org/info/rfc8106>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <<https://www.rfc-editor.org/info/rfc8141>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

Appendix A. Groups Registration and Lookup

The RD-Groups usage pattern allows announcing application groups inside an RD.

Groups are represented by endpoint registrations. Their base address is a multicast address, and they SHOULD be entered with the endpoint type "core.rd-group". The endpoint name can also be referred to as a group name in this context.

The registration is inserted into the RD by a Commissioning Tool, which might also be known as a group manager here. It performs third party registration and registration updates.

The links it registers SHOULD be available on all members that join the group. Depending on the application, members that lack some resource MAY be permissible if requests to them fail gracefully.

The following example shows a CT registering a group with the name "lights" which provides two resources. The directory resource path /rd is an example RD location discovered in a request similar to Figure 5. The group address in the example is constructed from [RFC3849]'s reserved 2001:db8:: prefix as a unicast-prefix based site-local address (see [RFC3306]).

```
Req: POST coap://rd.example.com/rd?ep=lights&et=core.rd-group
      &base=coap://[ff35:30:2001:db8:f1::8000:1]
Content-Format: 40
Payload:
</light>;rt="tag:example.com,2020:light";
      if="tag:example.net,2020:actuator",
</color-temperature>;if="tag:example.net,2020:parameter";u=K

Res: 2.01 Created
Location-Path: /rd/12
```

Figure 27: Example registration of a group

In this example, the group manager can easily permit devices that have no writable color-temperature to join, as they would still respond to brightness changing commands. Had the group instead contained a single resource that sets brightness and color temperature atomically, endpoints would need to support both properties.

The resources of a group can be looked up like any other resource, and the group registrations (along with any additional registration parameters) can be looked up using the endpoint lookup interface.

The following example shows a client performing an endpoint lookup for all groups.

```
Req: GET /rd-lookup/ep?et=core.rd-group

Res: 2.05 Content
Payload:
</rd/12>;ep=lights&et=core.rd-group;
      base="coap://[ff35:30:2001:f1:db8::8000:1]";rt=core.rd-ep
```

Figure 28: Example lookup of groups

The following example shows a client performing a lookup of all resources of all endpoints (groups) with et=core.rd-group.

```
Req: GET /rd-lookup/res?et=core.rd-group

Res: 2.05 Content
Payload:
<coap://[ff35:30:2001:db8:f1::8000:1]/light>;
      rt="tag:example.com,2020:light";
      if="tag:example.net,2020:actuator",
<coap://[ff35:30:2001:db8:f1::8000:1]/color-temperature>;
      if="tag:example.net,2020:parameter";u=K,
```

Figure 29: Example lookup of resources inside groups

Appendix B. Web links and the Resource Directory

Understanding the semantics of a link-format document and its URI references is a journey through different documents ([RFC3986] defining URIs, [RFC6690] defining link-format documents based on [RFC8288] which defines Link header fields, and [RFC7252] providing the transport). This appendix summarizes the mechanisms and semantics at play from an entry in `"/.well-known/core"` to a resource lookup.

This text is primarily aimed at people entering the field of Constrained Restful Environments from applications that previously did not use web mechanisms.

B.1. A simple example

Let's start this example with a very simple host, `"2001:db8:f0::1"`. A client that follows classical CoAP Discovery ([RFC7252] Section 7), sends the following multicast request to learn about neighbours supporting resources with resource-type `"temperature"`.

The client sends a link-local multicast:

```
Req: GET coap://[ff02::fd]:5683/.well-known/core?rt=temperature
Res: 2.05 Content
Payload:
</sensors/temp>;rt=temperature;ct=0
```

Figure 30: Example of direct resource discovery

where the response is sent by the server, `"[2001:db8:f0::1]:5683"`.

While the client -- on the practical or implementation side -- can just go ahead and create a new request to `"[2001:db8:f0::1]:5683"` with Uri-Path: `"sensors"` and `"temp"`, the full resolution steps for insertion into and retrieval from the RD without any shortcuts are:

B.1.1. Resolving the URIs

The client parses the single returned record. The link's target (sometimes called `"href"`) is `"/sensors/temp"`, which is a relative URI that needs resolving. The base URI `<coap://[ff02::fd]:5683/.well-known/core>` is used to resolve the reference `/sensors/temp` against.

The Base URI of the requested resource can be composed from the options of the CoAP GET request by following the steps of [RFC7252] section 6.5 (with an addition at the end of 8.2) into `"coap://[2001:db8:f0::1]/.well-known/core"`.

Because `"/sensors/temp"` starts with a single slash, the record's target is resolved by replacing the path `"/.well-known/core"` from the Base URI (section 5.2 [RFC3986]) with the relative target URI `"/sensors/temp"` into `"coap://[2001:db8:f0::1]/sensors/temp"`.

B.1.2. Interpreting attributes and relations

Some more information but the record's target can be obtained from the payload: the resource type of the target is "temperature", and its content format is text/plain (ct=0).

A relation in a web link is a three-part statement that specifies a named relation between the so-called "context resource" and the target resource, like `"_This page_ has _its table of contents_ at _/toc.html_"`. In link format documents, there is an implicit "host relation" specified with default parameter: `rel="hosts"`.

In our example, the context resource of the link is implied to be `"coap://[2001:db8:f0::1]"` by the default value of the anchor (see Appendix B.4). A full English expression of the "host relation" is:

`'"coap://[2001:db8:f0::1]" is hosting the resource "coap://[2001:db8:f0::1]/sensors/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'`

B.2. A slightly more complex example

Omitting the `"rt=temperature"` filter, the discovery query would have given some more records in the payload:

Req: GET `coap://[ff02::fd]:5683/.well-known/core`

Res: 2.05 Content

Payload:

```
</sensors/temp>;rt=temperature;ct=0,
</sensors/light>;rt=light-lux;ct=0,
</t>;anchor="/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;anchor="/sensors/temp";
rel=describedby
```

Figure 31: Extended example of direct resource discovery

Parsing the third record, the client encounters the "anchor" parameter. It is a URI relative to the Base URI of the request and is thus resolved to `"coap://[2001:db8:f0::1]/sensors/temp"`. That is the context resource of the link, so the "rel" statement is not about the target and the Base URI any more, but about the target and the resolved URI. Thus, the third record could be read as `"coap://[2001:db8:f0::1]/sensors/temp"` has an alternate representation at `"coap://[2001:db8:f0::1]/t"`.

Following the same resolution steps, the fourth record can be read as `"coap://[2001:db8:f0::1]/sensors/temp"` is described by `"http://www.example.com/sensors/t123"`.

B.3. Enter the Resource Directory

The RD tries to carry the semantics obtainable by classical CoAP discovery over to the resource lookup interface as faithfully as possible.

For the following queries, we will assume that the simple host has used Simple Registration to register at the RD that was announced to it, sending this request from its UDP port `"[2001:db8:f0::1]:6553"`:

Req: POST `coap://[2001:db8:f01::ff]/.well-known/rd?ep=simple-host1`

Res: 2.04 Changed

Figure 32: Example of a simple registration

The RD would have accepted the registration, and queried the simple host's `"/.well-known/core"` by itself. As a result, the host is registered as an endpoint in the RD with the name `"simple-host1"`. The registration is active for 90000 seconds, and the endpoint registration Base URI is `"coap://[2001:db8:f0::1]"` following the resolution steps described in Appendix B.1.1. It should be remarked that the Base URI constructed that way always yields a URI of the form: `scheme://authority without path suffix`.

If the client now queries the RD as it would previously have issued a multicast request, it would go through the RD discovery steps by fetching `"coap://[2001:db8:f0::ff]/.well-known/core?rt=core.rd-lookup-res"`, obtain `"coap://[2001:db8:f0::ff]/rd-lookup/res"` as the resource lookup endpoint, and ask it for all temperature resources:

Req: GET coap://[2001:db8:f0::ff]/rd-lookup/res?rt=temperature

Res: 2.05 Content

Payload:

<coap://[2001:db8:f0::1]/sensors/temp>;rt=temperature;ct=0

Figure 33: Example exchange performing resource lookup

This is not literally the same response that it would have received from a multicast request, but it contains the equivalent statement:

'"coap://[2001:db8:f0::1]" is hosting the resource "coap://[2001:db8:f0::1]/sensors/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'

To complete the examples, the client could also query all resources hosted at the endpoint with the known endpoint name "simple-host1":

Req: GET coap://[2001:db8:f0::ff]/rd-lookup/res?ep=simple-host1

Res: 2.05 Content

Payload:

<coap://[2001:db8:f0::1]/sensors/temp>;rt=temperature;ct=0,
 <coap://[2001:db8:f0::1]/sensors/light>;rt=light-lux;ct=0,
 <coap://[2001:db8:f0::1]/t>;
 anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=alternate,
 <http://www.example.com/sensors/t123>;
 anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=describedby

Figure 34: Extended example exchange performing resource lookup

All the target and anchor references are already in absolute form there, which don't need to be resolved any further.

Had the simple host done an equivalent full registration with a base= parameter (e.g. "?ep=simple-host1&base=coap+tcp://simple-host1.example.com"), that context would have been used to resolve the relative anchor values instead, giving

<coap+tcp://simple-host1.example.com/sensors/temp>;rt=temperature;ct=0

Figure 35: Example payload of a response to a resource lookup with a dedicated base URI

and analogous records.

B.4. A note on differences between link-format and Link header fields

While link-format and Link header fields look very similar and are based on the same model of typed links, there are some differences between [RFC6690] and [RFC8288]. When implementing an RD or interacting with an RD, care must be taken to follow the [RFC6690] behavior whenever application/link-format representations are used.

- * "Default value of anchor": Both under [RFC6690] and [RFC8288], relative references in the term inside the angle brackets (the target) and the anchor attribute are resolved against the relevant base URI (which usually is the URI used to retrieve the entity), and independent of each other.

When, in an [RFC8288] Link header, the anchor attribute is absent, the link's context is the URI of the selected representation (and usually equal to the base URI).

In [RFC6690] links, if the anchor attribute is absent, the default value is the Origin of (for all relevant cases: the URI reference "/" resolved against) the link's target.

- * There is no percent encoding in link-format documents.

A link-format document is a UTF-8 encoded string of Unicode characters and does not have percent encoding, while Link header fields are practically ASCII strings that use percent encoding for non-ASCII characters, stating the encoding explicitly when required.

For example, while a Link header field in a page about a Swedish city might read

```
Link: </temperature/Malm%C3%B6>;rel=live-environment-data
```

a link-format document from the same source might describe the link as

```
</temperature/Malmö>;rel=live-environment-data
```

Appendix C. Limited Link Format

The CoRE Link Format as described in [RFC6690] has been interpreted differently by implementers, and a strict implementation rules out some use cases of an RD (e.g. base values with path components in combination with absent anchors).

This appendix describes a subset of link format documents called Limited Link Format. The one rule herein is not very limiting in practice -- all examples in RFC6690, and all deployments the authors are aware of already stick to them -- but ease the implementation of RD servers.

It is applicable to representations in the application/link-format media type, and any other media types that inherit [RFC6690] Section 2.1.

A link format representation is in Limited Link format if, for each link in it, the following applies:

All URI references either follow the URI or the path-absolute ABNF rule of RFC3986 (i.e. target and anchor each either start with a scheme or with a single slash).

Authors' Addresses

Christian Amsüss (editor)
Hollandstr. 12/4
1020
Austria

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

Zach Shelby
ARM
150 Rose Orchard
San Jose, 95134
United States of America

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

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View, 94043
United States of America

Phone: +1-707-502-5136
Email: Michael.Koster@smarththings.com

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

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

Peter van der Stok
consultant

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 14 January 2021

A. Keränen
Ericsson
C. Bormann
Universität Bremen TZI
13 July 2020

SenML Data Value Content-Format Indication
draft-ietf-core-senml-data-ct-02

Abstract

The Sensor Measurement Lists (SenML) media type supports multiple types of values, from numbers to text strings and arbitrary binary data values. In order to simplify processing of the data values this document proposes to specify a new SenML field for indicating the Content-Format of the data.

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 14 January 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Terminology	3
3. SenML Content-Format ("ct") Field	3
4. SenML Base Content-Format ("bct") Field	4
5. Examples	4
6. Security Considerations	4
7. IANA Considerations	5
8. References	5
8.1. Normative References	5
8.2. Informative References	6
Acknowledgements	6
Authors' Addresses	6

1. Introduction

The Sensor Measurement Lists (SenML) media type [RFC8428] can be used to send various different kinds of data. In the example given in Figure 1, a temperature value, an indication whether a lock is open, and a data value (with SenML field "vd") read from an NFC reader is sent in a single SenML pack.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:", "n":"temp", "u":"Cel", "v":7.1},
  {"n":"open", "vb":false},
  {"n":"nfc-reader", "vd":"aGkgCg"}
]
```

Figure 1: SenML pack with unidentified binary data

The receiver is expected to know how to interpret the data in the "vd" field based on the context, e.g., name of the data source and out-of-band knowledge of the application. However, this context may not always be easily available to entities processing the SenML pack. To facilitate automatic interpretation it is useful to be able to indicate an Internet media type and content-coding right in the SenML Record. The CoAP Content-Format (Section 12.3 in [RFC7252]) provides just this information; enclosing a Content-Format number (in this case number 60 as defined for content-type application/cbor in [RFC7049]) in the Record is illustrated in Figure 2. All registered CoAP Content-Formats are listed in the Content-Formats subregistry of the CoRE Parameters registry [IANA.core-parameters].

```
{"n":"nfc-reader", "vd":"gmNmb28YKg", "ct":"60"}
```

Figure 2: SenML Record with binary data identified as CBOR

In this example SenML Record the data value contains a string "foo" and a number 42 encoded in a CBOR [RFC7049] array. Since the example above uses the JSON format of SenML, the data value containing the binary CBOR value is base64-encoded. The data value after base64 decoding is shown with CBOR diagnostic notation in Figure 3.

```
82          # array(2)
  63        # text(3)
    666F6F  # "foo"
  18 2A     # unsigned(42)
```

Figure 3: Example Data Value in CBOR diagnostic notation

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.

Readers should also be familiar with the terms and concepts discussed in [RFC8428]. Awareness of terminology issues discussed in [I-D.bormann-core-media-content-type-format] can also be very helpful.

3. SenML Content-Format ("ct") Field

When a SenML Record contains a Data Value field ("vd"), the Record MAY also include a Content-Format indication field. The Content-Format indication uses label "ct" and a string value with either a CoAP Content-Format identifier in decimal form with no leading zeros except for the value "0" itself (representing an unsigned integer in the range of 0-65535, similar to the CoRE Link Format [RFC6690] "ct" attribute) or with a string containing a Content-Type and optionally a Content-Coding (see below).

The CoAP Content-Format identifier provides a simple and efficient way to indicate the type of the data. Since some Internet media types and their content coding and parameter alternatives do not have assigned CoAP Content-Format identifiers, using Content-Type and Content-Coding is also allowed. Both methods use a string value in the "ct" field to keep its data type consistent across uses. When the "ct" field contains only digits, it is interpreted as a CoAP Content-Format identifier.

To indicate that a Content-Coding is used with a Content-Type, the Content-Coding value (e.g., "deflate" [RFC7230]) is appended to the Content Type (media type and parameters, if any), separated by a "@" sign. For example: "text/plain; charset=utf-8@deflate". If Content-Coding is not specified with a Content-Type (no "@" sign is present outside any media type parameters), the identity (i.e., no) transformation is used.

4. SenML Base Content-Format ("bct") Field

The Base Content-Format Field, label "bct", provides a default value for the Content-Format Field (label "ct") within its range. The range of the base field includes the Record containing it, up to (but not including) the next Record containing a "bct" field, if any, or up to the end of the pack otherwise. Resolution (Section 4.6 of [RFC8428]) of this base field is performed by adding its value with the label "ct" to all Records in this range that carry a "vd" field but do not already contain a Content-Format ("ct") field.

5. Examples

The following examples are valid values for the "ct" and "bct" fields (explanation/comments in parenthesis):

- * "60" (CoAP Content-Format for "application/cbor")
- * "0" (CoAP Content-Format for "text/plain" with parameter "charset=utf-8")
- * "application/json" (JSON Content-Type - equivalent to "50" CoAP Content-Format identifier)
- * "application/json@deflate" (JSON Content-Type with "deflate" as Content-Coding - equivalent to "11050" CoAP Content-Format identifier)
- * "text/csv" (Comma-Separated Values (CSV) [RFC4180] Content-Type)
- * "text/csv@gzip" (CSV with "gzip" as Content-Coding)

6. Security Considerations

The indication of a media type in the data does not exempt a consuming application from properly checking its inputs. Also, the ability for an attacker to supply crafted SenML data that specify media types chosen by the attacker may expose vulnerabilities of handlers for these media types to the attacker.

7. IANA Considerations

(Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification and remove this note.)

IANA is requested to assign new labels in the "SenML Labels" subregistry of the SenML registry [IANA.senml] (as defined in [RFC8428]) for the Content-Format indication as per Table 1:

Name	Label	JSON Type	XML Type	Reference
Base Content-Format	bct	String	string	RFC-AAAA
Content-Format	ct	String	string	RFC-AAAA

Table 1: IANA Registration for new SenML Labels

8. References

8.1. Normative References

- [IANA.senml] IANA, "Sensor Measurement Lists (SenML)", <<http://www.iana.org/assignments/senml>>.
- [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>>.
- [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>>.
- [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>>.

- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

8.2. Informative References

- [I-D.bormann-core-media-content-type-format] Bormann, C., "On Media-Types, Content-Types, and related terminology", Work in Progress, Internet-Draft, draft-bormann-core-media-content-type-format-01, 8 July 2019, <<http://www.ietf.org/internet-drafts/draft-bormann-core-media-content-type-format-01.txt>>.
- [IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <<http://www.iana.org/assignments/core-parameters>>.
- [RFC4180] Shafranovich, Y., "Common Format and MIME Type for Comma-Separated Values (CSV) Files", RFC 4180, DOI 10.17487/RFC4180, October 2005, <<https://www.rfc-editor.org/info/rfc4180>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.

Acknowledgements

The authors would like to thank Sergio Abreu for the discussions leading to the design of this extension and Isaac Rivera for reviews and feedback. Klaus Hartke suggested not burdening this draft with a separate mandatory-to-implement version of the fields.

Authors' Addresses

Ari Keränen
Ericsson
FI-02420 Jorvas
Finland

Email: ari.keranen@ericsson.com

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

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 24 April 2022

A. Keränen
Ericsson
C. Bormann
Universität Bremen TZI
21 October 2021

SenML Data Value Content-Format Indication
draft-ietf-core-senml-data-ct-07

Abstract

The Sensor Measurement Lists (SenML) media type supports multiple types of values, from numbers to text strings and arbitrary binary data values. In order to facilitate processing of binary data values, this document specifies a pair of new SenML fields for indicating the content format of those binary data values, i.e., their Internet media type including parameters as well as any content codings applied.

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 24 April 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Evolution	3
2. Terminology	4
3. SenML Content-Format ("ct") Field	5
4. SenML Base Content-Format ("bct") Field	6
5. Examples	6
6. ABNF	7
7. Security Considerations	8
8. IANA Considerations	8
9. References	9
9.1. Normative References	9
9.2. Informative References	10
Acknowledgements	11
Authors' Addresses	11

1. Introduction

The Sensor Measurement Lists (SenML) media types [RFC8428] can be used to send various kinds of data. In the example given in Figure 1, a temperature value, an indication whether a lock is open, and a data value (with SenML field "vd") read from an NFC reader is sent in a single SenML pack. The example is given in SenML JSON representation, so the "vd" (data value) field is encoded as a base64url string (without padding), as per Section 5 of [RFC8428].

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:", "n":"temp", "u":"Cel", "v":7.1},
  {"n":"open", "vb":false},
  {"n":"nfc-reader", "vd":"aGkgCg"}
]
```

Figure 1: SenML pack with unidentified binary data

The receiver is expected to know how to interpret the data in the "vd" field based on the context, e.g., name of the data source and out-of-band knowledge of the application. However, this context may not always be easily available to entities processing the SenML pack,

especially if the pack is propagated over time and via multiple entities. To facilitate automatic interpretation it is useful to be able to indicate an Internet media type and, optionally, content codings right in the SenML Record.

The CoAP Content-Format (Section 12.3 of [RFC7252]) provides this information in the form of a single unsigned integer; enclosing a Content-Format number (in this case number 60 as defined for content-type application/cbor in [RFC8949]) in the Record is illustrated in Figure 2. All registered CoAP Content-Format numbers are listed in the COAP Content-Formats registry [IANA.core-parameters] as specified by Section 12.3 of [RFC7252]. Note that, at the time of writing, the structure of this registry only provides for zero or one content codings; nothing in the present document needs to change if the registry is extended to allow sequences of content codings.

```
{"n":"nfc-reader", "vd":"gmNmb28YKg", "ct":"60"}
```

Figure 2: SenML Record with binary data identified as CBOR

In this example SenML Record, the data value contains a string "foo" and a number 42 encoded in a CBOR [RFC8949] array. Since the example above uses the JSON format of SenML, the data value containing the binary CBOR value is base64-encoded (Section 5 of [RFC4648]). The data value after base64 decoding is shown with CBOR diagnostic notation in Figure 3.

```
82          # array(2)
 63          # text(3)
  666F66F   # "foo"
 18 2A      # unsigned(42)
```

Figure 3: Example Data Value in CBOR diagnostic notation

1.1. Evolution

As with SenML in general, there is no expectation that the creator of a SenML pack knows (or has negotiated with) each consumer of that pack, which may be very remote in space and particularly in time. This means that the SenML creator in general has no way to know whether the consumer knows:

- * each specific media-type-name used
- * each parameter and each parameter value used
- * each content coding in use

* each Content-Format number in use for a combination of these

What SenML, as well as the new fields defined here, guarantees is that a recipient implementation knows when it needs to be updated to understand these field values and the values controlled by them; registries are used to evolve these name spaces in a controlled way. SenML packs can be processed by a consumer while not understanding all the information in them, and information can generally be preserved in this processing such that it is useful for further consumers.

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.

Media Type: A registered label for representations (byte strings) prepared for interchange, identified by a Media-Type-Name [RFC1590], [RFC6838].

Media-Type-Name: A combination of a type-name and a subtype-name registered in [IANA.media-types] as per [RFC6838], conventionally identified by the two names separated by a slash.

Content-Type: A Media-Type-Name, optionally associated with parameters (Section 5 of [RFC2045], separated from the Media-Type-Name and from each other by a semicolon). In HTTP and many other protocols, used in a Content-Type header field.

content coding: A name registered in the HTTP Content Coding registry [IANA.http-parameters] as specified by Sections 16.6.1 and 18.6 of [I-D.ietf-httpbis-semantics], indicating an encoding transformation with semantics further specified in Section 8.4.1 of [I-D.ietf-httpbis-semantics]. Confusingly, in HTTP, content coding values are found in a header field called "Content-Encoding", however "content coding" is the correct term for the process and the registered values.

content format: the combination of a Content-Type and zero or more content codings, identified by (1) a numeric identifier defined in the COAP Content-Formats registry [IANA.core-parameters] as per Section 12.3 of [RFC7252] (referred to as Content-Format number), or (2) a Content-Format-String.

Content-Format-String: the string representation of the combination

of a Content-Type and zero or more content codings.

Content-Format-Spec: the string representation of a content format; either a Content-Format-String or the (decimal) string representation of a Content-Format number.

Readers should also be familiar with the terms and concepts discussed in [RFC8428].

3. SenML Content-Format ("ct") Field

When a SenML Record contains a Data Value field ("vd"), the Record MAY also include a Content-Format indication field, using label "ct". The value of this field is a Content-Format-Spec, i.e., one of:

- * a CoAP Content-Format number in decimal form with no leading zeros (except for the value "0" itself). This value represents an unsigned integer in the range of 0-65535, similar to the "ct" attribute defined in Section 7.2.1 of [RFC7252] for CoRE Link Format [RFC6690]).
- * or a Content-Format-String containing a Content-Type and zero or more content codings (see below).

The syntax of this field is formally defined in Section 6.

The CoAP Content-Format number provides a simple and efficient way to indicate the type of the data. Since some Internet media types and their content coding and parameter alternatives do not have assigned CoAP Content-Format numbers, using Content-Type and zero or more content codings is also allowed. Both methods use a string value in the "ct" field to keep its data type consistent across uses. When the "ct" field contains only digits, it is interpreted as a CoAP Content-Format number.

To indicate that one or more content codings are used with a Content-Type, each of the content coding values is appended to the Content-Type value (media type and parameters, if any), separated by a "@" sign, in the order of the content codings were applied (the same order as in Section 8.4 of [I-D.ietf-httpbis-semantics]). For example (using a content coding value of "deflate" as defined in Section 8.4.1.2 of [I-D.ietf-httpbis-semantics]):

```
text/plain; charset=utf-8@deflate
```

If no "@" sign is present after the media type and parameters, then no content coding has been specified, and the "identity" content coding is used -- no encoding transformation is employed.

4. SenML Base Content-Format ("bct") Field

The Base Content-Format Field, label "bct", provides a default value for the Content-Format Field (label "ct") within its range. The range of the base field includes the Record containing it, up to (but not including) the next Record containing a "bct" field, if any, or up to the end of the pack otherwise. The process of resolving (Section 4.6 of [RFC8428]) this base field is performed by adding its value with the label "ct" to all Records in this range that carry a "vd" field but do not already contain a Content-Format ("ct") field.

Figure 4 shows a variation of Figure 2 with multiple records, with the "nfc-reader" records resolving to the base field value "60" and the "iris-photo" record overriding this with the "image/png" media type (actual data left out for brevity).

```
[
  { "n": "nfc-reader", "vd": "gmNmb28YKg",
    "bct": "60", "bt": 1627430700 },
  { "n": "nfc-reader", "vd": "gmNiYXIYKw", "t": 10 },
  { "n": "iris-photo", "vd": ".....", "ct": "image/png", "t": 10 },
  { "n": "nfc-reader", "vd": "gmNiYXoYLA", "t": 20 }
]
```

Figure 4: SenML pack with bct field

5. Examples

The following examples are valid values for the "ct" and "bct" fields (explanation/comments in parentheses):

- * "60" (CoAP Content-Format number for "application/cbor")
- * "0" (CoAP Content-Format number for "text/plain" with parameter "charset=utf-8")
- * "application/json" (JSON Content-Type -- equivalent to "50" CoAP Content-Format number)
- * "application/json@deflate" (JSON Content-Type with "deflate" as content coding -- equivalent to "11050" CoAP Content-Format number)
- * "application/json@deflate@aes128gcm" (JSON Content-Type with "deflate" followed by "aes128gcm" as content codings)
- * "text/csv" (Comma-Separated Values (CSV) [RFC4180] Content-Type)

* "text/csv;header=present@gzip" (CSV with header row, using "gzip" as content coding)

6. ABNF

This specification provides a formal definition of the syntax of Content-Format-Spec strings using ABNF notation [RFC5234], which contains three new rules and a number of rules collected and adapted from various RFCs [I-D.ietf-httpbis-semantics] [RFC6838] [RFC5234] [RFC8866].

; New in this document

Content-Format-Spec = Content-Format-Number / Content-Format-String

Content-Format-Number = "0" / (POS-DIGIT *DIGIT)

Content-Format-String = Content-Type *("@" Content-Coding)

; Cleaned up from [RFC-httpbis-semantics],

; leaving only SP as blank space,

; removing legacy 8-bit characters, and

; leaving the parameter as mandatory with each semicolon:

Content-Type = Media-Type-Name *(*SP ";" *SP parameter)

parameter = token "=" (token / quoted-string)

token = 1*tchar

tchar = "!" / "#" / "\$" / "%" / "&" / "'" / "*" /
/ "+" / "-" / "." / "^" / "_" / "`" / "|" / "~"
/ DIGIT / ALPHA

quoted-string = %x22 *(qdtext / quoted-pair) %x22

qdtext = SP / %x21 / %x23-5B / %x5D-7E

quoted-pair = "\" (SP / VCHAR)

; Adapted from section 8.4.1 of [RFC-httpbis-semantics]

Content-Coding = token

; Adapted from various specs

Media-Type-Name = type-name "/" subtype-name

; RFC 6838

type-name = restricted-name

subtype-name = restricted-name

restricted-name = restricted-name-first *126restricted-name-chars

```

restricted-name-first = ALPHA / DIGIT
restricted-name-chars = ALPHA / DIGIT / "!" / "#" /
                        "$" / "&" / "-" / "^" / "_"
restricted-name-chars =/ "." ; Characters before first dot always
                           ; specify a facet name
restricted-name-chars =/ "+" ; Characters after last plus always
                           ; specify a structured syntax suffix

; Boilerplate from RFC 5234 and RFC 8866

DIGIT      = %x30-39          ; 0 9
POS-DIGIT  = %x31-39          ; 1 9
ALPHA      = %x41-5A / %x61-7A ; A Z / a z
SP         = %x20
VCHAR      = %x21-7E          ; printable ASCII (no SP)

```

Figure 5: ABNF syntax of Content-Format-Spec

```

// RFC editor: Please replace [RFC-httpbis-semantics] by what gets
// published from [I-D.ietf-httpbis-semantics].

```

7. Security Considerations

The indication of a media type in the data does not exempt a consuming application from properly checking its inputs. Also, the ability for an attacker to supply crafted SenML data that specify media types chosen by the attacker may expose vulnerabilities of handlers for these media types to the attacker. This includes "decompression bombs", compressed data that is crafted to decompress to extremely large data items.

8. IANA Considerations

(Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification and remove this note.)

IANA is requested to assign new labels in the "SenML Labels" subregistry of the SenML registry [IANA.senml] (as defined in Section 12.2 of [RFC8428]) for the Content-Format indication as per Table 1:

Name	Label	JSON Type	XML Type	Reference
Base Content-Format	bct	String	string	RFC-AAAA
Content-Format	ct	String	string	RFC-AAAA

Table 1: IANA Registration for new SenML Labels

Note that as per Section 12.2 of [RFC8428], no CBOR labels or EXI schemaId values (EXI ID column) are supplied.

9. References

9.1. Normative References

- [I-D.ietf-httpbis-semantics]
 Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021,
<https://www.ietf.org/archive/id/draft-ietf-httpbis-semantics-19.txt>.
- [IANA.core-parameters]
 IANA, "Constrained RESTful Environments (CoRE) Parameters",
<https://www.iana.org/assignments/core-parameters>.
- [IANA.http-parameters]
 IANA, "Hypertext Transfer Protocol (HTTP) Parameters",
<https://www.iana.org/assignments/http-parameters>.
- [IANA.media-types]
 IANA, "Media Types",
<https://www.iana.org/assignments/media-types>.
- [IANA.senml]
 IANA, "Sensor Measurement Lists (SenML)",
<https://www.iana.org/assignments/senml>.
- [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>.

- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

9.2. Informative References

- [RFC1590] Postel, J., "Media Type Registration Procedure", RFC 1590, DOI 10.17487/RFC1590, March 1994, <<https://www.rfc-editor.org/info/rfc1590>>.
- [RFC4180] Shafranovich, Y., "Common Format and MIME Type for Comma-Separated Values (CSV) Files", RFC 4180, DOI 10.17487/RFC4180, October 2005, <<https://www.rfc-editor.org/info/rfc4180>>.
- [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>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [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>>.

- [RFC8866] Begen, A., Kyzivat, P., Perkins, C., and M. Handley, "SDP: Session Description Protocol", RFC 8866, DOI 10.17487/RFC8866, January 2021, <<https://www.rfc-editor.org/info/rfc8866>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

Acknowledgements

The authors would like to thank Sérgio Abreu for the discussions leading to the design of this extension and Isaac Rivera for reviews and feedback. Klaus Hartke suggested not burdening this draft with a separate mandatory-to-implement version of the fields. Alexey Melnikov, Jim Schaad, and Thomas Fossati provided helpful comments at Working-Group last call. Marco Tiloca asked for clarifying and using the term Content-Format-Spec.

Authors' Addresses

Ari Keränen
Ericsson
FI-02420 Jorvas
Finland

Email: ari.keranen@ericsson.com

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

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

Network Working Group
Internet-Draft
Updates: 8428 (if approved)
Intended status: Standards Track
Expires: 19 May 2021

C. Bormann
Universitaet Bremen TZI
15 November 2020

SenML Features and Versions
draft-ietf-core-senml-versions-01

Abstract

This short document updates RFC 8428, Sensor Measurement Lists (SenML), by specifying the use of independently selectable "SenML Features" and mapping them to SenML version numbers.

Discussion Venues

This note is to be removed before publishing as an RFC.

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

Source for this draft and an issue tracker can be found at <https://github.com/core-wg/senml-versions> (<https://github.com/core-wg/senml-versions>).

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 19 May 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Feature Codes and the Version number	3
3. Features: Reserved0, Reserved1, Reserved2, Reserved3	3
4. Feature: Secondary Units	3
5. Security Considerations	4
6. IANA Considerations	4
7. Normative References	5
Acknowledgements	5
Author's Address	5

1. Introduction

The Sensor Measurement Lists (SenML) specification [RFC8428] provides a version number that is initially set to 10, without further specification on the way to make use of different version numbers.

The traditional idea of using a version number for evolving an interchange format presupposes a linear progression of that format. A more likely form of evolution of SenML is the addition of independently selectable "features" that can be added to the base version (version 10) in a fashion that these are mostly independent of each other. A recipient of a SenML pack can check the features it implements against those required by the pack, processing the pack only if all required features are provided in the implementation.

This short document specifies the use of SenML Features and maps them to SenML version number space, updating [RFC8428].

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.

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C [C], except that superscript notation (example for two to the power of 64: $2^{(64)}$) denotes exponentiation; in the plain text version of this draft, superscript notation is rendered by C-incompatible surrogate notation as seen in this example.

2. Feature Codes and the Version number

The present specification defines "SenML Features", each identified by a "feature name" (a text string) and a "feature code", an unsigned integer less than 53.

The specific version of a SenML pack is composed of a set of features. The SenML version number ("bver" field) is then a bitmap of these features, specifically the sum of, for each feature present, two taken to the power of the feature code of that feature.

$$\text{version} = \sum_{fc=0}^{52} \text{present}(fc) \times 2^{fc}$$

where $\text{present}(fc)$ is 1 if the feature with the feature code "fc" is present, 0 otherwise.

3. Features: Reserved0, Reserved1, Reserved2, Reserved3

For SenML Version 10 as described in [RFC8428], the feature codes 0 to 3 are already in use. Reserved1 (1) and Reserved3 (3) are always present and the features Reserved0 (0) and Reserved2 (2) are always absent, yielding a version number of 10 if no other feature is in use. These four reserved feature codes are not to be used with any more specific semantics except in a specification that updates the present specification.

4. Feature: Secondary Units

The feature "Secondary Units" (code number 4) indicates that secondary unit names [RFC8798] MAY be used in the "u" field of SenML Records, in addition to the primary unit names already allowed by [RFC8428].

Note that the most basic use of this feature simply sets the SenML version number to 26 ($10 + 2^4$).

5. Security Considerations

The security considerations of [RFC8428] apply. This specification provides structure to the interpretation of the SenML version number, which poses no additional security considerations except for some potential for surprise that version numbers do not simply increase linearly.

6. IANA Considerations

IANA is requested to create a new subregistry "SenML features" within the SenML registry [IANA.senml], with the registration policy "specification required" [RFC8126] and the columns:

- * Feature code (an unsigned integer less than 53)
- * Feature name (text)
- * Specification

The initial content of this registry is as follows:

Feature code	Feature name	Specification
0	Reserved0	RFCthis
1	Reserved1	RFCthis
2	Reserved2	RFCthis
3	Reserved3	RFCthis
4	Secondary Units	RFCthis

Table 1: Features defined for SenML at the time of writing

As the number of features that can be registered has a hard limit (48 codes left at the time of writing), the designated expert is specifically instructed to maintain a frugal regime of code point allocation, keeping code points available for SenML Features that are likely to be useful for non-trivial subsets of the SenML ecosystem. Quantitatively, the expert could for instance steer the allocation to not allocate more than 10 % of the remaining set per year.

7. Normative References

- [C] International Organization for Standardization,
"Information technology Programming languages C", ISO/
IEC 9899:2018, Fourth Edition, June 2018.
- [IANA.senml]
IANA, "Sensor Measurement Lists (SenML)",
<<http://www.iana.org/assignments/senml>>.
- [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>>.
- [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>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C.
Bormann, "Sensor Measurement Lists (SenML)", RFC 8428,
DOI 10.17487/RFC8428, August 2018,
<<https://www.rfc-editor.org/info/rfc8428>>.
- [RFC8798] Bormann, C., "Additional Units for Sensor Measurement
Lists (SenML)", RFC 8798, DOI 10.17487/RFC8798, June 2020,
<<https://www.rfc-editor.org/info/rfc8798>>.

Acknowledgements

Ari Keranen proposed to use the version number as a bitmap and provided further input on this specification.

Author's Address

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

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

CoRE
Internet-Draft
Updates: 8428 (if approved)
Intended status: Standards Track
Expires: 6 December 2021

C. Bormann
Universitaet Bremen TZI
4 June 2021

SenML Features and Versions
draft-ietf-core-senml-versions-05

Abstract

This short document updates RFC 8428, Sensor Measurement Lists (SenML), by specifying the use of independently selectable "SenML Features" and mapping them to SenML version numbers.

Discussion Venues

This note is to be removed before publishing as an RFC.

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

Source for this draft and an issue tracker can be found at <https://github.com/core-wg/senml-versions> (<https://github.com/core-wg/senml-versions>).

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 6 December 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Feature Codes and the Version number	3
2.1. Discussion	4
2.2. Updating RFC8428	4
3. Features: Reserved0, Reserved1, Reserved2, Reserved3	5
4. Feature: Secondary Units	5
5. Security Considerations	5
6. IANA Considerations	6
7. References	7
7.1. Normative References	7
7.2. Informative References	7
Acknowledgements	8
Author's Address	8

1. Introduction

The Sensor Measurement Lists (SenML) specification [RFC8428] provides a version number that is initially set to 10, without further specification on the way to make use of different version numbers.

The traditional idea of using a version number to indicate the evolution of an interchange format generally assumes an incremental progression of the version number as the format accretes additional features over time. However, in the case of SenML, it is expected that the likely evolution will be for independently selectable capability `_features_` to be added to the basic specification that is indicated by version number 10. To support this model, this document repurposes the single version number accompanying a pack of SenML records so that it is interpreted as a bitmap that indicates the set of features a recipient would need to have implemented to be able to process the pack.

This short document specifies the use of SenML Features and maps them to SenML version number space, updating [RFC8428].

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.

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C [C], including the "0b" prefix for binary numbers defined in Section 5.13.2 of the C++ language standard [Cplusplus], except that superscript notation (example for two to the power of 64: 2⁶⁴) denotes exponentiation; in the plain text version of this draft, superscript notation is rendered in paragraph text by C-incompatible surrogate notation as seen in this example, and in display math by a crude plaintext representation, as is the sum (Sigma) sign.

2. Feature Codes and the Version number

The present specification defines "SenML Features", each identified by a "feature name" (a text string) and a "feature code" (an unsigned integer less than 53).

The specific version of a SenML pack is composed of a set of features. The SenML version number ("bver" field) is then a bitmap of these features represented as an unsigned integer, specifically the sum of, for each feature present, two taken to the power of the feature code of that feature (Figure 1).

$$\text{version} = \sum_{fc=0}^{52} \text{present}(fc) \cdot 2^{fc}$$

Figure 1: Feature bitmap as a sum of feature bits

where present(fc) is 1 if the feature with the feature code "fc" is present, 0 otherwise. (The expression 2^{fc} can be implemented as "1 << fc" in C and related languages.)

RFC editor: Please check that, in the TXT version, no " " crept into the above due to xml2rfc bug 641, and remove this paragraph. If possible with today's RFCXML, add the Sigma character as a parenthesis after "sum" in the caption.

2.1. Discussion

Representing features as a bitmap within a number is quite efficient as long as feature codes are sparingly allocated (see also Section 6).

[illegible]

Most representations visible to engineers working with SenML will use decimal numbers, e.g., 26 (0b11010, 0x1a) for a version that adds the "Secondary Units" feature (Section 4). This is slightly unwieldy, but will be quickly memorized in practice.

As a general observation, ending up over time with dozens of individually selectable optional extensions may lead to too many variants of what is supported by different implementations, reducing interoperability. So, in practice, it is still desirable to batch up extensions that are expected to be supported together into a single feature bit, leading to a sort of hybrid between completely independent extensions and a linear version scheme. This is also another reason why a space of 48 remaining feature codes should suffice for a while.

2.2. Updating Section 4.4 of [RFC8428]

The last paragraph of Section 4.4 of [RFC8428] may be read to give the impression that SenML version numbers are totally ordered, i.e., that an implementation that understands version n also always understands all versions $k < n$. If this ever was true for SenML versions before 10, it certainly is no longer true with this specification.

Any SenML pack that sets feature bits beyond the first four will lead to a version number that actually is greater than 10, so the requirement in Section 4.4 of [RFC8428] will prevent false interoperability with version 10 implementations.

Implementations that do implement feature bits beyond the first four, i.e., versions greater than 10, will instead need to perform a bitwise comparison of the feature bitmap as described in this specification and ensure that all features indicated are understood before using the pack. E.g., an implementation that implements basic SenML (version number 10) plus only a future feature code 5, will accept version number 42, but would not be able to work with a pack indicating version number 26 (base specification plus feature code 4). (If the implementation `_requires_` feature code 5 without being backwards compatible, it will accept 42, but not 10.)

3. Features: Reserved0, Reserved1, Reserved2, Reserved3

For SenML Version 10 as described in [RFC8428], the feature codes 0 to 3 are already in use. Reserved1 (1) and Reserved3 (3) are always present and the features Reserved0 (0) and Reserved2 (2) are always absent, i.e., the four least significant bits set to 0b1010 indicate a version number of 10 if no other feature is in use. These four reserved feature codes are not to be used with any more specific semantics except in a specification that updates the present specification. (Note that Reserved0 and Reserved2 could be used in such a specification in a similar way to the way the feature codes 4 to 52 are in the present specification.)

4. Feature: Secondary Units

The feature "Secondary Units" (code number 4) indicates that secondary unit names [RFC8798] MAY be used in the "u" field of SenML Records, in addition to the primary unit names already allowed by [RFC8428].

Note that the most basic use of this feature simply sets the SenML version number to 26 ($10 + 2^4$).

5. Security Considerations

The security considerations of [RFC8428] apply. This specification provides structure to the interpretation of the SenML version number, which poses no additional security considerations except for some potential for surprise that version numbers do not simply increase linearly.

6. IANA Considerations

IANA is requested to create a new subregistry "SenML features" within the SenML registry [IANA.senml], with the registration policy "specification required" [RFC8126] and the columns:

- * Feature code (an unsigned integer less than 53)
- * Feature name (text)
- * Specification

To facilitate the use of feature names in programs, the designated expert is requested to ensure that feature names are usable as identifiers in most programming languages, after lower-casing the feature name in the registry entry and replacing whitespace with underscores or hyphens, and that they also are distinct in this form.

The initial content of this registry is as follows:

Feature code	Feature name	Specification
0	Reserved0	RFCthis
1	Reserved1	RFCthis
2	Reserved2	RFCthis
3	Reserved3	RFCthis
4	Secondary Units	RFCthis, [RFC8798]

Table 1: Features defined for SenML at the time of writing

As the number of features that can be registered has a hard limit (48 codes left at the time of writing), the designated expert is specifically instructed to maintain a frugal regime of code point allocation, keeping code points available for SenML Features that are likely to be useful for non-trivial subsets of the SenML ecosystem. Quantitatively, the expert could for instance steer the allocation to a target of not allocating more than 10 % of the remaining set per year.

Where the specification of the feature code is provided in a document that is separate from the specification of the feature itself (as with feature code 4 above), both specifications should be listed.

7. References

7.1. Normative References

- [C] International Organization for Standardization,
"Information technology Programming languages C", ISO/
IEC 9899:2018, Fourth Edition, June 2018,
<<https://www.iso.org/standard/74528.html>>.
- [Cplusplus] International Organization for Standardization,
"Programming languages C++", ISO/IEC 14882:2020, Sixth
Edition, December 2020,
<<https://www.iso.org/standard/79358.html>>.
- [IANA.senml] IANA, "Sensor Measurement Lists (SenML)",
<<http://www.iana.org/assignments/senml>>.
- [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>>.
- [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>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C.
Bormann, "Sensor Measurement Lists (SenML)", RFC 8428,
DOI 10.17487/RFC8428, August 2018,
<<https://www.rfc-editor.org/info/rfc8428>>.
- [RFC8798] Bormann, C., "Additional Units for Sensor Measurement
Lists (SenML)", RFC 8798, DOI 10.17487/RFC8798, June 2020,
<<https://www.rfc-editor.org/info/rfc8798>>.

7.2. Informative References

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493,
DOI 10.17487/RFC7493, March 2015,
<<https://www.rfc-editor.org/info/rfc7493>>.

Acknowledgements

Ari Keränen proposed to use the version number as a bitmap and provided further input on this specification. Jaime Jiménez helped clarify the document by providing a review. Elwyn Davies provided a detailed GENART review, with directly implementable text suggestions that now form part of this specification. Rob Wilton supplied comments one of which became the last paragraph of Section 2.1; Éric Vyncke helped with Section 2. Additional thanks go to the other IESG reviewers.

Author's Address

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

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

CoRE Working Group
Internet-Draft
Updates: 7252, 8323 (if approved)
Intended status: Standards Track
Expires: May 6, 2021

K. Hartke
Ericsson
M. Richardson
Sandelman
November 2, 2020

Extended Tokens and Stateless Clients
in the Constrained Application Protocol (CoAP)
draft-ietf-core-stateless-07

Abstract

This document provides considerations for alleviating CoAP clients and intermediaries of keeping per-request state. To facilitate this, this document additionally introduces a new, optional CoAP protocol extension for extended token lengths.

This document updates RFCs 7252 and 8323 with a new definition of the TKL field in the CoAP message header.

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 May 6, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	4
2. Extended Tokens	4
2.1. Extended Token Length (TKL) Field	4
2.2. Discovering Support	5
2.2.1. Extended-Token-Length Capability Option	5
2.2.2. Trial-and-Error	6
2.3. Intermediaries	8
3. Stateless Clients	9
3.1. Serializing Client State	9
3.2. Using Extended Tokens	10
3.3. Transmitting Messages	12
4. Stateless Intermediaries	12
4.1. Observing Resources	13
4.2. Block-Wise Transfers	13
4.3. Gateway Timeouts	14
4.4. Extended Tokens	14
5. Security Considerations	14
5.1. Extended Tokens	14
5.2. Stateless Clients and Intermediaries	14
6. IANA Considerations	16
6.1. CoAP Signaling Option Number	16
7. References	16
7.1. Normative References	16
7.2. Informative References	17
Appendix A. Updated Message Formats	18
A.1. CoAP over UDP	18
A.2. CoAP over TCP/TLS	20
A.3. CoAP over WebSockets	21
Acknowledgements	21
Authors' Addresses	22

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a RESTful application-layer protocol for constrained environments [RFC7228]. In CoAP, clients (or intermediaries in the client role) make requests to servers (or intermediaries in the server role), which satisfy the requests by returning responses.

While a request is ongoing, a client typically needs to keep some state that it requires for processing the response when that arrives. Identification of this state is done in CoAP by means of a token, an opaque sequence of bytes chosen by the client and included in the CoAP request, and that is returned by the server verbatim in any resulting CoAP response (Figure 1).

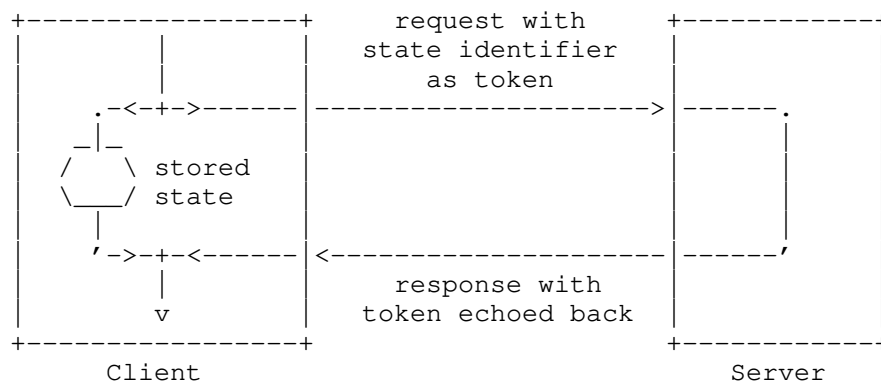


Figure 1: Token as an Identifier for Request State

In some scenarios, it can be beneficial to reduce the amount of state that is stored at the client at the cost of increased message sizes. A client can opt into this by serializing (parts of) its state into the token itself and then recovering this state from the token in the response (Figure 2).

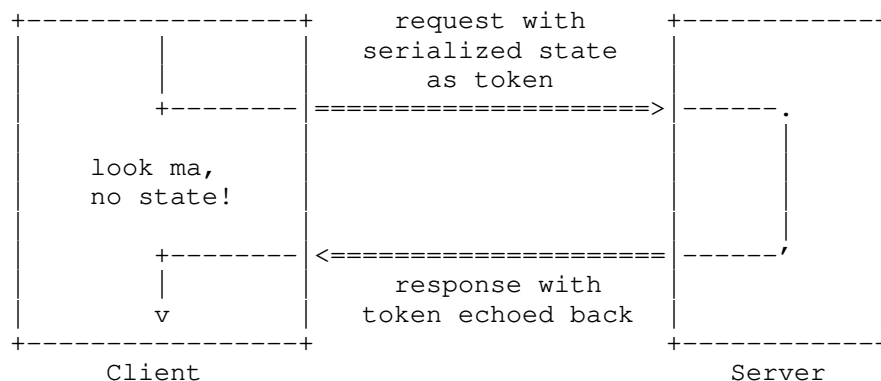


Figure 2: Token as Serialization of Request State

Section 3 of this document provides considerations for clients becoming "stateless" in this way. (The term "stateless" is in quotes here, because it's a bit oversimplified. Such clients still need to

maintain per-server state and other kinds of state. So it would be more accurate to just say that the clients are avoiding per-request state.)

Section 4 of this document extends the considerations for clients to intermediaries, which may not only want to avoid keeping state for the requests they send to servers but also for the requests they receive from clients.

The serialization of state into tokens is limited by the fact that both CoAP over UDP [RFC7252] and CoAP over reliable transports [RFC8323] restrict the maximum token length to 8 bytes. To overcome this limitation, Section 2 of this document first introduces a CoAP protocol extension for extended token lengths.

While the use case (avoiding per-request state) and the mechanism (extended token lengths) presented in this document are closely related, both can be used independently of each other: Some implementations may be able to fit their state in just 8 bytes; some implementations may have other use cases for extended token lengths.

1.1. Terminology

In this document, the term "stateless" refers to an implementation strategy for a client (or intermediary in the client role) that does not require it to keep state for the individual requests it sends to a server (or intermediary in the server role). The client still needs to keep state for each server it communicates with (e.g., for token generation, message retransmission, and congestion control).

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.

2. Extended Tokens

This document updates the message formats defined for CoAP over UDP [RFC7252] and CoAP over TCP, TLS, and WebSockets [RFC8323] with a new definition of the TKL field.

2.1. Extended Token Length (TKL) Field

The definition of the TKL field is updated as follows:

Token Length (TKL): 4-bit unsigned integer. A value between 0 and 12 inclusive indicates the length of the variable-length Token

field in bytes. The other three values are reserved for special constructs:

- 13: An 8-bit unsigned integer directly precedes the Token field and indicates the length of the Token field minus 13.
- 14: A 16-bit unsigned integer in network byte order directly precedes the Token field and indicates the length of the Token field minus 269.
- 15: Reserved. This value MUST NOT be sent and MUST be processed as a message format error.

All other fields retain their definitions.

The updated message formats are illustrated in Appendix A.

The new definition of the TKL field increases the maximum token length that can be represented in a message to 65804 bytes. However, the maximum token length that sender and recipient implementations support may be shorter. For example, a constrained node of Class 1 [RFC7228] might support extended token lengths only up to 32 bytes.

In CoAP over UDP, it is often beneficial to keep CoAP messages small enough to avoid IP fragmentation. The maximum practical token length may therefore also be influenced by the Path MTU. See Section 4.6 of RFC 7252 for details.

2.2. Discovering Support

Extended token lengths require support from server implementations. Support can be discovered by a client implementation in one of two ways:

- o Where Capabilities and Settings Messages (CSMs) are available, such as in CoAP over TCP, support can be discovered using the Extended-Token-Length Capability Option defined in Section 2.2.1.
- o Otherwise, such as in CoAP over UDP, support can only be discovered by trial-and-error, as described in Section 2.2.2.

2.2.1. Extended-Token-Length Capability Option

A server can use the elective Extended-Token-Length Capability Option to indicate the maximum token length it can accept in requests.

#	C	R	Applie s to	Name	Forma t	Length	Base Value
TB D			CSM	Extended-Token- Length	uint	0-3	8

C=Critical, R=Repeatable

Table 1: The Extended-Token-Length Capability Option

As per Section 3 of RFC 7252, the base value (and the value used when this option is not implemented) is 8.

The active value of the Extended-Token-Length Option is replaced each time the option is sent with a modified value. Its starting value is its base value.

The option value MUST NOT be less than 8 or greater than 65804. If an option value less than 8 is received, the option MUST be ignored. If an option value greater than 65804 is received, the option value MUST be set to 65804.

Any option value greater than 8 implies support for the new definition of the TKL field specified in Section 2.1. Indication of support by a server does not oblige a client to actually make use of token lengths greater than 8.

If a server receives a request with a token of a length greater than what it indicated in its Extended-Token-Length Option, it MUST handle the request as a message format error.

If a server receives a request with a token of a length less than or equal to what it indicated in its Extended-Token-Length Option but is unwilling or unable to handle the token at that time, it MUST NOT handle the request as a message format error. Instead, it SHOULD return a 5.03 (Service Unavailable) response.

The Extended-Token-Length Capability Option does not apply to responses. The sender of a request is simply expected not to use a token of a length greater than it is willing to accept in a response.

2.2.2. Trial-and-Error

A server implementation that does not support the updated definition of the TKL field specified in Section 2.1 will consider a request with a TKL field value outside the range 0 to 8 a message format

error and reject it (Section 3 of RFC 7252). A client can therefore determine support by sending a request with an extended token length and checking whether it is rejected by the server or not.

In CoAP over UDP, the way a request message is rejected depends on the message type. A Confirmable message with a message format error is rejected with a Reset message (Section 4.2 of RFC 7252). A Non-confirmable message with a message format error is either rejected with a Reset message or just silently ignored (Section 4.3 of RFC 7252). To reliably get a Reset message, it is therefore **REQUIRED** that clients use a Confirmable message for determining support.

As per RFC 7252, Reset messages are empty and do not contain a token; they only return the Message ID (Figure 3). They also do not contain any indication of what caused a message format error. To avoid any ambiguity, it is therefore **RECOMMENDED** that clients use a request that has no potential message format error other than the extended token length.

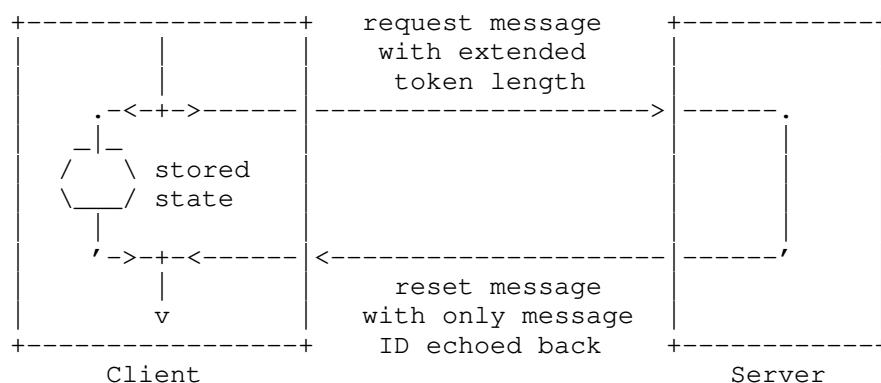


Figure 3: A Confirmable Request With an Extended Token is Rejected With a Reset Message if the Server Does Not Have Support

An example of a suitable request is a GET request in a Confirmable message that includes only an If-None-Match option and a token of the greatest length that the client intends to use. Any response with the same token echoed back indicates that tokens up to that length are supported by the server.

Since network addresses may change, a client **SHOULD NOT** assume that extended token lengths are supported by a server for an unlimited duration. Unless additional information is available, the client should assume that addresses (and therefore extended token lengths) are valid for a minimum of 1800s, and for a maximum of 86400s (1 day). A client may use additional forms of input into this

determination. For instance a client may assume a server which is in the same subnet as the client has a similar address lifetime as the client. The client may use DHCP lease times or Router Advertisements to set the limits. For servers which is not local, if the server was looked up using DNS, then the DNS resource record will have a Time To Live, and the extended token length should be kept for only that amount of time.

If a server supports extended token lengths but receives a request with a token of a length it is unwilling or unable to handle, it **MUST NOT** reject the message, as that would imply that extended token lengths are not supported at all. Instead, if the server cannot handle the request at the time, it **SHOULD** return a 5.03 (Service Unavailable) response; if the server will never be able to handle the request (e.g., because the token is too large), it **SHOULD** return a 4.00 (Bad Request) response.

Design Note: The requirement to return an error response when a token cannot be handled might seem somewhat contradictory, as returning the error response requires the server also to return the token it cannot handle. However, processing a request usually involves a number of steps from receiving the message to passing it to application logic. The idea is that a server implementing this extension supports large tokens at least in its first few processing steps, enough to return an error response rather than a Reset message.

Design Note: To make the trial-and-error-based discovery not too complicated, no effort is made to indicate the maximum supported token length. A client implementation would probably already choose the shortest token possible for the task (like being stateless as described in Section 3), so it would probably not be able to reduce the length any further anyway should a server indicate a lower limit.

2.3. Intermediaries

Tokens are a hop-by-hop feature: If there are one or more intermediaries between a client and a server, every token is scoped to the exchange between a node in the client role and the node in the server role that it is immediately interacting with.

When an intermediary receives a request, the only requirement is that it echoes the token back in any resulting response. There is no requirement or expectation that an intermediary passes a client's token on to a server or that an intermediary uses extended token lengths itself in its request to satisfy a request with an extended

token length. Discovery needs to be performed for each hop where extended token lengths are to be used.

3. Stateless Clients

A client can be alleviated of keeping per-request state as follows:

1. The client serializes (parts of) its per-request state into a sequence of bytes and sends those bytes as the token of its request to the server.
2. The server returns the token verbatim in the response to the client, which allows the client to recover the state and process the response as if it had kept the state locally.

As servers are just expected to return any token verbatim to the client, this implementation strategy for clients does not impact the interoperability of client and server implementations. However, there are a number of significant, non-obvious implications (e.g., related to security and other CoAP protocol features) that client implementations need take into consideration.

The following subsections discuss some of these considerations.

3.1. Serializing Client State

The format of the serialized state is generally an implementation detail of the client and opaque to the server. However, serialized state information is an attractive target for both unwanted nodes (e.g., on-path attackers) and wanted nodes (e.g., any configured forward proxy) on the path. The serialization format therefore needs to include security measures such as the following:

- o A client SHOULD protect the integrity of the state information serialized in a token.
- o Even when the integrity of the serialized state is protected, an attacker may still replay a response, making the client believe it sent the same request twice. For this reason, the client SHOULD implement replay protection (e.g., by using sequence numbers and a replay window). For replay protection, integrity protection is REQUIRED.
- o If processing a response without keeping request state is sensitive to the time elapsed since sending the request, then the client SHOULD include freshness information (e.g., a timestamp) in the serialized state and reject any response where the freshness information is insufficiently fresh.

- o Information in the serialized state may be privacy sensitive. A client SHOULD encrypt the serialized state if it contains privacy sensitive information that an attacker would not get otherwise.
- o When a client changes the format of the serialized state, it SHOULD prevent false interoperability with the previous format (e.g., by changing the key used for integrity protection or changing a field in the serialized state).

3.2. Using Extended Tokens

A client that depends on support for extended token lengths (Section 2) from the server to avoid keeping request state needs to perform a discovery of support (Section 2.2) before it can be stateless.

This discovery MUST be performed in a stateful way, i.e., keeping state for the request (Figure 4): If the client was stateless from the start and the server does not support extended tokens, then any error message could not be processed since the state would neither be present at the client nor returned in the Reset message (Figure 5).

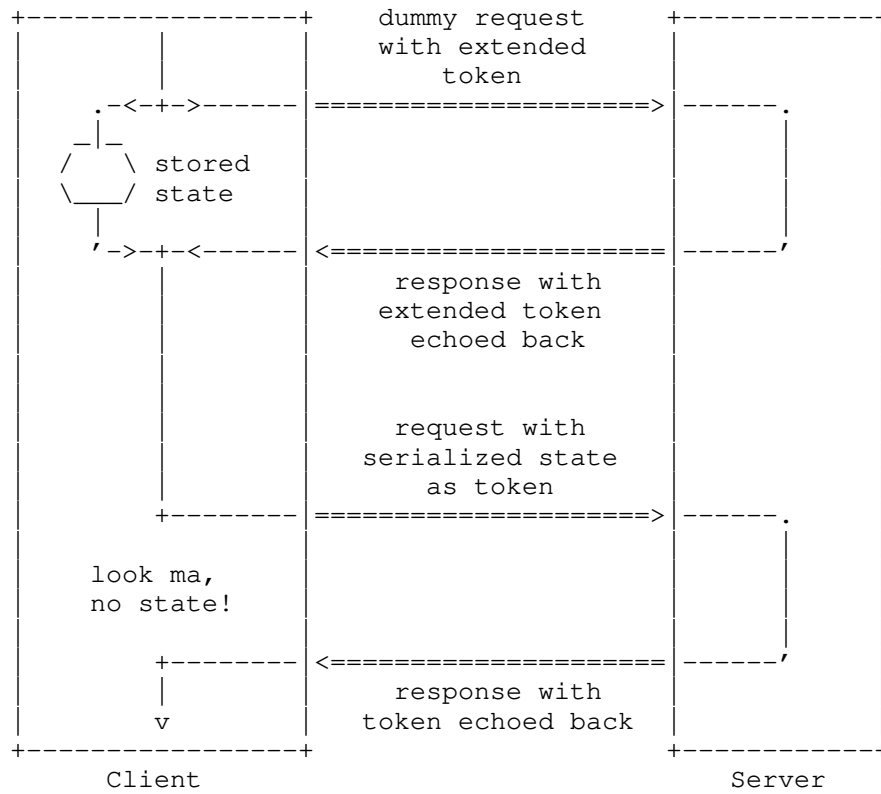


Figure 4: Depending on Extended Tokens for Being Stateless First Requires a Successful Stateful Discovery of Support

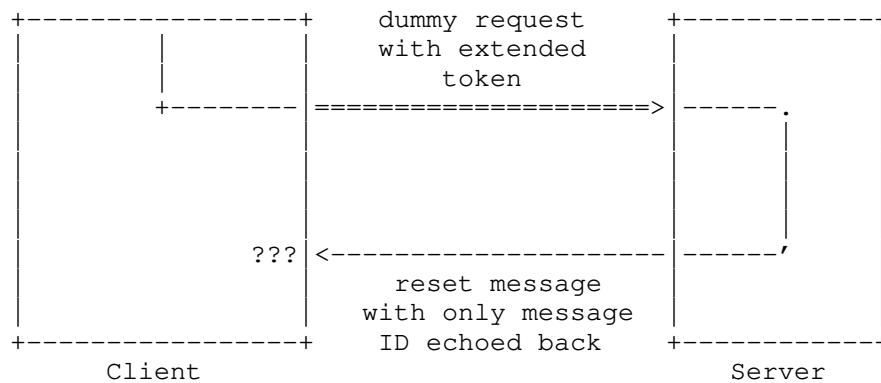


Figure 5: Stateless Discovery of Support Does Not Work

In environments where support can be reliably discovered through some other means, the discovery of support is OPTIONAL. An example for this is the Constrained Join Protocol (CoJP) in a 6TiSCH network [I-D.ietf-6tisch-minimal-security], where support for extended tokens is required from all relevant parties.

3.3. Transmitting Messages

In CoAP over UDP [RFC7252], a client has the choice between Confirmable and Non-confirmable messages for requests. When using Non-confirmable messages, a client does not have to keep any message exchange state, which can help in the goal of avoiding state. When using Confirmable messages, a client needs to keep message exchange state for performing retransmissions and handling Acknowledgement and Reset messages, however. Non-confirmable messages are therefore better suited for avoiding state. In any case, a client still needs to keep congestion control state, i.e., maintain state for each node it communicates with and enforce limits like NSTART.

As per Section 5.2 of RFC 7252, a client must be prepared to receive a response as a piggybacked response, a separate response, or Non-confirmable response, regardless of the message type used for the request. A stateless client MUST handle these response types as follows:

- o If a piggybacked response passes the checks for token integrity and freshness (Section 3.1), the client processes the message as specified in RFC 7252; otherwise, it processes the acknowledgement portion of the message as specified in RFC 7252 and silently discards the response portion.
- o If a separate response passes the checks for token integrity and freshness, the client processes the message as specified in RFC 7252; otherwise, it rejects the message as specified in Section 4.2 of RFC 7252.
- o If a Non-confirmable response passes the checks for token integrity and freshness, the client processes the message as specified in RFC 7252; otherwise, it rejects the message as specified in Section 4.3 of RFC 7252.

4. Stateless Intermediaries

Tokens are a hop-by-hop feature: If a client makes a request to an intermediary, that intermediary needs to store the client's token (along with the client's transport address) while it makes its own request towards the origin server and waits for the response. When the intermediary receives the response, it looks up the client's

token and transport address for the received request and sends an appropriate response to the client.

An intermediary might want to be "stateless" not only in its role as a client but also in its role as a server, i.e., be alleviated of storing the client information for the requests it receives.

Such an intermediary can be implemented by serializing the client information along with the request state into the token towards the origin server. When the intermediary receives the response, it can recover the client information from the token and use it to satisfy the client's request and therefore doesn't need to store it itself.

The following subsections discuss some considerations for this approach.

4.1. Observing Resources

One drawback of the approach is that an intermediary, without keeping request state, is unable to aggregate multiple requests for the same target resource, which can significantly reduce efficiency. In particular, when clients observe [RFC7641] the same resource, aggregating requests is REQUIRED (Section 3.1 of RFC 7641). This requirement cannot be satisfied without keeping request state.

Furthermore, an intermediary that does not keep track of the clients observing a resource is not able to determine whether these clients are still interested in receiving further notifications (Section 3.5 of RFC 7641) or want to cancel an observation (Section 3.6 of RFC 7641).

Therefore, an intermediary MUST NOT include an Observe Option in requests it sends without keeping both the request state for the requests it sends and the client information for the requests it receives.

4.2. Block-Wise Transfers

When using block-wise transfers [RFC7959], a server might not be able to distinguish blocks originating from different clients once they have been forwarded by an intermediary. Intermediaries need to ensure that this does not lead to inconsistent resource state by keeping distinct block-wise request operations on the same resource apart, e.g., utilizing the Request-Tag Option [I-D.ietf-core-echo-request-tag].

4.3. Gateway Timeouts

As per Section 5.7.1 of RFC 7252, an intermediary is REQUIRED to return a 5.04 (Gateway Timeout) response if it cannot obtain a response within a timeout. However, if an intermediary does not keep the client information for the requests it receives, it cannot return such a response. Therefore, in this case, the gateway cannot return such a response and as such cannot implement such a timeout.

4.4. Extended Tokens

A client may make use of extended token lengths in a request to an intermediary that wants to be "stateless". This means that such an intermediary may have to serialize potentially very large client information into its token towards the origin server. The tokens can grow even further when it progresses along a chain of intermediaries that all want to be "stateless".

Intermediaries SHOULD limit the size of client information they're serializing into their own tokens. An intermediary can do this, for example, by limiting the extended token lengths it accepts from its clients (see Section 2.2) or by keeping the client information locally when the client information exceeds the limit (i.e., not being "stateless").

5. Security Considerations

5.1. Extended Tokens

Tokens significantly larger than the 8 bytes specified in RFC 7252 have implications in particular for nodes with constrained memory size that need to be mitigated. A node in the server role supporting extended token lengths may be vulnerable to a denial-of-service when an attacker (either on-path or a malicious client) sends large tokens to fill up the memory of the node. Implementations need to be prepared to handle such messages.

5.2. Stateless Clients and Intermediaries

Transporting the state needed by a client to process a response as serialized state information in the token has several significant and non-obvious security and privacy implications that need to be mitigated; see Section 3.1 for recommendations.

In addition to the format requirements outlined there, implementations need to ensure that they are not vulnerable to maliciously crafted, delayed, or replayed tokens.

It is generally expected that the use of encryption, integrity protection, and replay protection for serialized state is appropriate.

In the absence of integrity and replay protection, an on-path attacker or rogue server/intermediary could return a state (either one modified in a reply, or an unsolicited one) that could alter the internal state of the client.

It is this reason that at least the use of integrity protection on the token is always recommended.

It maybe that in some very specific case, as a result of a careful and detailed analysis of any potential attacks, that there may be cases where such cryptographic protections do not add value. The authors of this document have not found such a use case as yet, but this is a local decision.

It should further be emphasized that the encrypted state is created by the sending node, and decrypted by the same node when receiving a response. The key is not shared with any other system. Therefore the choice of encryption scheme and the generation of the key for this system is purely a local matter.

When encryption is used, the use of AES-CCM [RFC3610] with a 64-bit tag is recommended, combined with a sequence number and a replay window. This choice is informed by available hardware acceleration of on many constrained systems. If a different algorithm is available accelerated on the sender, with similar strength, then it SHOULD be preferred. Where privacy of the state is not required, and encryption is not needed, HMAC-SHA-256 [RFC6234], combined with a sequence number and a replay window, may be used.

This size of the replay window depends upon the number of outstanding requests that need to be outstanding. This can be determined from the rate at which new ones are made, and the expected duration in which responses are expected.

For instance, given a CoAP ACK_TIMEOUT of 2s, and a request rate of 10 requests/second, any request that is not answered within 2s will be considered to have failed. Thus at most 20 request can be outstanding at a time, and any convenient replay window larger than 20 will work. As replay windows are often implemented with a sliding window and a bit, the use of a 32-bit window would be sufficient.

For use cases where requests are being relayed from another node, the request rate may be estimated by the total link capacity allocated for that kind of traffic. An alternate view would consider how many

IPv6 Neighbor Cache Entries (NCEs) the system can afford to allocate for this use.

When using an encryption mode that depends on a nonce, such as AES-CCM, repeated use of the same nonce under the same key causes the cipher to fail catastrophically.

If a nonce is ever used for more than one encryption operation with the same key, then the same key stream gets used to encrypt both plaintexts and the confidentiality guarantees are voided. Devices with low-quality entropy sources -- as is typical with constrained devices, which incidentally happen to be a natural candidate for the stateless mechanism described in this document -- need to carefully pick a nonce generation mechanism that provides the above uniqueness guarantee.

[RFC8613] appendix B.1.1 ("Sender Sequence Number") provides a model for how to maintain non-repeating nonces without causing excessive wear of flash.

6. IANA Considerations

6.1. CoAP Signaling Option Number

The following entries are added to the "CoAP Signaling Option Numbers" registry within the "CoRE Parameters" registry.

Applies to	Number	Name	Reference
7.01	TBD	Extended-Token-Length	[[this document]]

[[NOTE TO RFC EDITOR: Please replace "TBD" in this section and in Table 1 with the code point assigned by IANA.]]

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>>.

- [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>>.
- [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>>.
- [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>>.

7.2. Informative References

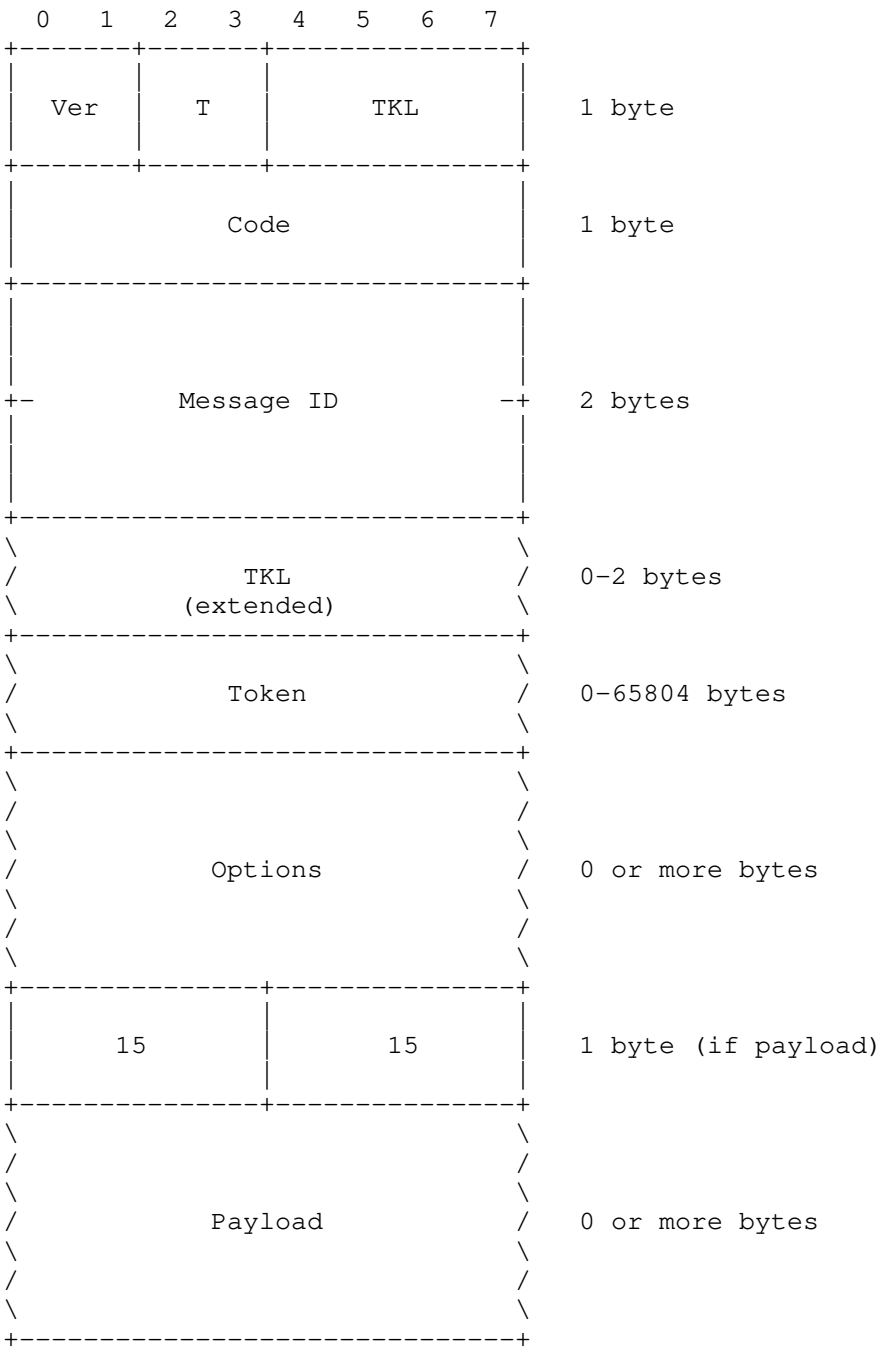
- [I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson,
"Constrained Join Protocol (CoJP) for 6TiSCH", draft-ietf-6tisch-minimal-security-15 (work in progress), December 2019.
- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-10 (work in progress), July 2020.
- [RFC3610] Whiting, D., Housley, R., and N. Ferguson, "Counter with CBC-MAC (CCM)", RFC 3610, DOI 10.17487/RFC3610, September 2003, <<https://www.rfc-editor.org/info/rfc3610>>.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<https://www.rfc-editor.org/info/rfc6234>>.

- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

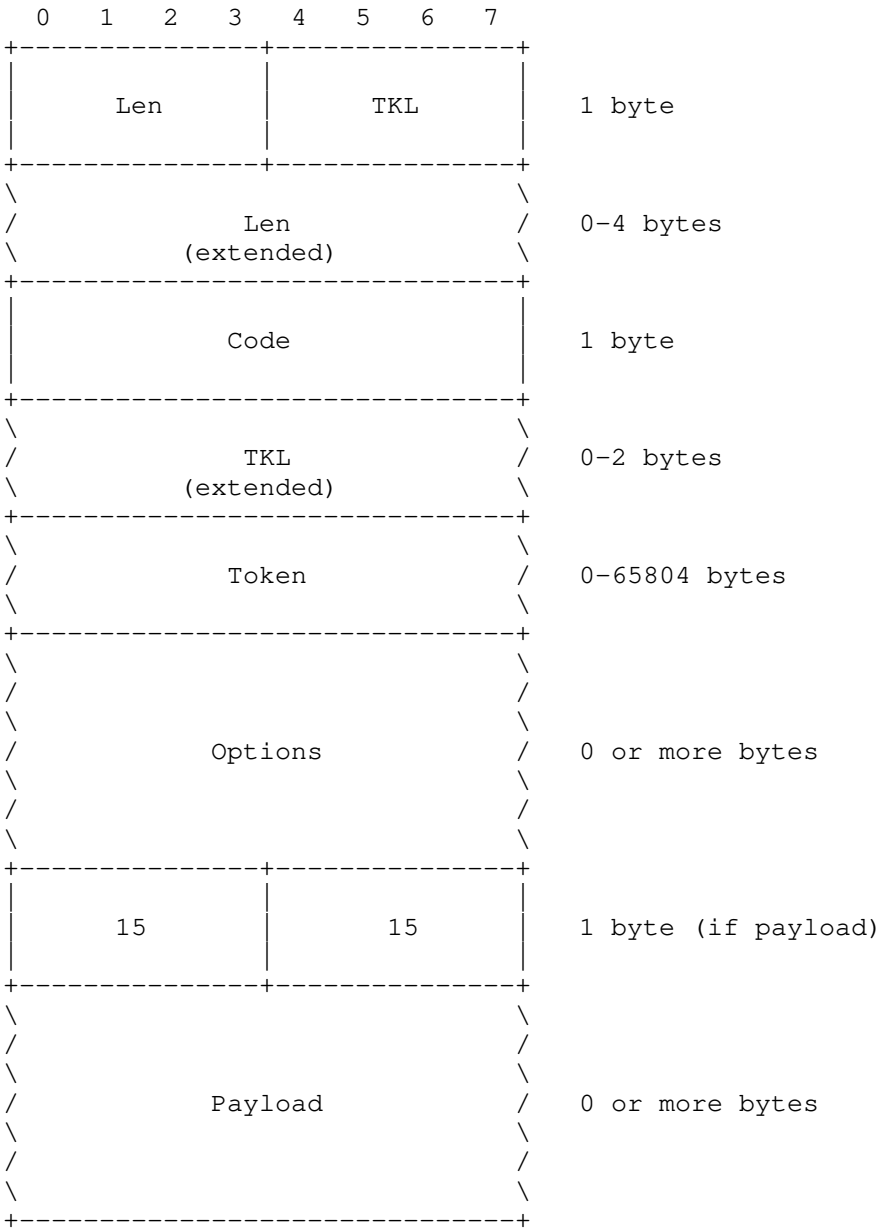
Appendix A. Updated Message Formats

In Section 2, this document updates the CoAP message formats by specifying a new definition of the TKL field in the message header. As an alternative presentation of this update, this appendix shows the CoAP message formats for CoAP over UDP [RFC7252] and CoAP over TCP, TLS, and WebSockets [RFC8323] with the new definition applied.

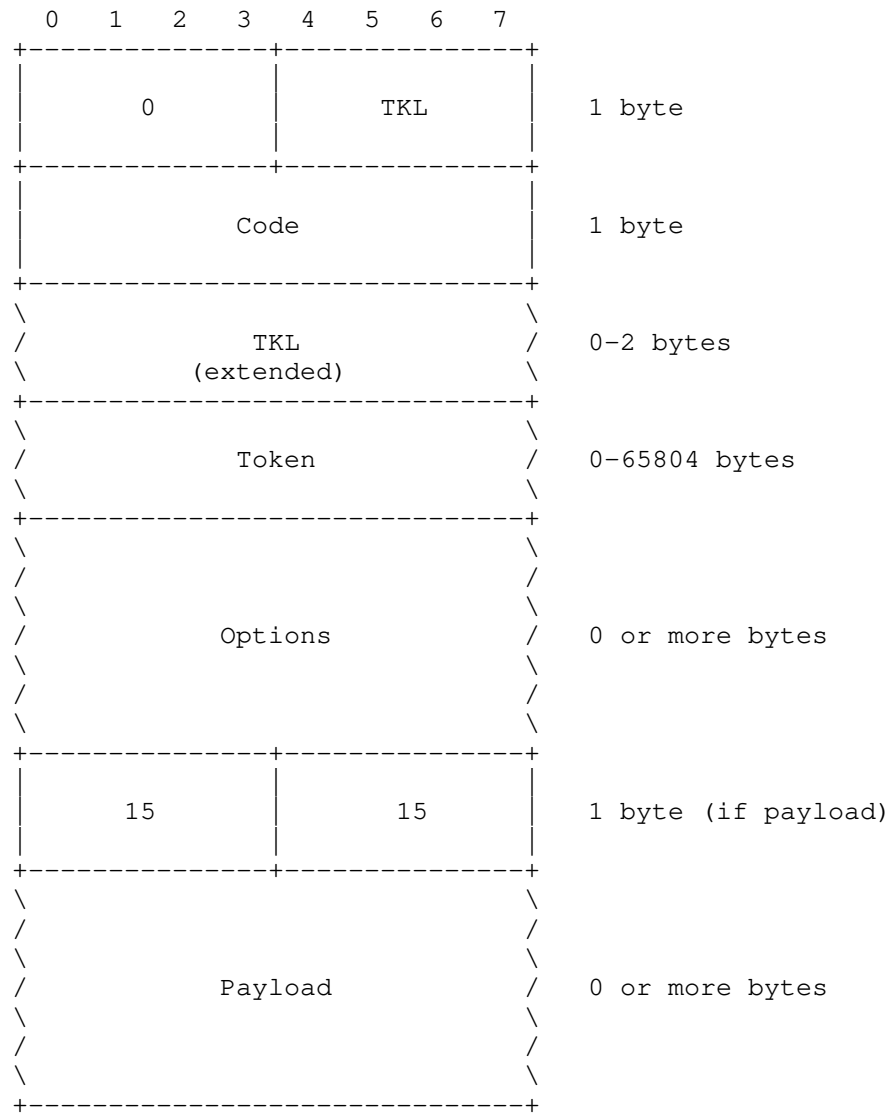
A.1. CoAP over UDP



A.2. CoAP over TCP/TLS



A.3. CoAP over WebSockets



Acknowledgements

This document is based on the requirements of and work on the Minimal Security Framework for 6TiSCH [I-D.ietf-6tisch-minimal-security] by Malisa Vucinic, Jonathan Simon, Kris Pister, and Michael Richardson.

Thanks to Christian Amsuss, Carsten Bormann, Roman Danyliw, Christer Holmberg, Benjamin Kaduk, Ari Keranen, Erik Kline, Murray Kucherawy, Warren Kumari, Barry Leiba, David Mandelberg, Dan Romascanu, Jim Schaad, Goran Selander, Malisa Vucinic, Eric Vyncke, and Robert Wilton for helpful comments and discussions that have shaped the document.

Special thanks to John Mattsson for his contributions to the security considerations of the document, and to Thomas Fossati for his in-depth review, copious comments, and suggested text.

Authors' Addresses

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

Michael C. Richardson
Sandelman Software Works

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

CoRE Working Group
Internet-Draft
Updates: 7252, 8323 (if approved)
Intended status: Standards Track
Expires: May 20, 2021

K. Hartke
Ericsson
M. Richardson
Sandelman
November 16, 2020

Extended Tokens and Stateless Clients
in the Constrained Application Protocol (CoAP)
draft-ietf-core-stateless-08

Abstract

This document provides considerations for alleviating CoAP clients and intermediaries of keeping per-request state. To facilitate this, this document additionally introduces a new, optional CoAP protocol extension for extended token lengths.

This document updates RFCs 7252 and 8323 with an extended definition of the TKL field in the CoAP message header.

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 May 20, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	4
2. Extended Tokens	4
2.1. Extended Token Length (TKL) Field	4
2.2. Discovering Support	5
2.2.1. Extended-Token-Length Capability Option	5
2.2.2. Trial-and-Error	6
2.3. Intermediaries	8
3. Stateless Clients	9
3.1. Serializing Client State	9
3.2. Using Extended Tokens	10
3.3. Transmitting Messages	12
4. Stateless Intermediaries	12
4.1. Observing Resources	13
4.2. Block-Wise Transfers	13
4.3. Gateway Timeouts	14
4.4. Extended Tokens	14
5. Security Considerations	14
5.1. Extended Tokens	14
5.2. Stateless Clients and Intermediaries	14
6. IANA Considerations	16
6.1. CoAP Signaling Option Number	16
7. References	16
7.1. Normative References	16
7.2. Informative References	17
Appendix A. Updated Message Formats	18
A.1. CoAP over UDP	18
A.2. CoAP over TCP/TLS	20
A.3. CoAP over WebSockets	21
Acknowledgements	21
Authors' Addresses	22

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a RESTful application-layer protocol for constrained environments [RFC7228]. In CoAP, clients (or intermediaries in the client role) make requests to servers (or intermediaries in the server role), which satisfy the requests by returning responses.

While a request is ongoing, a client typically needs to keep some state that it requires for processing the response when that arrives. Identification of this state is done in CoAP by means of a token, an opaque sequence of bytes chosen by the client and included in the CoAP request, and that is returned by the server verbatim in any resulting CoAP response (Figure 1).

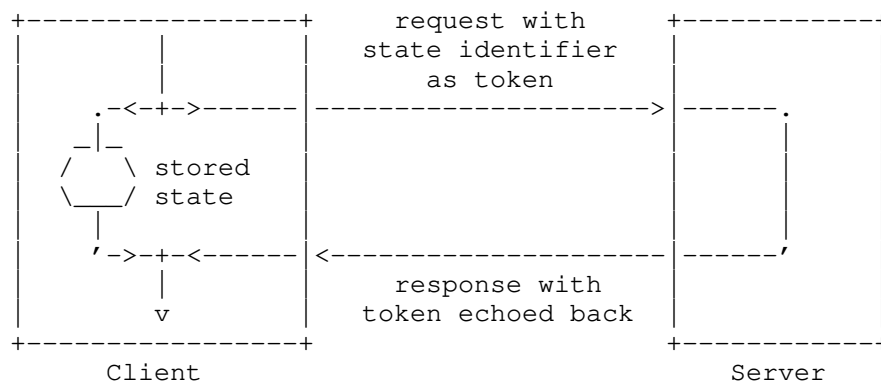


Figure 1: Token as an Identifier for Request State

In some scenarios, it can be beneficial to reduce the amount of state that is stored at the client at the cost of increased message sizes. A client can opt into this by serializing (parts of) its state into the token itself and then recovering this state from the token in the response (Figure 2).

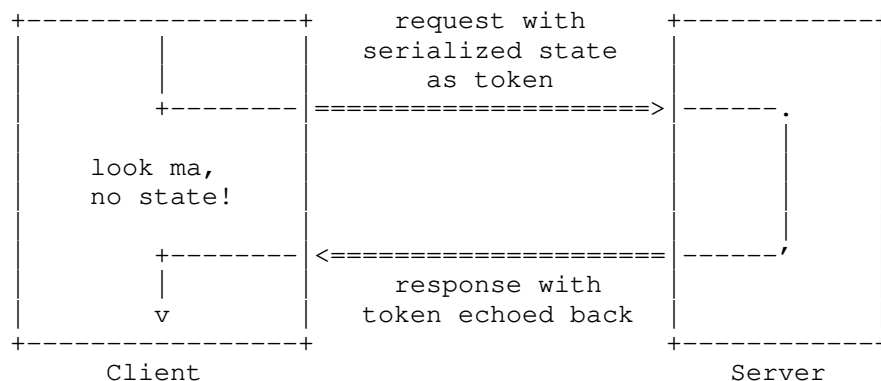


Figure 2: Token as Serialization of Request State

Section 3 of this document provides considerations for clients becoming "stateless" in this way. (The term "stateless" is in quotes here, because it's a bit oversimplified. Such clients still need to

maintain per-server state and other kinds of state. So it would be more accurate to just say that the clients are avoiding per-request state.)

Section 4 of this document extends the considerations for clients to intermediaries, which may not only want to avoid keeping state for the requests they send to servers but also for the requests they receive from clients.

The serialization of state into tokens is limited by the fact that both CoAP over UDP [RFC7252] and CoAP over reliable transports [RFC8323] restrict the maximum token length to 8 bytes. To overcome this limitation, Section 2 of this document first introduces a CoAP protocol extension for extended token lengths.

While the use case (avoiding per-request state) and the mechanism (extended token lengths) presented in this document are closely related, both can be used independently of each other: Some implementations may be able to fit their state in just 8 bytes; some implementations may have other use cases for extended token lengths.

1.1. Terminology

In this document, the term "stateless" refers to an implementation strategy for a client (or intermediary in the client role) that does not require it to keep state for the individual requests it sends to a server (or intermediary in the server role). The client still needs to keep state for each server it communicates with (e.g., for token generation, message retransmission, and congestion control).

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.

2. Extended Tokens

This document updates the message formats defined for CoAP over UDP [RFC7252] and CoAP over TCP, TLS, and WebSockets [RFC8323] with a new definition of the TKL field.

2.1. Extended Token Length (TKL) Field

The definition of the TKL field is updated as follows:

Token Length (TKL): 4-bit unsigned integer. A value between 0 and 12 inclusive indicates the length of the variable-length Token

field in bytes. The other three values are reserved for special constructs:

- 13: An 8-bit unsigned integer directly precedes the Token field and indicates the length of the Token field minus 13.
- 14: A 16-bit unsigned integer in network byte order directly precedes the Token field and indicates the length of the Token field minus 269.
- 15: Reserved. This value MUST NOT be sent and MUST be processed as a message format error.

All other fields retain their definitions.

The updated message formats are illustrated in Appendix A.

The new definition of the TKL field increases the maximum token length that can be represented in a message to 65804 bytes. However, the maximum token length that sender and recipient implementations support may be shorter. For example, a constrained node of Class 1 [RFC7228] might support extended token lengths only up to 32 bytes.

In CoAP over UDP, it is often beneficial to keep CoAP messages small enough to avoid IP fragmentation. The maximum practical token length may therefore also be influenced by the Path MTU. See Section 4.6 of RFC 7252 for details.

2.2. Discovering Support

Extended token lengths require support from server implementations. Support can be discovered by a client implementation in one of two ways:

- o Where Capabilities and Settings Messages (CSMs) are available, such as in CoAP over TCP, support can be discovered using the Extended-Token-Length Capability Option defined in Section 2.2.1.
- o Otherwise, such as in CoAP over UDP, support can only be discovered by trial-and-error, as described in Section 2.2.2.

2.2.1. Extended-Token-Length Capability Option

A server can use the elective Extended-Token-Length Capability Option to indicate the maximum token length it can accept in requests.

#	C	R	Applie s to	Name	Forma t	Length	Base Value
TB D			CSM	Extended-Token- Length	uint	0-3	8

C=Critical, R=Repeatable

Table 1: The Extended-Token-Length Capability Option

As per Section 3 of RFC 7252, the base value (and the value used when this option is not implemented) is 8.

The active value of the Extended-Token-Length Option is replaced each time the option is sent with a modified value. Its starting value is its base value.

The option value MUST NOT be less than 8 or greater than 65804. If an option value less than 8 is received, the option MUST be ignored. If an option value greater than 65804 is received, the option value MUST be set to 65804.

Any option value greater than 8 implies support for the new definition of the TKL field specified in Section 2.1. Indication of support by a server does not oblige a client to actually make use of token lengths greater than 8.

If a server receives a request with a token of a length greater than what it indicated in its Extended-Token-Length Option, it MUST handle the request as a message format error.

If a server receives a request with a token of a length less than or equal to what it indicated in its Extended-Token-Length Option but is unwilling or unable to handle the token at that time, it MUST NOT handle the request as a message format error. Instead, it SHOULD return a 5.03 (Service Unavailable) response.

The Extended-Token-Length Capability Option does not apply to responses. The sender of a request is simply expected not to use a token of a length greater than it is willing to accept in a response.

2.2.2. Trial-and-Error

A server implementation that does not support the updated definition of the TKL field specified in Section 2.1 will consider a request with a TKL field value outside the range 0 to 8 a message format

error and reject it (Section 3 of RFC 7252). A client can therefore determine support by sending a request with an extended token length and checking whether it is rejected by the server or not.

In CoAP over UDP, the way a request message is rejected depends on the message type. A Confirmable message with a message format error is rejected with a Reset message (Section 4.2 of RFC 7252). A Non-confirmable message with a message format error is either rejected with a Reset message or just silently ignored (Section 4.3 of RFC 7252). To reliably get a Reset message, it is therefore REQUIRED that clients use a Confirmable message for determining support.

As per RFC 7252, Reset messages are empty and do not contain a token; they only return the Message ID (Figure 3). They also do not contain any indication of what caused a message format error. To avoid any ambiguity, it is therefore RECOMMENDED that clients use a request that has no potential message format error other than the extended token length.

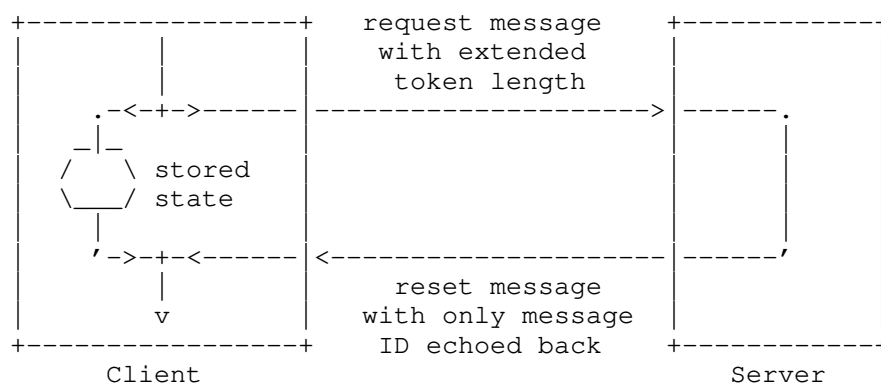


Figure 3: A Confirmable Request With an Extended Token is Rejected With a Reset Message if the Server Does Not Have Support

An example of a suitable request is a GET request in a Confirmable message that includes only an If-None-Match option and a token of the greatest length that the client intends to use. Any response with the same token echoed back indicates that tokens up to that length are supported by the server.

Since network addresses may change, a client SHOULD NOT assume that extended token lengths are supported by a server for an unlimited duration. Unless additional information is available, the client should assume that addresses (and therefore extended token lengths) are valid for a minimum of 1800 s, and for a maximum of 86400 s (1 day). A client may use additional forms of input into this

determination. For instance a client may assume a server which is in the same subnet as the client has a similar address lifetime as the client. The client may use DHCP lease times or Router Advertisements to set the limits. For servers that are not local, if the server was looked up using DNS, then the DNS resource record will have a Time To Live, and the extended token length should be kept for only that amount of time.

If a server supports extended token lengths but receives a request with a token of a length it is unwilling or unable to handle, it **MUST NOT** reject the message, as that would imply that extended token lengths are not supported at all. Instead, if the server cannot handle the request at the time, it **SHOULD** return a 5.03 (Service Unavailable) response; if the server will never be able to handle the request (e.g., because the token is too large), it **SHOULD** return a 4.00 (Bad Request) response.

Design Note: The requirement to return an error response when a token cannot be handled might seem somewhat contradictory, as returning the error response requires the server also to return the token it cannot handle. However, processing a request usually involves a number of steps from receiving the message to passing it to application logic. The idea is that a server implementing this extension supports large tokens at least in its first few processing steps, enough to return an error response rather than a Reset message.

Design Note: To make the trial-and-error-based discovery not too complicated, no effort is made to indicate the maximum supported token length. A client implementation would probably already choose the shortest token possible for the task (like being stateless as described in Section 3), so it would probably not be able to reduce the length any further anyway should a server indicate a lower limit.

2.3. Intermediaries

Tokens are a hop-by-hop feature: If there are one or more intermediaries between a client and a server, every token is scoped to the exchange between a node in the client role and the node in the server role that it is immediately interacting with.

When an intermediary receives a request, the only requirement is that it echoes the token back in any resulting response. There is no requirement or expectation that an intermediary passes a client's token on to a server or that an intermediary uses extended token lengths itself in its request to satisfy a request with an extended

token length. Discovery needs to be performed for each hop where extended token lengths are to be used.

3. Stateless Clients

A client can be alleviated of keeping per-request state as follows:

1. The client serializes (parts of) its per-request state into a sequence of bytes and sends those bytes as the token of its request to the server.
2. The server returns the token verbatim in the response to the client, which allows the client to recover the state and process the response as if it had kept the state locally.

As servers are just expected to return any token verbatim to the client, this implementation strategy for clients does not impact the interoperability of client and server implementations. However, there are a number of significant, non-obvious implications (e.g., related to security and other CoAP protocol features) that client implementations need take into consideration.

The following subsections discuss some of these considerations.

3.1. Serializing Client State

The format of the serialized state is generally an implementation detail of the client and opaque to the server. However, serialized state information is an attractive target for both unwanted nodes (e.g., on-path attackers) and wanted nodes (e.g., any configured forward proxy) on the path. The serialization format therefore needs to include security measures such as the following:

- o A client SHOULD protect the integrity of the state information serialized in a token.
- o Even when the integrity of the serialized state is protected, an attacker may still replay a response, making the client believe it sent the same request twice. For this reason, the client SHOULD implement replay protection (e.g., by using sequence numbers and a replay window). For replay protection, integrity protection is REQUIRED.
- o If processing a response without keeping request state is sensitive to the time elapsed since sending the request, then the client SHOULD include freshness information (e.g., a timestamp) in the serialized state and reject any response where the freshness information is insufficiently fresh.

- o Information in the serialized state may be privacy sensitive. A client SHOULD encrypt the serialized state if it contains privacy sensitive information that an attacker would not get otherwise.
- o When a client changes the format of the serialized state, it SHOULD prevent false interoperability with the previous format (e.g., by changing the key used for integrity protection or changing a field in the serialized state).

3.2. Using Extended Tokens

A client that depends on support for extended token lengths (Section 2) from the server to avoid keeping request state needs to perform a discovery of support (Section 2.2) before it can be stateless.

This discovery MUST be performed in a stateful way, i.e., keeping state for the request (Figure 4): If the client was stateless from the start and the server does not support extended tokens, then any error message could not be processed since the state would neither be present at the client nor returned in the Reset message (Figure 5).

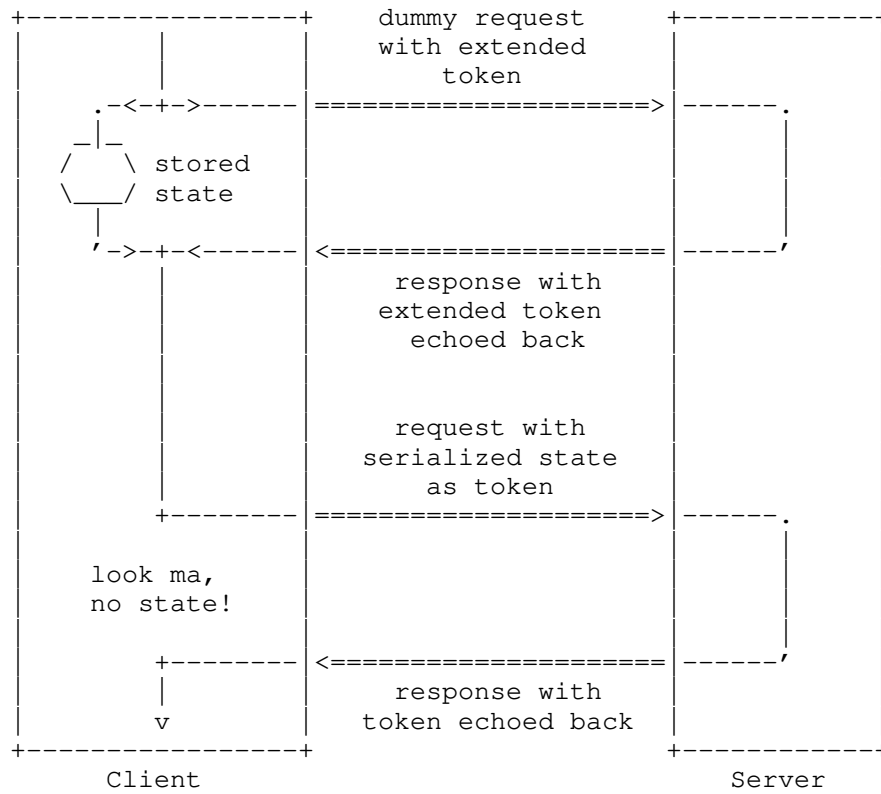


Figure 4: Depending on Extended Tokens for Being Stateless First Requires a Successful Stateful Discovery of Support

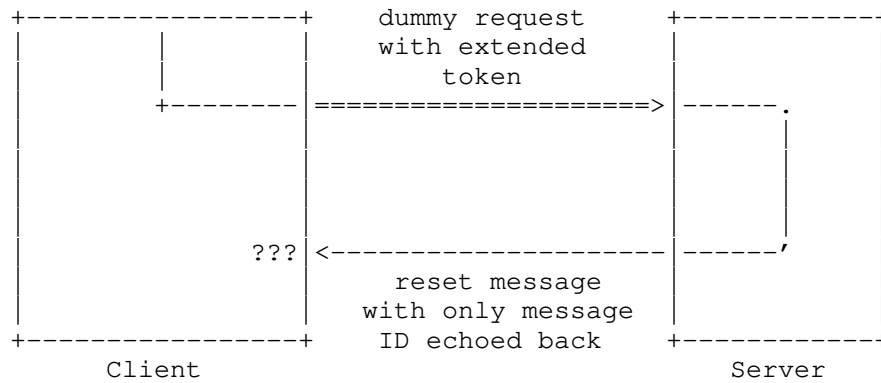


Figure 5: Stateless Discovery of Support Does Not Work

In environments where support can be reliably discovered through some other means, the discovery of support is OPTIONAL. An example for this is the Constrained Join Protocol (CoJP) in a 6TiSCH network [I-D.ietf-6tisch-minimal-security], where support for extended tokens is required from all relevant parties.

3.3. Transmitting Messages

In CoAP over UDP [RFC7252], a client has the choice between Confirmable and Non-confirmable messages for requests. When using Non-confirmable messages, a client does not have to keep any message exchange state, which can help in the goal of avoiding state. When using Confirmable messages, a client needs to keep message exchange state for performing retransmissions and handling Acknowledgement and Reset messages, however. Non-confirmable messages are therefore better suited for avoiding state. In any case, a client still needs to keep congestion control state, i.e., maintain state for each node it communicates with and enforce limits like NSTART.

As per Section 5.2 of RFC 7252, a client must be prepared to receive a response as a piggybacked response, a separate response, or Non-confirmable response, regardless of the message type used for the request. A stateless client MUST handle these response types as follows:

- o If a piggybacked response passes the checks for token integrity and freshness (Section 3.1), the client processes the message as specified in RFC 7252; otherwise, it processes the acknowledgement portion of the message as specified in RFC 7252 and silently discards the response portion.
- o If a separate response passes the checks for token integrity and freshness, the client processes the message as specified in RFC 7252; otherwise, it rejects the message as specified in Section 4.2 of RFC 7252.
- o If a Non-confirmable response passes the checks for token integrity and freshness, the client processes the message as specified in RFC 7252; otherwise, it rejects the message as specified in Section 4.3 of RFC 7252.

4. Stateless Intermediaries

Tokens are a hop-by-hop feature: If a client makes a request to an intermediary, that intermediary needs to store the client's token (along with the client's transport address) while it makes its own request towards the origin server and waits for the response. When the intermediary receives the response, it looks up the client's

token and transport address for the received request and sends an appropriate response to the client.

An intermediary might want to be "stateless" not only in its role as a client but also in its role as a server, i.e., be alleviated of storing the client information for the requests it receives.

Such an intermediary can be implemented by serializing the client information along with the request state into the token towards the origin server. When the intermediary receives the response, it can recover the client information from the token and use it to satisfy the client's request and therefore doesn't need to store it itself.

The following subsections discuss some considerations for this approach.

4.1. Observing Resources

One drawback of the approach is that an intermediary, without keeping request state, is unable to aggregate multiple requests for the same target resource, which can significantly reduce efficiency. In particular, when clients observe [RFC7641] the same resource, aggregating requests is REQUIRED (Section 3.1 of RFC 7641). This requirement cannot be satisfied without keeping request state.

Furthermore, an intermediary that does not keep track of the clients observing a resource is not able to determine whether these clients are still interested in receiving further notifications (Section 3.5 of RFC 7641) or want to cancel an observation (Section 3.6 of RFC 7641).

Therefore, an intermediary MUST NOT include an Observe Option in requests it sends without keeping both the request state for the requests it sends and the client information for the requests it receives.

4.2. Block-Wise Transfers

When using block-wise transfers [RFC7959], a server might not be able to distinguish blocks originating from different clients once they have been forwarded by an intermediary. Intermediaries need to ensure that this does not lead to inconsistent resource state by keeping distinct block-wise request operations on the same resource apart, e.g., utilizing the Request-Tag Option [I-D.ietf-core-echo-request-tag].

4.3. Gateway Timeouts

As per Section 5.7.1 of RFC 7252, an intermediary is REQUIRED to return a 5.04 (Gateway Timeout) response if it cannot obtain a response within a timeout. However, if an intermediary does not keep the client information for the requests it receives, it cannot return such a response. Therefore, in this case, the gateway cannot return such a response and as such cannot implement such a timeout.

4.4. Extended Tokens

A client may make use of extended token lengths in a request to an intermediary that wants to be "stateless". This means that such an intermediary may have to serialize potentially very large client information into its token towards the origin server. The tokens can grow even further when it progresses along a chain of intermediaries that all want to be "stateless".

Intermediaries SHOULD limit the size of client information they are serializing into their own tokens. An intermediary can do this, for example, by limiting the extended token lengths it accepts from its clients (see Section 2.2) or by keeping the client information locally when the client information exceeds the limit (i.e., not being "stateless").

5. Security Considerations

5.1. Extended Tokens

Tokens significantly larger than the 8 bytes specified in RFC 7252 have implications in particular for nodes with constrained memory size that need to be mitigated. A node in the server role supporting extended token lengths may be vulnerable to a denial-of-service when an attacker (either on-path or a malicious client) sends large tokens to fill up the memory of the node. Implementations need to be prepared to handle such messages.

5.2. Stateless Clients and Intermediaries

Transporting the state needed by a client to process a response as serialized state information in the token has several significant and non-obvious security and privacy implications that need to be mitigated; see Section 3.1 for recommendations.

In addition to the format requirements outlined there, implementations need to ensure that they are not vulnerable to maliciously crafted, delayed, or replayed tokens.

It is generally expected that the use of encryption, integrity protection, and replay protection for serialized state is appropriate.

In the absence of integrity and replay protection, an on-path attacker or rogue server/intermediary could return a state (either one modified in a reply, or an unsolicited one) that could alter the internal state of the client.

It is for this reason that at least the use of integrity protection on the token is always recommended.

It maybe that in some very specific case, as a result of a careful and detailed analysis of any potential attacks, that there may be cases where such cryptographic protections do not add value. The authors of this document have not found such a use case as yet, but this is a local decision.

It should further be emphasized that the encrypted state is created by the sending node, and decrypted by the same node when receiving a response. The key is not shared with any other system. Therefore the choice of encryption scheme and the generation of the key for this system is purely a local matter.

When encryption is used, the use of AES-CCM [RFC3610] with a 64-bit tag is recommended, combined with a sequence number and a replay window. This choice is informed by available hardware acceleration of on many constrained systems. If a different algorithm is available accelerated on the sender, with similar or stronger strength, then it SHOULD be preferred. Where privacy of the state is not required, and encryption is not needed, HMAC-SHA-256 [RFC6234], combined with a sequence number and a replay window, may be used.

This size of the replay window depends upon the number of requests that need to be outstanding. This can be determined from the rate at which new ones are made, and the expected duration in which responses are expected.

For instance, given a CoAP MAX_TRANSMIT_WAIT of 93 s (Section 4.8.2 of [RFC7252], any request that is not answered within 93 s will be considered to have failed. At a request rate of one request per 10 s, at most 10 ($\text{ceil}(9.3)$) requests can be outstanding at a time, and any convenient replay window larger than 20 will work. As replay windows are often implemented with a sliding window and a bit, the use of a 32-bit window would be sufficient.

For use cases where requests are being relayed from another node, the request rate may be estimated by the total link capacity allocated

for that kind of traffic. An alternate view would consider how many IPv6 Neighbor Cache Entries (NCEs) the system can afford to allocate for this use.

When using an encryption mode that depends on a nonce, such as AES-CCM, repeated use of the same nonce under the same key causes the cipher to fail catastrophically.

If a nonce is ever used for more than one encryption operation with the same key, then the same key stream gets used to encrypt both plaintexts and the confidentiality guarantees are voided. Devices with low-quality entropy sources -- as is typical with constrained devices, which incidentally happen to be a natural candidate for the stateless mechanism described in this document -- need to carefully pick a nonce generation mechanism that provides the above uniqueness guarantee.

[RFC8613] appendix B.1.1 ("Sender Sequence Number") provides a model for how to maintain non-repeating nonces without causing excessive wear of flash memory.

6. IANA Considerations

6.1. CoAP Signaling Option Number

The following entries are added to the "CoAP Signaling Option Numbers" registry within the "CoRE Parameters" registry.

Applies to	Number	Name	Reference
7.01	TBD	Extended-Token-Length	[[this document]]

[[NOTE TO RFC EDITOR: Please replace "TBD" in this section and in Table 1 with the code point assigned by IANA.]]

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>>.

- [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>>.
- [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>>.
- [RFC8323] Bormann, C., Lemay, S., Tschafenig, 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>>.

7.2. Informative References

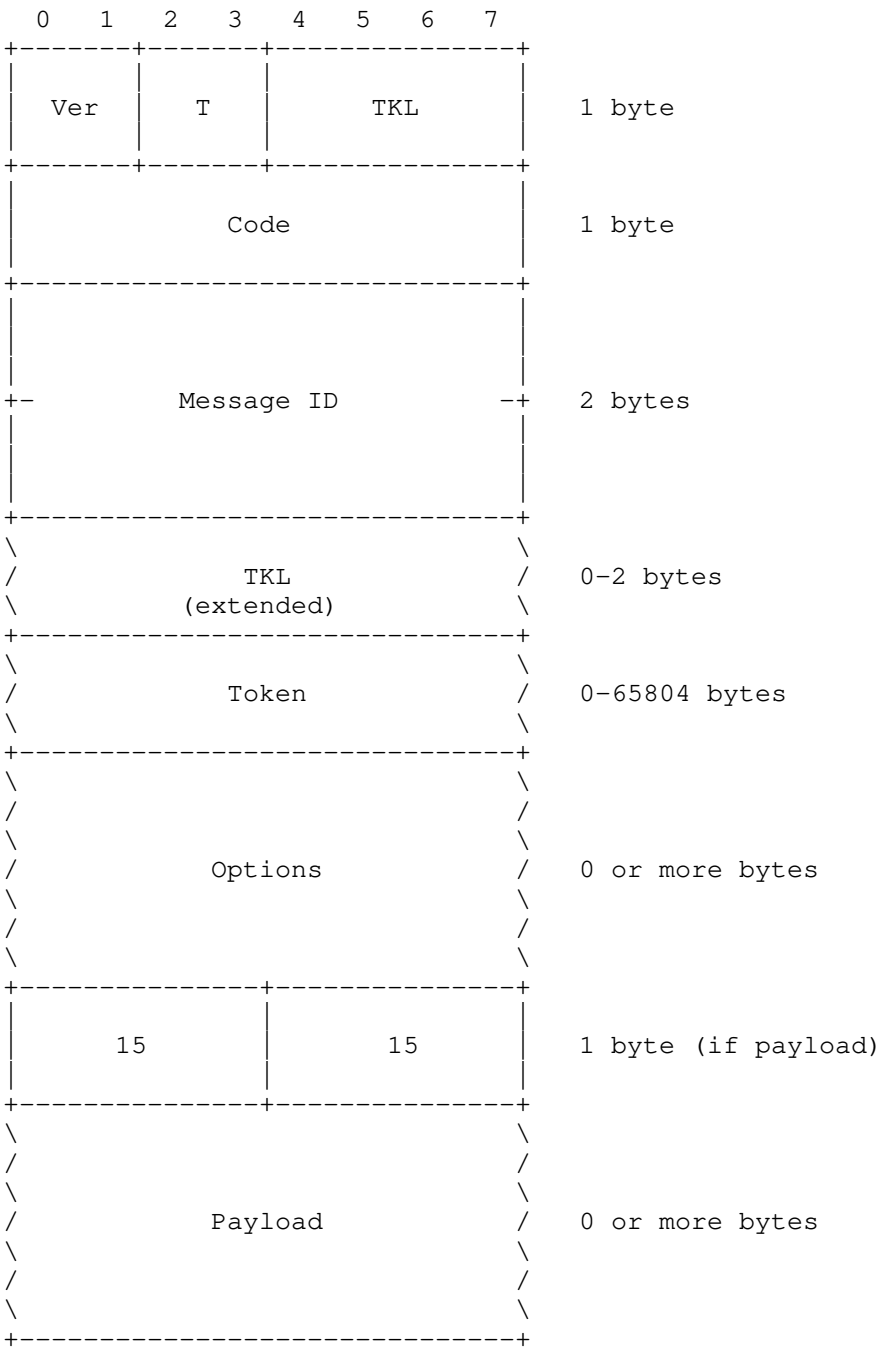
- [I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson,
"Constrained Join Protocol (CoJP) for 6TiSCH", draft-ietf-6tisch-minimal-security-15 (work in progress), December 2019.
- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-11 (work in progress), November 2020.
- [RFC3610] Whiting, D., Housley, R., and N. Ferguson, "Counter with CBC-MAC (CCM)", RFC 3610, DOI 10.17487/RFC3610, September 2003, <<https://www.rfc-editor.org/info/rfc3610>>.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<https://www.rfc-editor.org/info/rfc6234>>.

- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

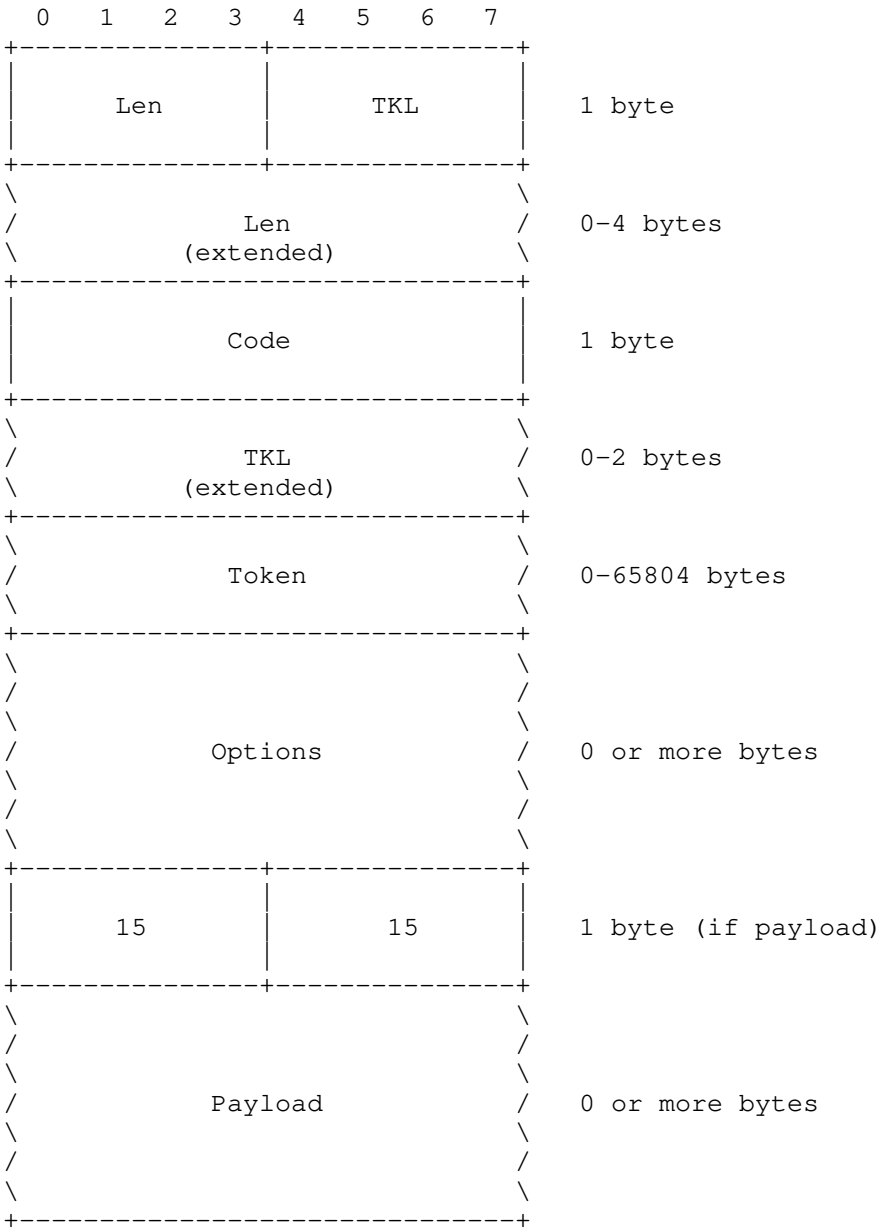
Appendix A. Updated Message Formats

In Section 2, this document updates the CoAP message formats by specifying a new definition of the TKL field in the message header. As an alternative presentation of this update, this appendix shows the CoAP message formats for CoAP over UDP [RFC7252] and CoAP over TCP, TLS, and WebSockets [RFC8323] with the new definition applied.

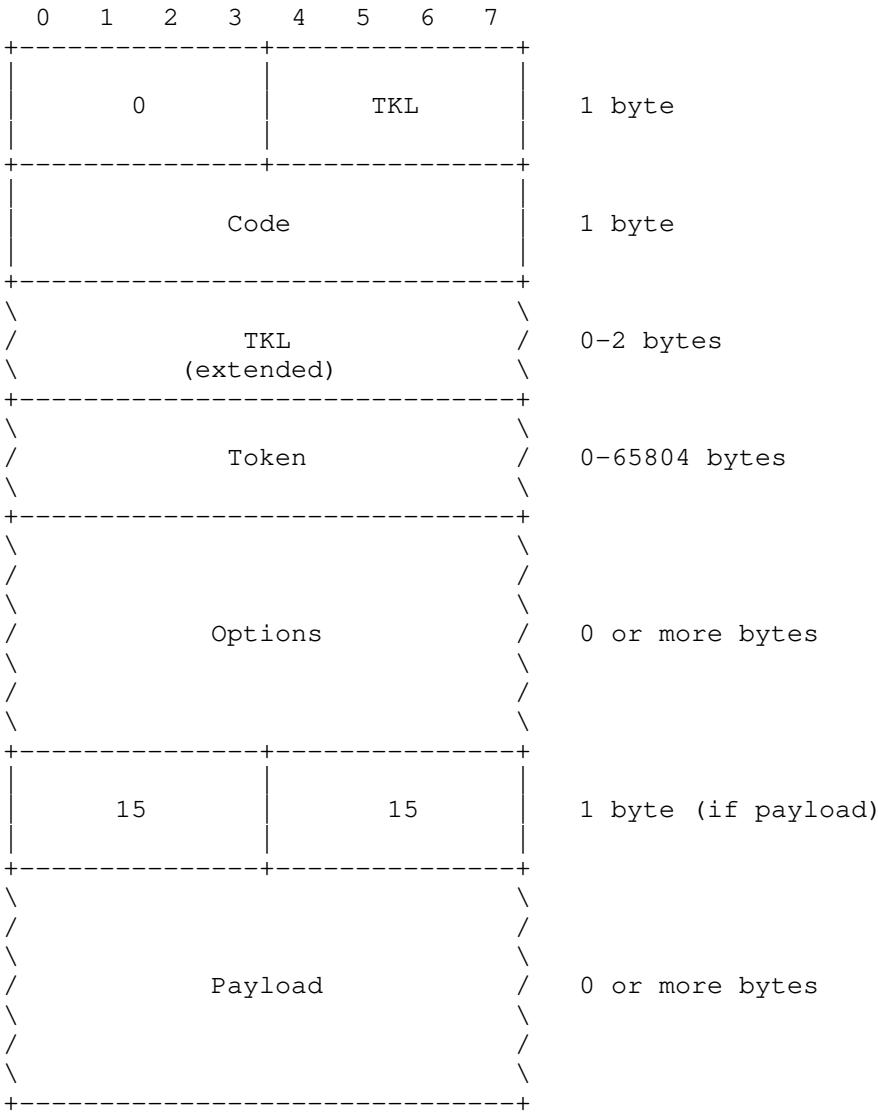
A.1. CoAP over UDP



A.2. CoAP over TCP/TLS



A.3. CoAP over WebSockets



Acknowledgements

This document is based on the requirements of and work on the Minimal Security Framework for 6TiSCH [I-D.ietf-6tisch-minimal-security] by Malisa Vucinic, Jonathan Simon, Kris Pister, and Michael Richardson.

Thanks to Christian Amsuss, Carsten Bormann, Roman Danyliw, Christer Holmberg, Benjamin Kaduk, Ari Keranen, Erik Kline, Murray Kucherawy, Warren Kumari, Barry Leiba, David Mandelberg, Dan Romascanu, Jim Schaad, Goran Selander, Malisa Vucinic, Eric Vyncke, and Robert Wilton for helpful comments and discussions that have shaped the document.

Special thanks to John Mattsson for his contributions to the security considerations of the document, and to Thomas Fossati for his in-depth review, copious comments, and suggested text.

Authors' Addresses

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

Michael C. Richardson
Sandelman Software Works

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 6 May 2021

F. Palombini
Ericsson
M. Tiloca
R. Hoeglund
RISE AB
S. Hristozov
Fraunhofer AISEC
G. Selander
Ericsson
2 November 2020

Combining EDHOC and OSCORE
draft-palombini-core-oscore-edhoc-01

Abstract

This document defines possible optimization approaches for combining the lightweight authenticated key exchange protocol EDHOC run over CoAP with the first subsequent OSCORE transaction. This combination reduces the number of round trips required to set up an OSCORE Security Context and complete an OSCORE transaction using that context.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at
<https://github.com/EricssonResearch/oscore-edhoc>
(<https://github.com/EricssonResearch/oscore-edhoc>).

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 6 May 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Background	3
3. EDHOC in OSCORE	5
3.1. Signalling in a New EDHOC Option	6
3.2. Signalling in the OSCORE Option	8
4. Security Considerations	9
5. IANA Considerations	9
6. Normative References	9
Acknowledgments	10
Authors' Addresses	10

1. Introduction

This document presents possible optimization approaches to combine the lightweight authenticated key exchange protocol EDHOC [I-D.ietf-lake-edhoc], when running over CoAP [RFC7252], with the first subsequent OSCORE [RFC8613] transaction.

This allows for a minimum number of round trips necessary to setup the OSCORE Security Context and complete an OSCORE transaction, for example when an IoT device gets configured in a network for the first time.

The number of protocol round trips impacts the minimum number of flights, which can have a substantial impact on performance with certain radio technologies.

Without this optimization, it is not possible, not even in theory, to achieve the minimum number of flights. This optimization makes it possible also in practice, since the last message of the EDHOC protocol can be made relatively small (see Section 1 of [I-D.ietf-lake-edhoc]), thus allowing additional OSCORE protected CoAP data within target MTU sizes.

The goal of this document is to provide details on different alternatives for transporting and processing the necessary data, gather opinions on the different approaches, and select only one of those.

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.

The reader is expected to be familiar with terms and concepts defined in CoAP [RFC7252], CBOR [I-D.ietf-cbor-7049bis], OSCORE [RFC8613] and EDHOC [I-D.ietf-lake-edhoc].

2. Background

EDHOC is a 3-message key exchange protocol. Section 7.1 of [I-D.ietf-lake-edhoc] specifies how to transport EDHOC over CoAP: the EDHOC data (referred to as "EDHOC messages") are transported in the payload of CoAP requests and responses.

This draft deals with the case of the Initiator acting as CoAP Client and the Responder acting as CoAP Server. (The case of the Initiator acting as CoAP server cannot be optimized in this way.) That is, the CoAP Client sends a POST request containing the EDHOC message 1 to a reserved resource at the CoAP Server. This triggers the EDHOC exchange on the CoAP Server, which replies with a 2.04 (Changed) Response containing the EDHOC message 2. Finally, the EDHOC message 3 is sent by the CoAP Client in a CoAP POST request to the same resource used for the EDHOC message 1. The Content-Format of these CoAP messages is set to "application/edhoc".

After this exchange takes place, and after successful verifications specified in the EDHOC protocol, the Client and Server derive the OSCORE Security Context, as specified in Section 7.1.1 of [I-D.ietf-lake-edhoc]. Then, they are ready to use OSCORE.

This sequential way of running EDHOC and then OSCORE is specified in Figure 1. As shown in the figure, this mechanism is executed in 3 round trips.

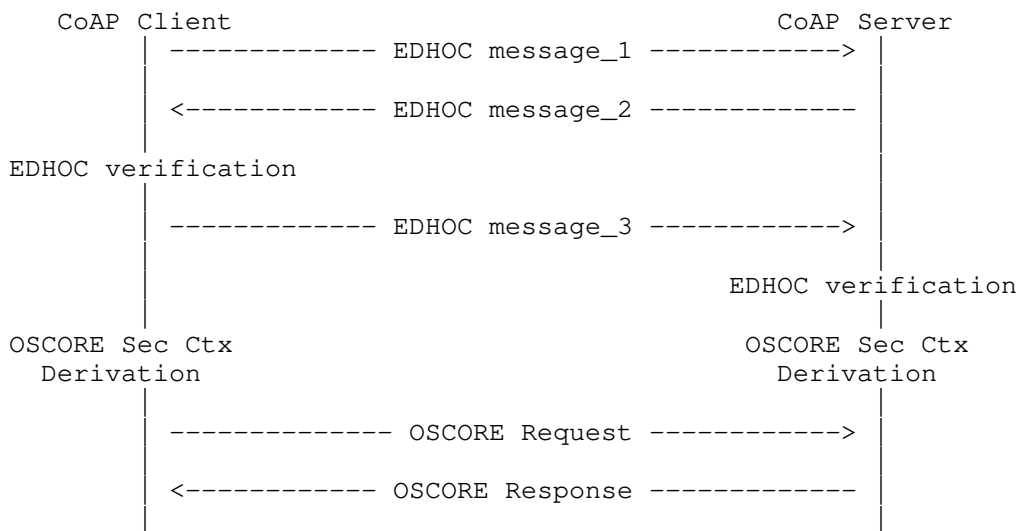


Figure 1: EDHOC and OSCORE run sequentially

The number of roundtrips can be minimized: after receiving the EDHOC message 2, the CoAP Client has all the information needed to derive the OSCORE Security Context before sending the EDHOC message 3.

This means that the Client can potentially send at the same time both the EDHOC message 3 and the subsequent OSCORE Request. On a semantic level, this approach practically requires to send two separate REST requests at the same time.

The high level message flow of running EDHOC and OSCORE combined is shown in Figure 2.

Defining the specific details of how to transport the data and of their processing order is the goal of this specification.

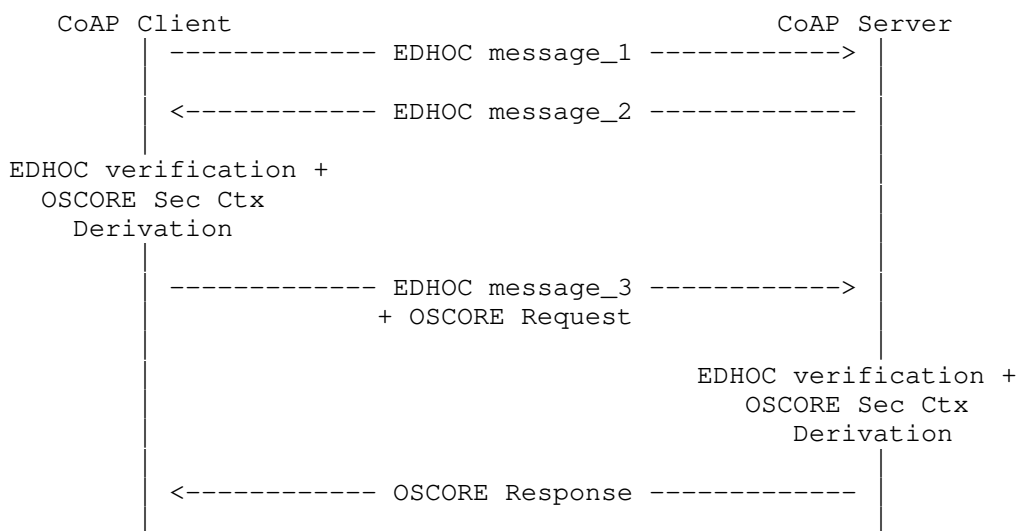


Figure 2: EDHOC and OSCORE combined

3. EDHOC in OSCORE

This approach consists in sending the EDHOC message 3 inside an OSCORE message (i.e., an OSCORE protected CoAP message).

The resulting OSCORE + EDHOC request is in practice the OSCORE Request from Figure 1, sent to a protected resource and with the correct CoAP method and options, with the addition that it also transports the EDHOC message 3.

As the EDHOC message 3 may be too large to be included in a CoAP Option, e.g. if containing a large public key certificate chain, it would have to be transported in the CoAP payload.

In particular, the payload of the OSCORE + EDHOC request is formatted as a CBOR sequence of two CBOR byte strings: the EDHOC message 3 and the OSCORE ciphertext of the original OSCORE Request, in this order, both encoded as CBOR byte strings.

Note that the OSCORE ciphertext is not computed over the EDHOC message 3, which is not protected by OSCORE. That is, the client first prepares the OSCORE Request as in Figure 1. Then, it reformats the payload to include also the EDHOC message 3, as defined above. The result is the OSCORE + EDHOC request to send.

The usage of this approach is indicated by a signalling information in the OSCORE + EDHOC request, which can be either a new EDHOC Option (see Section 3.1) or the OSCORE Option with a particular Flag Bit set (see Section 3.2).

When receiving such a request, the Server needs to perform the following processing, in addition to the EDHOC, OSCORE and CoAP processing:

1. Check the signalling information to identify that this is an OSCORE + EDHOC request.
2. Extract the EDHOC message 3 from the payload of the OSCORE + EDHOC request, as the value of the first CBOR byte string in the CBOR sequence.
3. Execute the EDHOC processing on the EDHOC message 3, including verifications and the OSCORE Security Context derivation.
4. Extract the OSCORE ciphertext from the payload of the OSCORE + EDHOC request, as the value of the second CBOR byte string in the CBOR sequence. Then, set the CoAP payload of the request to the extracted ciphertext.
5. Decrypt and verify the OSCORE protected CoAP request resulting from step 4, as defined by OSCORE.
6. Process the CoAP request resulting from step 5.

The following sections expand on the two ways of signalling that the EDHOC message is transported in the OSCORE message.

3.1. Signalling in a New EDHOC Option

One way to signal that the Server has to extract and process the EDHOC message 3 before processing the OSCORE protected CoAP request is to define a new CoAP Option, called the EDHOC Option.

The presence of this option means that the message contains EDHOC data in the payload, that must be extracted and processed before the rest of the message can be processed.

In particular, the EDHOC message 3 has to be extracted from the CoAP payload, as the first element of a CBOR sequence wrapped in a CBOR byte string.

The Option is critical, Safe-to-Forward, and part of the Cache-Key.

The Option value is always empty. If any value is sent, the value is simply ignored.

The Option MUST occur at most once.

The Option is of Class U for OSCORE.

Figure 3 shows the format for a CoAP message containing both the OSCORE ciphertext and EDHOC message 3, using the newly defined EDHOC option for signaling.

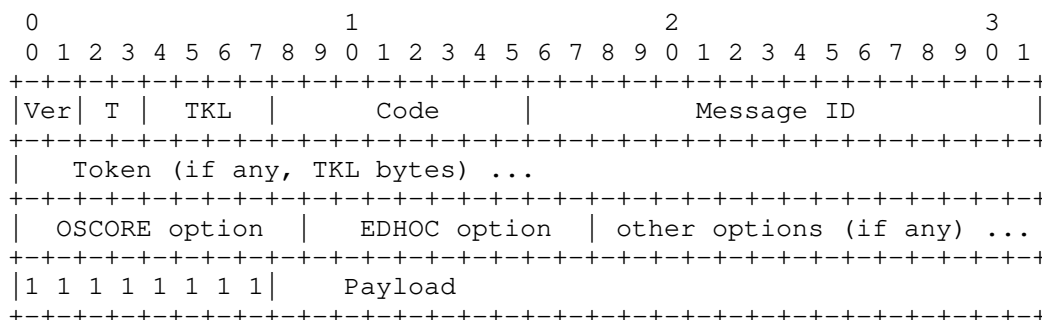


Figure 3: CoAP message for EDHOC and OSCORE combined - signaled with the EDHOC Option

An example based on the OSCORE test vector from Appendix C.4 of [RFC8613] and the EDHOC test vector from Appendix B.2 of [I-D.ietf-lake-edhoc] is given in Figure 4. The example assumes that the EDHOC option is registered with CoAP option number 13.

- o OSCORE option value: 0x0914 (2 bytes)
- o ciphertext: 0x612f1092f1776f1c1668b3825e (13 bytes)
- o EDHOC option value: - (0 bytes)
- o EDHOC message 3: 085253c3991999a5ffb86921e99b607c067770e0 (20 bytes)

From there:

- o Protected CoAP request (OSCORE message): 0x44025d1f00003974396c6f63616c686f737462 0914 04 ff 54085253C3991999A5FFB86921E99B607C067770E0 4d612f1092f1776f1c1668b3825e (58 bytes)

Figure 4: CoAP message for EDHOC and OSCORE combined - signaled with the EDHOC Option

3.2. Signalling in the OSCORE Option

Another way to signal that the EDHOC message 3 is to be extracted from the CoAP payload as the first element of a CBOR sequence wrapped in a CBOR byte string, and that the processing defined in Section 3 is to be executed, is to use one of the OSCORE Flag Bits of the OSCORE Option.

Bit Position: 1

Name: EDHOC

Description: Set to 1 if the payload is a sequence of EDHOC message 3 and OSCORE ciphertext.

Reference: this document

The OSCORE Option value with the EDHOC bit set is given in Figure 5.

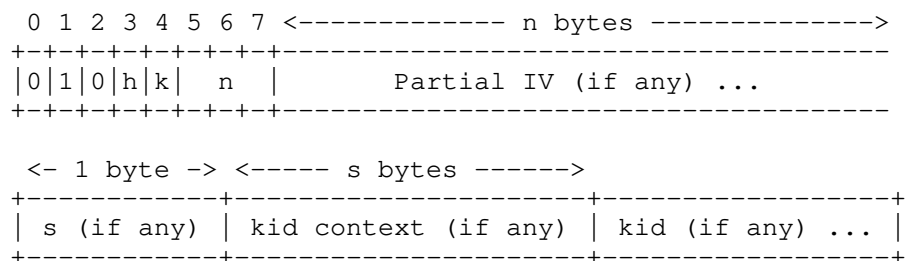


Figure 5: The OSCORE Option Value with the EDHOC bit set

Figure 6 shows the format for a CoAP message containing both the OSCORE ciphertext and EDHOC message 3, using the Flag Bit 1 in the OSCORE Option for signaling.

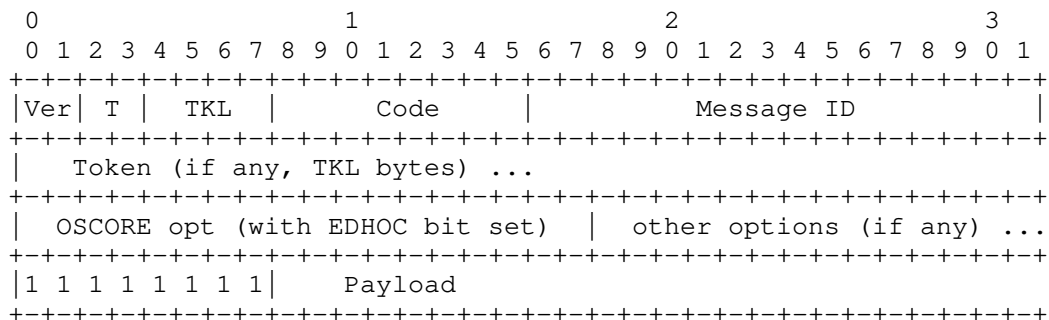


Figure 6: CoAP message for EDHOC and OSCORE combined - signaled within the OSCORE option

An example based on the OSCORE test vector from Appendix C.4 of [RFC8613] and the EDHOC test vector from Appendix B.2 of [I-D.ietf-lake-edhoc] is given in Figure 7.

- o OSCORE option value without EDHOC bit set: 0x0914 (2 bytes)
- o OSCORE option value with EDHOC bit set: 0x4914 (2 bytes)
- o ciphertext: 0x612f1092f1776f1c1668b3825e (13 bytes)
- o EDHOC message 3: 085253c3991999a5ffb86921e99b607c067770e0 (20 bytes)

From there:

- o Protected CoAP request (OSCORE message): 0x44025d1f00003974396c6f63616c686f737462 4914 ff 54085253c3991999a5ffb86921e99b607c067770e0 4d612f1092f1776f1c1668b3825e (58 bytes)

Figure 7: CoAP message for EDHOC and OSCORE combined - signaled within the OSCORE Option

4. Security Considerations

The same security considerations from OSCORE [RFC8613] and EDHOC [I-D.ietf-lake-edhoc] hold for this document.

TODO (more considerations)

5. IANA Considerations

Depending on the option chosen, this document will either register a new CoAP Option number to the CoAP Option Number registry, or a new bit to the OSCORE Flag Bits registry.

6. Normative References

[I-D.ietf-cbor-7049bis]

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", Work in Progress, Internet-Draft, draft-ietf-cbor-7049bis-16, 30 September 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-cbor-7049bis-16.txt>>.

- [I-D.ietf-lake-edhoc]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-edhoc-01, 2 August 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-lake-edhoc-01.txt>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [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>>.
- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

Acknowledgments

The authors sincerely thank Christian Amsuess, Klaus Hartke, Jim Schaad and Malisa Vucinic for their feedback and comments in the discussion leading up to this draft.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Francesca Palombini
Ericsson

Email: francesca.palombini@ericsson.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden

Email: marco.tiloca@ri.se

Rikard Hoeglund
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden

Email: rikard.hoglund@ri.se

Stefan Hristozov
Fraunhofer AISEC

Email: stefan.hristozov@aisec.fraunhofer.de

Goeran Selander
Ericsson

Email: goran.selander@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 23, 2021

F. Palombini
Ericsson
M. Tiloca
R. Hoeglund
RISE AB
S. Hristozov
Fraunhofer AISEC
G. Selander
Ericsson
February 19, 2021

Combining EDHOC and OSCORE
draft-palombini-core-oscore-edhoc-02

Abstract

This document defines an optimization approach for combining the lightweight authenticated key exchange protocol EDHOC run over CoAP with the first subsequent OSCORE transaction. This combination reduces the number of round trips required to set up an OSCORE Security Context and to complete an OSCORE transaction using that Security Context.

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 23, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Background	3
3. EDHOC Option	5
4. EDHOC Combined with OSCORE	6
4.1. Client Processing	6
4.2. Server Processing	7
5. Example of EDHOC + OSCORE Request	9
6. Security Considerations	10
7. IANA Considerations	10
7.1. CoAP Option Numbers Registry	10
8. Normative References	10
Acknowledgments	11
Authors' Addresses	11

1. Introduction

This document defines an optimization approach to combine the lightweight authenticated key exchange protocol EDHOC [I-D.ietf-lake-edhoc], when running over CoAP [RFC7252], with the first subsequent OSCORE [RFC8613] transaction.

This allows for a minimum number of round trips necessary to setup the OSCORE Security Context and complete an OSCORE transaction, for example when an IoT device gets configured in a network for the first time.

This optimization is desirable, since the number of protocol round trips impacts the minimum number of flights, which in turn can have a substantial impact on the latency of conveying the first OSCORE request, when using certain radio technologies.

Without this optimization, it is not possible, not even in theory, to achieve the minimum number of flights. This optimization makes it possible also in practice, since the last message of the EDHOC protocol can be made relatively small (see Section 1 of [I-D.ietf-lake-edhoc]), thus allowing additional OSCORE protected CoAP data within target MTU sizes.

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.

The reader is expected to be familiar with terms and concepts defined in CoAP [RFC7252], CBOR [RFC8949], CBOR sequences [RFC8742], OSCORE [RFC8613] and EDHOC [I-D.ietf-lake-edhoc].

2. Background

EDHOC is a 3-message key exchange protocol. Section 7.2 of [I-D.ietf-lake-edhoc] specifies how to transport EDHOC over CoAP: the EDHOC data (referred to as "EDHOC messages") are transported in the payload of CoAP requests and responses.

This draft deals with the case of the Initiator acting as CoAP Client and the Responder acting as CoAP Server; instead, the case of the Initiator acting as CoAP Server cannot be optimized by using this approach.

That is, the CoAP Client sends a POST request containing EDHOC message_1 to a reserved resource at the CoAP Server. This triggers the EDHOC exchange on the CoAP Server, which replies with a 2.04 (Changed) Response containing EDHOC message_2. Finally, the CoAP Client sends EDHOC message_3, as a CoAP POST request to the same resource used for EDHOC message_1. The Content-Format of these CoAP messages may be set to "application/edhoc".

After this exchange takes place, and after successful verifications specified in the EDHOC protocol, the Client and Server derive the OSCORE Security Context, as specified in Section 7.2.1 of [I-D.ietf-lake-edhoc]. Then, they are ready to use OSCORE.

This sequential way of running EDHOC and then OSCORE is specified in Figure 1. As shown in the figure, this mechanism takes 3 round trips to complete.

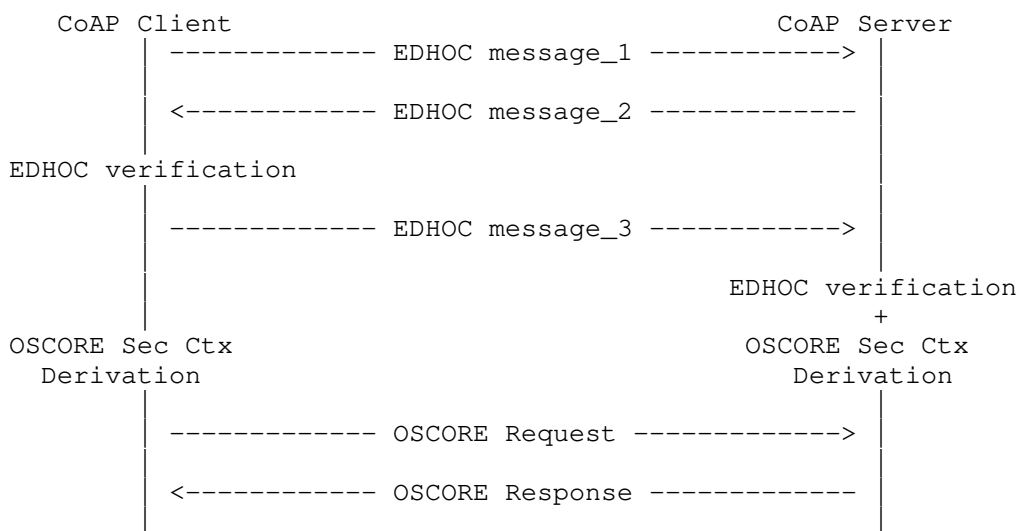


Figure 1: EDHOC and OSCORE run sequentially

The number of roundtrips can be minimized as follows. Already after receiving EDHOC message_2 and before sending EDHOC message_3, the CoAP Client has all the information needed to derive the OSCORE Security Context.

This means that the Client can potentially send at the same time both EDHOC message_3 and the subsequent OSCORE Request. On a semantic level, this approach practically requires to send two separate REST requests at the same time.

The high level message flow of running EDHOC and OSCORE combined is shown in Figure 2.

Defining the specific details of how to transport the data and of their processing order is the goal of this specification, as defined in Section 4.

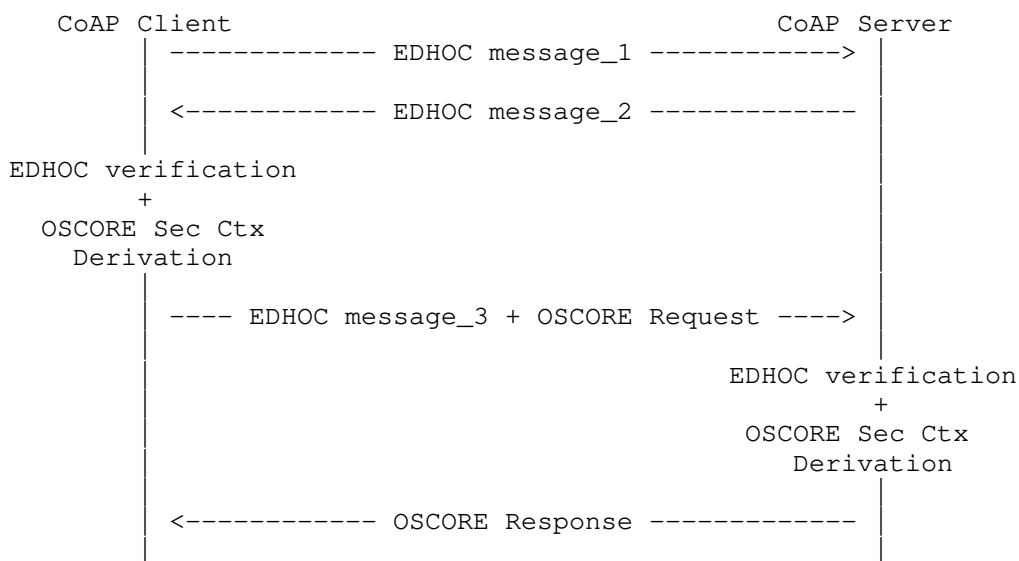


Figure 2: EDHOC and OSCORE combined

3. EDHOC Option

This section defines the EDHOC Option, used in a CoAP request to signal that the request combines EDHOC message_3 and OSCORE protected data.

The EDHOC Option has the properties summarized in Figure 3, which extends Table 4 of [RFC7252]. The option is Critical, Safe-to-Forward, and part of the Cache-Key. The option MUST occur at most once and is always empty. If any value is sent, the value is simply ignored. The option is intended only for CoAP requests and is of Class U for OSCORE [RFC8613].

No.	C	U	N	R	Name	Format	Length	Default
TBD13	x				EDHOC	Empty	0	(none)

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

Figure 3: The EDHOC Option.

The presence of this option means that the message payload contains also EDHOC data, that must be extracted and processed as defined in Section 4.2, before the rest of the message can be processed.

Figure 4 shows the format of a CoAP message containing both the EDHOC data and the OSCORE ciphertext, using the newly defined EDHOC option for signalling.

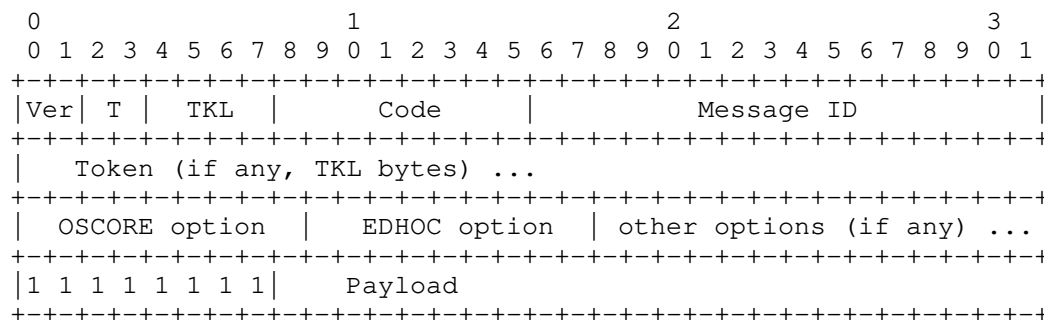


Figure 4: CoAP message for EDHOC and OSCORE combined - signalled with the EDHOC Option

4. EDHOC Combined with OSCORE

The approach defined in this specification consists of sending EDHOC message_3 inside an OSCORE protected CoAP message.

The resulting EDHOC + OSCORE request is in practice the OSCORE Request from Figure 1, sent to a protected resource and with the correct CoAP method and options, with the addition that it also transports EDHOC message_3.

Since EDHOC message_3 may be too large to be included in a CoAP Option, e.g. if containing a large public key certificate chain, it has to be transported through the CoAP payload.

The use of this approach is explicitly signalled by including an EDHOC Option (see Section 3) in the EDHOC + OSCORE request.

4.1. Client Processing

The Client prepares an EDHOC + OSCORE request as follows.

1. Compose EDHOC message_3 as per Section 5.4.2 of [I-D.ietf-lake-edhoc].

Since the Client is the EDHOC Initiator and the used Correlation Method is 1 (see Section 3.2.4 of [I-D.ietf-lake-edhoc]), the EDHOC message_3 always includes the Connection Identifier C_R and CIPHERTEXT_3. Note that C_R is the OSCORE Sender ID of the

Client, encoded as a `bstr_identifier` (see Section 5.1 of [I-D.ietf-lake-edhoc]).

2. Encrypt the original CoAP request as per Section 8.1 of [RFC8613], using the new OSCORE Security Context established after receiving EDHOC message_2.

Note that the OSCORE ciphertext is not computed over EDHOC message_3, which is not protected by OSCORE. That is, the result of this step is the OSCORE Request as in Figure 1.

3. Build a CBOR sequence [RFC8742] composed of two CBOR byte strings in the following order.
 - * The first CBOR byte string is the CIPHERTEXT_3 of the EDHOC message_3 resulting from step 3.
 - * The second CBOR byte string has as value the OSCORE ciphertext of the OSCORE protected CoAP request resulting from step 2.
4. Compose the EDHOC + OSCORE request, as the OSCORE protected CoAP request resulting from step 2, where the payload is replaced with the CBOR sequence built at step 3.
5. Signal the usage of this approach within the EDHOC + OSCORE request, by including the new EDHOC Option defined in Section 3.

4.2. Server Processing

When receiving an EDHOC + OSCORE request, the Server performs the following steps.

1. Check the presence of the EDHOC option defined in Section 3, to determine that the received request is an EDHOC + OSCORE request. If this is the case, the Server continues with the steps defined below.
2. Extract CIPHERTEXT_3 from the payload of the EDHOC + OSCORE request, as the first CBOR byte string in the CBOR sequence.
3. Rebuild EDHOC message_3, as a CBOR sequence composed of two CBOR byte strings in the following order.
 - * The first CBOR byte string is the 'kid' of the Client indicated in the OSCORE option of the EDHOC + OSCORE request, encoded as a `bstr_identifier` (see Section 5.1 of [I-D.ietf-lake-edhoc]).

- * The second CBOR byte string is the CIPHERTEXT_3 retrieved at step 2.
- 4. Perform the EDHOC processing on the EDHOC message_3 rebuilt at step 3, including verifications, and the OSCORE Security Context derivation, as per Section 5.4.3 and Section 7.2.1 of [I-D.ietf-lake-edhoc], respectively.
- 5. Extract the OSCORE ciphertext from the payload of the EDHOC + OSCORE request, as the value of the second CBOR byte string in the CBOR sequence.
- 6. Rebuild the OSCORE protected CoAP request as the EDHOC + OSCORE request, where the payload is replaced with the OSCORE ciphertext resulting from step 5.
- 7. Decrypt and verify the OSCORE protected CoAP request resulting from step 6, as per Section 8.2 of [RFC8613], by using the new OSCORE Security Context established at step 4.
- 8. Process the CoAP request resulting from step 7.

If steps 4 (EDHOC processing) and 7 (OSCORE processing) are both successfully completed, the Server MUST reply with an OSCORE protected response, in order for the Client to achieve key confirmation (see Section 5.4.2 of [I-D.ietf-lake-edhoc]). The usage of EDHOC message_4 as defined in Section 7.1 of [I-D.ietf-lake-edhoc] is not applicable to the approach defined in this specification.

If step 4 (EDHOC processing) fails, the server discontinues the protocol as per Section 5.4.3 of [I-D.ietf-lake-edhoc] and sends an EDHOC error message, formatted as defined in Section 6.1 of [I-D.ietf-lake-edhoc]. In particular, the CoAP response conveying the EDHOC error message:

- o MUST have Content-Format set to application/edhoc defined in Section 9.5 of [I-D.ietf-lake-edhoc].
- o MUST specify a CoAP error response code, i.e. 4.00 (Bad Request) in case of client error (e.g. due to a malformed EDHOC message_3), or 5.00 (Internal Server Error) in case of server error (e.g. due to failure in deriving EDHOC key material).

If step 4 (EDHOC processing) is successfully completed but step 7 (OSCORE processing) fails, the same OSCORE error handling applies as defined in Section 8.2 of [RFC8613].

5. Example of EDHOC + OSCORE Request

An example based on the OSCORE test vector from Appendix C.4 of [RFC8613] and the EDHOC test vector from Appendix B.2 of [I-D.ietf-lake-edhoc] is given in Figure 5. In particular, the example assumes that:

- o The used OSCORE Partial IV is 0, consistently with the first request protected with the new OSCORE Security Context.
- o The OSCORE Sender ID of the Client is 0x20. This corresponds to the EDHOC Connection Identifier C_R, which is encoded as the bstr_identifier 0x08 in EDHOC message_3.
- o The EDHOC option is registered with CoAP option number 13.
 - o OSCORE option value: 0x090020 (3 bytes)
 - o EDHOC option value: - (0 bytes)
 - o C_R: 0x20 (1 byte)
 - o CIPHERTEXT_3: 0x5253c3991999a5ffb86921e99b607c067770e0 (19 bytes)
 - o EDHOC message_3: 0x08 5253c3991999a5ffb86921e99b607c067770e0 (20 bytes)
 - o OSCORE ciphertext: 0x612f1092f1776f1c1668b3825e (13 bytes)

From there:

- o Protected CoAP request (OSCORE message):


```

0x44025d1f          ; CoAP 4-byte header
00003974           ; Token
39 6c6f63616c686f7374 ; Uri-Host Option: "localhost"
63 090020          ; OSCORE Option
40                 ; EDHOC Option
ff 5253c3991999a5ffb86921e99b607c067770e0
4d612f1092f1776f1c1668b3825e
(57 bytes)
```

Figure 5: Example of CoAP message with EDHOC and OSCORE combined

6. Security Considerations

The same security considerations from OSCORE [RFC8613] and EDHOC [I-D.ietf-lake-edhoc] hold for this document.

TODO (more considerations)

7. IANA Considerations

This document has the following actions for IANA.

7.1. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry defined in [RFC7252] within the "CoRE Parameters" registry.

[

The CoAP option numbers 13 and 21 are both consistent with the properties of the EDHOC Option defined in Section 3, and they both allow the EDHOC Option to always result in an overall size of 1 byte. This is because:

- o The EDHOC option is always empty, i.e. with zero-length value; and
- o Since the OSCORE option with option number 9 is always present in the CoAP request, the EDHOC option would be encoded with a maximum delta of 4 or 12, depending on its option number being 13 or 21.

At the time of writing, the CoAP option numbers 13 and 21 are both unassigned in the "CoAP Option Numbers" registry, as first available and consistent option numbers for the EDHOC option.

]

Number	Name	Reference
TBD13	EDHOC	[[this document]]

8. Normative References

[I-D.ietf-lake-edhoc]

Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-ietf-lake-edhoc-03 (work in progress), December 2020.

- [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>>.
- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <<https://www.rfc-editor.org/info/rfc8742>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

Acknowledgments

The authors sincerely thank Christian Amsuess, Klaus Hartke, Jim Schaad and Malisa Vucinic for their feedback and comments in the discussion leading up to this draft.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Francesca Palombini
Ericsson

Email: francesca.palombini@ericsson.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Rikard Hoeglund
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: rikard.hoglund@ri.se

Stefan Hristozov
Fraunhofer AISEC

Email: stefan.hristozov@aisec.fraunhofer.de

Goeran Selander
Ericsson

Email: goran.selander@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

M. Tiloca
RISE AB
E. Dijk
IoTconsultancy.nl
November 02, 2020

Proxy Operations for CoAP Group Communication
draft-tiloca-core-groupcomm-proxy-02

Abstract

This document specifies the operations performed by a forward-proxy, when using the Constrained Application Protocol (CoAP) in group communication scenarios. Proxy operations involve the processing of individual responses from servers, as reply to a single request sent by the client over unicast to the proxy, and then distributed by the proxy over IP multicast to the servers. When receiving the different responses via the proxy, the client is able to distinguish them and their origin servers, by acquiring their addressing 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 May 6, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. The Multicast-Signaling Option	4
3. The Response-Forwarding Option	5
4. Requirements and Objectives	6
5. Protocol Description	7
5.1. Request Sending	7
5.1.1. Supporting Observe	9
5.2. Request Processing at the Proxy	9
5.2.1. Supporting Observe	10
5.3. Request and Response Processing at the Server	10
5.3.1. Supporting Observe	10
5.4. Response Processing at the Proxy	10
5.4.1. Supporting Observe	11
5.5. Response Processing at the Client	12
5.5.1. Supporting Observe	13
6. Chain of Proxies	13
6.1. Request Processing at the Proxy	14
6.1.1. Supporting Observe	15
6.2. Response Processing at the Proxy	16
6.2.1. Supporting Observe	16
7. Security Considerations	17
7.1. Client Authentication	17
7.2. Multicast-Signaling Option	18
7.3. Response-Forwarding Option	19
8. IANA Considerations	19
8.1. CoAP Option Numbers Registry	19
9. References	19
9.1. Normative References	20
9.2. Informative References	21
Appendix A. Using OSCORE Between Client and Proxy	21
A.1. Protecting the Request	22
A.2. Verifying the Request	22
A.3. Protecting the Response	23
A.4. Verifying the Response	23
Acknowledgments	23
Authors' Addresses	23

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] allows the presence of forward-proxies, as intermediary entities supporting clients to perform requests on their behalf.

CoAP supports also group communication over IP multicast [I-D.ietf-core-groupcomm-bis], where a group request can be addressed to multiple recipient servers, each of which may reply with an individual unicast response. As discussed in Section 2.3.3 of [I-D.ietf-core-groupcomm-bis], this group communication scenario poses a number of issues and limitations to proxy operations.

In particular, the client sends a single unicast request to the proxy, which the proxy forwards to a group of servers over IP multicast. Later on, the proxy delivers back to the client multiple responses to the original unicast request. As defined by [RFC7252], the multiple responses are delivered to the client inside separate CoAP messages, all matching (by Token) to the client's original unicast request. A possible alternative approach of performing aggregation of responses into a single CoAP response would require a specific aggregation content-format, which is not available yet. Both these approaches have open issues.

This specification considers the former approach, i.e. the proxy forwards the individual responses to a CoAP group request back to the client. The described method addresses all the related issues raised in Section 2.3.3 of [I-D.ietf-core-groupcomm-bis]. To this end, a dedicated signaling protocol is defined, using two new CoAP options.

In particular, the client explicitly confirms its support for receiving multiple responses to a proxied unicast request, i.e. one per origin server, and for how long it is willing to wait for responses. Also, when forwarding a response to the client, the proxy indicates the addressing information of the origin server. This enables the client to distinguish the multiple, different responses by origin and to possibly contact one or more of the servers by sending individual unicast requests, optionally bypassing the forward-proxy.

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 terms and concepts defined in CoAP [RFC7252], Group Communication for CoAP [I-D.ietf-core-groupcomm-bis], CBOR [I-D.ietf-cbor-7049bis], OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

2. The Multicast-Signaling Option

The Multicast-Signaling Option defined in this section has the properties summarized in Figure 1, which extends Table 4 of [RFC7252].

Since the option is not Safe-to-Forward, the column "N" indicates a dash for "not applicable". The value of the Multicast-Signaling Option specifies a timeout value in seconds, encoded as an unsigned integer (see Section 3.2 of [RFC7252]).

No.	C	U	N	R	Name	Format	Length	Default
TBD1		x	-		Multicast-Signaling	uint	0-5	(none)

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

Figure 1: The Multicast-Signaling Option.

This document specifically defines how this option is used by a client, to indicate to a forward-proxy its support for and interest in receiving multiple responses to a proxied CoAP group request, i.e. one per origin server, and for how long it is willing to wait for receiving responses via that proxy (see Section 5.1 and Section 5.2).

The client, when sending a CoAP group request to a proxy via IP unicast, to be forwarded by the proxy to a targeted group of servers, includes the Multicast-Signaling Option into the request. The option value indicates after what time period in seconds the client will stop accepting responses matching its original unicast request, with the exception of notifications if CoAP Observe is used [RFC7641]. This allows the intermediary proxy to stop forwarding responses back to the client, if received from the servers later than a timeout expiration.

The Multicast-Signaling Option is of class U in terms of OSCORE processing (see Section 4.1 of [RFC8613]).

3. The Response-Forwarding Option

The Response-Forwarding Option defined in this section has the properties summarized in Figure 2, which extends Table 4 of [RFC7252]. The option is intended only for CoAP responses, and builds on the Base-Uri option from Section 3 of [I-D.bormann-coap-misc].

Since the option is intended only for responses, the column "N" indicates a dash for "not applicable".

No.	C	U	N	R	Name	Format	Length	Default
TBD2			-		Response-Forwarding	(*)	9-24	(none)

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

(*) See below.

Figure 2: The Response-Forwarding Option.

This document specifically defines how this option is used by a proxy that forwards a request originated by a client over IP multicast.

Upon receiving a response to that request from a server, the proxy includes the Response-Forwarding Option into the response sent to the origin client (see Section 5). The proxy uses the option to indicate to the client the addressing information of the server generating the response.

The client can use the addressing information of the server specified in the option to identify the response originator and possibly send it individual requests later on, either directly or via the proxy as CoAP unicast requests.

The option value is the byte serialization of a CBOR array 'srv_info', which includes the following elements.

- o 'srv_addr': this element is a CBOR byte string, with value the unicast IP address of the server. This element is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)". This element MUST be present.

- o 'srv_port': this element is a CBOR unsigned integer, with value the destination port number where to send unicast requests to the server. This element MAY be present.

The CDDL notation [RFC8610] provided below describes the 'srv_info' CBOR array using the format above.

```
srv_info = [
  srv_addr : #6.260(bstr), ; IP address of the server
  ? srv_port : uint, ; Port number of the server
]
```

If the 'srv_info' array does not include the element 'srv_port', it is assumed that the port number where to send unicast requests to the server is the same port number specified in the group URI of the original unicast CoAP group request sent to the proxy (see Section 5.1).

The Response-Forwarding Option is of class U in terms of OSCORE processing (see Section 4.1 of [RFC8613]).

4. Requirements and Objectives

This specification assumes that the following requirements are fulfilled.

- o REQ1. The CoAP proxy is explicitly configured (allow-list) to allow proxied CoAP group requests from specific client(s).
- o REQ2. The CoAP proxy MUST identify a client sending a CoAP group request, in order to verify whether the client is allowed-listed to do so. For example, this can rely on one of the following.
 - * A DTLS channel [RFC6347][I-D.ietf-tls-dtls13] between the client and the proxy, where the client has also been authenticated during the secure channel establishment.
 - * A pairwise OSCORE Security Context between the client and the proxy, as described in Appendix A.
- o REQ3. If secure, end-to-end communication is required between the client and the servers in the CoAP group, exchanged messages MUST be protected by using Group OSCORE [I-D.ietf-core-oscore-groupcomm], as discussed in Section 5.2 of [I-D.ietf-core-groupcomm-bis]. This requires the client and the servers to have previously joined the correct OSCORE group, for instance by using the approach described in [I-D.ietf-ace-key-groupcomm-oscore]. The correct OSCORE group to

join can be pre-configured or alternatively discovered, for instance by using the approach described in [I-D.tiloca-core-oscore-discovery].

This specification defines how to achieve the following objectives.

- o OBJ1. The CoAP proxy gets an indication from the client that it is in fact interested to and capable to receive multiple responses to its unicast request containing a CoAP group URI.
- o OBJ2. The CoAP proxy learns how long it should wait for responses to a proxied request, before starting to ignore following responses (except for notifications, if CoAP Observe is used [RFC7641]).
- o OBJ3. The CoAP proxy returns individual unicast responses to the client, each of which matches the original unicast request to the proxy.
- o OBJ4. The CoAP client is able to distinguish the different responses to the original unicast request, as well as their corresponding origin servers.
- o OBJ5. The CoAP client is enabled to optionally contact one or more of the responding origin servers in the future, either directly or via the CoAP proxy.

5. Protocol Description

This section specifies the steps of the signaling protocol.

5.1. Request Sending

In order to send a request addressed to a group of servers via the CoAP proxy, the client proceeds as follows.

1. The client prepares a request addressed to the CoAP proxy. The request specifies the group URI as a string in the Proxi-URI option, or by using the Proxy-Scheme option with the group URI constructed from the URI-* options (see Section 2.3.3 of [I-D.ietf-core-groupcomm-bis]).
2. The client MUST retain the Token value used for this original unicast request beyond the reception of a first response matching it. To this end, the client follows the same rules for Token retention defined for multicast requests in Section 2.3.1 of [I-D.ietf-core-groupcomm-bis].

In particular, the client picks an amount of time T it is fine to wait for before freeing up the Token value. Specifically, the value of T MUST be such that:

- * $T < T_r$, where T_r is the amount of time that the client is fine to wait for before potentially reusing the Token value. Note that T_r MUST NOT be less than `MIN_TOKEN_REUSE_TIME` defined in Section 2.3.1 of [I-D.ietf-core-groupcomm-bis].
 - * T should be at least the expected worst-case time taken by the request and response processing on the forward-proxy and on the servers in the addressed CoAP group.
 - * T should be at least the expected worst-case round-trip delay between the client and the forward-proxy, as well as between the proxy and the origin servers.
3. The client MUST include the Multicast-Signaling Option defined in Section 2 into the unicast request to send to the proxy. The option value specifies an amount of time $T' < T$. The difference ($T - T'$) should be at least the expected worst-case round-trip time between the client and the forward-proxy.
- The client can specify $T' = 0$ as option value, thus indicating to be not interested in receiving responses from the origin servers through the proxy. In such a case, the client SHOULD also include a No-Response Option [RFC7967] with value 26 (suppress all response codes), if it supports the option.
- Consistently, if the unicast request to send to the proxy already included a No-Response Option with value 26, the client SHOULD specify $T' = 0$ as value of the Multicast-Signaling Option.
4. The client processes the request as defined in [I-D.ietf-core-groupcomm-bis], and also as in [I-D.ietf-core-oscore-groupcomm] when secure group communication is used between the client and the servers.
5. If OSCORE is used to protect the leg between the client and the proxy (see REQ2 in Section 4), the client (further) protects the unicast request as resulting at the end of step 4. In particular, the client uses the pairwise OSCORE Security Context it has with the proxy, as described in Appendix A.1.
6. The client sends the request to the proxy as a unicast CoAP message.

The exact method that the client uses to estimate the worst-case processing times and round-trip delays mentioned above is out of the scope of this specification. However, such a method is expected to be already used by the client when generally determining a good Token lifetime and reuse interval.

5.1.1. Supporting Observe

When using CoAP Observe [RFC7641], the client follows what is specified in Section 2.3.5 of [I-D.ietf-core-groupcomm-bis], with the difference that it sends a unicast request to the proxy, to be forwarded to the group of servers, as defined in Section 5.1 of this specification.

Furthermore, the client especially follows what is specified in Section 5 of [RFC7641], i.e. it registers its interest to be an observer with the proxy, as if it was communicating with the servers.

5.2. Request Processing at the Proxy

Upon receiving the request from the client, the proxy proceeds as follows.

1. If OSCORE is used to protect the leg between the client and the proxy (see REQ2 in Section 4), the proxy decrypts the request using the pairwise OSCORE Security Context it has with the client, as described in Appendix A.2.
2. The proxy identifies the client, and verifies that the client is in fact allowed-listed to have its requests proxied to CoAP group URIs.
3. The proxy verifies the presence of the Multicast-Signaling Option, as a confirmation that the client is fine to receive multiple responses matching the same original request.

If the Multicast-Signaling Option is not present, the proxy MUST stop processing the request and MUST reply to the client with a 4.00 (Bad Request) response. The response MUST include a diagnostic payload, specifying that the Multicast-Signaling Option was missing and has to be included.

4. The proxy retrieves the value T' from the Multicast-Signaling Option, and then removes the option from the client's request.
5. The proxy forwards the client's request to the group of servers. In particular, the proxy sends it as a CoAP group request over IP multicast, addressed to the group URI specified by the client.

6. The proxy sets a timeout with the value T' retrieved from the Multicast-Signaling Option of the original unicast request.

In case $T' > 0$, the proxy will ignore responses to the forwarded group request coming from servers, if received after the timeout expiration, with the exception of Observe notifications (see Section 5.4).

In case $T' = 0$, the proxy will ignore all responses to the forwarded group request coming from servers.

5.2.1. Supporting Observe

When using CoAP Observe [RFC7641], the proxy takes the role of the client and registers its own interest to observe the target resource with the servers as per Section 5 of [RFC7641].

When doing so, the proxy especially follows what is specified for the client in Section 2.3.5 of [I-D.ietf-core-groupcomm-bis], by forwarding the group request to the servers over IP multicast, as defined in Section 5.2 of this specification.

5.3. Request and Response Processing at the Server

Upon receiving the group request from the proxy, a server proceeds as follows.

1. The server processes the group request as defined in [I-D.ietf-core-groupcomm-bis], and also as in [I-D.ietf-core-oscore-groupcomm] when secure group communication is used between the client and the server.
2. The server processes the response to be forwarded back to the client as defined in [I-D.ietf-core-groupcomm-bis], and also as in [I-D.ietf-core-oscore-groupcomm] when secure group communication is used between the client and the server.

5.3.1. Supporting Observe

When using CoAP Observe [RFC7641], the server especially follows what is specified in Section 2.3.5 of [I-D.ietf-core-groupcomm-bis] and Section 5 of [RFC7641].

5.4. Response Processing at the Proxy

Upon receiving a response matching the group request before the amount of time T' has elapsed, the proxy proceeds as follows.

1. The proxy MUST include the Response-Forwarding Option defined in Section 3 into the response. The proxy specifies as option value the addressing information of the server generating the response, encoded as defined in Section 3. In particular:
 - * The 'srv_addr' element of the 'srv_info' array MUST specify the server IPv6 address if the multicast request was destined for an IPv6 multicast address, and MUST specify the server IPv4 address if the multicast request was destined for an IPv4 address.
 - * If present, the 'srv_port' element of the 'srv_info' array MUST specify the port number of the server as the source port number of the response. This element MUST be present if the source port number of the response differs from the port number specified in the group URI of the original unicast CoAP group request (see Section 5.1). Otherwise, the 'srv_port' element MAY be omitted.
2. If OSCORE is used to protect the leg between the client and the proxy (see REQ2 in Section 4), the proxy (further) protects the response using the pairwise OSCORE Security Context it has with the client, as described in Appendix A.3.
3. The proxy forwards the response back to the client.

Upon timeout expiration, i.e. T' seconds after having sent the group request over IP multicast, the proxy frees up its local Token value associated to that request. Thus, following late responses to the same group request will be discarded and not forwarded back to the client.

5.4.1. Supporting Observe

When using CoAP Observe [RFC7641], the proxy acts as a client registered with the servers, as described earlier in Section 5.2.1.

Furthermore, the proxy takes the role of a server when forwarding notifications from origin servers back to the client. To this end, the proxy follows what is specified in Section 2.3.5 of [I-D.ietf-core-groupcomm-bis] and Section 5 of [RFC7641], with the following additions.

- o At step 1 in Section 5.4, the proxy includes the Response-Forwarding Option in every notification, including non-2.xx notifications resulting in removing the proxy from the list of observers of the origin server.

- o The proxy frees up its Token value used for a group observation only if, after the timeout expiration, no 2.xx (Success) responses matching the group request and also including an Observe option have been received from any origin server. After that, as long as observations are active with servers in the group for the target resource of the group request, notifications from those servers are forwarded back to the client, as defined in Section 5.4.

Finally, the proxy SHOULD regularly verify that the client is still interested in receiving observe notifications for a group observation. To this end, the proxy can rely on the same approach discussed for servers in Section 2.3.5 of [I-D.ietf-core-groupcomm-bis], with more details available in Section 4.5 of [RFC7641].

5.5. Response Processing at the Client

Upon receiving from the proxy a response matching the original unicast request before the amount of time T has elapsed, the client proceeds as follows.

1. The client processes the response as defined in [I-D.ietf-core-groupcomm-bis].
2. If OSCORE is used to protect the leg between the client and the proxy (see REQ2 in Section 4), the client decrypts the response using the pairwise OSCORE Security Context it has with the proxy, as described in Appendix A.4.
3. If secure group communication is used between the client and the servers, the client processes the response, possibly as outcome of step 2, as defined in [I-D.ietf-core-oscore-groupcomm].
4. The client identifies the origin server, whose addressing information is specified as value of the Response-Forwarding Option. If the port number is omitted in the value of the Response-Forwarding Option, the client MUST assume that the port number where to send unicast requests to the server is the same port number specified in the group URI of the original unicast CoAP group request sent to the proxy (see Section 5.1).

In particular, the client is able to distinguish different responses as originated by different servers. Optionally, the client may contact one or more of those servers individually, i.e. directly (bypassing the proxy) or indirectly (via a proxied CoAP unicast request).

Upon the timeout expiration, i.e. T seconds after having sent the original unicast request to the proxy, the client frees up its local Token value associated to that request. Note that, upon this timeout expiration, the Token value is not eligible for possible reuse yet (see Section 5.1). Thus, until the actual amount of time before enabling Token reuse has elapsed, following late responses to the same request forwarded by the proxy will be discarded, as not matching (by Token) any active request from the client.

5.5.1. Supporting Observe

When using CoAP Observe [RFC7641], the client frees up its Token value only if, after the timeout expiration, no 2.xx (Success) responses matching the original unicast request and also including an Observe option have been received.

Instead, if at least one such response has been received, the client continues receiving those notifications as forwarded by the proxy, as long as the observation for the target resource of the original unicast request is active.

6. Chain of Proxies

A client may be interested to access a resource at a group of origin servers which is reached through a chain of two or more proxies.

That is, these proxies are configured into a chain, where each non-last proxy is configured to forward CoAP (multicast) requests to the next hop towards the origin servers. Also, each non-first proxy is configured to forward back CoAP responses to (the previous hop proxy towards) the origin client.

This section specifies how the signaling protocol defined in Section 5 is used in that setting. Except for the last proxy before the origin servers, every other proxy in the chain takes the role of client with respect to the next hop towards the origin servers. Also, every proxy in the chain takes the role of server towards the previous proxy closer to the origin client.

The requirements REQ1 and REQ2 defined in Section 4 MUST be fulfilled for each proxy in the chain. That is, every proxy in the chain has to be explicitly configured (allow-list) to allow proxied group requests from specific senders, and MUST identify those senders upon receiving their group request. For the first proxy in the chain, that sender is the origin client. For each other proxy in the chain, that sender is the previous hop proxy closer the origin client. In either case, a proxy can identify the sender of a group request by the same means mentioned in Section 4.

6.1. Request Processing at the Proxy

Upon receiving a group request to be forwarded to a CoAP group URIs, a proxy proceed as follows.

If the proxy is the last one in the chain, i.e. it is the last hop before the origin servers, the proxy performs the steps defined in Section 5.2, with no modifications.

Otherwise, the proxy performs the steps defined in Section 5.2, with the following differences.

- o At steps 1-3, "client" refers to the origin client for the first proxy in the chain; or to the previous hop proxy closer to the origin client, otherwise.
- o At step 4, the proxy rather performs the following actions.
 1. The proxy retrieves the value T' from the Multicast-Signaling Option, and does not remove the option.
 2. In case $T' > 0$, the proxy picks an amount of time T it is fine to wait for before freeing up its local Token value to use with the next hop towards the origin servers. To this end, the proxy MUST follow what defined at step 2 of Section 5.1 for the origin client, with the following differences.
 - + T MUST be greater than the retrieved value T' , i.e. $T' < T$.
 - + The worst-case message processing time takes into account all the next hops towards the origin servers, as well as the origin servers themselves.
 - + The worst-case round-trip delay takes into account all the legs between the proxy and the origin servers.
 3. In case $T' > 0$, the proxy replaces the value of the Multicast-Signaling Option with a new value T'' , such that:
 - + $T'' < T$. The difference $(T - T'')$ should be at least the expected worst-case round-trip time between the proxy and the next hop towards the origin servers.
 - + $T'' < T'$. The difference $(T' - T'')$ should be at least the expected worst-case round-trip time between the proxy and the (previous hop proxy closer to the) origin client.

If the proxy is not able to determine a value T'' that fulfills both the requirements above, the proxy MUST stop processing the request and MUST respond with a 5.05 (Proxying Not Supported) error response to the previous hop proxy closer to the origin client. The proxy SHOULD include a Multicast-Signaling Option, set to the minimum value T' that would be acceptable in the Multicast-Signaling Option of a request to forward.

Upon receiving such an error response, any proxy in the chain MAY send an updated request to the next hop towards the origin servers, specifying in the Multicast-Signaling Option a value T' greater than in the previous request. If this does not happen, the proxy receiving the error response MUST also send a 5.05 (Proxying Not Supported) error response to the previous hop proxy closer to the origin client. Like the received one, also this error response SHOULD include a Multicast-Signaling Option, set to the minimum value T' acceptable by the proxy sending the error response.

- o At step 5, the proxy forwards the request to the next hop towards the origin servers.
- o At step 6, the proxy sets a timeout with the value T' retrieved from the Multicast-Signaling Option of the request received from the (previous hop proxy closer to the) origin client.

In case $T' > 0$, the proxy will ignore responses to the forwarded group request coming from the (next hop towards the) origin servers, if received after the timeout expiration, with the exception of Observe notifications (see Section 5.4).

In case $T' = 0$, the proxy will ignore all responses to the forwarded group request coming from the (next hop towards the) origin servers.

6.1.1. Supporting Observe

When using CoAP Observe [RFC7641], what defined in Section 5.2.1 applies for the last proxy in the chain, i.e. the last hop before the origin servers.

Any other proxy in the chain acts as a client and registers its own interest to observe the target resource with the next hop towards the origin servers, as per Section 5 of [RFC7641].

6.2. Response Processing at the Proxy

Upon receiving a response matching the group request before the amount of time T' has elapsed, the proxy proceeds as follows.

If the proxy is the last one in the chain, i.e. it is the last hop before the origin servers, the proxy performs the steps defined in Section 5.4, with no modifications.

Otherwise, the proxy performs the steps defined in Section 5.4, with the following differences.

- o The proxy skips step 1. In particular, the proxy **MUST NOT** remove, alter or replace the Response-Forwarding Option.
- o At steps 2-3, "client" refers to the origin client for the first proxy in the chain; or to the previous hop proxy closer to the origin client, otherwise.

Upon timeout expiration, i.e. T seconds after having sent the group request to the next hop towards the origin servers, the proxy frees up its local Token value associated to that request. Thus, following late responses to the same group request will be discarded and not forwarded back to the (previous hop proxy closer to the) origin client.

6.2.1. Supporting Observe

When using CoAP Observe [RFC7641], what defined in Section 5.4.1 applies for the last proxy in the chain, i.e. the last hop before the origin servers.

As to any other proxy in the chain, the following applies.

- o The proxy acts as a client registered with the next hop towards the origin servers, as described earlier in Section 6.1.1.
- o The proxy takes the role of a server when forwarding notifications from the next hop to the origin servers back to the (previous hop proxy closer to the) origin client, as per Section 5 of [RFC7641].
- o The proxy frees up its Token value used for a group observation only if, after the timeout expiration, no 2.xx (Success) responses matching the group request and also including an Observe option have been received from the next hop towards the origin servers. After that, as long as the observation for the target resource of the group request is active with the next hop towards the origin servers in the group, notifications from that hop are forwarded

back to the (previous hop proxy closer to the) origin client, as defined in Section 6.2.

- o The proxy SHOULD regularly verify that the (previous hop proxy closer to the) origin client is still interested in receiving observe notifications for a group observation. To this end, the proxy can rely on the same approach defined in Section 4.5 of [RFC7641].

7. Security Considerations

The security considerations from [RFC7252][I-D.ietf-core-groupcomm-bis][RFC8613][I-D.ietf-core-oscure-groupcomm] hold for this document.

When a chain of proxies is used (see Section 6), the secure communication between any two adjacent hops is independent.

When Group OSCORE is used for end-to-end secure group communication between the origin client and the origin servers, this security association is unaffected by the possible presence of a proxy or a chain of proxies.

Furthermore, the following additional considerations hold.

7.1. Client Authentication

As per the requirement REQ2 (see Section 4), the client has to authenticate to the proxy when sending a group request to forward. This leverages an established security association between the client and the proxy, that the client uses to protect the group request, before sending it to the proxy.

Note that, if the group request is (also) protected with Group OSCORE, i.e. end-to-end between the client and the servers, the proxy can authenticate the client by successfully verifying the counter signature embedded in the group request. This requires that, for each client to authenticate, the proxy stores the public key used by that client in the OSCORE group, which in turn would require a form of active synchronization between the proxy and the Group Manager for that group [I-D.ietf-core-oscure-groupcomm].

Nevertheless, the client and the proxy SHOULD still rely on a full-fledged, pairwise secure association. In addition to ensuring the integrity of group requests sent to the proxy (see Section 7.2 and Section 7.3), this prevents the proxy from forwarding replayed group requests with a valid counter signature, as possibly injected by an active, on-path adversary.

The same considerations apply when a chain of proxies is used (see Section 6), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

7.2. Multicast-Signaling Option

The Multicast-Signaling Option is of class U for OSCORE [RFC8613]. Hence, also when Group OSCORE is used between the client and the servers [I-D.ietf-core-oscore-groupcomm], a proxy is able to access the option value and retrieve the timeout value T' , as well as to remove the option altogether before forwarding the group request to the servers. When a chain of proxies is used (see Section 6), this also allows each proxy but the last one in the chain to update the option value, as an indication for the next hop towards the origin servers (see Section 6.1).

The security association between the client and the proxy **MUST** provide message integrity, so that further intermediaries between the two as well as on-path active adversaries are not able to remove the option or alter its content, before the group request reaches the proxy. Removing the option would otherwise result in not forwarding the group request to the servers. Instead, altering the option content would result in the proxy accepting and forwarding back responses for an amount of time different than the one actually indicated by the client.

The security association between the client and the proxy **SHOULD** also provide message confidentiality. Otherwise, further intermediaries between the two as well as on-path passive adversaries would be able to simply access the option content, and thus learn for how long the client is willing to receive responses from the servers in the group via the proxy. This may in turn be used to perform a more efficient, selective suppression of responses from the servers.

When the client (further) protects the unicast request sent to the proxy using OSCORE (see Appendix A) and/or with DTLS, both message integrity and message confidentiality are achieved in the leg between the client and the proxy.

The same considerations above about security associations apply when a chain of proxies is used (see Section 6), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

7.3. Response-Forwarding Option

The Response-Forwarding Option is of class U for OSCORE [RFC8613]. Hence, also when Group OSCORE is used between the client and the servers [I-D.ietf-core-oscore-groupcomm], the proxy that has forwarded the group request to the servers is able to include the option into a server response, before forwarding this response back to the (previous hop proxy closer to the) origin client.

Since the security association between the client and the proxy provides message integrity, any further intermediaries between the two or on-path active adversaries are not able to undetectably remove the Response-Forwarding Option from a forwarded server response. This ensures that the client can correctly distinguish the different responses and identify their corresponding origin server.

When the proxy (further) protects the response forwarded back to the client using OSCORE (see Appendix A) and/or with DTLS, message integrity is achieved in the leg between the client and the proxy.

The same considerations above about security associations apply when a chain of proxies is used (see Section 6), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

8. IANA Considerations

This document has the following actions for IANA.

8.1. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry defined in [RFC7252] within the "CoRE Parameters" registry.

Number	Name	Reference
TBD1	Multicast-Signaling	[[this document]]
TBD2	Response-Forwarding	[[this document]]

9. References

9.1. Normative References

- [I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work in progress), September 2020.
- [I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-ietf-core-groupcomm-bis-02 (work in progress), November 2020.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-10 (work in progress), November 2020.
- [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>>.
- [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>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

9.2. Informative References

- [I-D.bormann-coap-misc]
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-27 (work in progress), November 2014.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-09 (work in progress), November 2020.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-38 (work in progress), May 2020.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-07 (work in progress), November 2020.
- [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>>.
- [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. Using OSCORE Between Client and Proxy

This section describes how OSCORE is used to protect messages exchanged by an origin client and a proxy, using their pairwise OSCORE Security Context.

This is especially convenient for the communication scenario addressed in this document, when the origin client already supports and uses Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect messages end-to-end with the origin servers.

The following focuses on the origin client originating the group request and a single proxy as its immediate next hop. When a chain of proxies is used (see Section 6), the same independently applies between each pair of proxies in the chain, where the proxy forwarding

the group request acts as client and the next hop towards the origin servers acts as server.

A.1. Protecting the Request

Before sending the CoAP request to the proxy, the origin client protects it using the pairwise OSCORE Security Context it has with the proxy.

To this end, the origin client processes the CoAP request as defined in Section 8.1 of [RFC8613], with the following differences.

- o The Proxy-Uri option, if present, is not decomposed and recomposed as defined in Section 4.1.3.3 of [RFC8613].
- o The following options, if present, are processed as Class E.
 - * Proxy-Uri, Proxy-Scheme, Uri-Host and Uri-Port, defined in [RFC7252].
 - * OSCORE, defined in [RFC8613], which is present if Group OSCORE is used between the origin client and the origin servers, to achieve end-to-end secure group communication.
 - * Multicast-Signaling Option, defined in Section 2 of this specification.

As per [RFC8613], the resulting message includes an outer OSCORE Option, which reflects the usage of the pairwise OSCORE Security Context between the origin client and the proxy.

A.2. Verifying the Request

The proxy verifies the CoAP request as defined in Section 8.2 of [RFC8613]. Note that the Multicast-Signaling Option is retrieved during the decryption process, and added to the decrypted request.

If secure group communication is also used between the origin client and the origin servers, the request resulting from the previous step and to be forwarded to the origin servers is also already protected with Group OSCORE [I-D.ietf-core-oscore-groupcomm]. Consequently, it includes an outer OSCORE Option, which reflects the usage of the group OSCORE Security Context between the origin client and the origin servers.

A.3. Protecting the Response

The proxy protects the CoAP response received from a server, using the pairwise OSCORE Security Context it has with the origin client.

To this end, the proxy processes the CoAP response as defined in Section 8.3 of [RFC8613], with the difference that the OSCORE Option, if present, is processed as Class E. This is the case if Group OSCORE is used between the origin client and the origin servers, to achieve end-to-end secure group communication.

Furthermore, the Response-Forwarding Option defined in Section 3 of this specification is also processed as Class E.

As per [RFC8613], the resulting message to be forwarded back to the origin client includes an outer OSCORE Option, which reflects the usage of the pairwise OSCORE Security Context between the origin client and the proxy.

A.4. Verifying the Response

The origin client verifies the CoAP response received from the proxy as defined in Section 8.4 of [RFC8613]. Note that, the Response-Forwarding Option is retrieved during the decryption process, and added to the decrypted response.

If secure group communication is also used between the origin client and the origin servers, the response resulting from the previous step is protected with Group OSCORE [I-D.ietf-core-oscore-groupcomm]. Consequently, it includes an outer OSCORE Option, which reflects the usage of the group OSCORE Security Context between the origin client and the origin servers.

Acknowledgments

The authors sincerely thank Christian Amsuess, Jim Schaad and Goeran Selander for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Esko Dijk
IoTconsultancy.nl
_____\n
Utrecht
The Netherlands

Email: esko.dijk@iotconsultancy.nl

CoRE Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: 8 September 2022

M. Tiloca
RISE AB
E. Dijk
IoTconsultancy.nl
7 March 2022

Proxy Operations for CoAP Group Communication
draft-tiloca-core-groupcomm-proxy-06

Abstract

This document specifies the operations performed by a proxy, when using the Constrained Application Protocol (CoAP) in group communication scenarios. Such a proxy processes a single request sent by a client over unicast, and distributes the request over IP multicast to a group of servers. Then, the proxy collects the individual responses from those servers and relays those responses back to the client, in a way that allows the client to distinguish the responses and their origin servers through embedded addressing information. This document updates RFC7252 with respect to caching of response messages at proxies.

Discussion Venues

This note is to be removed before publishing as an RFC.

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

Source for this draft and an issue tracker can be found at <https://gitlab.com/crimson84/draft-tiloca-core-groupcomm-proxy>.

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
1.1. Terminology	5
2. The Multicast-Timeout Option	5
3. The Response-Forwarding Option	6
3.1. Encoding of Server Address	8
3.2. Default Values of the Server Port Number	9
4. Requirements and Objectives	10
5. Protocol Description	11
5.1. Request Sending at the Client	11
5.1.1. Request Sending	11
5.1.2. Supporting Observe	12
5.2. Request Processing at the Proxy	13
5.2.1. Request Processing	13
5.2.2. Supporting Observe	14
5.3. Request and Response Processing at the Server	14
5.3.1. Request and Response Processing	14
5.3.2. Supporting Observe	14
5.4. Response Processing at the Proxy	14
5.4.1. Response Processing	15
5.4.2. Supporting Observe	15
5.5. Response Processing at the Client	16
5.5.1. Response Processing	16
5.5.2. Supporting Observe	18
5.6. Example	18
6. Reverse-Proxies	20
6.1. Processing on the Client Side	20
6.2. Processing on the Proxy Side	20
7. Caching	22
7.1. Freshness Model	23
7.2. Validation Model	25

7.2.1.	Proxy-Servers Revalidation with Unicast Requests . .	25
7.2.2.	Proxy-Servers Revalidation with Group Requests . . .	26
7.3.	Client-Proxy Revalidation with Group Requests	27
7.4.	Caching of End-To-End Protected Responses at Proxies . .	29
7.4.1.	Deterministic Requests to Achieve Cacheability . . .	29
7.4.2.	Validation of Responses	30
8.	Chain of Proxies	31
8.1.	Request Processing at the Proxy	31
8.1.1.	Supporting Observe	33
8.2.	Response Processing at the Proxy	33
8.2.1.	Supporting Observe	34
9.	HTTP-CoAP Proxies	35
9.1.	The HTTP Multicast-Timeout Header Field	35
9.2.	The HTTP Response-Forwarding Header Field	36
9.3.	The HTTP Group-ETag Header Field	37
9.4.	Request Sending at the Client	37
9.5.	Request Processing at the Proxy	38
9.6.	Response Processing at the Proxy	38
9.7.	Response Processing at the Client	39
9.8.	Example	40
9.9.	Streamed Delivery of Responses to the Client	41
9.10.	Reverse-Proxies	42
9.10.1.	Processing on the Client Side	42
9.10.2.	Processing on the Proxy Side	42
10.	Security Considerations	42
10.1.	Client Authentication	43
10.2.	Multicast-Timeout Option	43
10.3.	Response-Forwarding Option	44
10.4.	Group-ETag Option	45
10.5.	HTTP-to-CoAP Proxies	46
11.	IANA Considerations	46
11.1.	CoAP Option Numbers Registry	46
11.2.	CoAP Transport Information Registry	47
11.3.	Header Field Registrations	47
12.	References	48
12.1.	Normative References	48
12.2.	Informative References	50
Appendix A.	Examples with Reverse-Proxy	51
A.1.	Example 1	52
A.2.	Example 2	55
A.3.	Example 3	57
Acknowledgments	59
Authors' Addresses	59

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] allows the presence of proxies, as intermediary entities supporting clients by performing requests on their behalf and relaying back responses.

CoAP supports also group communication over IP multicast [I-D.ietf-core-groupcomm-bis], where a group request can be addressed to multiple recipient servers, each of which may reply with an individual unicast response. As discussed in Section 3.5 of [I-D.ietf-core-groupcomm-bis], this group communication scenario poses a number of issues and limitations to proxy operations.

In particular, the client sends to the proxy a single unicast request, which the proxy forwards to a group of servers over IP multicast. Later on, the proxy replies to the client's original unicast request, by relaying back the responses from the servers.

As per [RFC7252], a CoAP-to-CoAP proxy relays those responses to the client as separate CoAP messages, all matching (by Token) with the client's original unicast request. A possible alternative approach for aggregating those responses into a single CoAP response sent to the client would require a specific aggregation content-format, which is not available yet. Both these approaches have open issues.

This document considers the former approach. That is, after forwarding a CoAP group request from the client to the group of CoAP servers, the proxy relays the individual responses back to the client as separate CoAP messages. The described method addresses all the related issues raised in Section 3.5 of [I-D.ietf-core-groupcomm-bis]. To this end, a dedicated signaling protocol is defined, using two new CoAP options.

Using this protocol, the client explicitly confirms its intent to perform a proxied group request and its support for receiving multiple responses as a result, i.e., one or more from each origin server. Also, the client signals for how long it is willing to wait for responses. When relaying to the client a response to the group request, the proxy indicates the addressing information of the origin server. This enables the client to distinguish, multiple different responses by origin and to possibly contact one or more of the respective servers by sending individual unicast request(s) to the indicated address(es). In doing these follow-up unicast requests, the client may optionally bypass the proxy.

This document also defines how the proposed protocol is used between an HTTP client and an HTTP-CoAP cross-proxy, in order to forward an HTTP group request from the client to a group of CoAP servers, and relay back the individual CoAP responses as HTTP responses.

Finally, this document defines a caching model for proxies and specifies how they can serve a group request by using cached responses. Therefore, this document updates [RFC7252].

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 terms and concepts defined in CoAP [RFC7252], Group Communication for CoAP [I-D.ietf-core-groupcomm-bis], CBOR [RFC8949], OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

Unless specified otherwise, the term "proxy" refers to a CoAP-to-CoAP forward-proxy, as defined in Section 5.7.2 of [RFC7252].

2. The Multicast-Timeout Option

The Multicast-Timeout Option defined in this section has the properties summarized in Figure 1, which extends Table 4 of [RFC7252].

Since the option is not Safe-to-Forward, the column "N" indicates a dash for "not applicable". The value of the Multicast-Timeout Option specifies a timeout value in seconds, encoded as an unsigned integer (see Section 3.2 of [RFC7252]).

No.	C	U	N	R	Name	Format	Length	Default
TBD1		x	-		Multicast-Timeout	uint	0-4	(none)

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

Figure 1: The Multicast-Timeout Option.

This document specifically defines how this option is used by a client in a CoAP request, to indicate to a proxy its support for and interest in receiving multiple responses to a proxied CoAP group request, i.e., one or more from each origin server, and for how long it is willing to wait for receiving responses via that proxy (see Section 5.1.1 and Section 5.2.1).

When sending a CoAP group request to a proxy via IP unicast, to be forwarded by the proxy to a targeted group of servers, the client includes the Multicast-Timeout Option into the request. The option value indicates after how much time in seconds the client will stop accepting responses matching its original unicast request, with the exception of notifications if the CoAP Observe Option [RFC7641] is used in the same request. This allows the proxy to stop relaying responses back to the client, if those are received from servers after the indicated amount of time has elapsed.

The Multicast-Timeout Option is of class U in terms of OSCORE processing (see Section 4.1 of [RFC8613]).

3. The Response-Forwarding Option

The Response-Forwarding Option defined in this section has the properties summarized in Figure 2, which extends Table 4 of [RFC7252]. The option is intended only for inclusion in CoAP responses, and builds on the Base-Uri option from Section 3 of [I-D.bormann-coap-misc].

Since the option is intended only for responses, the column "N" indicates a dash for "not applicable".

No.	C	U	N	R	Name	Format	Length	Default
TBD2			-		Response-Forwarding	(*)	10-25	(none)

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

(*) See below.

Figure 2: The Response-Forwarding Option.

This document specifically defines how this option is used by a proxy that can perform proxied CoAP group requests.

Upon receiving a response to such request from a server, the proxy includes the Response-Forwarding Option into the response sent to the origin client (see Section 5). The proxy uses the option to indicate the addressing information where the client can send an individual request intended to that origin server.

In particular, the client can use the addressing information specified in the option to identify the response originator and possibly send it individual requests later on, either directly, or indirectly via the proxy, as unicast requests.

The option value is set to the byte serialization of the CBOR array 'tp_info' defined in Section 2.2.1 of [I-D.ietf-core-observe-multicast-notifications], including only the set of elements 'srv_addr'. In turn, the set includes the integer 'tp_id' identifying the used transport protocol, and further elements whose number, format and encoding depend on the value of 'tp_id'.

The value of 'tp_id' MUST be taken from the "Value" column of the "CoAP Transport Information" registry defined in Section 14.5 of [I-D.ietf-core-observe-multicast-notifications]. The elements of 'srv_addr' following 'tp_id' are specified in the corresponding entry of the Registry, under the "Server Addr" column.

If the server is reachable through CoAP transported over UDP, the 'tp_info' array includes the following elements, encoded as defined in Section 2.2.1.1 of [I-D.ietf-core-observe-multicast-notifications].

- * 'tp_id': the CBOR integer with value 1. This element MUST be present.
- * 'srv_host': a CBOR byte string, encoding the unicast IP address of the server. This element is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)". This element MUST be present.
- * 'srv_port': a CBOR unsigned integer or the CBOR simple value "null" (0xf6). This element MAY be present.

If present as a CBOR unsigned integer, it has as value the destination UDP port number to use for individual requests to the server.

If present as the CBOR simple value "null" (0xf6), the client MUST assume that the same port number specified in the group URI of the original unicast CoAP group request sent to the proxy (see Section 5.1.1) can be used for individual requests to the server.

If not present, the client MUST assume that the default port number 5683 defined in [RFC7252] can be used as the destination UDP port number for individual requests to the server.

The CDDL notation [RFC8610] provided below describes the 'tp_info' CBOR array using the format defined above.

```
tp_info = [  
    tp_id : 1,                ; UDP as transport protocol  
    srv_host : #6.260(bstr),   ; IP address where to reach the server  
    ? srv_port : uint / null    ; Port number where to reach the server  
]
```

At present, 'tp_id' is expected to take only value 1 (UDP) when using forward proxies, UDP being the only currently available transport for CoAP to work over IP multicast. While additional multicast-friendly transports may be defined in the future, other current transport protocols can still be useful in applications relying on a reverse-proxy (see Section 6).

The rest of this section considers the new values of 'tp_id' registered by this document (see Section 11.2), and specifies:

- * The encoding for the elements of 'tp_info' following 'tp_id' (see Section 3.1).
- * The port number assumed by the client if the element 'srv_port' of 'tp_info' is not present (see Section 3.2).

The Response-Forwarding Option is of class U in terms of OSCORE processing (see Section 4.1 of [RFC8613]).

3.1. Encoding of Server Address

This document defines some values used as transport protocol identifiers, whose respective new entries are included in the "CoAP Transport Information" registry defined in Section 14.5 of [I-D.ietf-core-observe-multicast-notifications].

For each of these values, the following table summarizes the elements specified under the "Srv Addr" and "Req Info" columns of the registry, together with their CBOR encoding and short description.

While not listed here for brevity, the element 'tp_id' is always present as a CBOR integer in the element set "Srv Addr".

'tp_id' Values	Element Set	Element	CBOR Type	Description
2, 3, 4, 5, 6	Srv Addr	srv_host	#6.260(bstr) (*)	Address of the server
		srv_port	uint / null	Port number of the server
	Req Info	cli_host	#6.260(bstr) (*)	Address of the client
		cli_port	uint	Port number of the client

* The CBOR byte string is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)".

3.2. Default Values of the Server Port Number

If the 'srv_port' element of the 'tp_info' array is not present, the client MUST assume the following value as port number where to send individual requests intended to the server, based on the value of 'tp_id'.

- * If 'tp_id' is equal to 1, i.e., CoAP over UDP, the default port number 5683 as defined in [RFC7252].
- * If 'tp_id' is equal to 2, i.e., CoAP over UDP secured with DTLS, the default port number 5684 as defined in [RFC7252].
- * If 'tp_id' is equal to 3, i.e., CoAP over TCP, the default port number 5683 as defined in [RFC8323].
- * If 'tp_id' is equal to 4, i.e., CoAP over TCP secured with TLS, the default port number 5684 as defined in [RFC8323].
- * If 'tp_id' is equal to 5, i.e., CoAP over WebSockets, the default port number 80 as defined in [RFC8323].
- * If 'tp_id' is equal to 6, i.e., CoAP over WebSockets secured with TLS, the default port number 443 as defined in [RFC8323].

4. Requirements and Objectives

In this section, the word "proxy" is not limited to forward-proxies. Instead, it comprises also reverse-proxies and HTTP-to-CoAP proxies.

This document assumes that the following requirements are fulfilled.

- * REQ1. The proxy is explicitly configured (allow-list) to perform proxied group requests on behalf of specific allowed client(s).
- * REQ2. The proxy MUST identify a client sending a unicast group request to be proxied, in order to verify whether the client is allowed-listed to do so. For example, this can rely on one of the following security associations.
 - A TLS [RFC8446] or DTLS [RFC6347][I-D.ietf-tls-dtls13] channel between the client and the proxy, where the client has been authenticated during the secure channel establishment.
 - A pairwise OSCORE [RFC8613] Security Context between the client and the proxy, as defined in [I-D.tiloca-core-oscore-capable-proxies].
- * REQ3. If secure, end-to-end communication is required between the client and the servers in the CoAP group, exchanged messages MUST be protected by using Group OSCORE [I-D.ietf-core-oscore-groupcomm], as discussed in Section 5 of [I-D.ietf-core-groupcomm-bis]. This requires the client and the servers to have previously joined the correct OSCORE group, for instance by using the approach described in [I-D.ietf-ace-key-groupcomm-oscore]. The correct OSCORE group to join can be pre-configured or alternatively discovered, for instance by using the approach described in [I-D.tiloca-core-oscore-discovery].

This document defines how to achieve the following objectives.

- * OBJ1. The proxy gets an indication from the client that the client is in fact interested in and capable to handle multiple responses to a proxied group request. With particular reference to a unicast CoAP group request sent to the proxy, this means that the client is capable to receive those responses as separate CoAP responses, each matching with the original unicast request.
- * OBJ2. The proxy learns for how long it should wait for responses to a proxied group request, before starting to ignore following responses to it (except for notifications, if a CoAP Observe Option is used [RFC7641]).

- * OBJ3. The proxy relays to the client any multiple responses to the proxied group request. With particular reference to a client's original CoAP unicast request sent to the proxy, those responses are sent to the client as separate CoAP responses, each matching with the original unicast request.
- * OBJ4. The client is able to distinguish the different responses to the proxied group request, as well as their corresponding origin servers.
- * OBJ5. The client is enabled to optionally contact one or more of the responding origin servers in the future, either directly or via the proxy.

5. Protocol Description

This section specifies the steps of the signaling protocol.

5.1. Request Sending at the Client

This section defines the operations performed by the client, for sending a request targeting a group of servers via the proxy.

5.1.1. Request Sending

The client proceeds according to the following steps.

1. The client prepares a unicast CoAP group request addressed to the proxy. The request specifies the group URI where the request has to be forwarded to, as a string in the Proxi-URI option or by using the Proxy-Scheme option with the group URI constructed from the URI-* options (see Section 3.5.1 of [I-D.ietf-core-groupcomm-bis]).
2. The client MUST retain the Token value used for this original unicast request beyond the reception of a first CoAP response matching with it. To this end, the client follows the same rules for Token retention defined for multicast CoAP requests in Section 3.1.5 of [I-D.ietf-core-groupcomm-bis].

In particular, the client picks an amount of time T that it is fine to wait for before freeing up the Token value. Specifically, the value of T MUST be such that:

- * $T < T_r$, where T_r is the amount of time that the client is fine to wait for before potentially reusing the Token value. Note that T_r MUST NOT be less than `MIN_TOKEN_REUSE_TIME` defined in Section 3.1.5 of [I-D.ietf-core-groupcomm-bis].

- * T should be at least the expected worst-case time taken by the request and response processing on the proxy and on the servers in the addressed CoAP group.
 - * T should be at least the expected worst-case round-trip delay between the client and the proxy plus the worst-case round-trip delay between the proxy and any one of the origin servers.
3. The client MUST include the Multicast-Timeout Option defined in Section 2 into the unicast request to send to the proxy. The option value specifies an amount of time $T' < T$. The difference ($T - T'$) should be at least the expected worst-case round-trip time between the client and the proxy.

The client can specify $T' = 0$ as option value, thus indicating to be not interested in receiving responses from the origin servers through the proxy. In such a case, the client SHOULD also include a No-Response Option [RFC7967] with value 26 (suppress all response codes), if it supports the option.

Consistently, if the unicast request to send to the proxy already included a No-Response Option with value 26, the client SHOULD specify $T' = 0$ as value of the Multicast-Timeout Option.

4. The client processes the request as defined in [I-D.ietf-core-groupcomm-bis], and also as in [I-D.ietf-core-oscore-groupcomm] when secure group communication is used between the client and the servers.
5. The client sends the request to the proxy as a unicast CoAP message. When doing so, the client protects the request according to the security association it has with the proxy.

The exact method that the client uses to estimate the worst-case processing times and round-trip delays mentioned above is out of the scope of this document. However, such a method is expected to be already used by the client when generally determining an appropriate Token lifetime and reuse interval.

5.1.2. Supporting Observe

When using CoAP Observe [RFC7641], the client follows what is specified in Section 3.7 of [I-D.ietf-core-groupcomm-bis], with the difference that it sends a unicast request to the proxy, to be forwarded to the group of servers, as defined in Section 5.1.1 of this document.

Furthermore, the client especially follows what is specified in Section 5 of [RFC7641], i.e., it registers its interest to be an observer with the proxy, as if it was communicating with the servers.

5.2. Request Processing at the Proxy

This section defines the operations performed by the proxy, when receiving a request to forward to a group of servers.

5.2.1. Request Processing

Upon receiving the request from the client, the proxy proceeds according to the following steps.

1. The proxy decrypts the request, according to the security association it has with the client.
2. The proxy identifies the client, and verifies that the client is in fact allowed-listed to have its requests proxied to CoAP group URIs.
3. The proxy verifies the presence of the Multicast-Timeout Option, as a confirmation that the client is fine to receive multiple CoAP responses matching with the same original request.

If the Multicast-Timeout Option is not present, the proxy MUST stop processing the request and MUST reply to the client with a 4.00 (Bad Request) response. The response MUST include a Multicast-Timeout Option with an empty (zero-length) value, indicating that the Multicast-Timeout Option was missing and has to be included in the request. As per Section 5.9.2 of [RFC7252] The response SHOULD include a diagnostic payload.

4. The proxy retrieves the value T' from the Multicast-Timeout Option, and then removes the option from the client's request.
5. The proxy forwards the client's request to the group of servers. In particular, the proxy sends it as a CoAP group request over IP multicast, addressed to the group URI specified by the client.
6. The proxy sets a timeout with the value T' retrieved from the Multicast-Timeout Option of the original unicast request.

In case $T' > 0$, the proxy will ignore responses to the forwarded group request coming from servers, if received after the timeout expiration, with the exception of Observe notifications (see Section 5.4).

In case $T' = 0$, the proxy will ignore all responses to the forwarded group request coming from servers.

If the proxy supports caching of responses, it can serve the original unicast request also by using cached responses, as per Section 7.

5.2.2. Supporting Observe

When using CoAP Observe [RFC7641], the proxy takes the role of the client and registers its own interest to observe the target resource with the servers as per Section 5 of [RFC7641].

When doing so, the proxy especially follows what is specified for the client in Section 3.7 of [I-D.ietf-core-groupcomm-bis], by forwarding the group request to the servers over IP multicast as defined in Section 5.2.1 of this document.

5.3. Request and Response Processing at the Server

This section defines the operations performed by the server, when receiving a group request from the proxy.

5.3.1. Request and Response Processing

Upon receiving the request from the proxy, the server proceeds according to the following steps.

1. The server processes the group request as defined in [I-D.ietf-core-groupcomm-bis], and also as in [I-D.ietf-core-oscore-groupcomm] when secure group communication is used between the client and the server.
2. The server processes the response to be relayed to the client as defined in [I-D.ietf-core-groupcomm-bis], and also as in [I-D.ietf-core-oscore-groupcomm] when secure group communication is used between the client and the server.

5.3.2. Supporting Observe

When using CoAP Observe [RFC7641], the server especially follows what is specified in Section 3.7 of [I-D.ietf-core-groupcomm-bis] and Section 5 of [RFC7641].

5.4. Response Processing at the Proxy

This section defines the operations performed by the proxy, when receiving a response matching with a forwarded group request.

5.4.1. Response Processing

Upon receiving a response matching with the group request before the amount of time T' has elapsed, the proxy proceeds according to the following steps.

1. The proxy MUST include the Response-Forwarding Option defined in Section 3 into the response. The proxy specifies as option value the addressing information of the server generating the response, encoded as defined in Section 3. In particular:
 - * The 'srv_addr' element of the 'srv_info' array MUST specify the server IPv6 address if the multicast request was destined for an IPv6 multicast address, and MUST specify the server IPv4 address if the multicast request was destined for an IPv4 multicast address.
 - * If present, the 'srv_port' element of the 'srv_info' array MUST specify the port number of the server as the source port number of the response. This element MUST be present if the source port number of the response differs from the default port number for the transport protocol specified in the 'tp_id' element.
2. The proxy forwards the response back to the client. When doing so, the proxy protects the response according to the security association it has with the client.

As discussed in Section 3.1.6 of [I-D.ietf-core-groupcomm-bis], it is possible that a same server replies with multiple responses to the same group request, i.e., with the same Token. As long as the proxy forwards responses to a group request back to the origin client, the proxy MUST follow the steps defined above and forward also such multiple responses "as they come".

Upon timeout expiration, i.e., T' seconds after having sent the group request over IP multicast, the proxy frees up its local Token value associated with that request. Thus, following late responses to the same group request will be discarded and not forwarded back to the client.

5.4.2. Supporting Observe

When using CoAP Observe [RFC7641], the proxy acts as a client registered with the servers, as described earlier in Section 5.2.2.

Furthermore, the proxy takes the role of a server when forwarding notifications from origin servers back to the client. To this end, the proxy follows what is specified in Section 3.7 of [I-D.ietf-core-groupcomm-bis] and Section 5 of [RFC7641], with the following additions.

- * At step 1 in Section 5.4, the proxy includes the Response-Forwarding Option in every notification, including non-2.xx notifications resulting in removing the proxy from the list of observers of the origin server.
- * The proxy frees up its Token value used for a group observation only if, after the timeout expiration, no 2.xx (Success) responses matching with the group request and also including an Observe option have been received from any origin server. After that, as long as observations are active with servers in the group for the target resource of the group request, notifications from those servers are forwarded back to the client, as defined in Section 5.4, and the Token value used for the group observation is not freed during this time.

Finally, the proxy SHOULD regularly verify that the client is still interested in receiving observe notifications for a group observation. To this end, the proxy can rely on the same approach discussed for servers in Section 3.7 of [I-D.ietf-core-groupcomm-bis], with more details available in Section 4.5 of [RFC7641].

5.5. Response Processing at the Client

This section defines the operations performed by the client, when receiving a response matching with a request that targeted a group of servers via the proxy.

5.5.1. Response Processing

Upon receiving from the proxy a response matching with the original unicast request before the amount of time T has elapsed, the client proceeds according to the following steps.

1. The client processes the response as defined in [I-D.ietf-core-groupcomm-bis]. When doing so, the client decrypts the response according to the security association it has with the proxy.

2. If secure group communication is used end-to-end between the client and the servers, the client processes the response resulting at the end of step 1, as defined in [I-D.ietf-core-oscore-groupcomm].
3. The client identifies the origin server, whose addressing information is specified as value of the Response-Forwarding Option. If the 'srv_port' element of the 'tp_info' array in the Response-Forwarding Option is not present or specifies the CBOR simple value "null" (0xf6), then the client determines the port number where to send unicast requests to the server -- in case this is needed -- as defined in Section 3. In the former case, the assumed default port number depends on the transport protocol specified by the 'tp_id' element of the 'tp_info' array (see Section 3.2).

In particular, the client is able to distinguish different responses as originated by different servers. Optionally, the client may contact one or more of those servers individually, i.e., directly (bypassing the proxy) or indirectly (via a proxied unicast request).

In order to individually reach an origin server again through the proxy, the client is not required to understand or support the transport protocol indicated in the Response-Forwarding Option, as used between the proxy and the origin server, in case it differs from "UDP" (1). That is, using the IPv4/IPv6 address value and optional port value from the Response-Forwarding Option, the client simply creates the correct URI for the individual request, by means of the Proxy-Uri or Uri-Scheme Option in the unicast request to the proxy. The client uses the transport protocol it knows, and has used before, to send the request to the proxy.

As discussed in Section 3.1.6 of [I-D.ietf-core-groupcomm-bis], it is possible that the client receives multiple responses to the same group request, i.e., with the same Token, from the same origin server. The client normally processes at the CoAP layer each of those responses from the same origin server, and decides how to exactly handle them depending on its available context information (see Section 3.1.6 of [I-D.ietf-core-groupcomm-bis]).

Upon the timeout expiration, i.e., T seconds after having sent the original unicast request to the proxy, the client frees up its local Token value associated with that request. Note that, upon this timeout expiration, the Token value is not eligible for possible reuse yet (see Section 5.1.1). Thus, until the actual amount of time before enabling Token reusage has elapsed, any following late

responses to the same request forwarded by the proxy will be discarded, as these are not matching (by Token) with any active request from the client.

5.5.2. Supporting Observe

When using CoAP Observe [RFC7641], the client frees up its Token value only if, after the timeout T expiration, no 2.xx (Success) responses matching with the original unicast request and also including an Observe option have been received.

Instead, if at least one such response has been received, the client continues receiving those notifications as forwarded by the proxy, as long as the observation for the target resource of the original unicast request is active.

5.6. Example

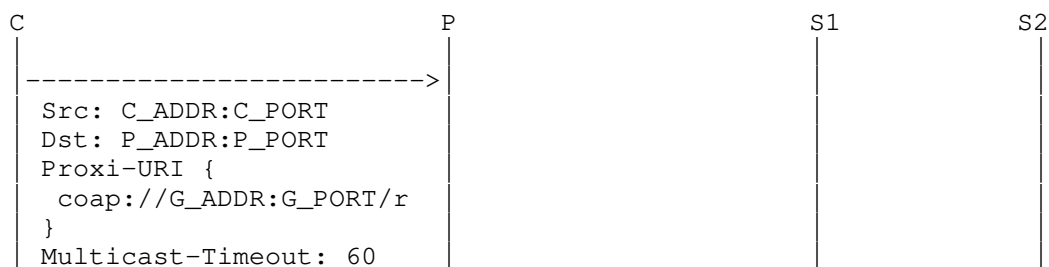
The example in this section refers to the following actors.

- * One origin client C, with address C_ADDR and port number C_PORT.
- * One proxy P, with address P_ADDR and port number P_PORT.
- * Two origin servers S1 and S2, where the server Sx has address Sx_ADDR and port number Sx_PORT.

The origin servers are members of a CoAP group with IP multicast address G_ADDR and port number G_PORT. Also, the origin servers are members of a same application group, and share the same resource /r.

The communication between C and P is based on CoAP over UDP, as per [RFC7252]. The communication between P and the origin servers is based on CoAP over UDP and IP multicast, as per [I-D.ietf-core-groupcomm-bis].

Finally, 'bstr(X)' denotes a CBOR byte string where its value is the byte serialization of X.



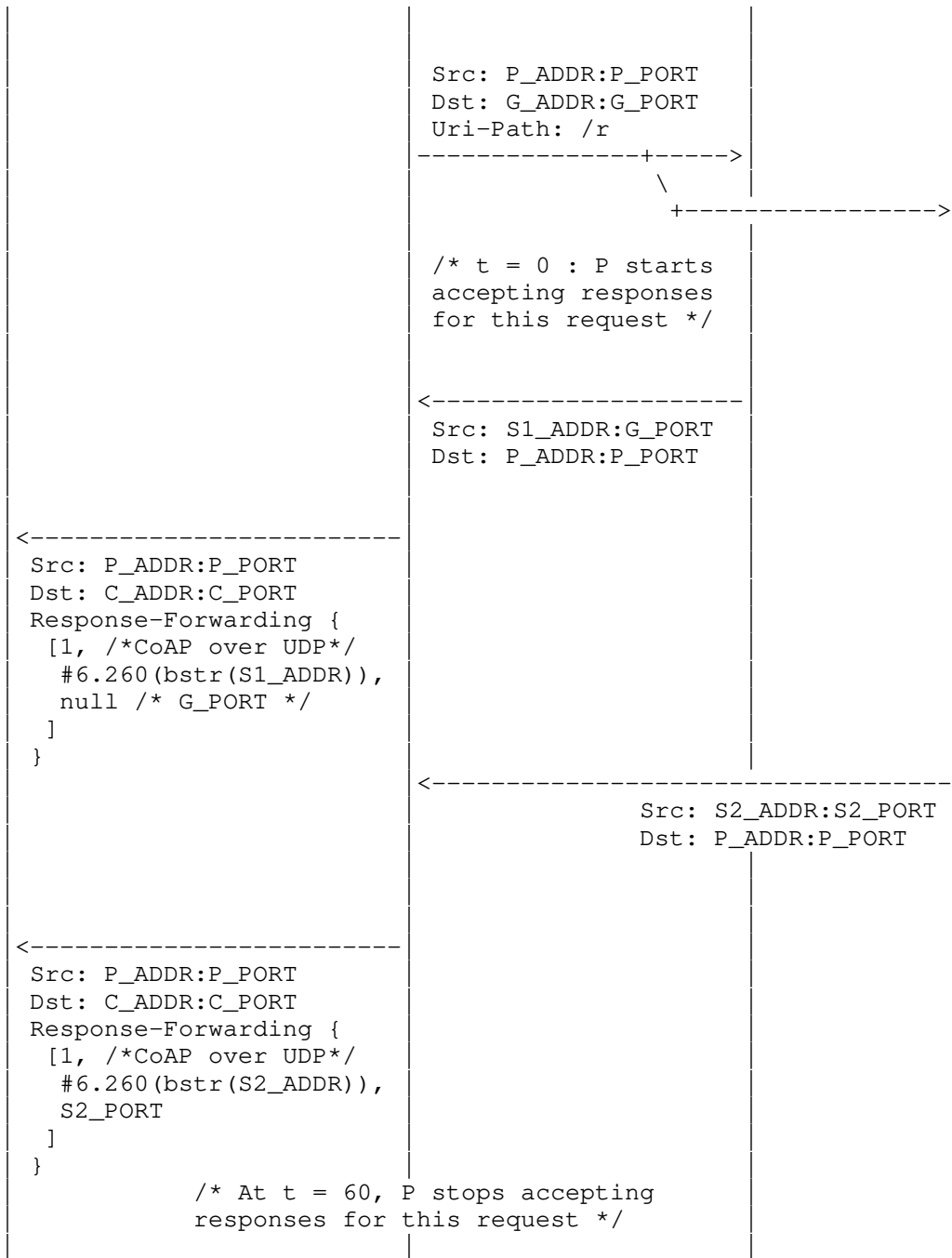


Figure 3: Workflow example with a forward-proxy

6. Reverse-Proxies

The use of reverse-proxies in group communication scenarios is defined in Section 3.5.2 of [I-D.ietf-core-groupcomm-bis].

This section clarifies how the Multicast-Timeout Option is effective also in such a context, in order for:

- * The proxy to explicitly reveal itself as a reverse-proxy to the client.
- * The client to indicate to the proxy of being aware that it is communicating with a reverse-proxy, and for how long it is willing to receive responses to a proxied group request.

This practically addresses the additional issues compared to the case with a forward-proxy, as compiled in Section 3.5.2 of [I-D.ietf-core-groupcomm-bis]. A reverse-proxy may also operate without support of the Multicast-Timeout Option, as defined in that section.

Appendix A provides examples with a reverse-proxy.

6.1. Processing on the Client Side

If a client sends a CoAP request intended to a group of servers and is aware of actually communicating with a reverse-proxy, then the client SHOULD perform the steps defined in Section 5.1.1. In particular, this results in a request sent to the proxy including a Multicast-Timeout Option.

An exception is the case where the reverse-proxy has a pre-configured timeout value `T_PROXY`, as the default timeout value to use for when to stop accepting responses from the servers, after the reception of the original unicast request from the client. In this case, a client aware of such a configuration MAY omit the Multicast-Timeout Option in the request sent to the proxy.

The client processes the CoAP responses forwarded back by the proxy as defined in Section 5.5.

6.2. Processing on the Proxy Side

If the proxy receives a CoAP request and determines that it should be forwarded to a group of servers over IP multicast, then the proxy performs the steps defined in Section 5.2.

In particular, when such a request does not include a Multicast-Timeout Option, the proxy SHOULD explicitly reveal itself as a reverse-proxy, by sending a 4.00 (Bad Request) response including a Multicast-Timeout Option with empty (zero-length) value.

An exception is the case where the reverse-proxy has a pre-configured timeout value `T_PROXY`, as default timeout value to use for when to stop accepting responses from the servers, after the reception of the original unicast request from the client. In this case, the proxy MAY replace the steps 3 and 4 in Section 5.2.1 with the following step.

A. The proxy verifies the presence of the Multicast-Timeout Option, as a confirmation that the client is willing to receive multiple CoAP responses matching with the same original request. Then, the proxy performs the following actions.

- * If the Multicast-Timeout Option is present, the proxy retrieves the value `T'` from the Multicast-Timeout Option, and then removes the option from the client's request. That is, the timeout value indicated in the option overrides the pre-configured timeout value `T_PROXY`.
- * If the Multicast-Timeout option is not present, the proxy checks that, according to its local configuration, both the following conditions hold for the client (which, at this point, has been successfully authenticated).
 - `COND_1` : The client is aware of the default timeout value `T_PROXY` pre-configured at the proxy.
 - `COND_2` : The client is able to process multiple responses to the same request.

These conditions are expected to hold for clients that are locally registered at the proxy, successfully authenticated and allowed-listed to have their requests proxied to CoAP group URIs.

If the proxy is able to successfully assert that both the two conditions hold, then the proxy considers the value `T'` as equal to `T_PROXY` and proceeds to step 5.

If the proxy is not able to successfully assert that both the two conditions hold, the proxy MUST stop processing the request and MUST reply to the client with a 4.00 (Bad Request) response. The response MUST include a Multicast-Timeout Option with an empty (zero-length) value, indicating that the Multicast-Timeout Option was missing and has to be included in the request. As per Section 5.9.2 of [RFC7252] The response SHOULD include a diagnostic payload.

The proxy processes the CoAP responses forwarded back to the client as defined in Section 5.4.

7. Caching

A proxy MAY cache responses to a group request, as defined in Section 5.7.1 of [RFC7252]. In particular, the same rules apply to determine the set of request options used as "Cache-Key", and to determine the max-age values offered for responses served from the cache.

A cache entry is associated with one server and stores one response from that server, regardless whether it is a response to a unicast request or to a group request. The following two types of requests can produce a hit to a cache entry.

- * A matching request intended to that server, i.e., to the corresponding unicast URI.

When the stored response is a response to a unicast request to the server, the unicast URI of the matching request is the same target URI used for the original unicast request.

When the stored response is a response to a group request to the CoAP group, the unicast URI of the matching request is the target URI obtained by replacing the authority part of the group URI in the original group request with the transport-layer source address and port number of the response.

- * A matching group request intended to the CoAP group, i.e., to the corresponding group URI.

That is, a matching group request produces a hit to multiple cache entries, each of which associated with one of the CoAP servers currently member of the CoAP group.

Note that, as per the freshness model defined in Section 7.1, the proxy might serve a group request exclusively from its cached responses only when it knows all the CoAP servers that are current members of the CoAP group and it has a valid cache entry for each of them.

When forwarding a GET or FETCH group request to the servers in the CoAP group, the proxy behaves like a CoAP client as defined in Section 3.2 of [I-D.ietf-core-groupcomm-bis], with the following additions.

- * As discussed in Section 5.4.1, the proxy can receive multiple responses to the same group request from a same origin server, and forwards them back to the origin client "as they come". When this happens, each of such multiple responses is stored in the cache entry associated with the server "as it comes", possibly replacing an already stored response from that server.
- * As discussed in Section 7.4, when communications in the group are secured with Group OSCORE [I-D.ietf-core-oscore-groupcomm], additional means are required to enable cacheability of responses at the proxy.

The following subsections define the freshness model and validation model that the proxy uses for cached responses.

7.1. Freshness Model

The proxy relies on the same freshness model defined in Section 3.2.1 of [I-D.ietf-core-groupcomm-bis], by taking the role of a CoAP client with respect to the servers in the CoAP group.

In particular, when receiving a unicast group request from the client, the proxy MAY serve it by using exclusively cached responses without forwarding the group request to the servers in the CoAP group, but only if both the following conditions hold.

- * The proxy knows all the CoAP servers that are currently members of the CoAP group for which the group request is intended to.
- * The proxy's cache currently stores a fresh response for each of those CoAP servers.

The specific way that the proxy uses to determine the CoAP servers currently members of the target CoAP group is out of scope for this document. As possible examples, the proxy can synchronize with a group manager server; rely on well-known time patterns used in the application or in the network for the addition of new CoAP group members; observe group join requests or IGMP/MLD multicast group join messages, e.g., if embedded in a multicast router.

When forwarding the group request to the servers, the proxy may have fresh responses stored in its cache for (some of) those servers. In such a case, the proxy uses (also) those cached responses to serve the original unicast group request, as defined below.

- * The request processing in Section 5.2.1 is extended as follows.

After setting the timeout with value $T' > 0$ in step 6, the proxy checks whether its cache currently stores fresh responses to the group request. For each of such responses, the proxy compares the residual lifetime L of the corresponding cache entry against the value T' .

If a cached response X is such that $L < T'$, then the proxy forwards X back to the client at its earliest convenience. Otherwise, the proxy does not forward X back to the client right away, and rather waits for approaching the timeout expiration, as discussed in the next point.

- * The response processing in Section 5.4.1 is extended as follows.

Before the timeout with original value $T' > 0$ expires and the proxy stops accepting responses to the group request, the proxy checks whether it stores in its cache any fresh response X to the group request such that both the following conditions hold.

- The cache entry E storing X was already existing when the proxy forwarded the group request.
- The proxy has received no response to the forwarded group request from the server associated with E .

Then, the proxy sends back to the client each response X stored in its cache and selected as above, before the timeout expires.

Note that, from the forwarding of the group request until the timeout expiration, the proxy still forwards responses to the group request back to the client "as they come" (see Section 5.4.1). Also, such responses possibly refresh older responses from the same servers that the proxy has stored in its cache, as defined earlier in Section 7.

7.2. Validation Model

This section defines the revalidation of responses, separately between the proxy and the origin servers, as well as between the origin client and the proxy.

7.2.1. Proxy-Servers Revalidation with Unicast Requests

The proxy MAY revalidate a cached response by making a GET or FETCH request on the related unicast request URI, i.e., by taking the role of a CoAP client with respect to a server in the CoAP group.

As discussed in Section 7.4, this is however not possible for the proxy if communications in the group are secured end-to-end between origin client and origin servers by using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

[TODO

It can be actually possible to enable revalidation of responses between proxy and server, also in this case where Group OSCORE is used end-to-end between client and origin servers.

Fundamentally, this requires to define the possible use of the ETag option also as an outer option for OSCORE. Thus, in addition to the normal inner ETag, a server can add also an outer ETag option intended to the proxy.

Since validation of responses assumes that cacheability of responses is possible in the first place, it would be convenient to define the use of ETag as outer option in [I-D.amsuess-core-cachable-oscore].

In case OSCORE is also used between the proxy and an individual origin server as per [I-D.tiloca-core-oscore-capable-proxies], then the outer ETag option would be seamlessly protected with the OSCORE Security Context shared between the proxy and the origin server.

The following text can be used to replace the last paragraph above.

As discussed in Section 7.4, the following applies when Group OSCORE [I-D.ietf-core-oscore-groupcomm] is used to secure communications end-to-end between the origin client and the origin servers in the group.

- * Additional means are required to enable cacheability of responses at the proxy (see Section 7.4.1).
- * If a cached response included an outer ETag option intended to the proxy, then the proxy can perform revalidation of the cached response, by making a request to the unicast URI targeting the server, and including outer ETag Option(s).

This is possible also in case the proxy and the origin server use OSCORE to further protect the exchanged request and response, as defined in [I-D.tiloca-core-oscore-capable-proxies]. In such a case, the originally outer ETag option is protected with the OSCORE Security Context shared between the proxy and the origin server, before transferring the message over the communication leg between the proxy and origin server.

]

7.2.2. Proxy-Servers Revalidation with Group Requests

When forwarding a group request to the servers in the CoAP group, the proxy MAY revalidate one or more stored responses that it has cached.

To this end, the proxy relies on the same validation model defined in Section 3.2.2 of [I-D.ietf-core-groupcomm-bis] and using the ETag Option, by taking the role of a CoAP client with respect to the servers in the CoAP group.

As discussed in Section 7.4, this is however not possible for the proxy if communications in the group are secured end-to-end between origin client and origin servers by using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

[TODO

See the notes in Section 7.2.1.

The following text can be used to replace the last paragraph above.

As discussed in Section 7.4, the following applies when Group OSCORE [I-D.ietf-core-oscore-groupcomm] is used to secure communications end-to-end between the origin client and the origin servers in the group.

- * Additional means are required to enable cacheability of responses at the proxy (see Section 7.4.1).
- * If a cached response included an outer ETag option intended to the proxy, then the proxy can perform revalidation of the cached response, by making a request to the group URI targeting the CoAP group, and including outer ETag Option(s).

This is possible also in case the proxy and the origin servers use Group OSCORE to further protect the exchanged request and response, as defined in [I-D.tiloca-core-oscore-capable-proxies]. In such a case, the originally outer ETag option is protected with the Group OSCORE Security Context shared between the proxy and the origin server, before transferring the message over the communication leg between the proxy and origin server.

]

7.3. Client-Proxy Revalidation with Group Requests

A client MAY revalidate the full set of responses to a group request by leveraging the corresponding cache entries at the proxy. To this end, this document defines the new Group-ETag Option.

The Group-ETag Option has the properties summarized in Figure 4, which extends Table 4 of [RFC7252]. The Group-ETag Option is elective, safe to forward, part of the cache key, and repeatable.

The option is intended for group requests sent to a proxy to be forwarded to the servers in a CoAP group, as well as for the associated responses.

No.	C	U	N	R	Name	Format	Length	Default
TBD3				x	Group-ETag	opaque	1-8	(none)

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

Figure 4: The Group-ETag Option.

The Group-ETag Option has the same properties of the ETag Option defined in Section 5.10.6 of [RFC7252].

The Group-ETag Option is of class U in terms of OSCORE processing (see Section 4.1 of [RFC8613]).

A proxy MUST NOT provide this form of validation if it is not in a position to serve a group request by using exclusively cached responses, i.e., without sending the group request to the servers in the CoAP group (see Section 7.1).

If the proxy supports this form of response revalidation, the following applies.

- * The proxy defines J as a joint set including all the cache entries currently storing fresh responses that satisfy a group request. A set J is "complete" if it includes a valid cache entry for each of the CoAP servers currently members of the CoAP group.
- * When the set J becomes "complete", the proxy assigns it an entity-tag value. The proxy MUST update the current entity-tag value, when J is "complete" and one of its cache entry is updated.
- * When forwarding to the client a 2.05 (Content) response to a GET or FETCH group request, the proxy MAY include one Group-ETag Option, in case the set J is "complete". Such a response MUST NOT include more than one Group-ETag Option. The option value specifies the entity-tag value currently associated with the set J.

When sending to the proxy a GET or FETCH request to be forwarded to the servers in the CoAP group, the client MAY include one or more Group-ETag Options. Each option specifies one entity-tag value, applicable to the set J of cache entries that can be hit by the group request.

The proxy MAY perform the following actions, in case the group request produces a hit to the cache entry of each CoAP server currently member of the CoAP group, i.e., the set J associated with the group request is "complete".

- * The proxy checks whether the current entity-tag value of the set J matches with one of the entity-tag values specified in the Group-ETag Options of the unicast group request from the client.

- * In case of positive match, the proxy replies with a single 2.03 (Valid) response. This response has no payload and MUST include one Group-ETag Option, specifying the current entity-tag value of the set J.

That is, the 2.03 (Valid) response from the proxy indicates to the client that the stored responses identified by the entity-tag given in the response's Group-ETag Option can be reused, after updating each of them as described in Section 5.9.1.3 of [RFC7252]. In effect, the client can determine if any of the stored representations from the respective cache entries at the proxy is current, without needing to transfer any of them again.

7.4. Caching of End-To-End Protected Responses at Proxies

When using Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect communications end-to-end between a client and multiple servers in the group, it is normally not possible for an intermediary proxy to cache protected responses.

In fact, when starting from the same plain CoAP message, different clients generate different protected requests to send on the wire. This prevents different clients to generate potential cache hits, and thus makes response caching at the proxy pointless.

7.4.1. Deterministic Requests to Achieve Cacheability

For application scenarios that use secure group communication, it is still possible to achieve cacheability of responses at proxies, by using the approach defined in [I-D.amsuess-core-cachable-oscore] which is based on Deterministic Requests protected with the pairwise mode of Group OSCORE. This approach is limited to group requests that are safe (in the RESTful sense) to process and do not yield side effects at the server. As for any protected group request, it requires the clients and all the servers in the CoAP group to have already joined the correct OSCORE group.

Starting from the same plain CoAP request, this allows different clients in the OSCORE group to deterministically generate a same request protected with Group OSCORE, which is sent to the proxy for being forwarded to the CoAP group. The proxy can now effectively cache the resulting responses from the servers in the CoAP group, since the same plain CoAP request will result again in the same Deterministic Request and thus will produce a cache hit.

When caching of Group OSCORE secured responses is enabled at the proxy, the same as defined in Section 7 applies, with respect to cache entries and their lifetimes.

Note that different Deterministic Requests result in different cache entries at the proxy. This includes the case where different plain group requests differ only in their set of ETag Options, as defined in Section 3.2.2 of [I-D.ietf-core-groupcomm-bis].

That is, even though the servers would produce the same plain CoAP responses in reply to two different Deterministic Requests, those will result in different protected responses to each respective Deterministic Request, hence in different cache entries at the proxy.

Thus, given a plain group request, a client needs to reuse the same set of ETag Options, in order to send that group request as a Deterministic Request that can actually produce a cache hit at the proxy. However, while this would prevent the caching at the proxy to be inefficient and unnecessarily redundant, it would also limit the flexibility of end-to-end response revalidation for a client.

7.4.2. Validation of Responses

Response revalidation remains possible end-to-end between the client and the servers in the group, by means of including inner ETag Option(s) as defined in Sections 3.2 and 3.2.2 of [I-D.ietf-core-groupcomm-bis].

Furthermore, it remains possible for a client to attempt revalidating responses to a group request from a "complete" set of cache entries at the proxy, by using the Group-ETag Option as defined in Section 7.3.

When directly interacting with the servers in the CoAP group to refresh its cache entries, the proxy cannot rely on response revalidation anymore. This applies to both the case where the request is addressed to a single server and sent to the related unicast URI (see Section 7.2.1) or instead is a group request addressed to the CoAP group and sent to the related group URI (see Section 7.2.2).

[TODO

See the notes in Section 7.2.1.

The following text can be used to replace the last paragraph above.

When directly interacting with the servers in the CoAP group to refresh its cache entries, the proxy also remains able to perform response revalidation. That is, if a cached response included an outer ETag option intended to the proxy, then the proxy can perform

revalidation of the cached response, by making a request to the unicast URI addressed to a single server and sent to the related unicast URI (see Section 7.2.1) or a group request addressed to the CoAP group and sent to the related group URI (see Section 7.2.2).

]

8. Chain of Proxies

A client may be interested to access a resource at a group of origin servers which is reached through a chain of two or more proxies.

That is, these proxies are configured into a chain, where each non-last proxy is configured to forward (group) requests to the next hop towards the origin servers. Also, each non-first proxy is configured to forward back responses to (the previous hop proxy towards) the origin client.

This section specifies how the signaling protocol defined in Section 5 is used in that setting. Except for the last proxy before the origin servers, every other proxy in the chain takes the role of client with respect to the next hop towards the origin servers. Also, every proxy in the chain except the first takes the role of server towards the previous proxy closer to the origin client.

Accordingly, possible caching of responses at each proxy works as defined in Section 7 and Section 7.4. Also, possible revalidation of responses cached at each proxy and based on the Group-ETag option works as defined in Section 7.3 and Section 7.4.2.

The requirements REQ1 and REQ2 defined in Section 4 MUST be fulfilled for each proxy in the chain. That is, every proxy in the chain has to be explicitly configured (allow-list) to allow proxied group requests from specific senders, and MUST identify those senders upon receiving their group request. For the first proxy in the chain, that sender is the origin client. For each other proxy in the chain, that sender is the previous hop proxy closer to the origin client. In either case, a proxy can identify the sender of a group request by the same means mentioned in Section 4.

8.1. Request Processing at the Proxy

Upon receiving a group request to be forwarded to a CoAP group URIs, a proxy proceed as follows.

If the proxy is the last one in the chain, i.e., it is the last hop before the origin servers, the proxy performs the steps defined in Section 5.2, with no modifications.

Otherwise, the proxy performs the steps defined in Section 5.2, with the following differences.

- * At steps 1-3, "client" refers to the origin client for the first proxy in the chain; or to the previous hop proxy closer to the origin client, otherwise.
- * At step 4, the proxy rather performs the following actions.
 1. The proxy retrieves the value T' from the Multicast-Timeout Option, and does not remove the option.
 2. In case $T' > 0$, the proxy picks an amount of time T it is fine to wait for before freeing up its local Token value to use with the next hop towards the origin servers. To this end, the proxy MUST follow what is defined at step 2 of Section 5.1.1 for the origin client, with the following differences.
 - T MUST be greater than the retrieved value T' , i.e., $T' < T$.
 - The worst-case message processing time takes into account all the next hops towards the origin servers, as well as the origin servers themselves.
 - The worst-case round-trip delay takes into account all the legs between the proxy and the origin servers.
 3. In case $T' > 0$, the proxy replaces the value of the Multicast-Timeout Option with a new value T'' , such that:
 - $T'' < T$. The difference $(T - T'')$ should be at least the expected worst-case round-trip time between the proxy and the next hop towards the origin servers.
 - $T'' < T'$. The difference $(T' - T'')$ should be at least the expected worst-case round-trip time between the proxy and the (previous hop proxy closer to the) origin client.

If the proxy is not able to determine a value T'' that fulfills both the requirements above, the proxy MUST stop processing the request and MUST respond with a 5.05 (Proxying Not Supported) error response to the previous hop proxy closer to the origin client. The proxy SHOULD include a Multicast-Timeout Option, set to the minimum value T' that would be acceptable in the Multicast-Timeout Option of a group request to forward.

Upon receiving such an error response, any proxy in the chain MAY send an updated group request to the next hop towards the origin servers, specifying in the Multicast-Timeout Option a value T' greater than in the previous request. If this does not happen, the proxy receiving the error response MUST also send a 5.05 (Proxying Not Supported) error response to the previous hop proxy closer to the origin client. Like the received one, also this error response SHOULD include a Multicast-Timeout Option, set to the minimum value T' acceptable by the proxy sending the error response.

- * At step 5, the proxy forwards the request to the next hop towards the origin servers.
- * At step 6, the proxy sets a timeout with the value T' retrieved from the Multicast-Timeout Option of the request received from the (previous hop proxy closer to the) origin client.

In case $T' > 0$, the proxy will ignore responses to the forwarded group request coming from the (next hop towards the) origin servers, if received after the timeout expiration, with the exception of Observe notifications (see Section 5.4).

In case $T' = 0$, the proxy will ignore all responses to the forwarded group request coming from the (next hop towards the) origin servers.

8.1.1. Supporting Observe

When using CoAP Observe [RFC7641], what is defined in Section 5.2.2 applies for the last proxy in the chain, i.e., the last hop before the origin servers.

Any other proxy in the chain acts as a client and registers its own interest to observe the target resource with the next hop towards the origin servers, as per Section 5 of [RFC7641].

8.2. Response Processing at the Proxy

Upon receiving a response matching with the group request before the amount of time T' has elapsed, the proxy proceeds as follows.

If the proxy is the last one in the chain, i.e., it is the last hop before the origin servers, the proxy performs the steps defined in Section 5.4, with no modifications.

Otherwise, the proxy performs the steps defined in Section 5.4, with the following differences.

- * The proxy skips step 1. In particular, the proxy MUST NOT remove, alter or replace the Response-Forwarding Option.
- * At step 2, "client" refers to the origin client for the first proxy in the chain; or to the previous hop proxy closer to the origin client, otherwise.

As to the possible reception of multiple responses to the same group request from the same (next hop proxy towards the) origin server, the same as defined in Section 5.4.1 applies. That is, as long as the proxy forwards responses to a group request back to the (previous hop proxy closer to the) origin client, the proxy MUST follow the steps above and forward also such multiple responses "as they come".

Upon timeout expiration, i.e., T seconds after having forwarded the group request to the next hop towards the origin servers, the proxy frees up its local Token value associated with that request. Thus, following late responses to the same group request will be discarded and not forwarded back to the (previous hop proxy closer to the) origin client.

8.2.1. Supporting Observe

When using CoAP Observe [RFC7641], what is defined in Section 5.4.2 applies for the last proxy in the chain, i.e., the last hop before the origin servers.

As to any other proxy in the chain, the following applies.

- * The proxy acts as a client registered with the next hop towards the origin servers, as described earlier in Section 8.1.1.
- * The proxy takes the role of a server when forwarding notifications from the next hop to the origin servers back to the (previous hop proxy closer to the) origin client, as per Section 5 of [RFC7641].
- * The proxy frees up its Token value used for a group observation only if, after the timeout expiration, no 2.xx (Success) responses matching with the group request and also including an Observe option have been received from the next hop towards the origin servers. After that, as long as the observation for the target resource of the group request is active with the next hop towards the origin servers in the group, notifications from that hop are forwarded back to the (previous hop proxy closer to the) origin client, as defined in Section 8.2.

- * The proxy SHOULD regularly verify that the (previous hop proxy closer to the) origin client is still interested in receiving observe notifications for a group observation. To this end, the proxy can rely on the same approach defined in Section 4.5 of [RFC7641].

9. HTTP-CoAP Proxies

This section defines the components needed to use the signaling protocol specified in this document, when an HTTP client wishes to send a group request to the servers of a CoAP group, via an HTTP-CoAP cross-proxy.

The following builds on the mapping of the CoAP request/response model to HTTP and vice versa as defined in Section 10 of [RFC7252], as well as on the additional details about the HTTP-CoAP mapping defined in [RFC8075].

Furthermore, the components defined in Section 11 of [RFC8613] are also used to map and transport OSCORE-protected messages over HTTP. This allows an HTTP client to use Group OSCORE end-to-end with the servers in the CoAP group.

9.1. The HTTP Multicast-Timeout Header Field

The HTTP Multicast-Timeout header field (see Section 11.3) is used for carrying the content otherwise specified in the CoAP Multicast-Timeout Option defined in Section 2.

Using the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] and including the core ABNF syntax rule DIGIT (decimal digits) defined by that specification, the HTTP Multicast-Timeout header field value is as follows.

Multicast-Timeout = *DIGIT

When translating a CoAP message into an HTTP message, the HTTP Multicast-Timeout header field is set with the content of the CoAP Multicast-Timeout Option, or is left empty in case the option is empty.

The same applies in the opposite direction, when translating an HTTP message into a CoAP message.

9.2. The HTTP Response-Forwarding Header Field

The HTTP Response-Forwarding header field (see Section 11.3) is used for carrying the content otherwise specified in the CoAP Response-Forwarding Option defined in Section 3.

Using the Uniform Resource Identifier (URI) syntax components defined in [RFC3986], the HTTP Response-Forwarding header field value is as follows.

scheme = <scheme, see Section 3.1 of [RFC3986]>

authority = <authority, see Section 3.2 of [RFC3986]>

Response-Forwarding = scheme "://" authority

In particular:

- * The scheme component indicates the URI scheme otherwise specified in the CoAP Response-Forwarding Option, as per the 'tp_id' element of the 'tp_info' array (see Section 3). That is, the 'tp_id' element with integer value 1 results in the scheme "coap".
- * The authority component indicates the URI authority otherwise specified in the CoAP Response-Forwarding Option, as per the 'srv_host' and 'srv_port' elements of the 'tp_info' array (see Section 3).

When translating a CoAP message into an HTTP message, the HTTP Response-Forwarding header field is set to the URI specified in the CoAP Response-Forwarding Option, as per the rules defined above. In particular, consistently with what is defined in Section 3:

- * If the 'srv_port' element of the 'tp_info' array is present and specifies the CBOR simple value "null" (0xf6), the URI authority of the header field includes the same port number that was specified in the group URI where the group request was forwarded.
- * If the 'srv_port' element of the 'tp_info' array is not present, the URI authority of the header field includes the default port number for the transport protocol specified by the 'tp_id' element of the 'tp_info' array, as per Section 3.2.

When translating an HTTP message into a CoAP message, the CoAP Response-Forwarding Option is set to the URI specified by the HTTP Response-Forwarding header field. In particular, the URI is encoded according to the format specified in Section 3.

9.3. The HTTP Group-ETag Header Field

The HTTP Group-ETag header field (see Section 11.3) is used for carrying the content otherwise specified in the CoAP Group-ETag Option defined in Section 7.3.

Using the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] and including the following core ABNF syntax rules defined by that specification: ALPHA (letters) and DIGIT (decimal digits), the HTTP Group-ETag header field value is as follows.

```
group-etag-char = ALPHA / DIGIT / "-" / "_"
```

```
Group-ETag = 2*group-etag-char
```

When translating a CoAP message into an HTTP message, the HTTP Group-ETag header field is set to the value of the CoAP Group-ETag Option in base64url (see Section 5 of [RFC4648]) encoding without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].

When translating an HTTP message into a CoAP message, the CoAP Group-ETag Option is set to the value of the HTTP Group-ETag header field decoded from base64url (see Section 5 of [RFC4648]) without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].

9.4. Request Sending at the Client

The client proceeds according to the following steps.

1. The client prepares an HTTP request to send to the proxy via IP unicast, and to be forwarded by the proxy to the targeted group of CoAP servers over IP multicast. With reference to Section 5 of [RFC8075], the request is addressed to a Hosting HTTP URI, such that the proxy can extract the Target CoAP URI as the group URI where to forward the request.
2. The client determines the amount of time T that it is fine to wait for a response to the request from the proxy. Then, the client determines the amount of time $T' < T$, where the difference $(T - T')$ should be at least the expected worst-case round-trip time between the client and the proxy.
3. If Group OSCORE is used end-to-end between the client and the servers, the client translates the HTTP request into a CoAP request, as per [RFC8075]. Then, the client protects the resulting CoAP request by using Group OSCORE, as defined in

[I-D.ietf-core-oscore-groupcomm]. Finally, the protected CoAP request is mapped to HTTP as defined in Section 11.2 of [RFC8613]. Later on, the resulting HTTP request MUST be sent in compliance with the rules in Section 11.1 of [RFC8613].

4. The client includes the HTTP Multicast-Timeout header field in the request, specifying T' as its value. The client can specify $T' = 0$, thus indicating to be not interested in receiving responses from the origin servers through the proxy.
5. If the client wishes to revalidate responses to a previous group request from the corresponding cache entries at the proxy (see Section 7.3), the client includes one or multiple HTTP Group-ETag header fields in the request (see Section 9.3), each specifying an entity-tag value like they would in a corresponding CoAP Group E-Tag option.
6. The client sends the request to the proxy, as a unicast HTTP message. In particular, the client protects the request according to the security association it has with the proxy.

9.5. Request Processing at the Proxy

The proxy translates the HTTP request to a CoAP request, as per [RFC8075]. The additional rules for HTTP messages with the HTTP Multicast-Timeout header field and HTTP Group-ETag header field are defined in Section 9.1 and Section 9.3, respectively.

Once translated the HTTP request into a CoAP request, the proxy MUST perform the steps defined in Section 5.2. If the proxy supports caching of responses, it can serve the unicast request also by using cached responses as per Section 7, considering the CoAP request above as the potentially matching request.

In addition, in case the HTTP Multicast-Timeout header field had value 0, the proxy replies to the client with an HTTP response with status code 204 (No Content), right after forwarding the group request to the group of servers.

9.6. Response Processing at the Proxy

Upon receiving a CoAP response matching with the group request before the amount of time $T' > 0$ has elapsed, the proxy includes the Response-Forwarding Option in the response, as per step 1 of Section 5.4.1. Then, the proxy translates the CoAP response to an HTTP response, as per Section 10.1 of [RFC7252] and [RFC8075], as well as Section 11.2 of [RFC8613] if Group OSCORE is used end-to-end between the client and servers. The additional rules for CoAP

messages specifying the Response-Forwarding Option are defined in Section 9.2.

After that, the proxy stores the resulting HTTP response until the timeout with original value $T' > 0$ expires. If, before then, the proxy receives another response to the same group request from the same CoAP server, the proxy performs the steps above, and stores the resulting HTTP response by superseding the currently stored one from that server.

When the timeout expires, if no responses have been received from the servers, the proxy replies to the client's original unicast group request with an HTTP response with status code 204 (No Content).

Otherwise, the proxy relays to the client all the collected and stored HTTP responses to the group request, according to the following steps.

1. The proxy prepares a single HTTP batch response, which MUST have 200 (OK) status code and MUST have its HTTP Content-Type header field with value multipart/mixed [RFC2046].
2. For each stored individual HTTP response RESP, the proxy prepares a corresponding batch part to include in the HTTP batch response, such that:
 - * The batch part has its own HTTP Content-Type header field with value application/http [RFC7230].
 - * The body of the batch part is the individual HTTP response RESP, including its status code, headers and body.
3. The proxy includes each batch part prepared at step 2 in the HTTP batch response.
4. The proxy replies to the client's original unicast group request, by sending the HTTP batch response. When doing so, the proxy protects the response according to the security association it has with the client.

9.7. Response Processing at the Client

When it receives an HTTP response as a reply to the original unicast group request, the client proceeds as follows.

1. The client decrypts the response, according to the security association it has with the proxy.

2. From the resulting HTTP batch response, the client extracts the different batch parts.
3. From each of the extracted batch parts, the client extracts the body as one of the individual HTTP response RESP.
4. For each individual HTTP response RESP, the client performs the following steps.
 - * If Group OSCORE is used end-to-end between the client and servers, the client translates the HTTP response RESP into a CoAP response, as per Section 11.3 of [RFC8613]. Then, the client decrypts the resulting CoAP response by using Group OSCORE, as defined in [I-D.ietf-core-oscure-groupcomm]. Finally, the decrypted CoAP response is mapped to HTTP as per Section 10.2 of [RFC7252] as well as [RFC8075]. The additional rules for HTTP messages with the HTTP Response-Forwarding header field are defined in Section 9.2.
 - * The client delivers to the application the individual HTTP response.

Similarly to step 3 in Section 5.5.1, the client identifies the origin server that originated the CoAP response corresponding to the HTTP response RESP, by means of its addressing information specified as value of the HTTP Response-Forwarding header field. This allows the client to distinguish different individual HTTP responses as corresponding to different CoAP responses from the servers in the CoAP group.

9.8. Example

The examples in this section build on Section 5.6, with the difference that the origin client C is an HTTP client and the proxy P is an HTTP-CoAP cross-proxy. The examples are simply illustrative and are not to be intended as a test vector.

The following is an example of unicast group request sent by C to P. The URI mapping and notation are based on the "Simple Form" defined in Section 5.4.1 of [RFC8075].

```
POST https://proxy.url/hc/?target_uri=coap://G_ADDR:G_PORT/ HTTP/1.1
Content-Length: <REQUEST_TOTAL_CONTENT_LENGTH>
Content-Type: text/plain
Multicast-Timeout: 60
```

Body: Do that!

The following is an example of HTTP batch response sent by P to C, as a reply to the client's original unicast group request.

```
HTTP/1.1 200 OK
Content-Length: <BATCH_RESPONSE_TOTAL_CONTENT_LENGTH>
Content-Type: multipart/mixed; boundary=batch_foo_bar

--batch_foo_bar
Content-Type: application/http

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: <INDIVIDUAL_RESPONSE_1_CONTENT_LENGTH>
Response-Forwarding: coap://S1_ADDR:G_PORT

Body: Done!
--batch_foo_bar
Content-Type: application/http

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: <INDIVIDUAL_RESPONSE_2_CONTENT_LENGTH>
Response-Forwarding: coap://S2_ADDR:S2_PORT

Body: More than done!
--batch_foo_bar--
```

9.9. Streamed Delivery of Responses to the Client

[TODO

The proxy might still be able to forward back individual responses to the client in a streamed fashion.

Individual responses can be forwarded back one by one as they come (like a CoAP-to-CoAP proxy does), or as soon as a certain amount of them have been received from the servers.

This can be achieved by combining the Content-Type multipart/mixed used in the previous sections with the Transfer-Coding "chunked" specified in RFC 7230.

The above applies to HTTP 1.1, while HTTP/2 has its own mechanisms for data streaming.

]

9.10. Reverse-Proxies

In case an HTTP-to-CoAP proxy acts specifically as a reverse-proxy, the same principles defined in Section 6 applies, as specified below.

9.10.1. Processing on the Client Side

If an HTTP client sends a request intended to a group of servers and is aware of actually communicating with a reverse-proxy, then the client SHOULD perform the steps defined in Section 9.4. In particular, this results in a request sent to the proxy including a Multicast-Timeout header field.

An exception is the case where the reverse-proxy has a pre-configured timeout value `T_PROXY`, as the default timeout value to use for when to stop accepting responses from the servers, after the reception of the original unicast request from the client. In this case, a client aware of such a configuration MAY omit the Multicast-Timeout header field in the request sent to the proxy.

The client processes the HTTP response forwarded back by the proxy as defined in Section 9.7.

9.10.2. Processing on the Proxy Side

If the proxy receives a request and determines that it should be forwarded to a group of servers over IP multicast, then the same as defined in Section 9.5 applies, with the following difference.

Once translated the HTTP request into a CoAP request, the proxy performs what is defined in Section 6.2. Note that, in this case, the condition `COND_2` always holds, since the proxy is going to send to the client at most one response, i.e., the HTTP batch response (see Section 9.6).

The proxy processes the HTTP response sent to the client as defined in Section 9.6.

10. Security Considerations

The security considerations from [RFC7252][I-D.ietf-core-groupcomm-bis][RFC8613][I-D.ietf-core-oscore-groupcomm] hold for this document.

When a chain of proxies is used (see Section 8), the secure communication between any two adjacent hops is independent.

When Group OSCORE is used for end-to-end secure group communication between the origin client and the origin servers, this security association is unaffected by the possible presence of a proxy or a chain of proxies.

Furthermore, the following additional considerations hold.

10.1. Client Authentication

As per the requirement REQ2 (see Section 4), the client has to authenticate to the proxy when sending a group request to forward. This leverages an established security association between the client and the proxy, that the client uses to protect the group request, before sending it to the proxy.

If the group request is (also) protected end-to-end between the client and the servers using the group mode of Group OSCORE, the proxy can act as external signature checker (see Section 8.5 of [I-D.ietf-core-oscore-groupcomm]) and authenticate the client by successfully verifying the signature embedded in the group request. However, this requires that, for each client to authenticate, the proxy stores the authentication credential and public key included therein used by that client in the OSCORE group. This in turn would require a form of active synchronization between the proxy and the Group Manager for that group [I-D.ietf-core-oscore-groupcomm].

Nevertheless, the client and the proxy SHOULD still rely on a full-fledged pairwise secure association. In addition to ensuring the integrity of group requests sent to the proxy (see Section 10.2, Section 10.3 and Section 10.4), this prevents the proxy from forwarding replayed group requests with a valid signature, as possibly injected by an active, on-path adversary.

The same considerations apply when a chain of proxies is used (see Section 8), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

10.2. Multicast-Timeout Option

The Multicast-Timeout Option is of class U for OSCORE [RFC8613]. Hence, also when Group OSCORE is used between the client and the servers [I-D.ietf-core-oscore-groupcomm], a proxy is able to access the option value and retrieve the timeout value T' , as well as to remove the option altogether before forwarding the group request to the servers. When a chain of proxies is used (see Section 8), this also allows each proxy but the last one in the chain to update the option value, as an indication for the next hop towards the origin servers (see Section 8.1).

The security association between the client and the proxy MUST provide message integrity, so that further intermediaries between the two as well as on-path active adversaries are not able to remove the option or alter its content, before the group request reaches the proxy. Removing the option would otherwise result in not forwarding the group request to the servers. Instead, altering the option content would result in the proxy accepting and forwarding back responses for an amount of time different than the one actually indicated by the client.

The security association between the client and the proxy SHOULD also provide message confidentiality. Otherwise, any further intermediaries between the two as well as any on-path passive adversaries would be able to simply access the option content, and thus learn for how long the client is willing to receive responses from the servers in the group via the proxy. This may in turn be used to perform a more efficient, selective suppression of responses from the servers.

When the client protects the unicast request sent to the proxy using OSCORE (see [I-D.tiloca-core-oscore-capable-proxies]) and/or (D)TLS, both message integrity and message confidentiality are achieved in the leg between the client and the proxy.

The same considerations above about security associations apply when a chain of proxies is used (see Section 8), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

10.3. Response-Forwarding Option

The Response-Forwarding Option is of class U for OSCORE [RFC8613]. Hence, also when Group OSCORE is used between the client and the servers [I-D.ietf-core-oscore-groupcomm], the proxy that has forwarded the group request to the servers is able to include the option into a server response, before forwarding this response back to the (previous hop proxy closer to the) origin client.

Since the security association between the client and the proxy provides message integrity, any further intermediaries between the two as well as any on-path active adversaries are not able to undetectably remove the Response-Forwarding Option from a forwarded server response. This ensures that the client can correctly distinguish the different responses and identify their corresponding origin server.

When the proxy protects the response forwarded back to the client using OSCORE (see [I-D.tiloca-core-oscore-capable-proxies]) and/or (D)TLS, message integrity is achieved in the leg between the client and the proxy.

The same considerations above about security associations apply when a chain of proxies is used (see Section 8), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

10.4. Group-ETag Option

The Group-ETag Option is of class U for OSCORE [RFC8613]. Hence, also when Group OSCORE is used between the client and the servers [I-D.ietf-core-oscore-groupcomm], a proxy is able to access the option value and use it to possibly perform response revalidation at its cache entries associated with the servers in the CoAP group, as well as to remove the option altogether before forwarding the group request to the servers. When a chain of proxies is used (see Section 8), this also allows each proxy but the last one in the chain to update the option value, to possibly ask the next hop towards the origin servers to perform response revalidation at its cache entries.

The security association between the client and the proxy MUST provide message integrity, so that further intermediaries between the two as well as on-path active adversaries are not able to remove the option or alter its content, before the group request reaches the proxy. Removing the option would otherwise result in the proxy not performing response revalidation at its cache entries associated with the servers in the CoAP group, even though that was what the client asked for.

Altering the option content in a group request would result in the proxy replying with 2.05 (Content) responses conveying the full resource representations from its cache entries, rather than with a single 2.03 (Valid) response. Instead, altering the option content in a 2.03 (Valid) or 2.05 (Content) response would result in the client wrongly believing that the already stored or the just received representation, respectively, is also the current one, as per the entity value of the tampered Group-ETag Option.

The security association between the client and the proxy SHOULD also provide message confidentiality. Otherwise, any further intermediaries between the two as well as any on-path passive adversaries would be able to simply access the option content, and thus learn the rate and pattern according to which the group resource in question changes over time, as inferable from the entity values read over time.

When the client protects the unicast request sent to the proxy using OSCORE (see [I-D.tiloca-core-oscore-capable-proxies]) and/or (D)TLS, both message integrity and message confidentiality are achieved in the leg between the client and the proxy.

The same considerations above about security associations apply when a chain of proxies is used (see Section 8), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

When caching of Group OSCORE secured responses is enabled at the proxy, the same as defined in Section 7 applies, with respect to cache entries and the way they are maintained.

10.5. HTTP-to-CoAP Proxies

Consistently with what is discussed in Section 10.1, an HTTP client has to authenticate to the HTTP-to-CoAP proxy, and they SHOULD rely on a full-fledged pairwise secure association. This can rely on a TLS [RFC8446] channel as also recommended in Section 12.1 of [RFC8613] for when OSCORE is used with HTTP, or on a pairwise OSCORE [RFC8613] Security Context between the client and the proxy as defined in [I-D.tiloca-core-oscore-capable-proxies].

[TODO

Revisit security considerations from [RFC8075]

]

11. IANA Considerations

This document has the following actions for IANA.

11.1. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry within the "CoRE Parameters" registry group.

Number	Name	Reference
TBD1	Multicast-Timeout	[[this document]]
TBD2	Response-Forwarding	[[this document]]
TBD3	Group-ETag	[[this document]]

11.2. CoAP Transport Information Registry

IANA is asked to add the following entries to the "CoAP Transport Information" registry defined in Section 14.5 of [I-D.ietf-core-observe-multicast-notifications].

Transport Protocol	Description	Value	Srv Addr	Req Info	Reference
UDP secured with DTLS	UDP with DTLS is used as per RFC8323	2	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]
TCP	TCP is used as per RFC8323	3	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]
TCP secured with TLS	TCP with TLS is used as per RFC8323	4	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]
WebSockets	WebSockets are used as per RFC8323	5	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]
WebSockets secured with TLS	WebSockets with TLS are used as per RFC8323	6	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]

11.3. Header Field Registrations

IANA is asked to enter the following HTTP header fields to the "Message Headers" registry.

Header Field Name	Protocol	Status	Reference
Multicast-Timeout	http	standard	[This document]
Response-Forwarding	http	standard	[This document]
Group-ETag	http	standard	[This document]

12. References

12.1. Normative References

[I-D.ietf-core-groupcomm-bis]

Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-06, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-06.txt>>.

[I-D.ietf-core-observe-multicast-notifications]

Tiloca, M., Höglund, R., Amsüss, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-ietf-core-observe-multicast-notifications-03, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-observe-multicast-notifications-03.txt>>.

[I-D.ietf-core-oscore-groupcomm]

Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-14, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-14.txt>>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <<https://www.rfc-editor.org/info/rfc2046>>.

- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

- [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>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

12.2. Informative References

- [I-D.amsuess-core-cachable-oscore]
Amsüss, C. and M. Tiloca, "Cacheable OSCORE", Work in Progress, Internet-Draft, draft-amsuess-core-cachable-oscore-04, 6 March 2022, <<https://www.ietf.org/archive/id/draft-amsuess-core-cachable-oscore-04.txt>>.
- [I-D.bormann-coap-misc]
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", Work in Progress, Internet-Draft, draft-bormann-coap-misc-27, 14 November 2014, <<https://www.ietf.org/archive/id/draft-bormann-coap-misc-27.txt>>.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-13, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-13.txt>>.

- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <<https://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-43.txt>>.
- [I-D.tiloca-core-oscore-capable-proxies]
Tiloca, M. and R. Höglund, "OSCORE-capable Proxies", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-capable-proxies-02, 7 March 2022, <<https://www.ietf.org/archive/id/draft-tiloca-core-oscore-capable-proxies-02.txt>>.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. V. D. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-11, 7 March 2022, <<https://www.ietf.org/archive/id/draft-tiloca-core-oscore-discovery-11.txt>>.
- [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>>.
- [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>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Appendix A. Examples with Reverse-Proxy

The examples in this section refer to the following actors.

- * One origin client C, with address C_ADDR and port number C_PORT.
- * One proxy P, with address P_ADDR and server port number P_PORT.

- * Two origin servers S1 and S2, where the server Sx has address Sx_ADDR and port number Sx_PORT.

The origin servers are members of a CoAP group with IP multicast address G_ADDR and port number G_PORT. Also, the origin servers are members of a same application group, and share the same resource /r.

The communication between C and P is based on CoAP over TCP, as per [RFC8323]. The group communication between P and the origin servers is based on CoAP over UDP and IP multicast, as per [I-D.ietf-core-groupcomm-bis].

Finally, 'bstr(X)' denotes a CBOR byte string where its value is the byte serialization of X.

A.1. Example 1

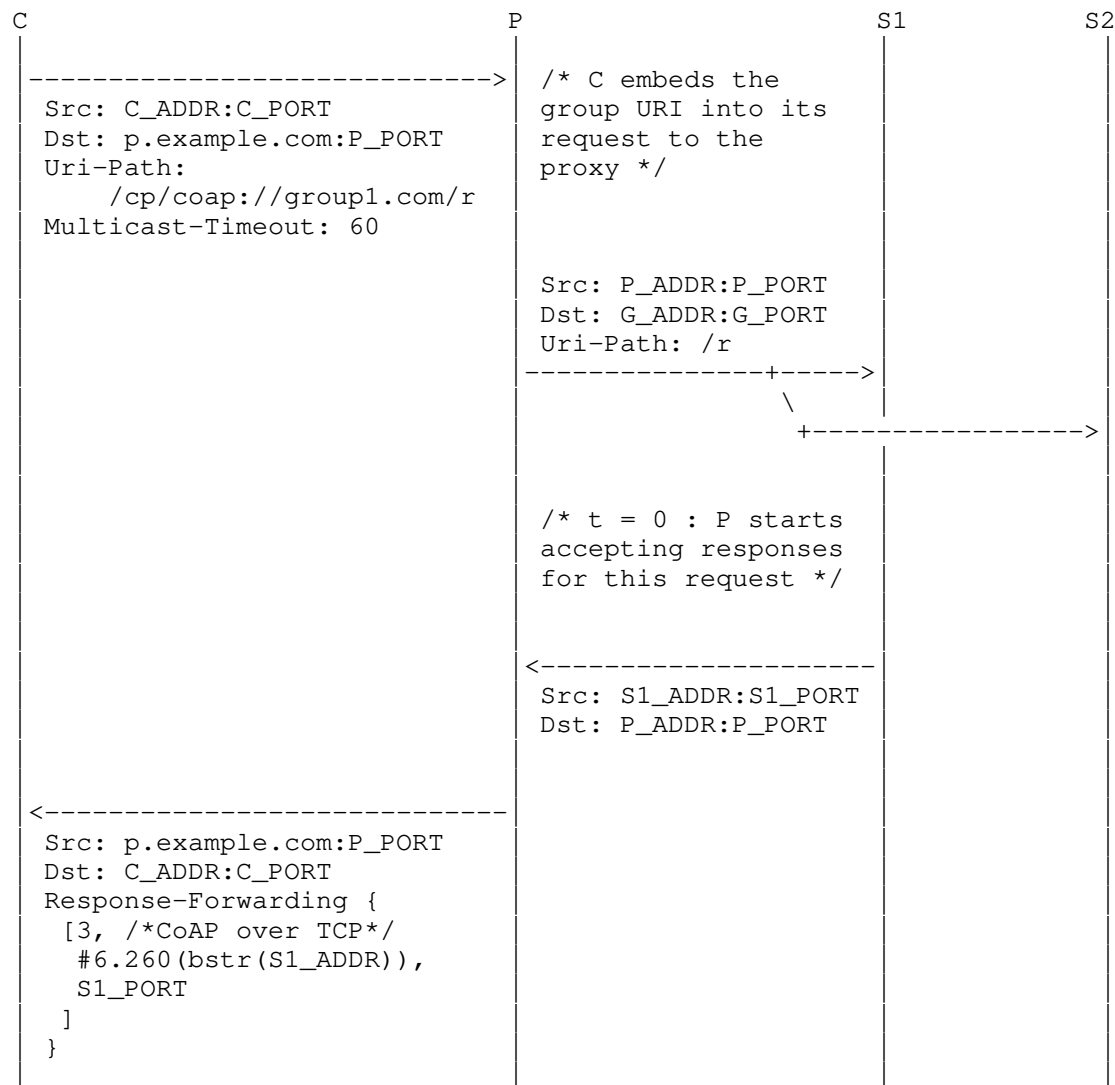
The example shown in Figure 5 considers a reverse-proxy P that provides access to both the whole group of servers {S1,S2} and also to each of those servers individually. The client C may not have a way to reach the servers directly (e.g., P is acting as a firewall). After the client C has received two responses to its group request sent via the proxy, it selects one server (S1) and requests another resource from it in unicast, again via the proxy.

In particular:

- * The client C encodes the group URI 'coap://group1.com/r' within the URI path of its request to P. This encoding follows the "default mapping" defined in Section 5.3 of [RFC8075] for HTTP-to-CoAP proxies, but now applied to a CoAP-to-CoAP proxy. The proxy P decodes the embedded group URI from the request.
- * The client's request URI path starts with '/cp', which is the resource on P that provides the CoAP proxy function. Since C in this example constructs the URI in its request including this resource '/cp', it is aware that it is requesting to a proxy.
- * Because the embedded group URI omits the CoAP port, P infers G_PORT to be the default port 5683 for the 'coap' scheme.
- * The hostname 'p.example.com' resolves to the proxy's unicast IPv6 address P_ADDR.
- * The hostname 'group1.com' resolves to the IPv6 multicast address G_ADDR. The proxy P performs this resolution upon receiving the request from C. P constructs the group request and sends it to the CoAP group at G_ADDR:G_PORT.

- * Typically S1_PORT and S2_PORT will be equal to G_PORT, but a server Sx is allowed to reply to the multicast request from another port number not equal to G_PORT. For this reason, the notation Sx_PORT is used.

Note that this type of reverse-proxy only requires one unicast IP address (P_ADDR) for the proxy, so it is well scalable to a large number of servers Sx. The type of reverse-proxy in the example in Appendix A.2 requires an additional IP address for each server Sx and also for each CoAP group that it supports.



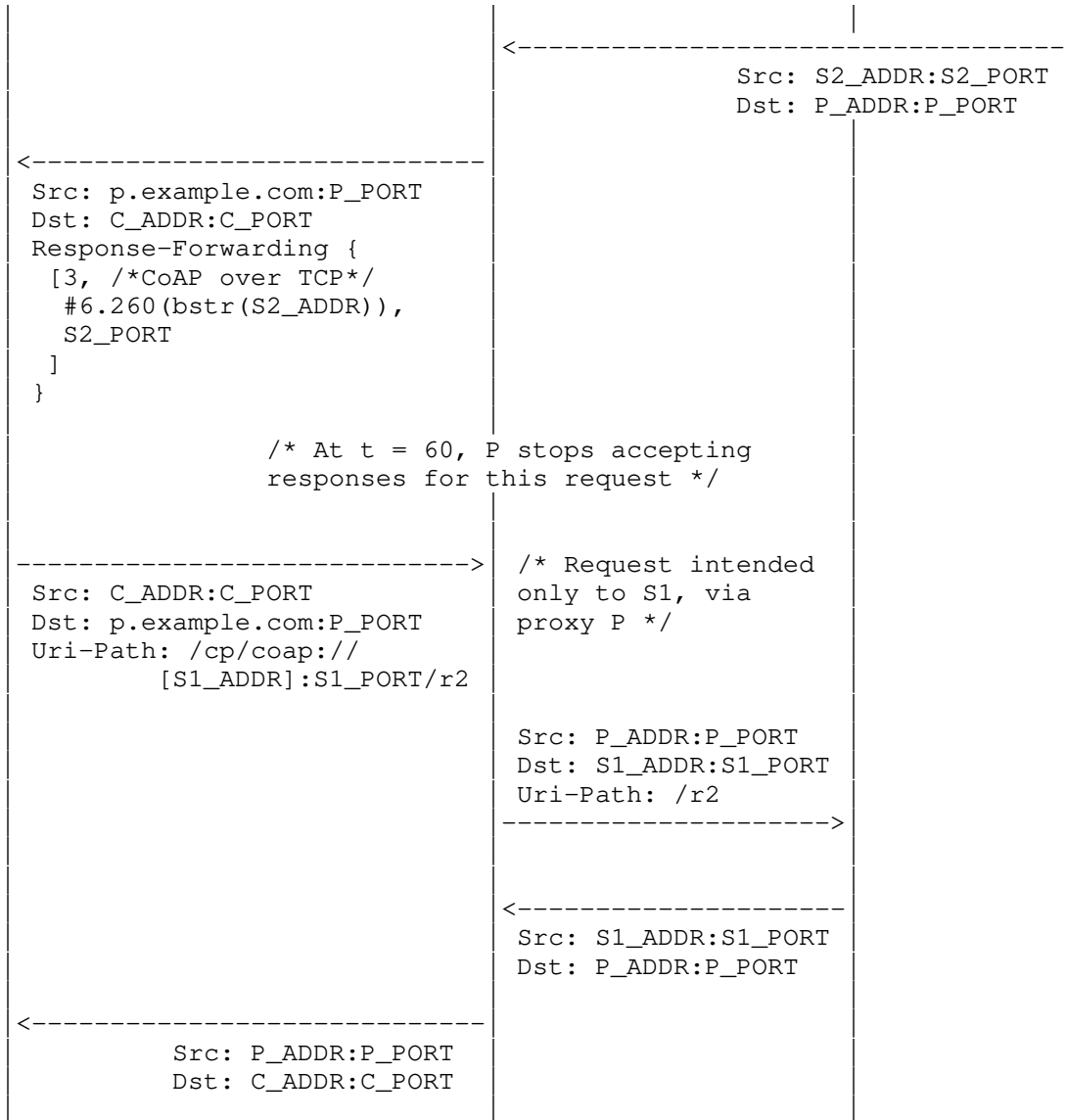


Figure 5: Workflow example with reverse-proxy that processes an embedded group URI in a client's request

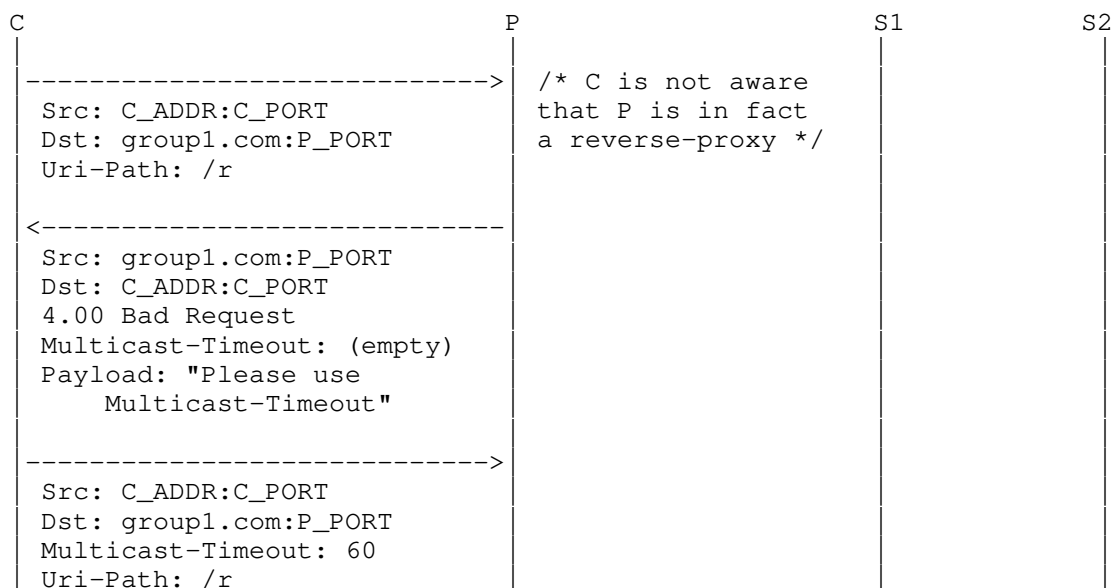
A.2. Example 2

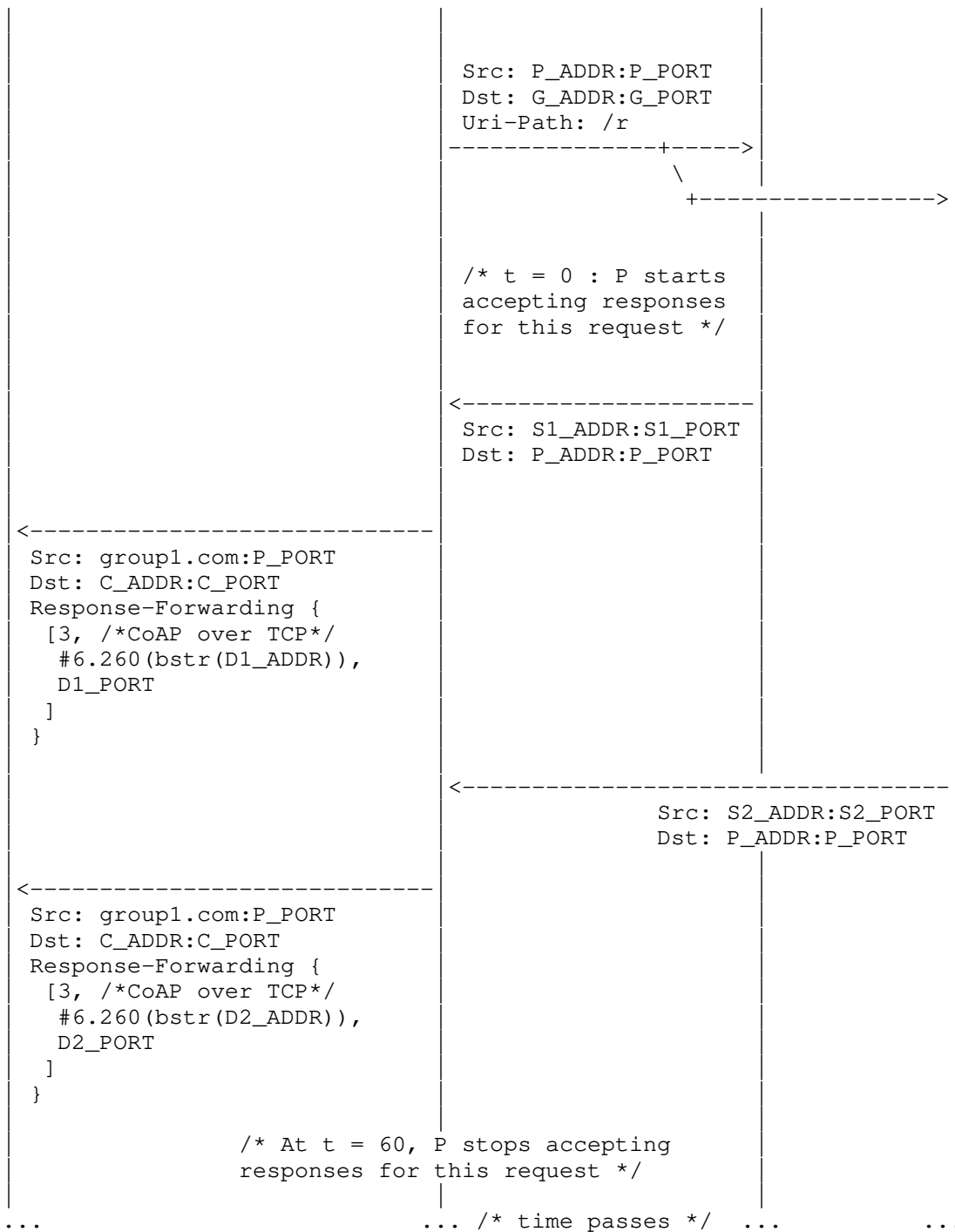
The example shown in Figure 6 considers a reverse-proxy that stands in for both the whole group of servers {S1,S2} and for each of those servers Sx. The client C may not have a way to reach the servers directly (e.g., P is acting as a firewall). After the client C has received two responses to its group request sent via the proxy, it selects one server (S1) and requests at a later time the same resource from it in unicast, again via the proxy.

In particular:

- * The hostname 'group1.com' resolves to the unicast address P_ADDR. The proxy forwards an incoming request to that address, for any resource i.e., URI path, towards the CoAP group at G_ADDR:G_PORT leaving the URI path unchanged.
- * The address Dx_ADDR and port number Dx_PORT are used by the proxy, which forwards an incoming request to that address towards the server at Sx_ADDR:Sx_PORT. The different Dx_ADDR are effectively 'proxy IP addresses' used to provide access to the servers.

Note that this type of reverse-proxy implementation requires the proxy to use (potentially) a large number of distinct IP addresses, hence it is not very scalable. Instead, the type of reverse-proxy shown in the example in Appendix A.1 uses only one IPv6 unicast address to provide access to all servers and all CoAP groups.





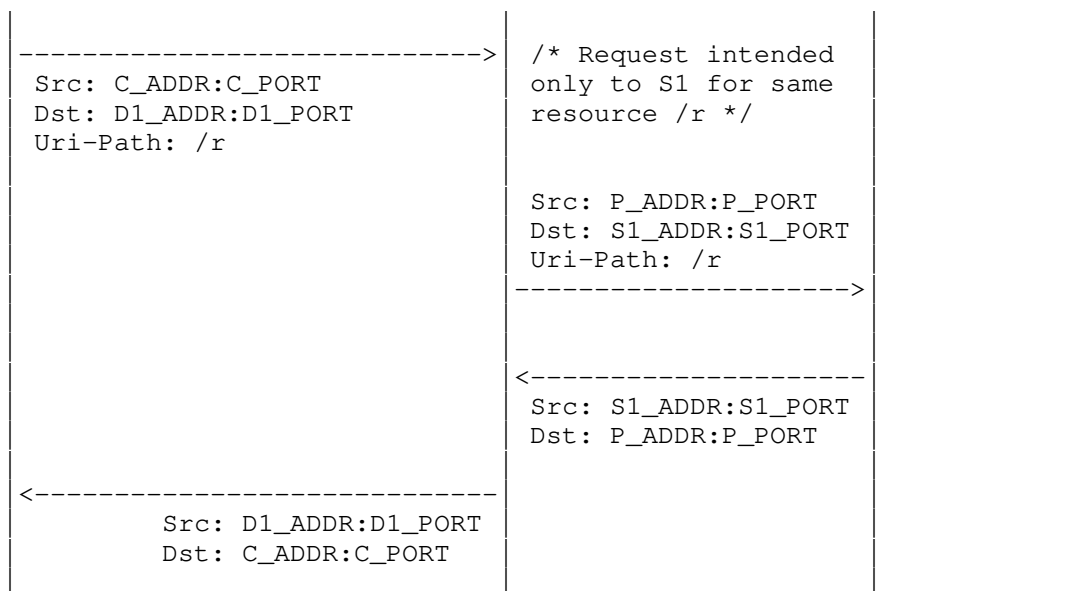


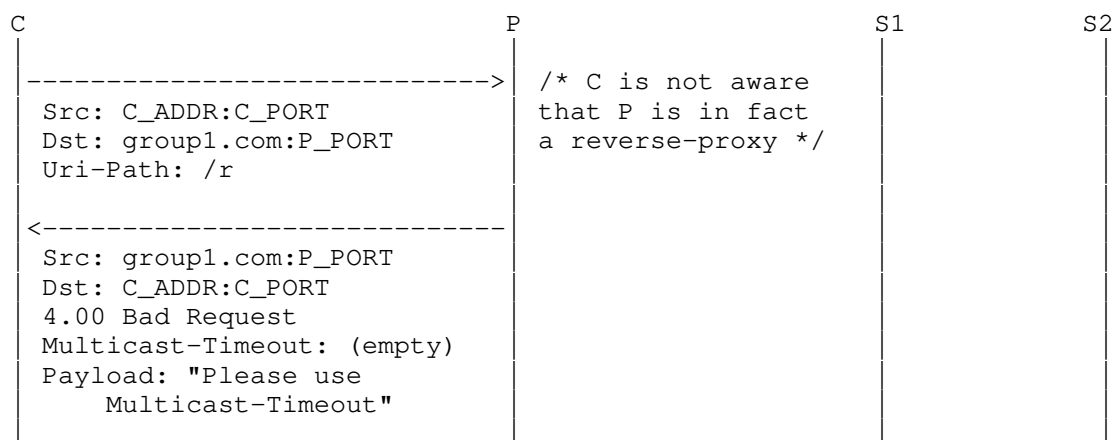
Figure 6: Workflow example with reverse-proxy standing in for both the whole group of servers and each individual server

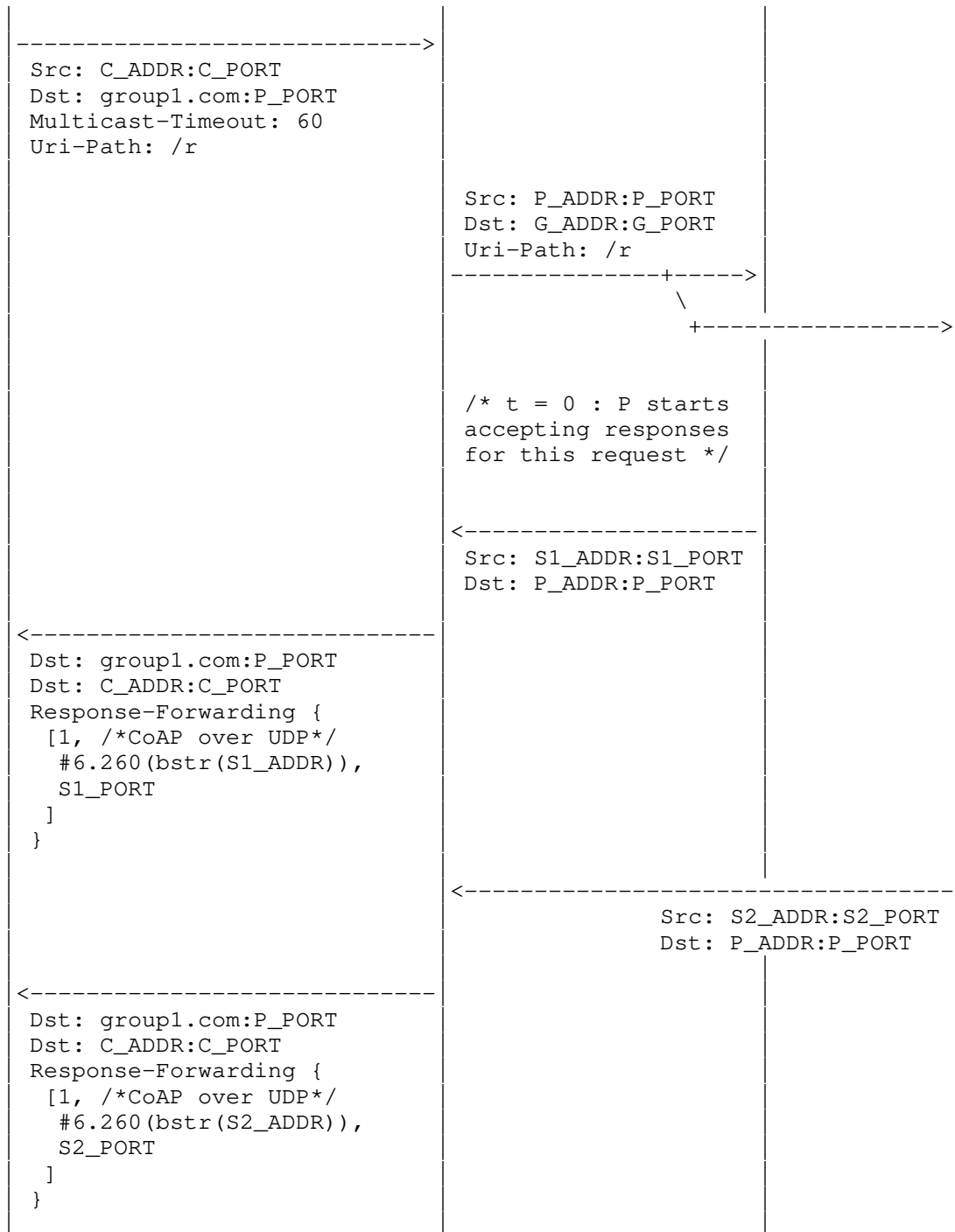
A.3. Example 3

The example shown in Figure 7 builds on the example in Appendix A.2.

However, it considers a reverse-proxy that stands in for only the whole group of servers, but not for each individual server S_x .

The final exchange between C and S1 occurs with CoAP over UDP.





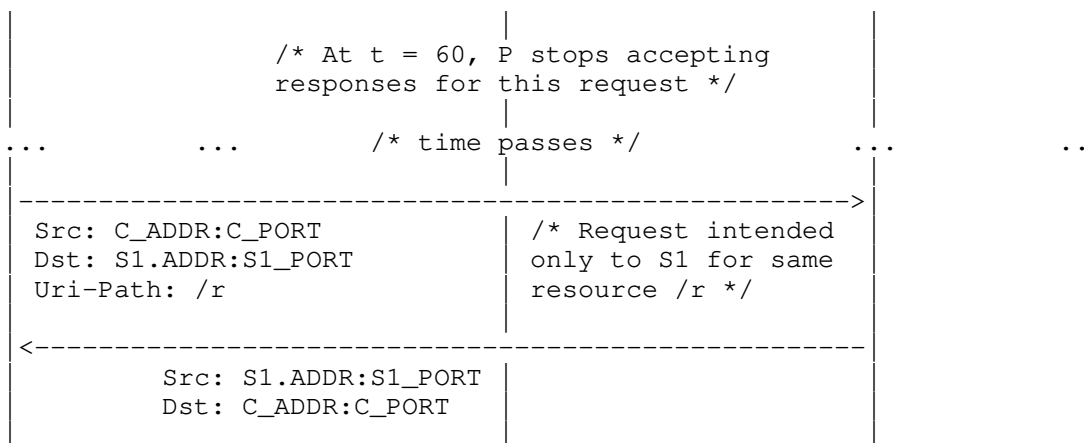


Figure 7: Workflow example with reverse-proxy standing in for only the whole group of servers, but not for each individual server

Acknowledgments

The authors sincerely thank Christian Amsuess, Jim Schaad and Goeran Selander for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

Esko Dijk
IoTconsultancy.nl

Utrecht
Email: esko.dijk@iotconsultancy.nl

CoRE Working Group
Internet-Draft
Updates: 7252, 7641 (if approved)
Intended status: Standards Track
Expires: May 6, 2021

M. Tiloca
R. Hoeglund
RISE AB
C. Amsuess
F. Palombini
Ericsson AB
November 02, 2020

Observe Notifications as CoAP Multicast Responses
draft-tiloca-core-observe-multicast-notifications-04

Abstract

The Constrained Application Protocol (CoAP) allows clients to "observe" resources at a server, and receive notifications as unicast responses upon changes of the resource state. In some use cases, such as based on publish-subscribe, it would be convenient for the server to send a single notification to all the clients observing a same target resource. This document defines how a CoAP server sends observe notifications as response messages over multicast, by synchronizing all the observers of a same resource on a same shared Token value. Besides, this document defines how Group OSCORE can be used to protect multicast notifications end-to-end from the CoAP server to the multiple observer clients.

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 May 6, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	5
2. Server-Side Requirements	5
2.1. Request	6
2.2. Informative Response	7
2.2.1. Encoding of Transport-Independent Message Information	8
2.2.2. Encoding of Transport-Specific Message Information	9
2.3. Notifications	10
2.4. Congestion Control	11
2.5. Cancellation	12
3. Client-Side Requirements	13
3.1. Request	13
3.2. Informative Response	13
3.3. Notifications	14
3.4. Cancellation	14
4. Example	15
5. Rough Counting of Clients in the Group Observation	17
5.1. Processing on the Client Side	17
5.2. Processing on the Server Side	18
5.2.1. Request for Feedback	19
5.2.2. Collection of Feedback	19
5.2.3. Processing of Feedback	20
6. Protection of Multicast Notifications with Group OSCORE	21
6.1. Signaling the OSCORE Group in the Informative Response	22
6.2. Server-Side Requirements	24
6.2.1. Registration	24
6.2.2. Informative Response	24
6.2.3. Notifications	25
6.2.4. Cancellation	25
6.3. Client-Side Requirements	26
6.3.1. Informative Response	26

6.3.2. Notifications	27
7. Example with Group OSCORE	27
8. Intermediaries	31
9. Intermediaries Together with End-to-End Security	33
9.1. The Listen-To-Multicast-Responses Option	34
9.2. Message Processing	35
10. Informative Response Parameters	36
11. Transport Protocol Identifiers	37
12. Security Considerations	38
12.1. Listen-To-Multicast-Responses Option	38
13. IANA Considerations	39
13.1. Media Type Registrations	39
13.2. CoAP Content-Formats Registry	40
13.3. Informative Response Parameters Registry	40
13.4. Transport Protocol Identifiers Registry	41
13.5. CoAP Option Numbers Registry	42
13.6. Expert Review Instructions	42
14. References	43
14.1. Normative References	43
14.2. Informative References	45
14.3. URIs	46
Appendix A. Different Sources for Group Observation Data	46
A.1. Topic Discovery in Publish-Subscribe Settings	46
A.2. Introspection at the Multicast Notification Sender	47
Appendix B. Pseudo-Code for Rough Counting of Clients	48
B.1. Client Side	48
B.2. Client Side - Optimized Version	49
B.3. Server Side	50
Appendix C. Example with a Proxy	52
Appendix D. Example with a Proxy and Group OSCORE	55
Acknowledgments	61
Authors' Addresses	61

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] has been extended with a number of mechanisms, including resource Observation [RFC7641]. This enables CoAP clients to register at a CoAP server as "observers" of a resource, and hence being automatically notified with an unsolicited response upon changes of the resource state.

CoAP supports group communication over IP multicast [I-D.ietf-core-groupcomm-bis]. This includes support for Observe registration requests over multicast, in order for clients to efficiently register as observers of a resource hosted at multiple servers.

However, in a number of use cases, using multicast messages for responses would also be desirable. That is, it would be useful that a server sends observe notifications for a same target resource to multiple observers as responses over IP multicast.

For instance, in CoAP publish-subscribe [I-D.ietf-core-coap-pubsub], multiple clients can subscribe to a topic, by observing the related resource hosted at the responsible broker. When a new value is published on that topic, it would be convenient for the broker to send a single multicast notification at once, to all the subscriber clients observing that topic.

A different use case concerns clients observing a same registration resource at the CoRE Resource Directory [I-D.ietf-core-resource-directory]. For example, multiple clients can benefit of observation for discovering (to-be-created) OSCORE groups [I-D.ietf-core-oscore-groupcomm], by retrieving from the Resource Directory updated links and descriptions to join them through the respective Group Manager [I-D.tiloca-core-oscore-discovery].

More in general, multicast notifications would be beneficial whenever several CoAP clients observe a same target resource at a CoAP server, and can be all notified at once by means of a single response message. However, CoAP does not currently define response messages over IP multicast. This specification fills this gap and provides the following twofold contribution.

First, it defines a method to deliver Observe notifications as CoAP responses over IP multicast. In the proposed method, the group of potential observers entrusts the server to manage the Token space for multicast notifications. By doing so, the server provides all the observers of a target resource with the same Token value to bind to their own observation. That Token value is then used in every multicast notification for the target resource. This is achieved by means of an informative unicast response sent by the server to each observer client.

Second, this specification defines how to use Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect multicast notifications end-to-end between the server and the observer clients. This is also achieved by means of the informative unicast response mentioned above, which additionally includes parameter values used by the server to protect every multicast notification for the target resource by using Group OSCORE. This provides a secure binding between each of such notifications and the observation of each of the clients.

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 terms and concepts described in CoAP [RFC7252], group communication for CoAP [I-D.ietf-core-groupcomm-bis], Observe [RFC7641], CBOR [I-D.ietf-cbor-7049bis], OSCORE [RFC8613], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This specification additionally defines the following terminology.

- o Traditional observation. A resource observation associated to a single observer client, as defined in [RFC7641].
- o Group observation. A resource observation associated to a group of clients. The server sends notifications for the group-observed resource over IP multicast to all the observer clients.
- o Phantom request. The CoAP request message that the server would have received to start or cancel a group observation on one of its resources. A phantom request is generated inside the server and does not hit the wire.
- o Informative response. A CoAP response message that the server sends to a given client via unicast, providing the client with information on a group observation.

2. Server-Side Requirements

The server can, at any time, start a group observation on one of its resources. Practically, the server may want to do that under the following circumstances.

- o In the absence of observations for the target resource, the server receives a registration request from a first client wishing to start a traditional observation on that resource.
- o When a certain amount of traditional observations has been established on the target resource, the server decides to make those clients part of a group observation on that resource.

The server maintains an observer counter for each group observation to a target resource, as a rough estimation of the observers actively taking part in the group observation.

The server initializes the counter to 0 when starting the group observation, and increments it after a new client starts taking part in that group observation. Also, the server should keep the counter up-to-date over time, for instance by using the method described in Section 5.

2.1. Request

Assuming it is reachable at the address SRV_ADDR and port number SRV_PORT, the server starts a group observation on one of its resources as defined below. The server intends to send multicast notifications for the target resource to the multicast IP address GRP_ADDR and port number GRP_PORT.

1. The server builds a phantom observation request, i.e. a GET request with an Observe option set to 0 (register).
2. The server selects an available value T, from the Token space of a CoAP endpoint used for messages having:
 - * As source address and port number, the IP multicast address GRP_ADDR and port number GRP_PORT.
 - * As destination address and port number, the server address SRV_ADDR and port number SRV_PORT, intended for accessing the target resource.

This Token space is under exclusive control of the server.

3. The server processes the phantom observation request above, without transmitting it on the wire. The request is addressed to the resource for which the server wants to start the group observation, as if sent by the group of observers, i.e. with GRP_ADDR as source address and GRP_PORT as source port.
4. Upon processing the self-generated phantom registration request, the server interprets it as an observe registration received from the group of potential observer clients. In particular, from then on, the server MUST use T as its own local Token value associated to that observation, with respect to the (previous hop towards the) clients.
5. The server does not immediately respond to the phantom observation request with a multicast notification sent on the

wire. The server stores the phantom observation request as is, throughout the lifetime of the group observation.

6. The server builds a CoAP response message INIT_NOTIF as initial multicast notification for the target resource, in response to the phantom observation request. This message is formatted as other multicast notifications (see Section 2.3) and MUST include the current representation of the target resource as payload. The server stores the message INIT_NOTIF and does not transmit it. The server considers this message as the latest multicast notification for the target resource, until it transmits a new multicast notification for that resource as a CoAP message on the wire. After that, the server deletes the message INIT_NOTIF.

2.2. Informative Response

After having started a group observation on a target resource, the server proceeds as follows.

For each traditional observation ongoing on the target resource, the server MAY cancel that observation. Then, the server considers the corresponding clients as now taking part in the group observation, for which it increases the corresponding observer counter accordingly.

The server sends to each of such clients an informative response message, encoded as a unicast response with response code 5.03 (Service Unavailable). As per [RFC7641], such a response does not include an Observe option. The response MUST be Confirmable and MUST NOT encode link-local addresses.

The Content-Format of the informative response is set to application/informative-response+cbor, as defined in Section 13.2. The payload of the informative response is a CBOR map which MUST include all the following parameters, whose CBOR labels are defined in Section 10.

- o 'ph_req', with value the byte serialization of the transport-independent information of the phantom observation request (see Section 2.1), encoded as a CBOR byte string. The value of the CBOR byte string is formatted as defined in Section 2.2.1.
- o 'last_notif', with value the byte serialization of the transport-independent information of the latest multicast notification for the target resource, encoded as a CBOR byte string. The value of the CBOR byte string is formatted as defined in Section 2.2.1.
- o 'tp_info', with value a CBOR array. This includes the transport-specific information required to correctly receive multicast

notifications bound to the phantom observation request. The CBOR array is formatted as defined in Section 2.2.2.

The CDDL notation [RFC8610] provided below describes the payload of the informative response.

```
informative_response_payload = {  
  1 => bstr, ; phantom request (transport-independent information)  
  2 => bstr, ; latest notification (transport-independent information)  
  3 => array ; transport-specific information  
}
```

Upon receiving a registration request to observe the target resource, the server does not create a corresponding individual observation for the requesting client. Instead, the server considers that client as now taking part in the group observation of the target resource, of which it increments the observer counter by 1. Then, the server replies to the client with the same informative response message defined above, which MUST be Confirmable.

Note that this also applies when, with no ongoing traditional observations on the target resource, the server receives a registration request from a first client and decides to start a group observation on the target resource.

2.2.1. Encoding of Transport-Independent Message Information

For both the parameters 'ph_req' and 'last_notif' in the informative response, the value of the byte string is the concatenation of the following components, in the order specified below.

When defining the value of each component, "CoAP message" refers to the phantom observation request for the 'ph_req' parameter, and to the corresponding latest multicast notification for the 'last_notif' parameter.

- o A single byte, with value the content of the Code field in the CoAP message.
- o The byte serialization of the complete sequence of CoAP options in the CoAP message.
- o If the CoAP message includes a non-zero length payload, the one-byte Payload Marker (0xff) followed by the payload.

2.2.2. Encoding of Transport-Specific Message Information

The CBOR array specified in the 'tp_info' parameter includes at least one element and is formatted as follows.

- o 'tp_id' : this element is a CBOR integer, which specifies the transport protocol used to transport the CoAP multicast notifications from the server. This element takes value from the "Transport Protocol Identifiers" sub-registry defined in Section 13.4 of this specification. This element MUST be present. The value of this element determines how many elements are required to follow in the CBOR array, as well as what information they convey, their encoding and their semantics.

This specification registers the integer value 1 ("UDP") to be used as value for the 'tp_id' element, when CoAP multicast notifications are transported over UDP as per [RFC7252] and [I-D.ietf-core-groupcomm-bis]. In such a case, the full encoding of the 'tp_info' CBOR array is as defined in Section 2.2.2.1.

Future specifications that consider CoAP multicast notifications transported over different transport protocols MUST:

- * Register an integer value to be used as value for the 'tp_id' array element, in the "Transport Protocol Identifiers" sub-registry defined in Section 13.4 of this specification.
- * Accordingly, define the elements of the 'tp_info' CBOR array following the 'tp_id' element, as to what information they convey, their encoding and their semantics.

2.2.2.1. UDP Transport-Specific Information

When CoAP multicast notifications are transported over UDP as per [RFC7252] and [I-D.ietf-core-groupcomm-bis], the server specifies the integer value 1 ("UDP") as value of the 'tp_id' element of the 'tp_info' CBOR array in the error informative response.

Then, following the 'tp_id' element, the rest of the 'tp_info' CBOR array is defined as follows.

- o 'token': this element is a CBOR byte string, with value the Token value of the phantom observation request generated by the server (see Section 2.1). Note that the same Token value is used for the multicast notifications bound to that phantom observation request (see Section 2.3). This element MUST be present.

- o 'srv_addr': this element is a CBOR byte string, with value the destination IP address of the phantom observation request. This parameter is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)". That is, the value of the CBOR byte string is the IP address SRV_ADDR of the server hosting the target resource, from where the server will send multicast notifications for the target resource. This element MUST be present.
- o 'srv_port': this element is a CBOR unsigned integer, with value the destination port number of the phantom observation request. That is, the specified value is the port number SRV_PORT, from where the server will send multicast notifications for the target resource. This element MUST be present.
- o 'cli_addr': this element is a CBOR byte string, with value the source IP address of the phantom observation request. This parameter is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)". That is, the value of the CBOR byte string is the IP multicast address GRP_ADDR, where the server will send multicast notifications for the target resource. This element MUST be present.
- o 'cli_port': this element is a CBOR unsigned integer, with value the source port number of the phantom observation request. That is, the specified value is the port number GRP_PORT, where the server will send multicast notifications for the target resource. This element is OPTIONAL. If not included, the default port number 5683 is assumed.

The CDDL notation [RFC8610] provided below describes the full 'tp_info' CBOR array using the format above.

```
tp_info = [
  tp_id : 1,          ; UDP as transport protocol
  token : bstr,       ; Token of phantom request and multicast notifications
  srv_addr : #6.260(bstr), ; Src. address of multicast notifications
  srv_port : uint,     ; Src. port of multicast notifications
  cli_addr : #6.260(bstr), ; Dst. address of multicast notifications
  ? cli_port : uint    ; Dst. port of multicast notifications
]
```

2.3. Notifications

Upon a change in the status of the target resource under group observation, the server sends a multicast notification, intended to all the clients taking part in the group observation of that

resource. In particular, each of such multicast notifications is formatted as follows.

- o It MUST be Non-confirmable.
- o It MUST include an Observe option, as per [RFC7641].
- o It MUST have the same Token value T of the phantom registration request that started the group observation. This Token value is specified in the 'token' element of the 'tp_info' parameter, in the informative response message sent to all the observer clients.

That is, every multicast notification for a target resource is not bound to the observation requests from the different clients, but rather to the phantom registration request associated to the whole set of clients taking part in the group observation of that resource.

- o It MUST be sent from the same IP address SRV_ADDR and port number SRV_PORT where: i) the original Observe registration requests are sent to by the clients; and ii) the corresponding informative responses are sent from by the server (see Section 2.2). These are indicated to the observer clients as value of the 'srv_addr' and 'srv_port' elements of the 'tp_info' parameter, in the informative response message (see Section 2.2.2.1). That is, redirection MUST NOT be used.
- o It MUST be sent to the IP multicast address GRP_ADDR and port number GRP_PORT. These are indicated to the observer clients as value of the 'cli_addr' and 'cli_port' elements of the 'tp_info' parameter, in the informative response message (see Section 2.2.2.1).

For each target resource with an active group observation, the server MUST store the latest multicast notification.

2.4. Congestion Control

In order to not cause congestion, the server should conservatively control the sending of multicast notifications. In particular:

- o The multicast notifications MUST be Non-confirmable.
- o In constrained environments such as low-power, lossy networks (LLNs), the server should only support multicast notifications for resources that are small. Following related guidelines from Section 2.2.4 of [I-D.ietf-core-groupcomm-bis], this can consist, for example, in having the payload of multicast notifications as

limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size, so that it fits into a single link-layer frame in case IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) (see Section 4 of [RFC4944]) is used.

- o The server SHOULD provide multicast notifications with the smallest possible IP multicast scope that fulfills the application needs. For example, following related guidelines from Section 2.2.4 of [I-D.ietf-core-groupcomm-bis], site-local scope is always preferred over global scope IP multicast, if this fulfills the application needs. Similarly, realm-local scope is always preferred over site-local scope, if this fulfills the application needs.
- o Following related guidelines from Section 4.5.1 of [RFC7641], the server SHOULD NOT send more than one multicast notification every 3 seconds, and SHOULD use an even less aggressive rate when possible (see also Section 3.1.2 of [RFC8085]). The transmission rate of multicast notifications should also take into account the avoidance of a possible "broadcast storm" problem [MOBICOM99]. This prevents a following, considerable increase of the channel load, whose origin would be likely attributed to a router rather than the server.

2.5. Cancellation

At any point in time, the server may want to cancel a group observation of a target resource. For instance, the server may realize that no clients or not enough clients are interested in taking part in the group observation anymore. A possible approach that the server can use to assess this is defined in Section 5.

In order to cancel the group observation, the server sends to itself a phantom cancellation request, i.e. a GET request with an Observe option set to 1 (deregister), without transmitting it on the wire. As per Section 3.6 of [RFC7641], all other options MUST be identical to those in the phantom registration request, except for the set of ETag Options. This request has the same Token value T of the phantom registration request, and is addressed to the resource for which the server wants to end the group observation, as if sent by the group of observers, i.e. with the multicast IP address GRP_ADDR as source address and the port number GRP_PORT as source port.

After that, the server sends a multicast response with response code 5.03 (Service Unavailable), signaling that the group observation has been terminated. The response has no payload, and is sent to the same multicast IP address GRP_ADDR and port number GRP_PORT used to send the multicast notifications related to the target resource. As

per [RFC7641], this response does not include an Observe option. Finally, the server releases the resources allocated for the group observation, and especially frees up the Token value T used at its CoAP endpoint.

3. Client-Side Requirements

3.1. Request

A client sends an observation request to the server as described in [RFC7641], i.e. a GET request with an Observe option set to 0 (register). The request MUST NOT encode link-local addresses. If the server is not configured to accept registrations on that target resource with a group observation, this would still result in a positive notification response to the client as described in [RFC7641].

3.2. Informative Response

Upon receiving the informative response defined in Section 2.2, the client proceeds as follows.

1. The client configures an observation of the target resource. To this end, it relies on a CoAP endpoint used for messages having:
 - * As source address and port number, the server address SRV_ADDR and port number SRV_PORT intended for accessing the target resource. These are specified as value of the 'srv_addr' and 'srv_port' elements of the 'tp_info' parameter, in the informative response (see Section 2.2.2.1).
 - * As destination address and port number, the IP multicast address GRP_ADDR and port number GRP_PORT. These are specified as value of the 'cli_addr' and 'cli_port' elements of the 'tp_info' parameter, in the informative response (see Section 2.2.2.1). If the 'cli_port' element is omitted in the 'tp_info' parameter, the client MUST assume the default port number 5683 as GRP_PORT.
2. The client rebuilds the phantom registration request, by using:
 - * The transport-independent information, specified in the 'ph_req' parameter of the informative response.
 - * The Token value T, specified in the 'token' element of the 'tp_info' parameter of the informative response.

3. The client stores the phantom registration request, as associated to the observation of the target resource. In particular, the client **MUST** use the Token value T of this phantom registration request as its own local Token value associated to that group observation, with respect to the server. The particular way to achieve this is implementation specific.
4. The client rebuilds the latest multicast notification, by using:
 - * The transport-independent information, specified in the 'last_notif' parameter of the informative response.
 - * The Token value T, specified in the 'token' element of the 'tp_info' parameter of the informative response.
5. Then, the client processes the latest multicast notification as defined in Section 3.2 of [RFC7641]. In particular, the value of the Observe option is used as initial baseline for notification reordering in this group observation.
6. If a traditional observation to the target resource is ongoing, the client **MAY** silently cancel it without notifying the server.

If any of the expected fields in the informative response are not present or malformed, the client **MAY** try sending a new registration request to the server (see Section 3.1). Otherwise, the client **SHOULD** explicitly withdraw from the group observation.

Appendix A describes possible alternative ways for clients to retrieve the phantom registration request and other information related to a group observation.

3.3. Notifications

After having successfully processed the informative response as defined in Section 3.2, the client will receive, accept and process multicast notifications about the state of the target resource from the server, as responses to the phantom registration request and with Token value T.

The client relies on the value of the Observe option for notification reordering, as defined in Section 3.4 of [RFC7641].

3.4. Cancellation

At a certain point in time, a client may become not interested in receiving further multicast notifications about a target resource. When this happens, the client can simply "forget" about being part of

the group observation for that target resource, as per Section 3.6 of [RFC7641].

When, later on, the server sends the next multicast notification, the client will not recognize the Token value T in the message. Since the multicast notification is Non-confirmable, it is OPTIONAL for the client to reject the multicast notification with a Reset message, as defined in Section 3.5 of [RFC7641].

In case the server has cancelled a group observation as defined in Section 2.5, the client simply forgets about the group observation and frees up the used Token value T for that endpoint, upon receiving the multicast error response defined in Section 2.5.

4. Example

The following example refers to two clients C_1 and C_2 that register to observe a resource /r at a Server S, which has address SRV_ADDR and listens to the port number SRV_PORT. Before the following exchanges occur, no clients are observing the resource /r, which has value "1234".

The server S sends multicast notifications to the IP multicast address GRP_ADDR and port number GRP_PORT, and starts the group observation upon receiving a registration request from a first client that wishes to start a traditional observation on the resource /r.

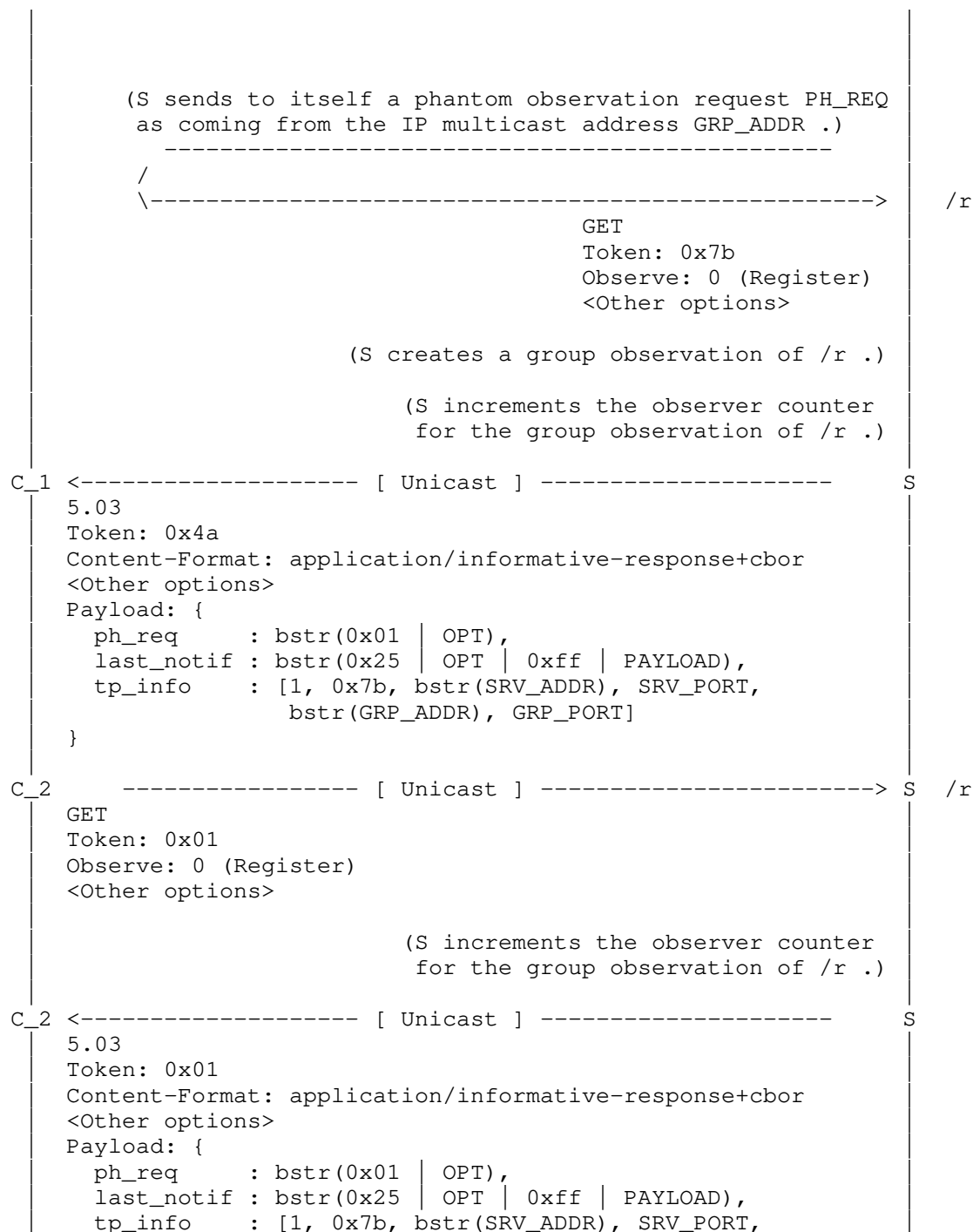
The following notation is used for the payload of the informative responses:

- o 'bstr(X)' denotes a CBOR byte string with value the byte serialization of X, with '|' denoting byte concatenation.
- o 'OPT' denotes a sequence of CoAP options. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.
- o 'PAYLOAD' denotes a CoAP payload. This refers to the latest multicast notification encoded by the 'last_notif' parameter.

```

C_1      ----- [ Unicast ] -----> S  /r
| GET
| Token: 0x4a
| Observe: 0 (Register)
| <Other options>
|
|                                     (S allocates the available Token value 0x7b .)

```



```

        bstr(GRP_ADDR), GRP_PORT]
    }

    (The value of the resource /r changes to "5678".)

C_1
+ <----- [ Multicast ] ----- S
C_2   (Destination address/port: GRP_ADDR/GRP_PORT)
    2.05
    Token: 0x7b
    Observe: 11
    Content-Format: application/cbor
    <Other options>
    Payload: : "5678"

```

5. Rough Counting of Clients in the Group Observation

To allow the server to keep an estimate of interested clients without creating undue traffic on the network, a new CoAP option is introduced, which SHOULD be supported by clients that listen to multicast responses.

The option is called Multicast-Response-Feedback-Divider. As summarized in Figure 1, the option is not critical but proxy-unsafe, and integer valued.

No.	C	U	N	R	Name	Format	Len.	Default
TBD		x			Multicast-Response-Feedback-Divider	uint	0-1	(none)

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

Figure 1: Multicast-Response-Feedback-Divider

The Multicast-Response-Feedback-Divider option is of class E for OSCORE [RFC8613][I-D.ietf-core-oscore-groupcomm].

5.1. Processing on the Client Side

Upon receiving a response with a Multicast-Response-Feedback-Divider option, a client SHOULD acknowledge its interest in continuing receiving multicast notifications for the target resource, as described below.

The client picks an integer random number I , from 0 inclusive to the number $Z = (2 ** Q)$ exclusive, where Q is the value specified in the option and $**$ is the exponentiation operator. If I is different than 0, the client takes no further action. Otherwise, the client should wait a random fraction of the Leisure time (see Section 8.2 of [RFC7252]), and then registers a regular unicast observation on the same target resource.

To this end, the client essentially follows the steps that got it originally subscribed to group notifications for the target resource. In particular, the client sends an observation request to the server, i.e. a GET request with an Observe option set to 0 (register). The request MUST be addressed to the same target resource, and MUST have the same destination IP address and port number used for the original registration request, regardless the source IP address and port number of the received multicast notification.

Since the observation registration is only done for its side effect of showing as an attempted observation at the server, the client MUST send the unicast request in a non confirmable way, and with the maximum No-Response setting [RFC7967]. In the request, the client MUST include a Multicast-Response-Feedback-Divider option, whose value MUST be empty (Option Length = 0). The client does not need to wait for responses, and can keep processing further notifications on the same token.

The client MUST ignore the Multicast-Response-Feedback-Divider option, if the multicast notification is retrieved from the 'last_notif' parameter of an informative response (see Section 2.2). A client includes the Multicast-Response-Feedback-Divider option only in a re-registration request triggered by the server as described above, and MUST NOT include it in any other request.

As the Multicast-Response-Feedback-Divider option is unsafe to forward, a proxy needs to answer it on its own, and is later counted as a single client.

Appendix B.1 provides a description in pseudo-code of the operations above performed by the client.

5.2. Processing on the Server Side

In order to avoid needless use of network resources, a server SHOULD keep a rough, updated count of the number of clients taking part in the group observation of a target resource. To this end, the server updates the value COUNT of the associated observer counter (see Section 2), for instance by using the method described below.

5.2.1. Request for Feedback

When it wants to obtain a new estimated count, the server considers a number M of confirmations it would like to receive from the clients. It is up to applications to define policies about how the server determines and possibly adjusts the value of M .

Then, the server computes the value $Q = \max(L, 0)$, where:

- o L is computed as $L = \text{ceil}(\log_2(N / M))$.
- o N is the current value of the observer counter, possibly rounded up to 1, i.e. $N = \max(\text{COUNT}, 1)$.

Finally, the server sets Q as the value of the Multicast-Response-Feedback-Divider option, which is sent within a successful multicast notification.

If several multicast notifications are sent in a burst fashion, it is RECOMMENDED for the server to include the Multicast-Response-Feedback-Divider option only in the first one of those notifications.

5.2.2. Collection of Feedback

The server collects unicast observation requests from the clients, for an amount of time of `MAX_CONFIRMATION_WAIT` seconds. During this time, the server regularly increments the observer counter when adding a new client to the group observation (see Section 2.2).

It is up to applications to define the value of `MAX_CONFIRMATION_WAIT`, which has to take into account the transmission time of the multicast notification and of unicast observation requests, as well as the leisure time of the clients, which may be hard to know or estimate for the server.

If this information is not known to the server, it is recommended to define `MAX_CONFIRMATION_WAIT` as follows.

`MAX_CONFIRMATION_WAIT = MAX_RTT + MAX_CLIENT_REQUEST_DELAY`

where `MAX_RTT` is as defined in Section 4.8.2 of [RFC7252] and has default value 202 seconds, while `MAX_CLIENT_REQUEST_DELAY` is equivalent to `MAX_SERVER_RESPONSE_DELAY` defined in Section 2.3.1 of [I-D.ietf-core-groupcomm-bis] and has default value 250 seconds. In the absence of more specific information, the server can thus consider a conservative `MAX_CONFIRMATION_WAIT` of 452 seconds.

If more information is available in deployments, a much shorter `MAX_CONFIRMATION_WAIT` can be set. This can be based on a realistic round trip time (replacing `MAX_RTT`) and on the largest leisure time configured on the clients (replacing `MAX_CLIENT_REQUEST_DELAY`), e.g. `DEFAULT_LEISURE = 5` seconds, thus shortening `MAX_CONFIRMATION_WAIT` to a few seconds.

5.2.3. Processing of Feedback

Once `MAX_CONFIRMATION_WAIT` seconds have passed, the server counts the `R` confirmations arrived as unicast observation requests to the target resource, since the multicast notification with the Multicast-Response-Feedback-Divider option has been sent. In particular, the server considers a unicast observation request as a confirmation from a client only if it includes a Multicast-Response-Feedback-Divider option with an empty value (Option Length = 0).

Then, the server computes a feedback indicator as $F = R * (2 ** Q)$, where `***` is the exponentiation operator. According to what defined by application policies, the server determines the next time when to ask clients for their confirmation, e.g. after a certain number of multicast notifications has been sent. For example, the decision can be influenced by the reception of no confirmations from the clients, i.e. $R = 0$, or by the value of the ratios (F/N) and (N/F) .

Finally, the server computes a new estimated count of the observers. To this end the server first consider `COUNT'` as the current value of the observer counter at this point in time. Note that `COUNT'` may be greater than the value `COUNT` used at the beginning of this process, if the server has incremented the observer counter upon adding new clients to the group observation (see Section 2.2).

In particular, the server computes the new estimated count value as $COUNT' + ((E - N) / D)$, where $D > 0$ is an integer value used as dampener. This step has to be performed atomically. That is, until this step is completed, the server MUST hold the processing of an observation request for the same target resource from a new client. Finally, the server considers the result as the current observer counter, and assesses it for possibly cancelling the group observation (see Section 2.5).

This estimate is skewed by packet loss, but it gives the server a sufficiently good estimation for further counts and for deciding when to cancel the group observation. It is up to applications to define policies about how the server takes the newly updated estimate into account and determines whether to cancel the group observation.

As an example, if the server currently estimates that $N = \text{COUNT} = 32$ observers are active and considers a constant $M = 8$, it sends out a notification with Multicast-Response-Feedback-Divider: 2. Then, out of 18 actually active clients, 5 send a re-registration request based on their random draw, of which one request gets lost, thus leaving 4 re-registration requests received by the server. Also, no new clients have been added to the group observation during this time, i.e. COUNT' is equal to COUNT . As a consequence, assuming that a dampener value $D = 1$ is used, the server computes the new estimated count value as $32 + (16 - 32) = 16$, and keeps the group observation active.

To produce a most accurate updated counter, a server can include a Multicast-Response-Feedback-Divider option with value $Q = 0$ in its multicast notifications, as if M is equal to N . This will trigger all the active clients to state their interest in continuing receiving notifications for the target resource. Thus, the amount R of arrived confirmations is affected only by possible packet loss.

Appendix B.3 provides a description in pseudo-code of the operations above performed by the server, including example behaviors for scheduling the next count update and deciding whether to cancel the group observation.

6. Protection of Multicast Notifications with Group OSCORE

A server can protect multicast notifications by using Group OSCORE [I-D.ietf-core-oscore-groupcomm], thus ensuring they are protected end-to-end with the observer clients. This requires that both the server and the clients interested in receiving multicast notifications from that server are members of the same OSCORE group.

In some settings, the OSCORE group to refer to can be pre-configured on the clients and the server. In such a case, a server which is aware of such pre-configuration can simply assume a client to be already member of the correct OSCORE group.

In any other case, the server MAY communicate to clients what OSCORE group they are required to join, by providing additional guidance in the informative response as described in Section 6.1. Note that clients can already be members of the right OSCORE group, in case they have previously joined it to securely communicate with the same and/or other servers to access their resources.

Both the clients and the server MAY join the OSCORE group by using the approach described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz]. Further details

on how to discover the OSCORE group and join it are out of the scope of this specification.

If multicast notifications are protected using Group OSCORE, the original registration requests and related unicast (notification) responses MUST also be secured, including and especially the informative responses from the server.

To this end, alternative security protocols than Group OSCORE, such as OSCORE [RFC8613] and/or DTLS [RFC6347][I-D.ietf-tls-dtls13], can be used to protect other exchanges via unicast between the server and each client, including the original client registration (see Section 3).

6.1. Signaling the OSCORE Group in the Informative Response

This section describes a mechanism for the server to communicate to the client what OSCORE group to join in order to decrypt and verify the multicast notifications protected with group OSCORE. The client MAY use the information provided by the server to start the ACE joining procedure described in [I-D.ietf-ace-key-groupcomm-oscore]. This mechanism is OPTIONAL to support for the client and server.

Additionally to what defined in Section 2, the CBOR map in the informative response payload contains the following fields, whose CBOR labels are defined in Section 10.

- o 'join_uri', with value the URI for joining the OSCORE group at the respective Group Manager, encoded as a CBOR text string. If the procedure described in [I-D.ietf-ace-key-groupcomm-oscore] is used for joining, this field specifically indicates the URI of the group-membership resource at the Group Manager.
- o 'sec_gp', with value the name of the OSCORE group, encoded as a CBOR text string.
- o Optionally, 'as_uri', with value the URI of the Authorization Server associated to the Group Manager for the OSCORE group, encoded as a CBOR text string.
- o Optionally, 'cs_alg', with value the COSE algorithm [I-D.ietf-cose-rfc8152bis-algs] used to countersign messages in the OSCORE group, encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Algorithms" registry [COSE.Algorithms].
- o Optionally, 'cs_alg_crv', with value the elliptic curve (if applicable) for the COSE algorithm [I-D.ietf-cose-rfc8152bis-algs]

used to countersign messages in the OSCORE group, encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Elliptic Curve" registry [COSE.Elliptic.Curves].

- o Optionally, 'cs_key_kty', with value the COSE key type [I-D.ietf-cose-rfc8152bis-struct] of countersignature keys used to countersign messages in the OSCORE group, encoded as a CBOR text string or an integer. The value is taken from the 'Value' column of the "COSE Key Types" registry [COSE.Key.Types].
- o Optionally, 'cs_key_crv', with value the elliptic curve (if applicable) of countersignature keys used to countersign messages in the OSCORE group, encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Elliptic Curve" registry [COSE.Elliptic.Curves].
- o Optionally, 'cs_kenc', with value the encoding of the public keys used in the OSCORE group, encoded as a CBOR integer. The value is taken from the 'Confirmation Key' column of the "CWT Confirmation Method" registry defined in [RFC8747]. Future specifications may define additional values for this parameter.
- o Optionally, 'alg', with value the COSE AEAD algorithm [I-D.ietf-cose-rfc8152bis-algs], encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Algorithms" registry [COSE.Algorithms].
- o Optionally, 'hkdf', with value the COSE HKDF algorithm [I-D.ietf-cose-rfc8152bis-algs], encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Algorithms" registry [COSE.Algorithms].

The values of 'cs_alg', 'cs_alg_crv', 'cs_key_kty', 'cs_key_crv' and 'cs_key_kenc' provide an early knowledge of the format and encoding of public keys used in the OSCORE group. Thus, the client does not need to ask the Group Manager for this information as a preliminary step before the (ACE) join process, or to perform a trial-and-error exchange with the Group Manager upon joining the group. Hence, the client is able to provide the Group Manager with its own public key in the correct expected format and encoding, at the very first step of the (ACE) join process.

The values of 'cs_alg', 'alg' and 'hkdf' provide an early knowledge of the algorithms used in the OSCORE group. Thus, the client is able to decide whether to actually proceed with the (ACE) join process, depending on its support for the indicated algorithms.

As mentioned above, since this mechanism is OPTIONAL, all the fields are OPTIONAL in the informative response. However, the 'join_uri' and 'sec_gp' fields MUST be present if the mechanism is implemented and used. If any of the fields are present without the 'join_uri' and 'sec_gp' fields present, the client MUST ignore these fields, since they would not be sufficient to start the (ACE) join procedure. When this happens, the client MAY try sending a new registration request to the server (see Section 3.1). Otherwise, the client SHOULD explicitly withdraw from the group observation.

6.2. Server-Side Requirements

When using Group OSCORE to protect multicast notifications, the server performs the operations described in Section 2, with the following differences.

6.2.1. Registration

The phantom registration request MUST be secured, by using Group OSCORE. In particular, the group mode of Group OSCORE defined in Section 8 of [I-D.ietf-core-oscore-groupcomm] MUST be used.

The server protects the phantom registration request as defined in Section 8.1 of [I-D.ietf-core-oscore-groupcomm], as if it was the actual sender, i.e. by using its Sender Context. As a consequence, the server consumes the current value of its Sender Sequence Number SN in the OSCORE group, and hence updates it to $SN^* = (SN + 1)$. Consistently, the OSCORE option in the phantom registration request includes:

- o As 'kid', the Sender ID of the server in the OSCORE group.
- o As 'piv', the previously consumed Sender Sequence Number value SN of the server in the OSCORE group, i.e. $(SN^* - 1)$.

6.2.2. Informative Response

The value of the CBOR byte string in the 'ph_req' parameter encodes the phantom observation request as a message protected with Group OSCORE (see Section 6.2.1). As a consequence: the specified Code is always 0.05 (FETCH); the sequence of CoAP options will be limited to the outer, non encrypted options; a payload is always present, as the authenticated ciphertext followed by the counter signature.

Similarly, the value of the CBOR byte string in the 'last_notif' parameter encodes the latest multicast notification as a message protected with Group OSCORE (see Section 6.2.3). This applies also

to the initial multicast notification INIT_NOTIF built in step 6 of Section 2.1.

Optionally, the informative response includes information on the OSCORE group to join, as additional parameters (see Section 6.1).

6.2.3. Notifications

The server MUST protect every multicast notification for the target resource with Group OSCORE. In particular, the group mode of Group OSCORE defined in Section 8 of [I-D.ietf-core-oscore-groupcomm] MUST be used.

The process described in Section 8.3 of [I-D.ietf-core-oscore-groupcomm] applies, with the following additions when building the two OSCORE 'external_aad' to encrypt and countersign the multicast notification (see Sections 4.3.1 and 4.3.2 of [I-D.ietf-core-oscore-groupcomm]).

- o The 'request_kid' is the 'kid' value in the OSCORE option of the phantom registration request, i.e. the Sender ID of the server.
- o The 'request_piv' is the 'piv' value in the OSCORE option of the phantom registration request, i.e. the consumed Sender Sequence Number SN of the server.
- o The 'request_kid_context' is the 'kid context' value in the OSCORE option of the phantom registration request, i.e. the Group Identifier value (Gid) of the OSCORE group used as ID Context.

Note that these same values are used to protect each and every multicast notification sent for the target resource under this group observation.

6.2.4. Cancellation

When cancelling a group observation (see Section 2.5), the phantom cancellation request MUST be secured, by using Group OSCORE. In particular, the group mode of Group OSCORE defined in Section 8 of [I-D.ietf-core-oscore-groupcomm] MUST be used.

Like defined in Section 6.2.1 for the phantom registration request, the server protects the phantom cancellation request as per Section 8.1 of [I-D.ietf-core-oscore-groupcomm], by using its Sender Context and consuming its current Sender Sequence number in the OSCORE group, from its Sender Context. The following, corresponding multicast error response defined in Section 2.5 is also protected

with Group OSCORE, as per Section 8.3 of [I-D.ietf-core-oscore-groupcomm].

Note that, differently from the multicast notifications, this multicast error response will be the only one securely paired with the phantom cancellation request.

6.3. Client-Side Requirements

When using Group OSCORE to protect multicast notifications, the client performs as described in Section 3, with the following differences.

6.3.1. Informative Response

Upon receiving the informative response from the server, the client performs as described in Section 3.2, with the following additions.

Once completed step 2, the client decrypts and verifies the rebuilt phantom registration request as defined in Section 8.2 of [I-D.ietf-core-oscore-groupcomm], with the following differences.

- o The client MUST NOT perform any replay check. That is, the client skips step 3 in Section 8.2 of [RFC8613].
- o If decryption and verification of the phantom registration request succeed:
 - * The client MUST NOT update the Replay Window in the Recipient Context associated to the server. That is, the client skips the second bullet of step 6 in Section 8.2 of [RFC8613].
 - * The client MUST NOT take any further process as normally expected according to [RFC7252]. That is, the client skips step 8 in Section 8.2 of [RFC8613]. In particular, the client MUST NOT deliver the phantom registration request to the application, and MUST NOT take any action in the Token space of its unicast endpoint, where the informative response has been received.
 - * The client stores the values of the 'kid', 'piv' and 'kid context' fields from the OSCORE option of the phantom registration request.
- o If decryption and verification of the phantom registration request fail, the client MAY try sending a new registration request to the server (see Section 3.1). Otherwise, the client SHOULD explicitly withdraw from the group observation.

Once completed step 4, the client also decrypts and verifies the rebuilt latest multicast notification, just like it would for the multicast notifications transmitted as CoAP messages on the wire (see Section 6.3.2). The client proceeds with step 5 if decryption and verification of the latest multicast notification succeed, or to step 6 otherwise.

6.3.2. Notifications

After having successfully processed the informative response as defined in Section 6.3.1, the client will decrypt and verify every multicast notification for the target resource as defined in Section 8.4 of [I-D.ietf-core-oscore-groupcomm], with the following difference.

The client MUST set the two 'external_aad' defined in Sections 4.3.1 and 4.3.2 of [I-D.ietf-core-oscore-groupcomm] as follows. The particular way to achieve this is implementation specific.

- o 'request_kid' takes the value of the 'kid' field from the OSCORE option of the phantom registration request (see Section 6.3.1).
- o 'request_piv' takes the value of the 'piv' field from the OSCORE option of the phantom registration request (see Section 6.3.1).
- o 'request_kid_context' takes the value of the 'kid context' field from the OSCORE option of the phantom registration request (see Section 6.3.1).

Note that these same values are used to decrypt and verify each and every multicast notification received for the target resource.

The replay protection and checking of multicast notifications is performed as specified in Section 4.1.3.5.2 of [RFC8613].

7. Example with Group OSCORE

The following example refers to two clients C_1 and C_2 that register to observe a resource /r at a Server S, which has address SRV_ADDR and listens to the port number SRV_PORT. Before the following exchanges occur, no clients are observing the resource /r, which has value "1234".

The server S sends multicast notifications to the IP multicast address GRP_ADDR and port number GRP_PORT, and starts the group observation upon receiving a registration request from a first client that wishes to start a traditional observation on the resource /r.

Pairwise communication over unicast are protected with OSCORE, while S protects multicast notifications with Group OSCORE. Specifically:

- o C_1 and S have a pairwise OSCORE Security Context. In particular, C_1 has 'kid' = 1 as Sender ID, and SN_1 = 101 as Sender Sequence Number. Also, S has 'kid' = 3 as Sender ID, and SN_3 = 301 as Sender Sequence Number.
- o C_2 and S have a pairwise OSCORE Security Context. In particular, C_2 has 'kid' = 2 as Sender ID, and SN_2 = 201 as Sender Sequence Number. Also, S has 'kid' = 4 as Sender ID, and SN_4 = 401 as Sender Sequence Number.
- o S is a member of the OSCORE group with name "myGroup", and 'kid context' = 0x57ab2e as Group ID. In the OSCORE group, S has 'kid' = 5 as Sender ID, and SN_5 = 501 as Sender Sequence Number.

The following notation is used for the payload of the informative responses:

- o 'bstr(X)' denotes a CBOR byte string with value the byte serialization of X, with '|' denoting byte concatenation.
- o 'OPT' denotes a sequence of CoAP options. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.
- o 'PAYLOAD' denotes an encrypted CoAP payload. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.
- o 'SIGN' denotes the counter signature appended to an encrypted CoAP payload. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.

```

C_1      ----- [ Unicast w/ OSCORE ] -----> S  /r
| 0.05 (FETCH)
| Token: 0x4a
| OSCORE: {kid: 1 ; piv: 101 ; ...}
| <Other class U/I options>
| 0xff
| Encrypted_payload {
|   0x01 (GET),
|   Observe: 0 (Register),
|   <Other class E options>
|

```

```

    }

    (S allocates the available Token value 0x7b .)

    (S sends to itself a phantom observation request PH_REQ
     as coming from the IP multicast address GRP_ADDR .)
    -----
    /
    \-----> /r

        0.05 (FETCH)
        Token: 0x7b
        OSCORE: {kid: 5 ; piv: 501 ;
                  kid context: 57ab2e; ...}
        <Other class U/I options>
        0xff
        Encrypted_payload {
            0x01 (GET),
            Observe: 0 (Register),
            <Other class E options>
        }
        <Counter signature>

    (S steps SN_5 in the Group OSCORE Sec. Ctx : SN_5 <== 502)

    (S creates a group observation of /r .)

    (S increments the observer counter
     for the group observation of /r .)

C_1 <----- [ Unicast w/ OSCORE ] ----- S
2.05 (Content)
Token: 0x4a
OSCORE: {piv: 301; ...}
<Other class U/I options>
0xff
Encrypted_payload {
    5.03 (Service Unavailable),
    Content-Format: application/informative-response+cbor,
    <Other class E options>,
    0xff,
    CBOR_payload {
        ph_req      : bstr(0x05 | OPT | 0xff | PAYLOAD | SIGN),
        last_notif  : bstr(0x25 | OPT | 0xff | PAYLOAD | SIGN),
        tp_info     : [1, 0x7b, bstr(SRV_ADDR), SRV_PORT,
                       bstr(GRP_ADDR), GRP_PORT],
        join_uri    : "coap://myGM/ace-group/myGroup",
    }

```

```

        sec_gp      : "myGroup"
    }
}

C_2  ----- [ Unicast w/ OSCORE ] -----> S  /r
0.05 (FETCH)
Token: 0x01
OSCORE: {kid: 2 ; piv: 201 ; ...}
<Other class U/I options>
0xff
Encrypted_payload {
    0x01 (GET),
    Observe: 0 (Register),
    <Other class E options>
}

(S increments the observer counter
for the group observation of /r .)

C_2 <----- [ Unicast w/ OSCORE ] ----- S
2.05 (Content)
Token: 0x01
OSCORE: {piv: 401; ...}
<Other class U/I options>
0xff,
Encrypted_payload {
    5.03 (Service Unavailable),
    Content-Format: application/informative-response+cbor,
    <Other class E options>,
    0xff,
    CBOR_payload {
        ph_req      : bstr(0x05 | OPT | 0xff | PAYLOAD | SIGN),
        last_notif   : bstr(0x25 | OPT | 0xff | PAYLOAD | SIGN),
        tp_info      : [1, 0x7b, bstr(SRV_ADDR), SRV_PORT,
                        bstr(GRP_ADDR), GRP_PORT],
        join_uri     : "coap://myGM/ace-group/myGroup",
        sec_gp       : "myGroup"
    }
}

(The value of the resource /r changes to "5678".)

C_1
+ <----- [ Multicast w/ Group OSCORE ] ----- S
C_2 (Destination address/port: GRP_ADDR/GRP_PORT)
| 2.05 (Content)

```

```
Token: 0x7b
OSCORE: {kid: 5; piv: 502 ;
         kid context: 57ab2e; ...}
<Other class U/I options>
0xff
Encrypted_payload {
  2.05 (Content),
  Observe: 11,
  Content-Format: application/cbor,
  <Other class E options>,
  0xff,
  CBOR_Payload : "5678"
}
<Counter signature>
```

The two external_aad used to encrypt and countersign the multicast notification above have 'request_kid' = 5, 'request_piv' = 501 and 'request_kid_context' = 0x57ab2e. These values are specified in the 'kid', 'piv' and 'kid context' field of the OSCORE option of the phantom observation request, which is encoded in the 'ph_req' parameter of the unicast informative response to the two clients. Thus, the two clients can build the two same external_aad for decrypting and verifying this multicast notification and the following ones.

8. Intermediaries

This section specifies how the approach presented in Section 2 and Section 3 works when a proxy is used between the clients and the server. In addition to what specified in Section 5.7 of [RFC7252] and Section 5 of [RFC7641], the following applies.

- o A client sends its original observation request to the proxy, which forwards it to the server. The server considers the proxy as taking part in the group observation for that target resource, or takes it as a confirmation of interest if it is already the case from previously forwarded observation requests. Then, the server sends the informative response to the proxy, to be forwarded back to the client.
- o Upon receiving an informative response, the proxy performs as specified in Section 3, with the difference that it does not consider the latest multicast notification encoded in the 'last_notif' field. In particular, by using the information retrieved from the informative response, the proxy configures an observation of the target resource at the origin server, acting as

a client directly taking part in the group observation. The proxy MUST NOT cache the informative response.

As a consequence, the proxy will listen to the IP multicast address and port number indicated by the server in the informative response, as 'cli_addr' and 'cli_port' element of the 'tp_info' parameter, respectively (see Section 2.2.2.1). Furthermore, multicast notifications will match the phantom request stored at the proxy, based on the Token value specified in the 'token' element of the 'tp_info' parameter in the informative response.

Note that the proxy configures the observation of the target resource at the server only once, when receiving the first informative response associated to a newly started group observation. That is, after forwarding observation requests from a following new client to be added to the same group observation, the proxy does not take any action other than forwarding the informative response back to that client.

- o When forwarding the informative response back to a client, the proxy adds that client to the list of its registered observers for the target resource, consistently with the previously received observation request. In particular, the Token value associated to this observation and to use with the client is the same Token value used in the original registration request that the client has sent to the proxy.
- o Upon receiving an informative response from the proxy, a client performs as defined in Section 3, with the following differences.
 - * In step 1, the client relies on a CoAP endpoint used for messages having:
 - + As source address and port number, the IP address and port number used by the proxy, i.e. the source address and port number of the informative response received from the proxy.
 - + As destination address and port number, the IP address and port number used by the client, i.e. the destination address and port number of the informative response received from the proxy.
 - * In step 2, the Token value used when rebuilding the phantom registration request is the same Token value used in the original registration request sent to the proxy. The client MUST use that Token value as its own local Token value associated to that group observation, with respect to the proxy. The particular way to achieve this is implementation

specific. The client does not consider the transport-specific information specified in the 'tp_info' parameter of the informative response.

- * In step 4, the Token value used when rebuilding the latest multicast notification is the same Token value used to rebuild the phantom registration request, as explained above.
- o Upon receiving a multicast notification from the server, the proxy forwards it back separately to each observer client over unicast. Note that the notification forwarded back to a certain client has the same Token value of the original observation request sent by that client to the proxy.

An example is provided in Appendix C.

In the general case with a chain of two or more proxies, every proxy in the chain takes the role of client with the (next hop towards the) origin server. Note that the proxy adjacent to the origin server is the only one in the chain that listens to an IP multicast address to receive notifications for the group observation. Furthermore, every proxy in the chain takes the role of server with the (previous hop towards the) origin client.

9. Intermediaries Together with End-to-End Security

As defined in Section 6, Group OSCORE can be used to protect multicast notifications end-to-end between the origin server and the clients. In such a case, additional actions are required when also the informative responses from the origin server are protected specifically end-to-end, by using OSCORE or Group OSCORE.

In fact, the proxy adjacent to the origin server is not able to access the encrypted payload of such informative responses. Hence, the proxy cannot retrieve the 'ph_req' and 'tp_info' parameters necessary to correctly receive multicast notifications and forward them back to the clients.

Then, differently from what defined in Section 8, each proxy receiving an informative response simply forwards it back to the client that has sent the corresponding observation request. Note that the proxy does not even realize the message to be an actual informative response, since the outer Code field is set to 2.05 (Content).

Once a client receives the informative response, it not only configures an observation of the target resource at the origin server. In addition, the client transmits the re-built phantom

request as intended to reach the proxy adjacent to the origin server. In particular, the client includes the new Listen-To-Multicast-Responses CoAP option defined in Section 9.1, to provide that proxy with the transport-specific information required for receiving multicast notifications for the group observation.

Details on the additional message exchange and processing are defined in Section 9.2.

9.1. The Listen-To-Multicast-Responses Option

To allow the proxy to listen to the multicast notifications sent by the server, a new CoAP option is introduced. This option MUST be supported by clients interested to take part in group observations through intermediaries, and by proxies that collect multicast notifications and forward them back to the observer clients.

The option is called Listen-To-Multicast-Responses and is intended only for requests. As summarized in Figure 2, the option is critical and proxy-unsafe.

No.	C	U	N	R	Name	Format	Len.	Default
TBD	x	x			Listen-To-Multicast-Responses	(*)	3-1024	(none)

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

(*) See below.

Figure 2: Listen-To-Multicast-Responses

The Listen-To-Multicast-Responses option includes the serialization of a CBOR array. This specifies transport-specific message information required for listening to the multicast notifications of a group observation, and intended to the proxy adjacent to the origin server sending those notifications. In particular, the serialized CBOR array has the same format specified in Section 2.2.2 for the 'tp_info' parameter of the informative response (see Section 2.2).

The Listen-To-Multicast-Responses option is of class U for OSCORE [RFC8613][I-D.ietf-core-oscore-groupcomm].

9.2. Message Processing

Compared to Section 8, the following additions apply when informative responses are protected end-to-end between the origin server and the clients.

After the origin server sends an informative response, each proxy simply forwards it back to the (previous hop towards the) origin client that has sent the observation request.

Once received the informative response, the origin client rebuilds the phantom request and accordingly configures an observation of the target resource, just as defined in Section 8. Then, before rebuilding the latest multicast notification from the 'last_notif' parameter of the informative response, the client performs the following additional actions.

- o The client builds a ticket request (see Section 3 of [I-D.amsuess-core-cachable-oscore]), as intended to reach the proxy adjacent to the origin server. The ticket request is formatted as follows.
 - * The Code field, the outer CoAP options and the encrypted payload concatenated with the counter signature are the same of the phantom request used for the group observation. That is, they are as specified in the 'ph_req' parameter of the received informative response.
 - * An outer Observe option is included and set to 0 (Register).
 - * The outer options Proxy-Scheme, Uri-Host and Uri-Port are included, and set to the same values they had in the original registration request sent by the client.
 - * The new option Listen-To-Multicast-Responses is included as an outer option. The value is set to the serialization of the CBOR array specified by the 'tp_info' parameter of the informative response.

Note that, except for transport-specific information such as the Token and Message ID values, every different client participating to the same group observation (hence rebuilding the same phantom request) will build the same ticket request.

Note also that, identically to the phantom request, the ticket request is still protected with Group OSCORE, i.e. it has the same OSCORE option, encrypted payload and counter signature.

Then, the client sends the ticket request to the next hop towards the origin server. Every proxy in the chain forwards the ticket request to the next hop towards the origin server, until the last proxy in the chain is reached. This last proxy, adjacent to the origin server, proceeds as follows.

- o The proxy MUST NOT further forward the ticket request to the origin server.
- o The proxy removes the Proxy-Scheme, Uri-Host and Uri-Port options from the ticket request.
- o The proxy removes the Listen-To-Multicast-Responses option from the ticket request, and extracts the conveyed transport-specific information.
- o The proxy rebuilds the phantom request associated to the group observation, by using the ticket request as directly providing the required transport-independent information. This includes the outer Code field, the outer CoAP options and the encrypted payload concatenated with the counter signature.
- o The proxy configures an observation of the target resource at the origin server, acting as a client directly taking part in the group observation. To this end, the proxy uses the rebuilt phantom request and the transport-specific information retrieved from the Listen-To-Multicast-Responses Option. The particular way to achieve this is implementation specific.

After that, the proxy will listen to the IP multicast address and port number indicated in the Listen-To-Multicast-Responses option, as 'cli_addr' and 'cli_port' element of the serialized CBOR array, respectively. Furthermore, multicast notifications will match the phantom request stored at the proxy, based on the Token value specified in the 'token' element of the serialized CBOR array in the Listen-To-Multicast-Responses option.

An example is provided in Appendix D.

10. Informative Response Parameters

This specification defines a number of fields used in the informative response message defined in Section 2.2.

The table below summarizes them and specifies the CBOR key to use instead of the full descriptive name. Note that the media type application/informative-response+cbor MUST be used when these fields are transported.

Name	CBOR Key	CBOR Type	Reference
ph_req	1	byte string	Section 2.2
last_notif	2	byte string	Section 2.2
tp_info	3	array	Section 2.2
join_uri	4	text string	Section 6.1
sec_gp	5	text string	Section 6.1
as_uri	6	text string	Section 6.1
cs_alg	7	int / text string	Section 6.1
cs_crv	8	int / text string	Section 6.1
cs_kty	9	int / text string	Section 6.1
cs_kenc	10	int	Section 6.1
alg	11	int / text string	Section 6.1
hkdf	12	int / text string	Section 6.1

11. Transport Protocol Identifiers

This specification defines some values of transport protocol identifiers used in the 'tp_info' parameter of the informative response message defined in Section 2.2 of this specification.

According to the encoding specified in Section 2.2.2, these values are used as first element of the CBOR array 'tp_info' of an informative response message.

The table below summarizes them, and specifies the integer value to use instead of the full descriptive name.

Name	Description	Value	Reference
Reserved	This value is reserved	0	
UDP	UDP is used to transport CoAP messages, as per [RFC7252] and [I-D.ietf-core-groupcomm-bis]	1	Section 2.2.1

12. Security Considerations

The same security considerations from [RFC7252][RFC7641][I-D.ietf-core-groupcomm-bis][RFC8613][I-D.ietf-core-oscore-groupcomm] hold for this document.

If multicast notifications are protected using Group OSCORE, the original registration requests and related unicast (notification) responses MUST also be secured, including and especially the informative responses from the server. This prevents on-path active adversaries from altering the conveyed IP multicast address and serialized phantom registration request. Thus, it ensures secure binding between every multicast notification for a same observed resource and the phantom registration request that started the group observation of that resource.

To this end, clients and servers SHOULD use OSCORE or Group OSCORE, so ensuring that the secure binding above is enforced end-to-end between the server and each observing client.

12.1. Listen-To-Multicast-Responses Option

The CoAP option Listen-To-Multicast-Responses defined in Section 9.1 is of class U for OSCORE and Group OSCORE [RFC8613][I-D.ietf-core-oscore-groupcomm].

This allows the proxy adjacent to the origin server to access the option value conveyed in a ticket request (see Section 9.2), and to retrieve from it the transport-specific information about a phantom request. By doing so, the proxy becomes able to configure an observation of the target resource and to receive multicast notifications matching to the phantom request.

Any proxy in the chain, as well as further possible intermediaries or on-path active adversaries, are thus able to remove the option or alter its content, before the ticket request reaches the proxy adjacent to the origin server.

Removing the option would result in the proxy adjacent to the origin server to not configure the group observation, if that has not happened yet. In such a case, the proxy would not receive the corresponding multicast notifications to be forwarded back to the clients.

Altering the option content would result in the proxy adjacent to the origin server to incorrectly configure a group observation (e.g., by indicating a wrong multicast IP address) hence preventing the correct reception of multicast notifications and their forwarding to the clients; or to configure bogus group observations that are currently not active on the origin server.

In order to prevent what described above, the ticket requests conveying the Listen-To-Multicast-Responses option can be additionally protected hop-by-hop.

13. IANA Considerations

This document has the following actions for IANA.

13.1. Media Type Registrations

This specification registers the media type 'application/informative-response+cbor' for error messages as informative response defined in Section 2.2 of this specification, when carrying parameters encoded in CBOR. This registration follows the procedures specified in [RFC6838].

- o Type name: application
- o Subtype name: informative-response+cbor
- o Required parameters: none
- o Optional parameters: none
- o Encoding considerations: Must be encoded as a CBOR map containing the parameters defined in Section 2.2 of [this document].
- o Security considerations: See Section 12 of [this document].
- o Interoperability considerations: n/a
- o Published specification: [this document]

- o Applications that use this media type: The type is used by CoAP servers and clients that support error messages as informative response defined in Section 2.2 of [this document].
- o Additional information: n/a
- o Person & email address to contact for further information: iesg@ietf.org [1]
- o Intended usage: COMMON
- o Restrictions on usage: None
- o Author: Marco Tiloca marco.tiloca@ri.se [2]
- o Change controller: IESG

13.2. CoAP Content-Formats Registry

IANA is asked to add the following entry to the "CoAP Content-Formats" sub-registry defined in Section 12.3 of [RFC7252], within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Media Type: application/informative-response+cbor

Encoding: -

ID: TBD

Reference: [this document]

13.3. Informative Response Parameters Registry

This specification establishes the "Informative Response Parameters" IANA registry. The registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 13.6.

The columns of this registry are:

- o Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.
- o CBOR Key: This is the value used as CBOR key of the item. These values MUST be unique. The value can be a positive integer, a negative integer, or a string.

- o **CBOR Type:** This contains the CBOR type of the item, or a pointer to the registry that defines its type, when that depends on another item.
- o **Reference:** This contains a pointer to the public specification for the item.

This registry has been initially populated by the values in Section 10. The "Reference" column for all of these entries refers to sections of this document.

13.4. Transport Protocol Identifiers Registry

This specification establishes the "Transport Protocol Identifiers" IANA sub-registry, within the "Informative Response Parameters" registry defined in Section 13.3 of this specification. The sub-registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 13.6.

The columns of this sub-registry are:

- o **Name:** This is a descriptive name that enables easier reference to the item. The name **MUST** be unique. It is not used in the encoding.
- o **Description:** Text giving an overview of the transport protocol referred by this item.
- o **Value:** CBOR abbreviation for the name of this transport protocol. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.
- o **Reference:** This contains a pointer to the public specification for the item.

This registry has been initially populated by the values in Section 11. The "Reference" column for all of these entries refers to sections of this document.

13.5. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry defined in [RFC7252] within the "CoRE Parameters" registry.

Number	Name	Reference
TBD	Multicast-Response-Feedback-Divider	[[this document]]
TBD	Listen-To-Multicast-Responses	[[this document]]

13.6. Expert Review Instructions

The IANA registries established in this document are defined as expert review. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- o Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments, code points in other ranges should not be assigned for testing.
- o Specifications are required for the standards track range of point assignment. Specifications should exist for specification required ranges, but early assignment before a specification is available is considered to be permissible. Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.
- o Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be

used on, and the number of code points left that encode to that size.

14. References

14.1. Normative References

- [COSE.Algorithms]
IANA, "COSE Algorithms",
<<https://www.iana.org/assignments/cose/cose.xhtml#algorithms>>.
- [COSE.Elliptic.Curves]
IANA, "COSE Elliptic Curves",
<<https://www.iana.org/assignments/cose/cose.xhtml#elliptic-curves>>.
- [COSE.Key.Types]
IANA, "COSE Key Types",
<<https://www.iana.org/assignments/cose/cose.xhtml#key-type>>.
- [I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work in progress), September 2020.
- [I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-ietf-core-groupcomm-bis-02 (work in progress), November 2020.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-10 (work in progress), November 2020.
- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", draft-ietf-cose-rfc8152bis-algs-12 (work in progress), September 2020.
- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", draft-ietf-cose-rfc8152bis-struct-14 (work in progress), September 2020.

- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

14.2. Informative References

- [I-D.amsuess-core-cachable-oscore]
Amsuess, C. and M. Tiloca, "Cachable OSCORE", draft-amsuess-core-cachable-oscore-00 (work in progress), July 2020.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-09 (work in progress), November 2020.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-35 (work in progress), June 2020.
- [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-09 (work in progress), September 2019.
- [I-D.ietf-core-coral]
Hartke, K., "The Constrained RESTful Application Language (CoRAL)", draft-ietf-core-coral-03 (work in progress), March 2020.
- [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-25 (work in progress), July 2020.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-38 (work in progress), May 2020.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-07 (work in progress), November 2020.

[MOBICOM99]

Ni, S., Tseng, Y., Chen, Y., and J. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network", Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking , August 1999, <<https://people.eecs.berkeley.edu/~culler/cs294-f03/papers/bcast-storm.pdf>>.

[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>>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.

14.3. URIs

[1] <mailto:iesg@ietf.org>

[2] <mailto:marco.tiloca@ri.se>

Appendix A. Different Sources for Group Observation Data

While the clients usually receive the phantom registration request and other information related to the group observation through an Informative Response, the same data can be made available through different services, such as the following ones.

A.1. Topic Discovery in Publish-Subscribe Settings

In a Publish-Subscribe scenario ([I-D.ietf-core-coap-pubsub]), a group observation can be discovered along with topic metadata. For instance, a discovery step can make the following metadata available.

This example assumes a CoRAL namespace [I-D.ietf-core-coral], that contains properties analogous to those in the content-format application/informative-response+cbor.

Request:

```
GET </ps/topics?rt=oic.r.temperature>
Accept: CoRAL
```

Response:

```
2.05 Content
Content-Format: CoRAL

rdf:type <http://example.org/pubsub/topic-list>
topic </ps/topics/1234> {
  ph_req h"0160.."
  last_notif h"256105.."
  tp_info [1, h"7b", h"20010db80100..0001", 5683,
           h"ff35003020010db8..1234", 5683]
}
```

With this information from the topic discovery step, the client can already set up its multicast address and start receiving multicast notifications.

In heavily asymmetric networks like municipal notification services, discovery and notifications do not necessarily need to use the same network link. For example, a departure monitor could use its (costly and usually-off) cellular uplink to discover the topics it needs to update its display to, and then listen on a LoRA-WAN interface for receiving the actual multicast notifications.

A.2. Introspection at the Multicast Notification Sender

For network debugging purposes, it can be useful to query a server that sends multicast responses as matching a phantom registration request.

Such an interface is left for other documents to specify on demand. As an example, a possible interface can be as follows, and rely on the already known Token value of intercepted multicast notifications, associated to a phantom registration request.

Request:

```
GET </.well-known/core/mc-sender?token=6464>
```

Response:

2.05 Content

Content-Format: application/informative-response+cbor

```
{
  'ph_req': h"0160..",
  'last_notif' : h"256105..",
  'tp_info': [1, h"7b", h"20010db80100..0001", 5683,
              h"ff35003020010db8..1234", 5683]
}
```

For example, a network sniffer could offer sending such a request when unknown multicast notifications are seen on a network. Consequently, it can associate those notifications with a URI, or decrypt them, if member of the correct OSCORE group.

Appendix B. Pseudo-Code for Rough Counting of Clients

This appendix provides a description in pseudo-code of the two algorithms used for the rough counting of active observers, as defined in Section 5.

In particular, Appendix B.1 describes the algorithm for the client side, while Appendix B.2 describes an optimized version for constrained clients. Finally, Appendix B.3 describes the algorithm for the server side.

B.1. Client Side

```
input:  int Q, // Value of the MRFD option
        int LEISURE_TIME, // DEFAULT_LEISURE from RFC 7252,
                          // unless overridden
```

```
output: None
```

```
int RAND_MIN = 0;
int RAND_MAX = (2**Q) - 1;
int I = randomInteger(RAND_MIN, RAND_MAX);

if (I == 0) {
    float fraction = randomFloat(0, 1);

    Timer t = new Timer();
    t.setAndStart(fraction * LEISURE_TIME);
    while(!t.isExpired());

    Request req = new Request();
    // Initialize as NON and with maximum
    // No-Response settings, set options ...

    Option opt = new Option(OBSERVE);
    opt.set(0);
    req.setOption(opt);

    opt = new Option(MRFD);
    req.setOption(opt);

    req.send(SRV_ADDR, SRV_PORT);
}
```

B.2. Client Side - Optimized Version

```
input:  int Q, // Value of the MRFD option
        int LEISURE_TIME, // DEFAULT_LEISURE from RFC 7252,
                          // unless overridden
```

```
output: None
```

```
const unsigned int UINT_BIT = CHAR_BIT * sizeof(unsigned int);
```

```
if (respond_to(Q) == true) {
    float fraction = randomFloat(0, 1);

    Timer t = new Timer();
    t.setAndStart(fraction * LEISURE_TIME);
    while(!t.isExpired());

    Request req = new Request();
    // Initialize as NON and with maximum
    // No-Response settings, set options ...

    Option opt = new Option(OBSERVE);
    opt.set(0);
    req.setOption(opt);

    opt = new Option(MRFD);
    req.setOption(opt);

    req.send(SRV_ADDR, SRV_PORT);
}
```

```
bool respond_to(int Q) {
    while (Q >= UINT_BIT) {
        if (rand() != 0) return false;
        Q -= UINT_BIT;
    }
    unsigned int mask = ~((~0u) << Q);
    unsigned int masked = mask & rand();
    return masked == 0;
}
```

B.3. Server Side

```
input:  int COUNT, // Current observer counter
        int M, // Desired number of confirmations
        int MAX_CONFIRMATION_WAIT,
        Response notification, // Multicast notification to send
```

```
output: int NEW_COUNT // Updated observer counter
```

```
int D = 4; // Dampener value
int RETRY_NEXT_THRESHOLD = 4;
float CANCEL_THRESHOLD = 0.2;

int N = max(COUNT, 1);
int Q = max(ceil(log2(N / M)), 0);
Option opt = new Option(MRFD);
opt.set(Q);

notification.setOption(opt);
<Finalize the notification message>
notification.send(GRP_ADDR, GRP_PORT);

Timer t = new Timer();
t.setAndStart(MAX_CONFIRMATION_WAIT); // Time t1
while(!t.isExpired());

// Time t2

int R = <number of requests to the target resource
        between t1 and t2, with the MRFD option>;

int E = R * (2**Q);

// Determine after how many multicast notifications
// the next count update will be performed
if ((R == 0) || (max(E/N, N/E) > RETRY_NEXT_THRESHOLD)) {
    <Next count update with the next multicast notification>
}
else {
    <Next count update after 10 multicast notifications>
}

// Compute the new count estimate
int COUNT_PRIME = <current value of the observer counter>;
int NEW_COUNT = COUNT_PRIME + ((E - N) / D);

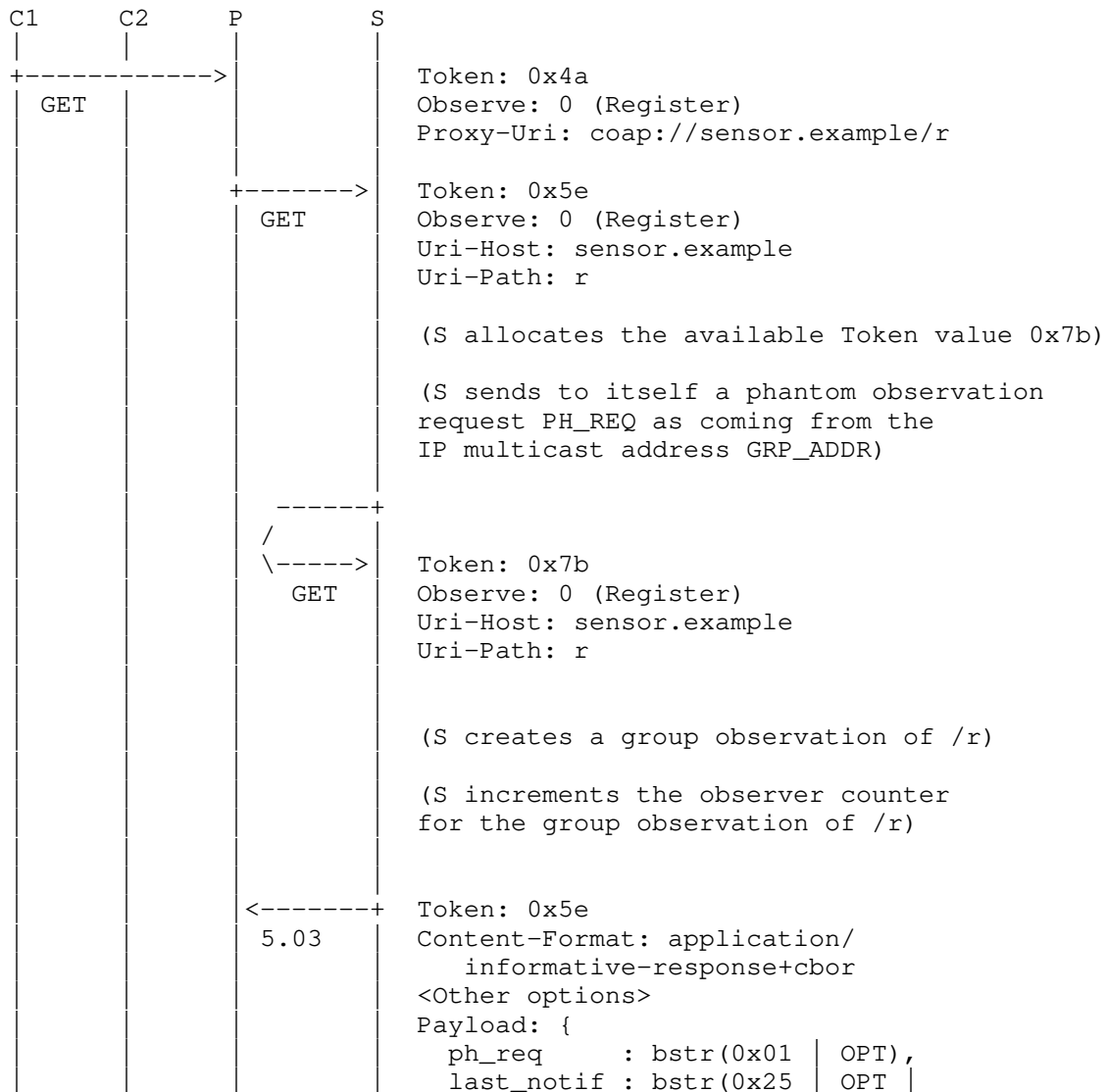
// Determine whether to cancel the group observation
if (NEW_COUNT < CANCEL_THRESHOLD) {
    <Cancel the group observation>;
    return 0;
}

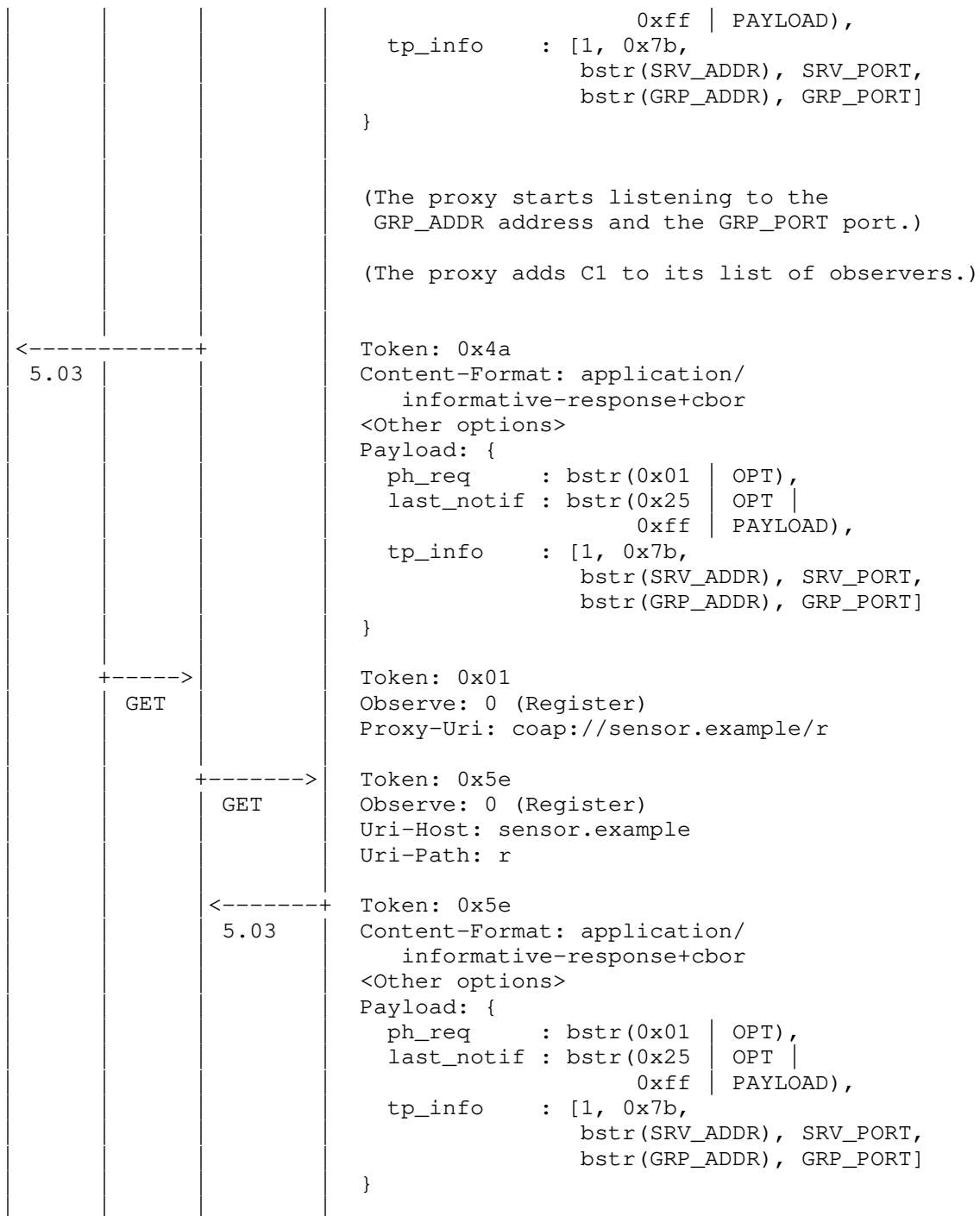
return NEW_COUNT;
```

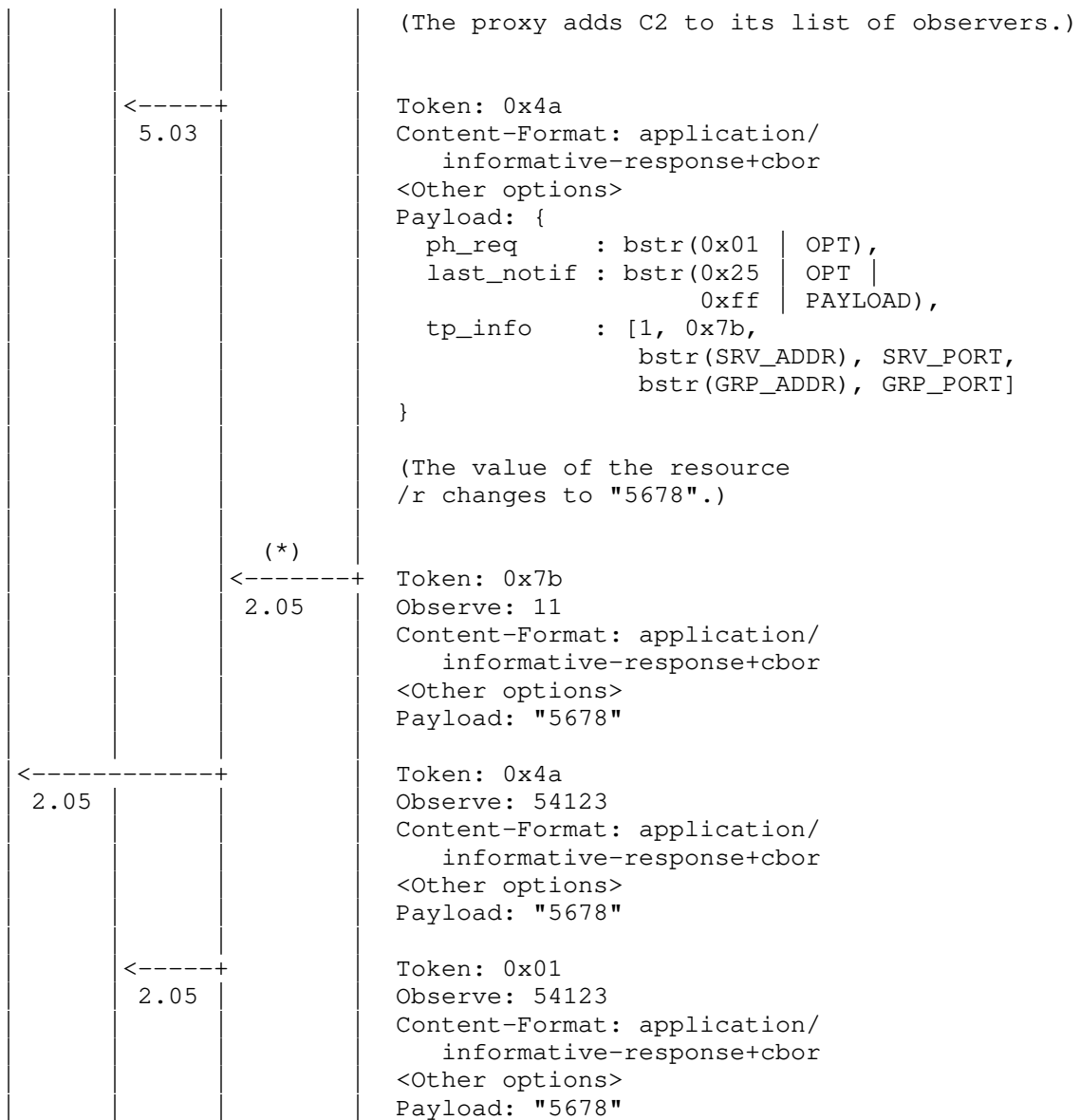
Appendix C. Example with a Proxy

This section provides an example when a proxy P is used between the clients and the server. The same assumptions and notation used in Section 4 are used for this example. In addition, the proxy has address PRX_ADDR and listens to the port number PRX_PORT.

Unless explicitly indicated, all messages transmitted on the wire are sent over unicast.







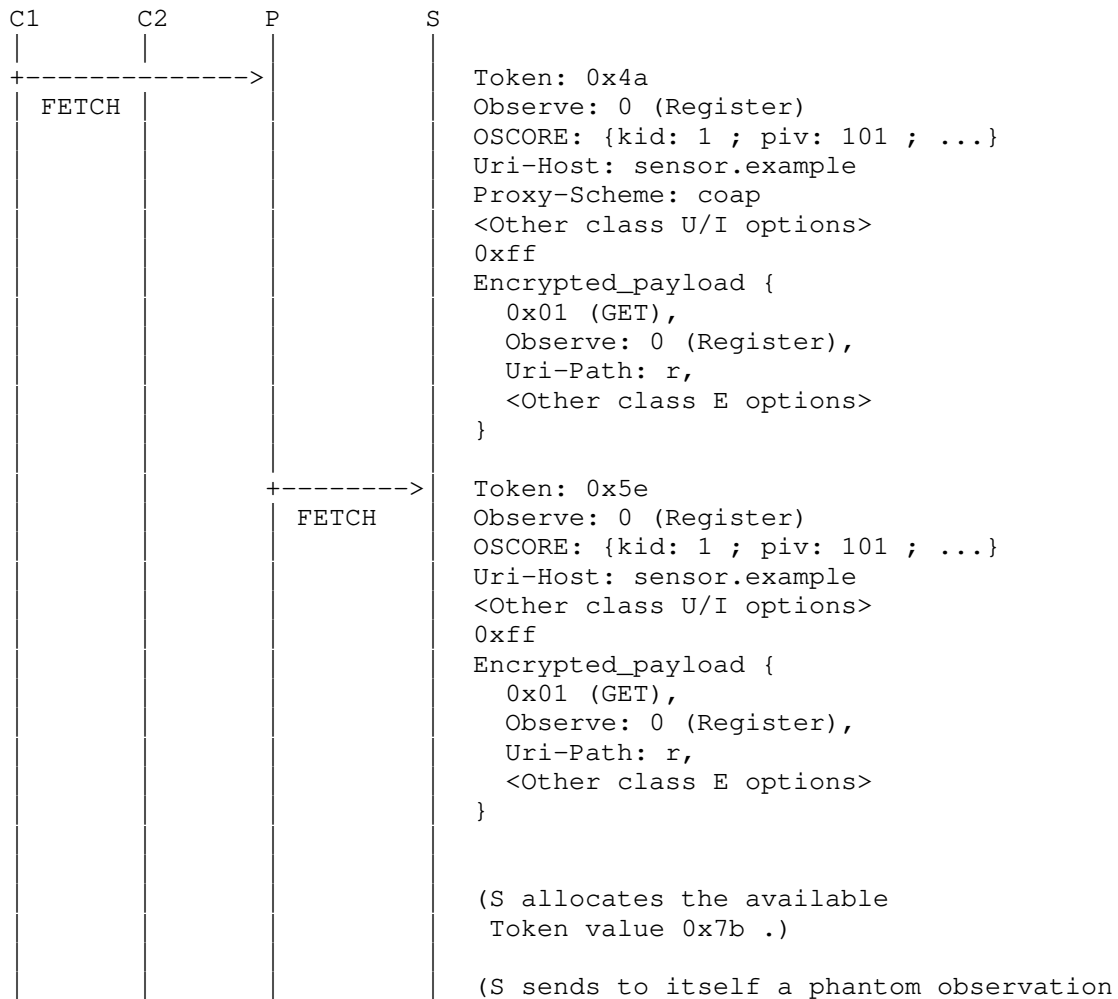
(*) Sent over IP multicast to GROUP_ADDR:GROUP_PORT

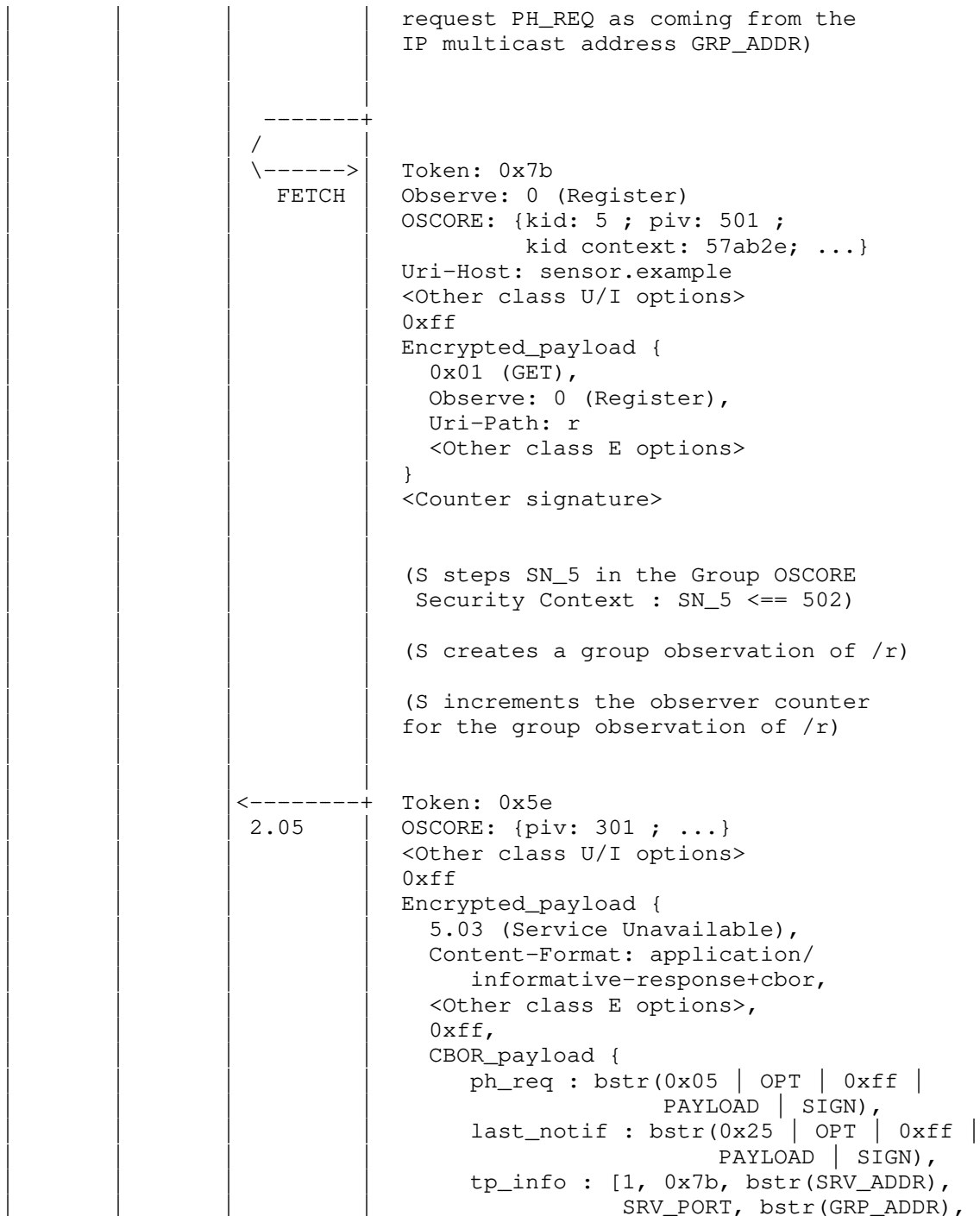
Appendix D. Example with a Proxy and Group OSCORE

This section provides an example when a proxy P is used between the clients and the server, and Group OSCORE is used to protect multicast notifications end-to-end between the server and the clients.

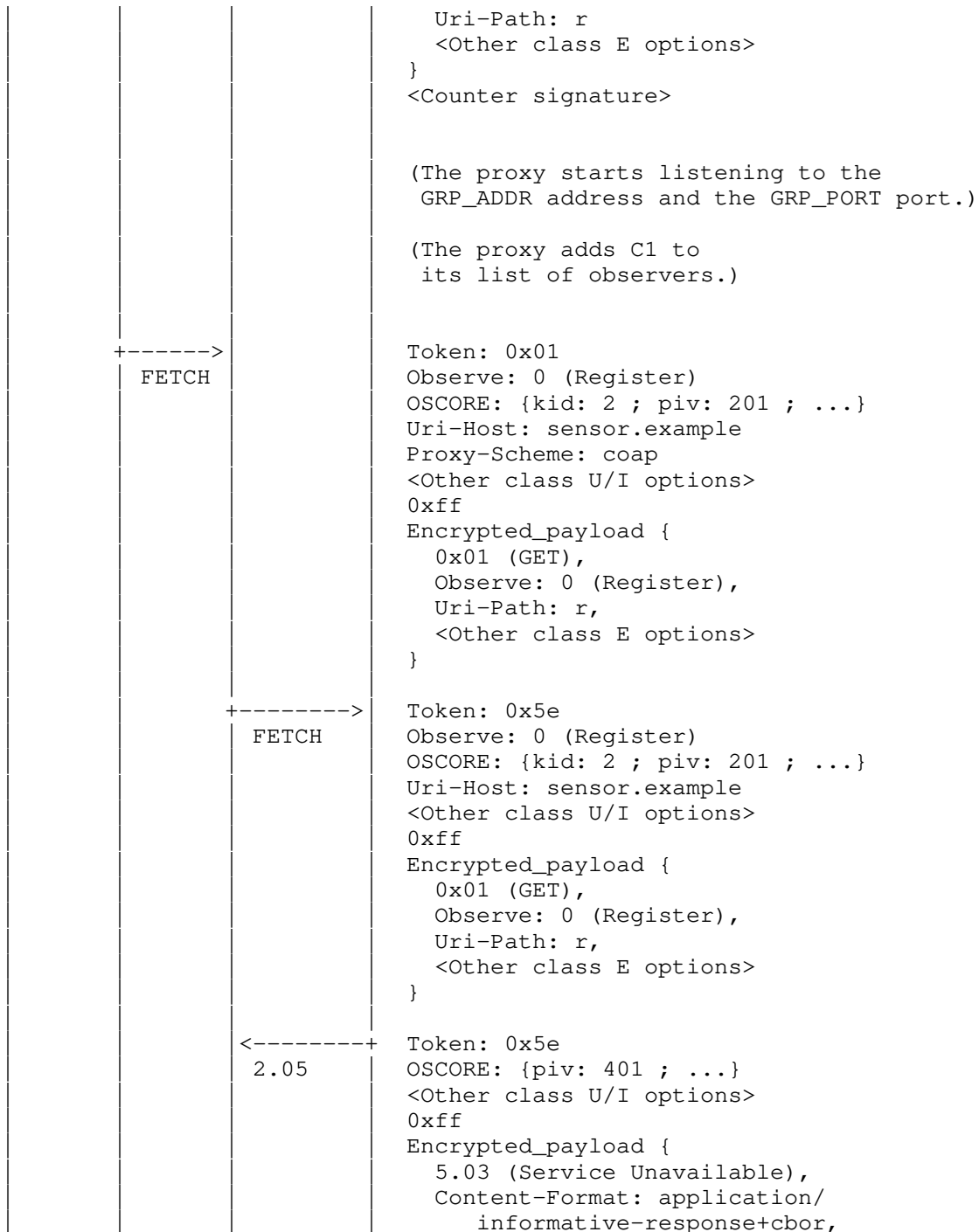
The same assumptions and notation used in Section 7 are used for this example. In addition, the proxy has address PRX_ADDR and listens to the port number PRX_PORT.

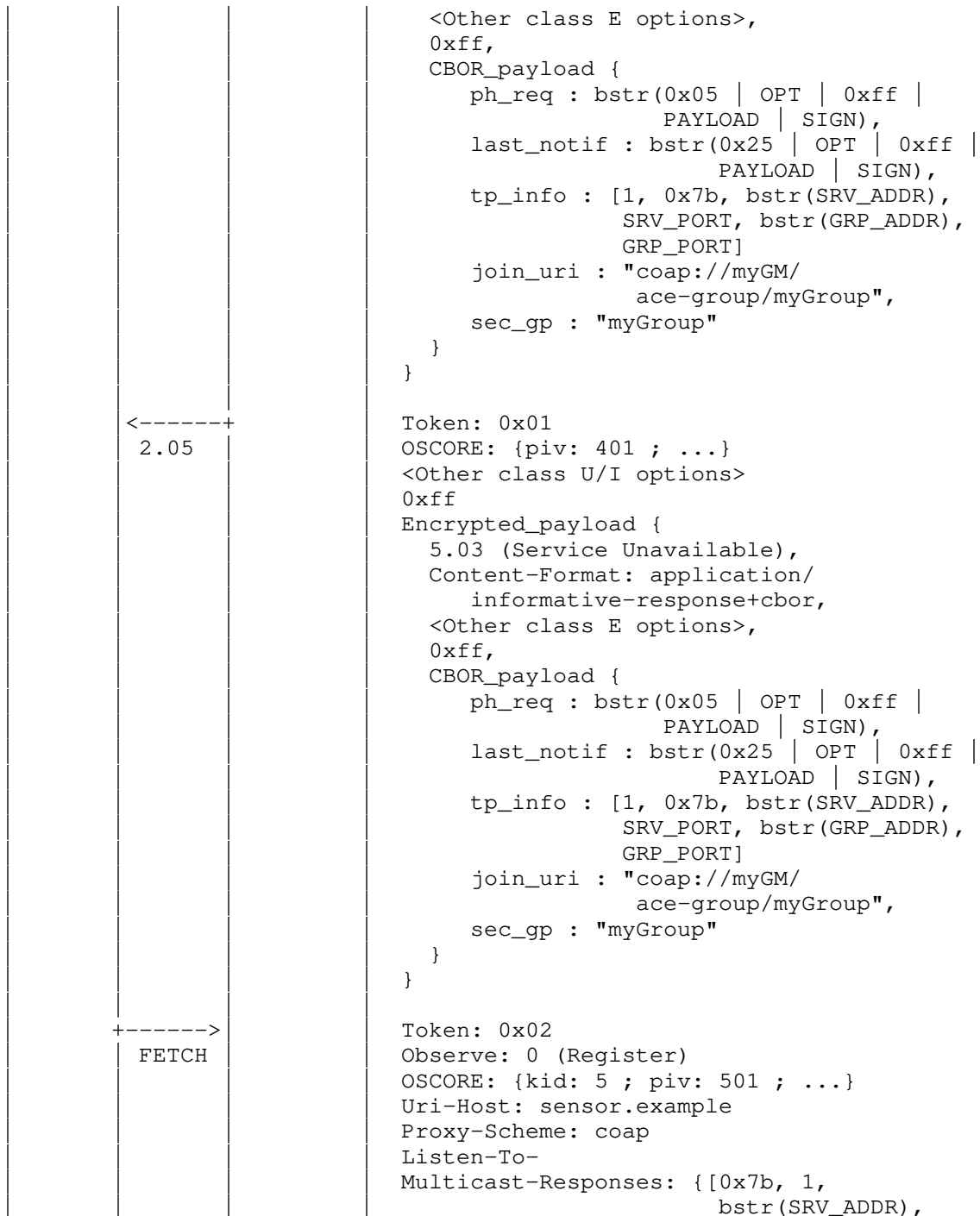
Unless explicitly indicated, all messages transmitted on the wire are sent over unicast and protected with OSCORE end-to-end between a client and the server.





			<pre> GRP_PORT] join_uri : "coap://myGM/ ace-group/myGroup", sec_gp : "myGroup" } } </pre>
←-----+ 2.05			<pre> Token: 0x4a OSCORE: {piv: 301 ; ...} <Other class U/I options> 0xff Encrypted_payload { 5.03 (Service Unavailable), Content-Format: application/ informative-response+cbor, <Other class E options>, 0xff, CBOR_payload { ph_req : bstr(0x05 OPT 0xff PAYLOAD SIGN), last_notif : bstr(0x25 OPT 0xff PAYLOAD SIGN), tp_info : [1, 0x7b, bstr(SRV_ADDR), SRV_PORT, bstr(GRP_ADDR), GRP_PORT] join_uri : "coap://myGM/ ace-group/myGroup", sec_gp : "myGroup" } } </pre>
-----+ FETCH			<pre> Token: 0x4b Observe: 0 (Register) OSCORE: {kid: 5 ; piv: 501 ; ...} Uri-Host: sensor.example Proxy-Scheme: coap Listen-To- Multicast-Responses: {[0x7b, 1, bstr(SRV_ADDR), SRV_PORT, bstr(GRP_ADDR), GRP_PORT] } <Other class U/I options> 0xff Encrypted_payload { 0x01 (GET), Observe: 0 (Register), </pre>





			SRV_PORT, bstr(GRP_ADDR), GRP_PORT] } <Other class U/I options> 0xff Encrypted_payload { 0x01 (GET), Observe: 0 (Register), Uri-Path: r <Other class E options> } <Counter signature> (The proxy adds C2 to its list of observers.) (The value of the resource /r changes to "5678".)
		(*)	
	2.05	<-----+	Token: 0x7b Observe: 11 OSCORE: {kid: 5; piv: 502 ; kid context: 57ab2e; ...} <Other class U/I options> 0xff Encrypted_payload { 2.05 (Content), Observe: 11, Content-Format: application/cbor, <Other class E options>, 0xff, CBOR_Payload : "5678" } <Counter signature>
	2.05	<-----+	Token: 0x4b Observe: 54123 OSCORE: {kid: 5; piv: 502 ; kid context: 57ab2e; ...} <Other class U/I options> 0xff

			<pre> Encrypted_payload { 2.05 (Content), Observe: 11, Content-Format: application/cbor, <Other class E options>, 0xff, CBOR_Payload : "5678" } <Counter signature> </pre>
	<pre> <-----+ 2.05 </pre>		<pre> Token: 0x02 Observe: 54123 OSCORE: {kid: 5; piv: 502 ; kid context: 57ab2e; ...} <Other class U/I options> 0xff Encrypted_payload { 2.05 (Content), Observe: 11, Content-Format: application/cbor, <Other class E options>, 0xff, CBOR_Payload : "5678" } <Counter signature> </pre>

(*) Sent over IP multicast to GROUP_ADDR:GROUP_PORT and protected with Group OSCORE end-to-end between the server and the clients.

Acknowledgments

The authors sincerely thank Carsten Bormann, Klaus Hartke, Jaime Jimenez, John Mattsson, Ludwig Seitz, Jim Schaad and Goeran Selander for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Rikard Hoeglund
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: rikard.hoglund@ri.se

Christian Amsuess
Hollandstr. 12/4
Vienna 1020
Austria

Email: christian@amsuess.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

CoRE Working Group
Internet-Draft
Updates: 7252, 7641 (if approved)
Intended status: Standards Track
Expires: August 26, 2021

M. Tiloca
R. Hoeglund
RISE AB
C. Amsuess
F. Palombini
Ericsson AB
February 22, 2021

Observe Notifications as CoAP Multicast Responses
draft-tiloca-core-observe-multicast-notifications-05

Abstract

The Constrained Application Protocol (CoAP) allows clients to "observe" resources at a server, and receive notifications as unicast responses upon changes of the resource state. In some use cases, such as based on publish-subscribe, it would be convenient for the server to send a single notification addressed to all the clients observing a same target resource. This document updates RFC7252 and RFC7641, and defines how a server sends observe notifications as response messages over multicast, synchronizing all the observers of a same resource on a same shared Token value. Besides, this document defines how Group OSCORE can be used to protect multicast notifications end-to-end between the server and the observer clients.

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 26, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	5
2. Server-Side Requirements	5
2.1. Request	6
2.2. Informative Response	7
2.2.1. Encoding of Transport-Specific Message Information	8
2.2.2. Encoding of Transport-Independent Message Information	11
2.3. Notifications	12
2.4. Congestion Control	13
2.5. Cancellation	13
3. Client-Side Requirements	14
3.1. Request	14
3.2. Informative Response	14
3.3. Notifications	16
3.4. Cancellation	16
4. Web Linking	16
5. Example	17
6. Rough Counting of Clients in the Group Observation	19
6.1. Processing on the Client Side	20
6.2. Processing on the Server Side	21
6.2.1. Request for Feedback	21
6.2.2. Collection of Feedback	22
6.2.3. Processing of Feedback	22
7. Protection of Multicast Notifications with Group OSCORE	24
7.1. Signaling the OSCORE Group in the Informative Response	24
7.2. Server-Side Requirements	26
7.2.1. Registration	27
7.2.2. Informative Response	27
7.2.3. Notifications	27
7.2.4. Cancellation	28
7.3. Client-Side Requirements	28

7.3.1. Informative Response	29
7.3.2. Notifications	30
8. Example with Group OSCORE	30
9. Intermediaries	34
10. Intermediaries Together with End-to-End Security	36
10.1. The Listen-To-Multicast-Responses Option	36
10.2. Message Processing	37
11. Informative Response Parameters	39
12. Transport Protocol Information	40
13. Security Considerations	41
13.1. Listen-To-Multicast-Responses Option	41
14. IANA Considerations	42
14.1. Media Type Registrations	42
14.2. CoAP Content-Formats Registry	43
14.3. Informative Response Parameters Registry	43
14.4. CoAP Transport Information Registry	44
14.5. CoAP Option Numbers Registry	45
14.6. Expert Review Instructions	45
15. References	46
15.1. Normative References	46
15.2. Informative References	48
15.3. URIs	50
Appendix A. Different Sources for Group Observation Data	50
A.1. Topic Discovery in Publish-Subscribe Settings	50
A.2. Introspection at the Multicast Notification Sender	51
Appendix B. Pseudo-Code for Rough Counting of Clients	52
B.1. Client Side	52
B.2. Client Side - Optimized Version	53
B.3. Server Side	53
Appendix C. OSCORE Group Self-Managed by the Server	55
Appendix D. Phantom Request as Deterministic Request	57
Appendix E. Example with a Proxy	58
Appendix F. Example with a Proxy and Group OSCORE	60
Acknowledgments	66
Authors' Addresses	67

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] has been extended with a number of mechanisms, including resource Observation [RFC7641]. This enables CoAP clients to register at a CoAP server as "observers" of a resource, and hence being automatically notified with an unsolicited response upon changes of the resource state.

CoAP supports group communication over IP multicast [I-D.ietf-core-groupcomm-bis]. This includes support for Observe registration requests over multicast, in order for clients to

efficiently register as observers of a resource hosted at multiple servers.

However, in a number of use cases, using multicast messages for responses would also be desirable. That is, it would be useful that a server sends observe notifications for a same target resource to multiple observers as responses over IP multicast.

For instance, in CoAP publish-subscribe [I-D.ietf-core-coap-pubsub], multiple clients can subscribe to a topic, by observing the related resource hosted at the responsible broker. When a new value is published on that topic, it would be convenient for the broker to send a single multicast notification at once, to all the subscriber clients observing that topic.

A different use case concerns clients observing a same registration resource at the CoRE Resource Directory [I-D.ietf-core-resource-directory]. For example, multiple clients can benefit of observation for discovering (to-be-created) OSCORE groups [I-D.ietf-core-oscore-groupcomm], by retrieving from the Resource Directory updated links and descriptions to join them through the respective Group Manager [I-D.tiloca-core-oscore-discovery].

More in general, multicast notifications would be beneficial whenever several CoAP clients observe a same target resource at a CoAP server, and can be all notified at once by means of a single response message. However, CoAP does not currently define response messages over IP multicast. This specification fills this gap and provides the following twofold contribution.

First, it updates [RFC7252] and [RFC7641], by defining a method to deliver Observe notifications as CoAP responses addressed to multiple clients, e.g. over IP multicast. In the proposed method, the group of potential observers entrusts the server to manage the Token space for multicast notifications. By doing so, the server provides all the observers of a target resource with the same Token value to bind to their own observation. That Token value is then used in every multicast notification for the target resource. This is achieved by means of an informative unicast response sent by the server to each observer client.

Second, this specification defines how to use Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect multicast notifications end-to-end between the server and the observer clients. This is also achieved by means of the informative unicast response mentioned above, which additionally includes parameter values used by the server to protect every multicast notification for the target

resource by using Group OSCORE. This provides a secure binding between each of such notifications and the observation of each of the clients.

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 terms and concepts described in CoAP [RFC7252], group communication for CoAP [I-D.ietf-core-groupcomm-bis], Observe [RFC7641], CBOR [RFC8949], OSCORE [RFC8613], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This specification additionally defines the following terminology.

- o Traditional observation. A resource observation associated to a single observer client, as defined in [RFC7641].
- o Group observation. A resource observation associated to a group of clients. The server sends notifications for the group-observed resource over IP multicast to all the observer clients.
- o Phantom request. The CoAP request message that the server would have received to start or cancel a group observation on one of its resources. A phantom request is generated inside the server and does not hit the wire.
- o Informative response. A CoAP response message that the server sends to a given client via unicast, providing the client with information on a group observation.

2. Server-Side Requirements

The server can, at any time, start a group observation on one of its resources. Practically, the server may want to do that under the following circumstances.

- o In the absence of observations for the target resource, the server receives a registration request from a first client wishing to start a traditional observation on that resource.
- o When a certain amount of traditional observations has been established on the target resource, the server decides to make those clients part of a group observation on that resource.

The server maintains an observer counter for each group observation to a target resource, as a rough estimation of the observers actively taking part in the group observation.

The server initializes the counter to 0 when starting the group observation, and increments it after a new client starts taking part in that group observation. Also, the server should keep the counter up-to-date over time, for instance by using the method described in Section 6.

This document does not describe a way for the client to influence the server's decision to start group observations. That is done on purpose: the specified mechanism is expected to be used in situations where sending individual notifications is not feasible, or not preferred beyond a certain number of clients observing a target resource. If applications arise where negotiation does make sense, they are welcome to specify additional means to opt in to multicast notifications.

2.1. Request

Assuming it is reachable at the address SRV_ADDR and port number SRV_PORT, the server starts a group observation on one of its resources as defined below. The server intends to send multicast notifications for the target resource to the multicast IP address GRP_ADDR and port number GRP_PORT.

1. The server builds a phantom observation request, i.e. a GET request with an Observe option set to 0 (register).
2. The server selects an available value T, from the Token space of a CoAP endpoint used for messages having:
 - * As source address and port number, the IP multicast address GRP_ADDR and port number GRP_PORT.
 - * As destination address and port number, the server address SRV_ADDR and port number SRV_PORT, intended for accessing the target resource.

This Token space is under exclusive control of the server.

3. The server processes the phantom observation request above, without transmitting it on the wire. The request is addressed to the resource for which the server wants to start the group observation, as if sent by the group of observers, i.e. with GRP_ADDR as source address and GRP_PORT as source port.

4. Upon processing the self-generated phantom registration request, the server interprets it as an observe registration received from the group of potential observer clients. In particular, from then on, the server **MUST** use T as its own local Token value associated to that observation, with respect to the (previous hop towards the) clients.
5. The server does not immediately respond to the phantom observation request with a multicast notification sent on the wire. The server stores the phantom observation request as is, throughout the lifetime of the group observation.
6. The server builds a CoAP response message INIT_NOTIF as initial multicast notification for the target resource, in response to the phantom observation request. This message is formatted as other multicast notifications (see Section 2.3) and **MUST** include the current representation of the target resource as payload. The server stores the message INIT_NOTIF and does not transmit it. The server considers this message as the latest multicast notification for the target resource, until it transmits a new multicast notification for that resource as a CoAP message on the wire. After that, the server deletes the message INIT_NOTIF.

2.2. Informative Response

After having started a group observation on a target resource, the server proceeds as follows.

For each traditional observation ongoing on the target resource, the server **MAY** cancel that observation. Then, the server considers the corresponding clients as now taking part in the group observation, for which it increases the corresponding observer counter accordingly.

The server sends to each of such clients an informative response message, encoded as a unicast response with response code 5.03 (Service Unavailable). As per [RFC7641], such a response does not include an Observe option. The response **MUST** be Confirmable and **MUST NOT** encode link-local addresses.

The Content-Format of the informative response is set to application/informative-response+cbor, defined in Section 14.2. The payload of the informative response is a CBOR map including the following parameters, whose CBOR labels are defined in Section 11.

- o 'tp_info', with value a CBOR array. This includes the transport-specific information required to correctly receive multicast notifications bound to the phantom observation request. The CBOR

array is formatted as defined in Section 2.2.1. This parameter MUST be included.

- o 'ph_req', with value the byte serialization of the transport-independent information of the phantom observation request (see Section 2.1), encoded as a CBOR byte string. The value of the CBOR byte string is formatted as defined in Section 2.2.2. This parameter MUST be included.
- o 'last_notif', with value the byte serialization of the transport-independent information of the latest multicast notification for the target resource, encoded as a CBOR byte string. The value of the CBOR byte string is formatted as defined in Section 2.2.2. This parameter MAY be included.

The CDDL notation [RFC8610] provided below describes the payload of the informative response.

```
informative_response_payload = {  
  1 => array, ; 'tp_info', i.e. transport-specific information  
  2 => bstr, ; 'ph_req' (transport-independent information)  
  ? 3 => bstr ; 'last_notif' (transport-independent information)  
}
```

Figure 1: Format of the informative response payload

Upon receiving a registration request to observe the target resource, the server does not create a corresponding individual observation for the requesting client. Instead, the server considers that client as now taking part in the group observation of the target resource, of which it increments the observer counter by 1. Then, the server replies to the client with the same informative response message defined above, which MUST be Confirmable.

Note that this also applies when, with no ongoing traditional observations on the target resource, the server receives a registration request from a first client and decides to start a group observation on the target resource.

2.2.1. Encoding of Transport-Specific Message Information

The CBOR array specified in the 'tp_info' parameter is formatted according to the following CDDL notation.

```
tp_info = [  
    srv_addr ; Addressing information of the server  
    ? req_info ; Request data extension  
]  
  
srv_addr = (  
    tp_id : int, ; Identifier of the used transport protocol  
    + elements ; Number, format and encoding  
                ; based on the value of 'tp_id'  
)  
  
req_info = (  
    + elements ; Number, format and encoding based on  
                ; the value of 'tp_id' in 'srv_addr'  
)
```

Figure 2: General format of 'tp_info'

The 'srv_addr' element of 'tp_info' specifies the addressing information of the server, and includes at least one element 'tp_id' which is formatted as follows.

- o 'tp_id' : this element is a CBOR integer, which specifies the transport protocol used to transport the CoAP response from the server, i.e. a multicast notification in this specification.

This element takes value from the "Value" column of the "CoAP Transport Information" registry defined in Section 14.4 of this specification. This element MUST be present. The value of this element determines:

- * How many elements are required to follow in 'srv_addr', as well as what information they convey, their encoding and their semantics.
- * How many elements are required in the 'req_info' element of the 'tp_info' array, as well as what information they convey, their encoding and their semantics.

This specification registers the integer value 1 ("UDP") to be used as value for the 'tp_id' element, when CoAP responses are transported over UDP. In such a case, the full encoding of the 'tp_info' CBOR array is as defined in Section 2.2.1.1.

Future specifications that consider CoAP multicast notifications transported over different transport protocols MUST:

- * Register an entry with an integer value to be used for 'tp_id', in the "CoAP Transport Information" registry defined in Section 14.4 of this specification.
- * Accordingly, define the elements of the 'tp_info' CBOR array, i.e. the elements following 'tp_id' in 'srv_addr' as well as the elements in 'req_info', as to what information they convey, their encoding and their semantics.

The 'req_info' element of 'tp_info' specifies transport-specific information related to a pertinent request message, i.e. the phantom observation request in this specification. The exact format of 'req_info' depends on the value of 'tp_id'.

Given a specific value of 'tp_id', the complete set of elements composing 'srv_addr' and 'req_info' in the 'tp_info' CBOR array is indicated by the two columns "Srv Addr" and "Req Info" of the "CoAP Transport Information" registry defined in Section 14.4, respectively.

2.2.1.1. UDP Transport-Specific Information

When CoAP multicast notifications are transported over UDP as per [RFC7252] and [I-D.ietf-core-groupcomm-bis], the server specifies the integer value 1 ("UDP") as value of 'tp_id' in the 'srv_addr' element of the 'tp_info' CBOR array in the error informative response. Then, the rest of the 'tp_info' CBOR array is defined as follows.

- o 'srv_addr' includes two more elements following 'tp_id':
 - * 'srv_host': this element is a CBOR byte string, with value the destination IP address of the phantom observation request. This parameter is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)". That is, the value of the CBOR byte string is the IP address SRV_ADDR of the server hosting the target resource, from where the server will send multicast notifications for the target resource. This element MUST be present.
 - * 'srv_port': this element is a CBOR unsigned integer, with value the destination port number of the phantom observation request. That is, the specified value is the port number SRV_PORT, from where the server will send multicast notifications for the target resource. This element MUST be present.
- o 'req_info' includes the following elements:

- * 'token': this element is a CBOR byte string, with value the Token value of the phantom observation request generated by the server (see Section 2.1). Note that the same Token value is used for the multicast notifications bound to that phantom observation request (see Section 2.3). This element MUST be present.
- * 'cli_addr': this element is a CBOR byte string, with value the source IP address of the phantom observation request. This parameter is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)". That is, the value of the CBOR byte string is the IP multicast address GRP_ADDR, where the server will send multicast notifications for the target resource. This element MUST be present.
- * 'cli_port': this element is a CBOR unsigned integer, with value the source port number of the phantom observation request. That is, the specified value is the port number GRP_PORT, where the server will send multicast notifications for the target resource. This element is OPTIONAL. If not included, the default port number 5683 is assumed.

The CDDL notation provided below describes the full 'tp_info' CBOR array using the format above.

```
tp_info = [
  tp_id      : 1,                ; UDP as transport protocol
  srv_host   : #6.260(bstr),     ; Src. address of multicast notifications
  srv_port   : uint,             ; Src. port of multicast notifications
  token      : bstr,             ; Token of the phantom request and
                                ; associated multicast notifications
  cli_addr   : #6.260(bstr),     ; Dst. address of multicast notifications
  ? cli_port : uint              ; Dst. port of multicast notifications
]
```

Figure 3: Format of 'tp_info' with UDP as transport protocol

2.2.2. Encoding of Transport-Independent Message Information

For both the parameters 'ph_req' and 'last_notif' in the informative response, the value of the byte string is the concatenation of the following components, in the order specified below.

When defining the value of each component, "CoAP message" refers to the phantom observation request for the 'ph_req' parameter, and to the corresponding latest multicast notification for the 'last_notif' parameter.

- o A single byte, with value the content of the Code field in the CoAP message.
- o The byte serialization of the complete sequence of CoAP options in the CoAP message.
- o If the CoAP message includes a non-zero length payload, the one-byte Payload Marker (0xff) followed by the payload.

2.3. Notifications

Upon a change in the status of the target resource under group observation, the server sends a multicast notification, intended to all the clients taking part in the group observation of that resource. In particular, each of such multicast notifications is formatted as follows.

- o It MUST be Non-confirmable.
- o It MUST include an Observe option, as per [RFC7641].
- o It MUST have the same Token value T of the phantom registration request that started the group observation. This Token value is specified in the 'token' element of 'req_info' under the 'tp_info' parameter, in the informative response message sent to all the observer clients.

That is, every multicast notification for a target resource is not bound to the observation requests from the different clients, but rather to the phantom registration request associated to the whole set of clients taking part in the group observation of that resource.

- o It MUST be sent from the same IP address SRV_ADDR and port number SRV_PORT where: i) the original Observe registration requests are sent to by the clients; and ii) the corresponding informative responses are sent from by the server (see Section 2.2). These are indicated to the observer clients as value of the 'srv_host' and 'srv_port' elements of 'srv_addr' under the 'tp_info' parameter, in the informative response message (see Section 2.2.1.1). That is, redirection MUST NOT be used.
- o It MUST be sent to the IP multicast address GRP_ADDR and port number GRP_PORT. These are indicated to the observer clients as value of the 'cli_addr' and 'cli_port' elements of 'req_info' under the 'tp_info' parameter, in the informative response message (see Section 2.2.1.1).

For each target resource with an active group observation, the server MUST store the latest multicast notification.

2.4. Congestion Control

In order to not cause congestion, the server should conservatively control the sending of multicast notifications. In particular:

- o The multicast notifications MUST be Non-confirmable.
- o In constrained environments such as low-power, lossy networks (LLNs), the server should only support multicast notifications for resources that are small. Following related guidelines from Section 2.2.4 of [I-D.ietf-core-groupcomm-bis], this can consist, for example, in having the payload of multicast notifications as limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size, so that it fits into a single link-layer frame in case IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) (see Section 4 of [RFC4944]) is used.
- o The server SHOULD provide multicast notifications with the smallest possible IP multicast scope that fulfills the application needs. For example, following related guidelines from Section 2.2.4 of [I-D.ietf-core-groupcomm-bis], site-local scope is always preferred over global scope IP multicast, if this fulfills the application needs. Similarly, realm-local scope is always preferred over site-local scope, if this fulfills the application needs.
- o Following related guidelines from Section 4.5.1 of [RFC7641], the server SHOULD NOT send more than one multicast notification every 3 seconds, and SHOULD use an even less aggressive rate when possible (see also Section 3.1.2 of [RFC8085]). The transmission rate of multicast notifications should also take into account the avoidance of a possible "broadcast storm" problem [MOBICOM99]. This prevents a following, considerable increase of the channel load, whose origin would be likely attributed to a router rather than the server.

2.5. Cancellation

At any point in time, the server may want to cancel a group observation of a target resource. For instance, the server may realize that no clients or not enough clients are interested in taking part in the group observation anymore. A possible approach that the server can use to assess this is defined in Section 6.

In order to cancel the group observation, the server sends to itself a phantom cancellation request, i.e. a GET request with an Observe option set to 1 (deregister), without transmitting it on the wire. As per Section 3.6 of [RFC7641], all other options MUST be identical to those in the phantom registration request, except for the set of ETag Options. This request has the same Token value T of the phantom registration request, and is addressed to the resource for which the server wants to end the group observation, as if sent by the group of observers, i.e. with the multicast IP address GRP_ADDR as source address and the port number GRP_PORT as source port.

After that, the server sends a multicast response with response code 5.03 (Service Unavailable), signaling that the group observation has been terminated. The response has no payload, and is sent to the same multicast IP address GRP_ADDR and port number GRP_PORT used to send the multicast notifications related to the target resource. As per [RFC7641], this response does not include an Observe option. Finally, the server releases the resources allocated for the group observation, and especially frees up the Token value T used at its CoAP endpoint.

3. Client-Side Requirements

3.1. Request

A client sends an observation request to the server as described in [RFC7641], i.e. a GET request with an Observe option set to 0 (register). The request MUST NOT encode link-local addresses. If the server is not configured to accept registrations on that target resource with a group observation, this would still result in a positive notification response to the client as described in [RFC7641].

3.2. Informative Response

Upon receiving the informative response defined in Section 2.2, the client proceeds as follows.

1. The client configures an observation of the target resource. To this end, it relies on a CoAP endpoint used for messages having:
 - * As source address and port number, the server address SRV_ADDR and port number SRV_PORT intended for accessing the target resource. These are specified as value of the 'srv_host' and 'srv_port' elements of 'srv_addr' under the 'tp_info' parameter, in the informative response (see Section 2.2.1.1).

- * As destination address and port number, the IP multicast address GRP_ADDR and port number GRP_PORT. These are specified as value of the 'cli_addr' and 'cli_port' elements of 'req_info' under the 'tp_info' parameter, in the informative response (see Section 2.2.1.1). If the 'cli_port' element is omitted in 'req_info', the client MUST assume the default port number 5683 as GRP_PORT.
2. The client rebuilds the phantom registration request, by using:
 - * The transport-independent information, specified in the 'ph_req' parameter of the informative response.
 - * The Token value T, specified in the 'token' element of 'req_info' under the 'tp_info' parameter of the informative response.
 3. The client stores the phantom registration request, as associated to the observation of the target resource. In particular, the client MUST use the Token value T of this phantom registration request as its own local Token value associated to that group observation, with respect to the server. The particular way to achieve this is implementation specific.
 4. If the informative response includes the parameter 'last_notif', the client rebuilds the latest multicast notification, by using:
 - * The transport-independent information, specified in the 'last_notif' parameter of the informative response.
 - * The Token value T, specified in the 'token' element of 'req_info' under the 'tp_info' parameter of the informative response.
 5. If the informative response includes the parameter 'last_notif', the client processes the multicast notification rebuilt in step 4 as defined in Section 3.2 of [RFC7641]. In particular, the value of the Observe option is used as initial baseline for notification reordering in this group observation.
 6. If a traditional observation to the target resource is ongoing, the client MAY silently cancel it without notifying the server.

If any of the expected fields in the informative response are not present or malformed, the client MAY try sending a new registration request to the server (see Section 3.1). Otherwise, the client SHOULD explicitly withdraw from the group observation.

Appendix A describes possible alternative ways for clients to retrieve the phantom registration request and other information related to a group observation.

3.3. Notifications

After having successfully processed the informative response as defined in Section 3.2, the client will receive, accept and process multicast notifications about the state of the target resource from the server, as responses to the phantom registration request and with Token value T.

The client relies on the value of the Observe option for notification reordering, as defined in Section 3.4 of [RFC7641].

3.4. Cancellation

At a certain point in time, a client may become not interested in receiving further multicast notifications about a target resource. When this happens, the client can simply "forget" about being part of the group observation for that target resource, as per Section 3.6 of [RFC7641].

When, later on, the server sends the next multicast notification, the client will not recognize the Token value T in the message. Since the multicast notification is Non-confirmable, it is OPTIONAL for the client to reject the multicast notification with a Reset message, as defined in Section 3.5 of [RFC7641].

In case the server has canceled a group observation as defined in Section 2.5, the client simply forgets about the group observation and frees up the used Token value T for that endpoint, upon receiving the multicast error response defined in Section 2.5.

4. Web Linking

The possible use of multicast notifications in a group observation may be indicated by a target "grp_obs" attribute in a web link [RFC8288] to a resource, e.g. using a link-format document [RFC6690] if the resource is accessible over CoAP.

The "grp_obs" attribute is a hint, indicating that the server might send multicast notifications for observations of the resource targeted by the link. Note that this is simply a hint, i.e. it does not include any information required to participate in a group observation, and to receive and process multicast notifications.

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

The example in Figure 4 shows a use of the "grp_obs" attribute: the client does resource discovery on a server and gets back a list of resources, one of which includes the "grp_obs" attribute indicating that the server might send multicast notifications for observations of that resource. The link-format notation (see Section 5 of [RFC6690]) is used.

```
REQ: GET /.well-known/core
```

```
RES: 2.05 Content
      </sensors/temp>;grp_obs,
      </sensors/light>;if="sensor"
```

Figure 4: The Web Link

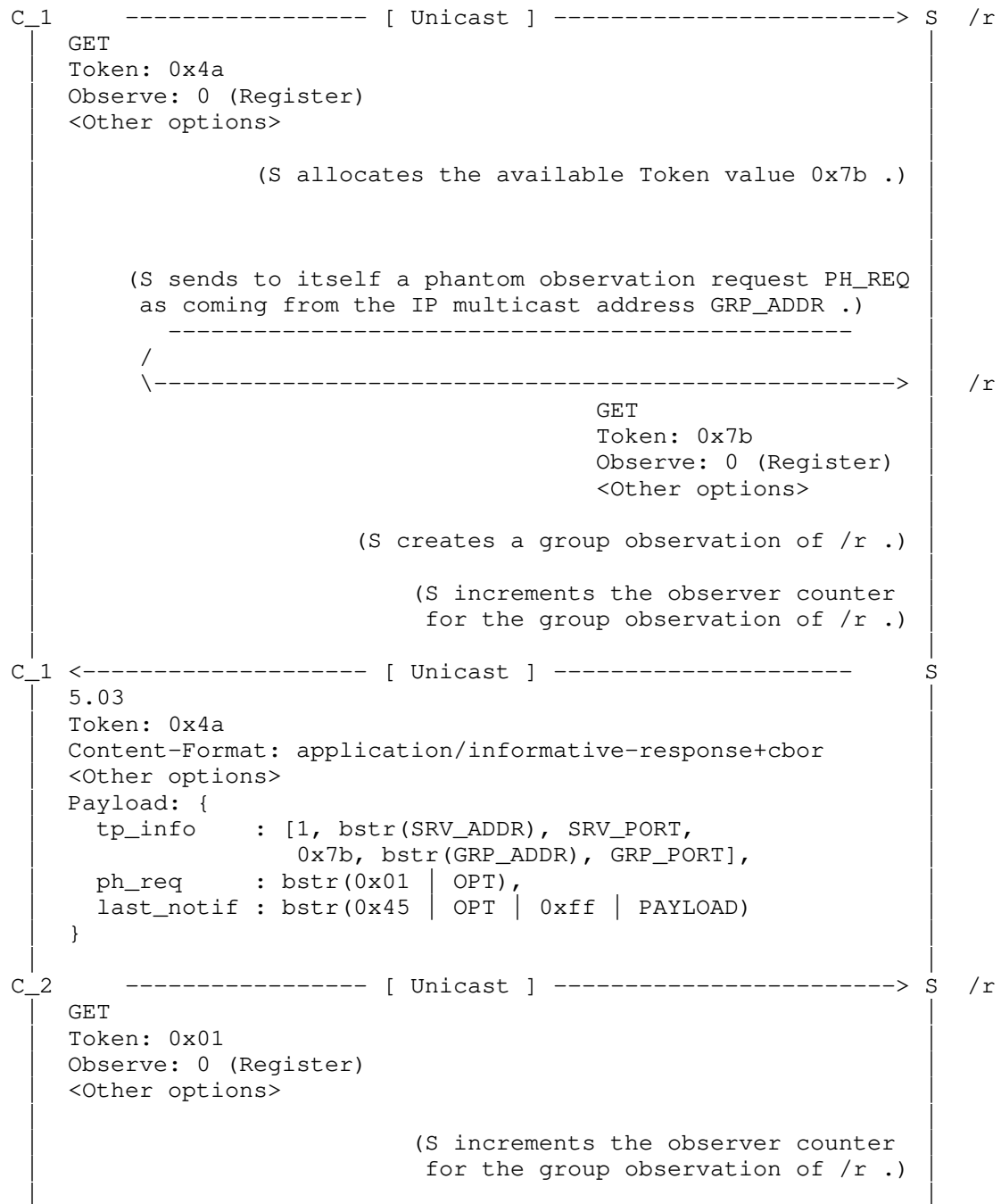
5. Example

The following example refers to two clients C_1 and C_2 that register to observe a resource /r at a Server S, which has address SRV_ADDR and listens to the port number SRV_PORT. Before the following exchanges occur, no clients are observing the resource /r, which has value "1234".

The server S sends multicast notifications to the IP multicast address GRP_ADDR and port number GRP_PORT, and starts the group observation upon receiving a registration request from a first client that wishes to start a traditional observation on the resource /r.

The following notation is used for the payload of the informative responses:

- o 'bstr(X)' denotes a CBOR byte string with value the byte serialization of X, with '|' denoting byte concatenation.
- o 'OPT' denotes a sequence of CoAP options. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.
- o 'PAYLOAD' denotes a CoAP payload. This refers to the latest multicast notification encoded by the 'last_notif' parameter.



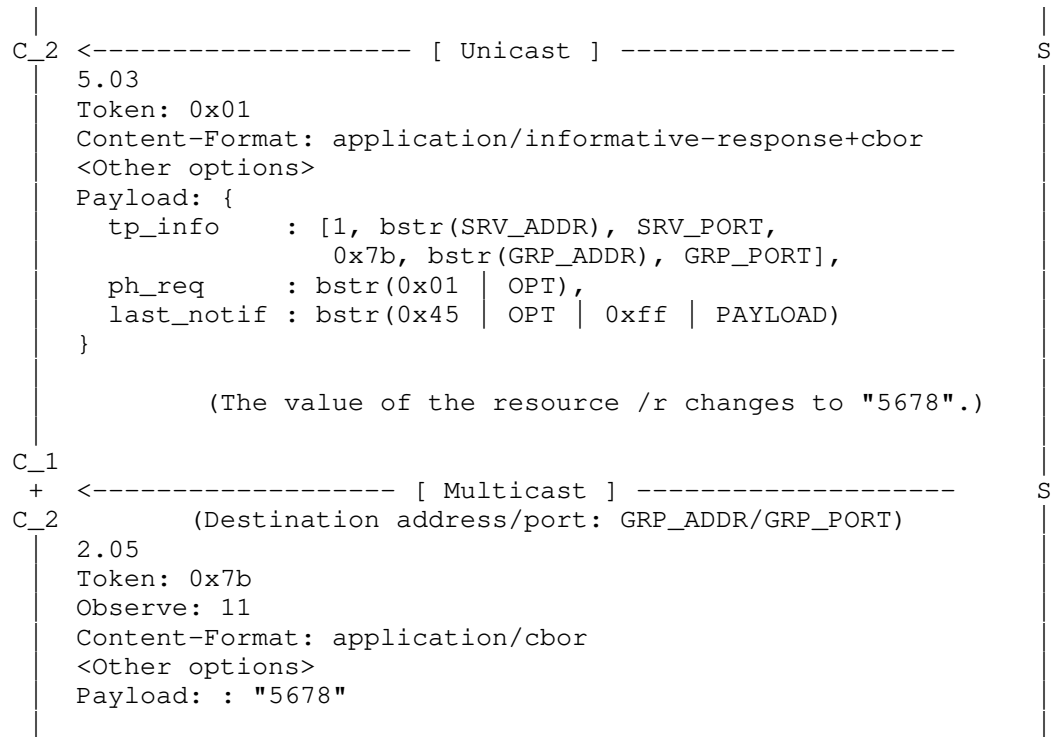


Figure 5: Example of group observation

6. Rough Counting of Clients in the Group Observation

To allow the server to keep an estimate of interested clients without creating undue traffic on the network, a new CoAP option is introduced, which SHOULD be supported by clients that listen to multicast responses.

The option is called Multicast-Response-Feedback-Divider. As summarized in Figure 6, the option is not critical but proxy-unsafe, and integer valued.

No.	C	U	N	R	Name	Format	Len.	Default
TBD		x			Multicast-Response-Feedback-Divider	uint	0-1	(none)

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

Figure 6: Multicast-Response-Feedback-Divider

The Multicast-Response-Feedback-Divider option is of class E for OSCORE [RFC8613] [I-D.ietf-core-oscore-groupcomm].

6.1. Processing on the Client Side

Upon receiving a response with a Multicast-Response-Feedback-Divider option, a client SHOULD acknowledge its interest in continuing receiving multicast notifications for the target resource, as described below.

The client picks an integer random number I , from 0 inclusive to the number $Z = (2 ** Q)$ exclusive, where Q is the value specified in the option and $**$ is the exponentiation operator. If I is different than 0, the client takes no further action. Otherwise, the client should wait a random fraction of the Leisure time (see Section 8.2 of [RFC7252]), and then registers a regular unicast observation on the same target resource.

To this end, the client essentially follows the steps that got it originally subscribed to group notifications for the target resource. In particular, the client sends an observation request to the server, i.e. a GET request with an Observe option set to 0 (register). The request MUST be addressed to the same target resource, and MUST have the same destination IP address and port number used for the original registration request, regardless the source IP address and port number of the received multicast notification.

Since the observation registration is only done for its side effect of showing as an attempted observation at the server, the client MUST send the unicast request in a non confirmable way, and with the maximum No-Response setting [RFC7967]. In the request, the client MUST include a Multicast-Response-Feedback-Divider option, whose value MUST be empty (Option Length = 0). The client does not need to wait for responses, and can keep processing further notifications on the same token.

The client MUST ignore the Multicast-Response-Feedback-Divider option, if the multicast notification is retrieved from the 'last_notif' parameter of an informative response (see Section 2.2). A client includes the Multicast-Response-Feedback-Divider option only in a re-registration request triggered by the server as described above, and MUST NOT include it in any other request.

As the Multicast-Response-Feedback-Divider option is unsafe to forward, a proxy needs to answer it on its own, and is later counted as a single client.

Appendix B.1 provides a description in pseudo-code of the operations above performed by the client.

6.2. Processing on the Server Side

In order to avoid needless use of network resources, a server SHOULD keep a rough, updated count of the number of clients taking part in the group observation of a target resource. To this end, the server updates the value COUNT of the associated observer counter (see Section 2), for instance by using the method described below.

6.2.1. Request for Feedback

When it wants to obtain a new estimated count, the server considers a number M of confirmations it would like to receive from the clients. It is up to applications to define policies about how the server determines and possibly adjusts the value of M .

Then, the server computes the value $Q = \max(L, 0)$, where:

- o L is computed as $L = \text{ceil}(\log_2(N / M))$.
- o N is the current value of the observer counter, possibly rounded up to 1, i.e. $N = \max(\text{COUNT}, 1)$.

Finally, the server sets Q as the value of the Multicast-Response-Feedback-Divider option, which is sent within a successful multicast notification.

If several multicast notifications are sent in a burst fashion, it is RECOMMENDED for the server to include the Multicast-Response-Feedback-Divider option only in the first one of those notifications.

6.2.2. Collection of Feedback

The server collects unicast observation requests from the clients, for an amount of time of `MAX_CONFIRMATION_WAIT` seconds. During this time, the server regularly increments the observer counter when adding a new client to the group observation (see Section 2.2).

It is up to applications to define the value of `MAX_CONFIRMATION_WAIT`, which has to take into account the transmission time of the multicast notification and of unicast observation requests, as well as the leisure time of the clients, which may be hard to know or estimate for the server.

If this information is not known to the server, it is recommended to define `MAX_CONFIRMATION_WAIT` as follows.

$$\text{MAX_CONFIRMATION_WAIT} = \text{MAX_RTT} + \text{MAX_CLIENT_REQUEST_DELAY}$$

where `MAX_RTT` is as defined in Section 4.8.2 of [RFC7252] and has default value 202 seconds, while `MAX_CLIENT_REQUEST_DELAY` is equivalent to `MAX_SERVER_RESPONSE_DELAY` defined in Section 2.3.1 of [I-D.ietf-core-groupcomm-bis] and has default value 250 seconds. In the absence of more specific information, the server can thus consider a conservative `MAX_CONFIRMATION_WAIT` of 452 seconds.

If more information is available in deployments, a much shorter `MAX_CONFIRMATION_WAIT` can be set. This can be based on a realistic round trip time (replacing `MAX_RTT`) and on the largest leisure time configured on the clients (replacing `MAX_CLIENT_REQUEST_DELAY`), e.g. `DEFAULT_LEISURE` = 5 seconds, thus shortening `MAX_CONFIRMATION_WAIT` to a few seconds.

6.2.3. Processing of Feedback

Once `MAX_CONFIRMATION_WAIT` seconds have passed, the server counts the `R` confirmations arrived as unicast observation requests to the target resource, since the multicast notification with the Multicast-Response-Feedback-Divider option has been sent. In particular, the server considers a unicast observation request as a confirmation from a client only if it includes a Multicast-Response-Feedback-Divider option with an empty value (Option Length = 0).

Then, the server computes a feedback indicator as $E = R * (2 ** Q)$, where `**` is the exponentiation operator. According to what defined by application policies, the server determines the next time when to ask clients for their confirmation, e.g. after a certain number of multicast notifications has been sent. For example, the decision can

be influenced by the reception of no confirmations from the clients, i.e. $R = 0$, or by the value of the ratios (E/N) and (N/E) .

Finally, the server computes a new estimated count of the observers. To this end the server first consider $COUNT'$ as the current value of the observer counter at this point in time. Note that $COUNT'$ may be greater than the value $COUNT$ used at the beginning of this process, if the server has incremented the observer counter upon adding new clients to the group observation (see Section 2.2).

In particular, the server computes the new estimated count value as $COUNT' + ((E - N) / D)$, where $D > 0$ is an integer value used as dampener. This step has to be performed atomically. That is, until this step is completed, the server MUST hold the processing of an observation request for the same target resource from a new client. Finally, the server considers the result as the current observer counter, and assesses it for possibly canceling the group observation (see Section 2.5).

This estimate is skewed by packet loss, but it gives the server a sufficiently good estimation for further counts and for deciding when to cancel the group observation. It is up to applications to define policies about how the server takes the newly updated estimate into account and determines whether to cancel the group observation.

As an example, if the server currently estimates that $N = COUNT = 32$ observers are active and considers a constant $M = 8$, it sends out a notification with Multicast-Response-Feedback-Divider: 2. Then, out of 18 actually active clients, 5 send a re-registration request based on their random draw, of which one request gets lost, thus leaving 4 re-registration requests received by the server. Also, no new clients have been added to the group observation during this time, i.e. $COUNT'$ is equal to $COUNT$. As a consequence, assuming that a dampener value $D = 1$ is used, the server computes the new estimated count value as $32 + (16 - 32) = 16$, and keeps the group observation active.

To produce a most accurate updated counter, a server can include a Multicast-Response-Feedback-Divider option with value $Q = 0$ in its multicast notifications, as if M is equal to N . This will trigger all the active clients to state their interest in continuing receiving notifications for the target resource. Thus, the amount R of arrived confirmations is affected only by possible packet loss.

Appendix B.3 provides a description in pseudo-code of the operations above performed by the server, including example behaviors for scheduling the next count update and deciding whether to cancel the group observation.

7. Protection of Multicast Notifications with Group OSCORE

A server can protect multicast notifications by using Group OSCORE [I-D.ietf-core-oscore-groupcomm], thus ensuring they are protected end-to-end with the observer clients. This requires that both the server and the clients interested in receiving multicast notifications from that server are members of the same OSCORE group.

In some settings, the OSCORE group to refer to can be pre-configured on the clients and the server. In such a case, a server which is aware of such pre-configuration can simply assume a client to be already member of the correct OSCORE group.

In any other case, the server MAY communicate to clients what OSCORE group they are required to join, by providing additional guidance in the informative response as described in Section 7.1. Note that clients can already be members of the right OSCORE group, in case they have previously joined it to securely communicate with the same and/or other servers to access their resources.

Both the clients and the server MAY join the OSCORE group by using the approach described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz]. Further details on how to discover the OSCORE group and join it are out of the scope of this specification.

If multicast notifications are protected using Group OSCORE, the original registration requests and related unicast (notification) responses MUST also be secured, including and especially the informative responses from the server.

To this end, alternative security protocols than Group OSCORE, such as OSCORE [RFC8613] and/or DTLS [RFC6347][I-D.ietf-tls-dtls13], can be used to protect other exchanges via unicast between the server and each client, including the original client registration (see Section 3).

7.1. Signaling the OSCORE Group in the Informative Response

This section describes a mechanism for the server to communicate to the client what OSCORE group to join in order to decrypt and verify the multicast notifications protected with group OSCORE. The client MAY use the information provided by the server to start the ACE joining procedure described in [I-D.ietf-ace-key-groupcomm-oscore]. This mechanism is OPTIONAL to support for the client and server.

Additionally to what defined in Section 2, the CBOR map in the informative response payload contains the following fields, whose CBOR labels are defined in Section 11.

- o 'join_uri', with value the URI for joining the OSCORE group at the respective Group Manager, encoded as a CBOR text string. If the procedure described in [I-D.ietf-ace-key-groupcomm-oscore] is used for joining, this field specifically indicates the URI of the group-membership resource at the Group Manager.
- o 'sec_gp', with value the name of the OSCORE group, encoded as a CBOR text string.
- o Optionally, 'as_uri', with value the URI of the Authorization Server associated to the Group Manager for the OSCORE group, encoded as a CBOR text string.
- o Optionally, 'cs_alg', with value the COSE algorithm [I-D.ietf-cose-rfc8152bis-algs] used to countersign messages in the OSCORE group, encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Algorithms" registry [COSE.Algorithms].
- o Optionally, 'cs_alg_crv', with value the elliptic curve (if applicable) for the COSE algorithm [I-D.ietf-cose-rfc8152bis-algs] used to countersign messages in the OSCORE group, encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Elliptic Curve" registry [COSE.Elliptic.Curves].
- o Optionally, 'cs_key_kty', with value the COSE key type [I-D.ietf-cose-rfc8152bis-struct] of countersignature keys used to countersign messages in the OSCORE group, encoded as a CBOR text string or an integer. The value is taken from the 'Value' column of the "COSE Key Types" registry [COSE.Key.Types].
- o Optionally, 'cs_key_crv', with value the elliptic curve (if applicable) of countersignature keys used to countersign messages in the OSCORE group, encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Elliptic Curve" registry [COSE.Elliptic.Curves].
- o Optionally, 'cs_kenc', with value the encoding of the public keys used in the OSCORE group, encoded as a CBOR integer. The value is taken from the 'Confirmation Key' column of the "CWT Confirmation Method" registry defined in [RFC8747]. Future specifications may define additional values for this parameter.

- o Optionally, 'alg', with value the COSE AEAD algorithm [I-D.ietf-cose-rfc8152bis-algs], encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Algorithms" registry [COSE.Algorithms].
- o Optionally, 'hkdf', with value the COSE HKDF algorithm [I-D.ietf-cose-rfc8152bis-algs], encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Algorithms" registry [COSE.Algorithms].

The values of 'cs_alg', 'cs_alg_crv', 'cs_key_kty', 'cs_key_crv' and 'cs_key_kenc' provide an early knowledge of the format and encoding of public keys used in the OSCORE group. Thus, the client does not need to ask the Group Manager for this information as a preliminary step before the (ACE) join process, or to perform a trial-and-error exchange with the Group Manager upon joining the group. Hence, the client is able to provide the Group Manager with its own public key in the correct expected format and encoding, at the very first step of the (ACE) join process.

The values of 'cs_alg', 'alg' and 'hkdf' provide an early knowledge of the algorithms used in the OSCORE group. Thus, the client is able to decide whether to actually proceed with the (ACE) join process, depending on its support for the indicated algorithms.

As mentioned above, since this mechanism is OPTIONAL, all the fields are OPTIONAL in the informative response. However, the 'join_uri' and 'sec_gp' fields MUST be present if the mechanism is implemented and used. If any of the fields are present without the 'join_uri' and 'sec_gp' fields present, the client MUST ignore these fields, since they would not be sufficient to start the (ACE) join procedure. When this happens, the client MAY try sending a new registration request to the server (see Section 3.1). Otherwise, the client SHOULD explicitly withdraw from the group observation.

Appendix C describes a possible alternative approach, where the server self-manages the OSCORE group, and provides the observer clients with the necessary keying material in the informative response. The approach in Appendix C MUST NOT be used together with the mechanism defined in this section for indicating what OSCORE group to join.

7.2. Server-Side Requirements

When using Group OSCORE to protect multicast notifications, the server performs the operations described in Section 2, with the following differences.

7.2.1. Registration

The phantom registration request MUST be secured, by using Group OSCORE. In particular, the group mode of Group OSCORE defined in Section 8 of [I-D.ietf-core-oscore-groupcomm] MUST be used.

The server protects the phantom registration request as defined in Section 8.1 of [I-D.ietf-core-oscore-groupcomm], as if it was the actual sender, i.e. by using its Sender Context. As a consequence, the server consumes the current value of its Sender Sequence Number SN in the OSCORE group, and hence updates it to $SN^* = (SN + 1)$. Consistently, the OSCORE option in the phantom registration request includes:

- o As 'kid', the Sender ID of the server in the OSCORE group.
- o As 'piv', the previously consumed Sender Sequence Number value SN of the server in the OSCORE group, i.e. $(SN^* - 1)$.

7.2.2. Informative Response

The value of the CBOR byte string in the 'ph_req' parameter encodes the phantom observation request as a message protected with Group OSCORE (see Section 7.2.1). As a consequence: the specified Code is always 0.05 (FETCH); the sequence of CoAP options will be limited to the outer, non encrypted options; a payload is always present, as the authenticated ciphertext followed by the counter signature.

Similarly, the value of the CBOR byte string in the 'last_notif' parameter encodes the latest multicast notification as a message protected with Group OSCORE (see Section 7.2.3). This applies also to the initial multicast notification INIT_NOTIF built in step 6 of Section 2.1.

Optionally, the informative response includes information on the OSCORE group to join, as additional parameters (see Section 7.1).

7.2.3. Notifications

The server MUST protect every multicast notification for the target resource with Group OSCORE. In particular, the group mode of Group OSCORE defined in Section 8 of [I-D.ietf-core-oscore-groupcomm] MUST be used.

The process described in Section 8.3 of [I-D.ietf-core-oscore-groupcomm] applies, with the following additions when building the two OSCORE 'external_aad' to encrypt and

countersign the multicast notification (see Sections 4.3.1 and 4.3.2 of [I-D.ietf-core-oscure-groupcomm]).

- o The 'request_kid' is the 'kid' value in the OSCORE option of the phantom registration request, i.e. the Sender ID of the server.
- o The 'request_piv' is the 'piv' value in the OSCORE option of the phantom registration request, i.e. the consumed Sender Sequence Number SN of the server.
- o The 'request_kid_context' is the 'kid context' value in the OSCORE option of the phantom registration request, i.e. the Group Identifier value (Gid) of the OSCORE group used as ID Context.

Note that these same values are used to protect each and every multicast notification sent for the target resource under this group observation.

7.2.4. Cancellation

When canceling a group observation (see Section 2.5), the phantom cancellation request MUST be secured, by using Group OSCORE. In particular, the group mode of Group OSCORE defined in Section 8 of [I-D.ietf-core-oscure-groupcomm] MUST be used.

Like defined in Section 7.2.1 for the phantom registration request, the server protects the phantom cancellation request as per Section 8.1 of [I-D.ietf-core-oscure-groupcomm], by using its Sender Context and consuming its current Sender Sequence number in the OSCORE group, from its Sender Context. The following, corresponding multicast error response defined in Section 2.5 is also protected with Group OSCORE, as per Section 8.3 of [I-D.ietf-core-oscure-groupcomm].

Note that, differently from the multicast notifications, this multicast error response will be the only one securely paired with the phantom cancellation request.

7.3. Client-Side Requirements

When using Group OSCORE to protect multicast notifications, the client performs as described in Section 3, with the following differences.

7.3.1. Informative Response

Upon receiving the informative response from the server, the client performs as described in Section 3.2, with the following additions.

Once completed step 2, the client decrypts and verifies the rebuilt phantom registration request as defined in Section 8.2 of [I-D.ietf-core-oscore-groupcomm], with the following differences.

- o The client MUST NOT perform any replay check. That is, the client skips step 3 in Section 8.2 of [RFC8613].
- o If decryption and verification of the phantom registration request succeed:
 - * The client MUST NOT update the Replay Window in the Recipient Context associated to the server. That is, the client skips the second bullet of step 6 in Section 8.2 of [RFC8613].
 - * The client MUST NOT take any further process as normally expected according to [RFC7252]. That is, the client skips step 8 in Section 8.2 of [RFC8613]. In particular, the client MUST NOT deliver the phantom registration request to the application, and MUST NOT take any action in the Token space of its unicast endpoint, where the informative response has been received.
 - * The client stores the values of the 'kid', 'piv' and 'kid context' fields from the OSCORE option of the phantom registration request.
- o If decryption and verification of the phantom registration request fail, the client MAY try sending a new registration request to the server (see Section 3.1). Otherwise, the client SHOULD explicitly withdraw from the group observation.
- o If the informative response includes the parameter 'last_notif', the client also decrypts and verifies the latest multicast notification rebuilt in step 4, just like it would for the multicast notifications transmitted as CoAP messages on the wire (see Section 7.3.2). The client proceeds with step 5 if decryption and verification of the latest multicast notification succeed, or to step 6 otherwise.

7.3.2. Notifications

After having successfully processed the informative response as defined in Section 7.3.1, the client will decrypt and verify every multicast notification for the target resource as defined in Section 8.4 of [I-D.ietf-core-oscore-groupcomm], with the following difference.

The client MUST set the two 'external_aad' defined in Sections 4.3.1 and 4.3.2 of [I-D.ietf-core-oscore-groupcomm] as follows. The particular way to achieve this is implementation specific.

- o 'request_kid' takes the value of the 'kid' field from the OSCORE option of the phantom registration request (see Section 7.3.1).
- o 'request_piv' takes the value of the 'piv' field from the OSCORE option of the phantom registration request (see Section 7.3.1).
- o 'request_kid_context' takes the value of the 'kid context' field from the OSCORE option of the phantom registration request (see Section 7.3.1).

Note that these same values are used to decrypt and verify each and every multicast notification received for the target resource.

The replay protection and checking of multicast notifications is performed as specified in Section 4.1.3.5.2 of [RFC8613].

8. Example with Group OSCORE

The following example refers to two clients C_1 and C_2 that register to observe a resource /r at a Server S, which has address SRV_ADDR and listens to the port number SRV_PORT. Before the following exchanges occur, no clients are observing the resource /r, which has value "1234".

The server S sends multicast notifications to the IP multicast address GRP_ADDR and port number GRP_PORT, and starts the group observation upon receiving a registration request from a first client that wishes to start a traditional observation on the resource /r.

Pairwise communication over unicast is protected with OSCORE, while S protects multicast notifications with Group OSCORE. Specifically:

- o C_1 and S have a pairwise OSCORE Security Context. In particular, C_1 has 'kid' = 1 as Sender ID, and SN_1 = 101 as Sender Sequence Number. Also, S has 'kid' = 3 as Sender ID, and SN_3 = 301 as Sender Sequence Number.

- o C_2 and S have a pairwise OSCORE Security Context. In particular, C_2 has 'kid' = 2 as Sender ID, and SN_2 = 201 as Sender Sequence Number. Also, S has 'kid' = 4 as Sender ID, and SN_4 = 401 as Sender Sequence Number.
- o S is a member of the OSCORE group with name "myGroup", and 'kid context' = 0x57ab2e as Group ID. In the OSCORE group, S has 'kid' = 5 as Sender ID, and SN_5 = 501 as Sender Sequence Number.

The following notation is used for the payload of the informative responses:

- o 'bstr(X)' denotes a CBOR byte string with value the byte serialization of X, with '|' denoting byte concatenation.
- o 'OPT' denotes a sequence of CoAP options. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.
- o 'PAYLOAD' denotes an encrypted CoAP payload. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.
- o 'SIGN' denotes the counter signature appended to an encrypted CoAP payload. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.

```

C_1      ----- [ Unicast w/ OSCORE ] -----> S  /r
|
| 0.05 (FETCH)
| Token: 0x4a
| OSCORE: {kid: 1 ; piv: 101 ; ...}
| <Other class U/I options>
| 0xff
| Encrypted_payload {
|   0x01 (GET),
|   Observe: 0 (Register),
|   <Other class E options>
| }
|
|                                     (S allocates the available Token value 0x7b .)
|

```

```

        (S sends to itself a phantom observation request PH_REQ
         as coming from the IP multicast address GRP_ADDR .)
    -----
    /
    \-----> /r

        0.05 (FETCH)
        Token: 0x7b
        OSCORE: {kid: 5 ; piv: 501 ;
                  kid context: 57ab2e; ...}
        <Other class U/I options>
        0xff
        Encrypted_payload {
            0x01 (GET),
            Observe: 0 (Register),
            <Other class E options>
        }
        <Counter signature>

        (S steps SN_5 in the Group OSCORE Sec. Ctx : SN_5 <== 502)

        (S creates a group observation of /r .)

        (S increments the observer counter
         for the group observation of /r .)

C_1 <----- [ Unicast w/ OSCORE ] ----- S
2.05 (Content)
Token: 0x4a
OSCORE: {piv: 301; ...}
<Other class U/I options>
0xff
Encrypted_payload {
    5.03 (Service Unavailable),
    Content-Format: application/informative-response+cbor,
    <Other class E options>,
    0xff,
    CBOR_payload {
        tp_info      : [1, bstr(SRV_ADDR), SRV_PORT,
                        0x7b, bstr(GRP_ADDR), GRP_PORT],
        ph_req       : bstr(0x05 | OPT | 0xff | PAYLOAD | SIGN),
        last_notif   : bstr(0x45 | OPT | 0xff | PAYLOAD | SIGN),
        join_uri     : "coap://myGM/ace-group/myGroup",
        sec_gp       : "myGroup"
    }
}

```

```

C_2      ----- [ Unicast w/ OSCORE ] -----> S /r
0.05 (FETCH)
Token: 0x01
OSCORE: {kid: 2 ; piv: 201 ; ...}
<Other class U/I options>
0xff
Encrypted_payload {
    0x01 (GET),
    Observe: 0 (Register),
    <Other class E options>
}

(S increments the observer counter
for the group observation of /r .)

C_2 <----- [ Unicast w/ OSCORE ] ----- S
2.05 (Content)
Token: 0x01
OSCORE: {piv: 401; ...}
<Other class U/I options>
0xff,
Encrypted_payload {
    5.03 (Service Unavailable),
    Content-Format: application/informative-response+cbor,
    <Other class E options>,
    0xff,
    CBOR_payload {
        tp_info      : [1, bstr(SRV_ADDR), SRV_PORT,
                        0x7b, bstr(GRP_ADDR), GRP_PORT],
        ph_req       : bstr(0x05 | OPT | 0xff | PAYLOAD | SIGN),
        last_notif   : bstr(0x45 | OPT | 0xff | PAYLOAD | SIGN),
        join_uri     : "coap://myGM/ace-group/myGroup",
        sec_gp       : "myGroup"
    }
}

(The value of the resource /r changes to "5678".)

C_1
+ <----- [ Multicast w/ Group OSCORE ] ----- S
C_2      (Destination address/port: GRP_ADDR/GRP_PORT)
2.05 (Content)
Token: 0x7b
OSCORE: {kid: 5; piv: 502 ;
        kid context: 57ab2e; ...}

```

```
<Other class U/I options>
0xff
Encrypted_payload {
  2.05 (Content),
  Observe: 11,
  Content-Format: application/cbor,
  <Other class E options>,
  0xff,
  CBOR_Payload : "5678"
}
<Counter signature>
```

Figure 7: Example of group observation with Group OSCORE

The two external_aad used to encrypt and countersign the multicast notification above have 'request_kid' = 5, 'request_piv' = 501 and 'request_kid_context' = 0x57ab2e. These values are specified in the 'kid', 'piv' and 'kid context' field of the OSCORE option of the phantom observation request, which is encoded in the 'ph_req' parameter of the unicast informative response to the two clients. Thus, the two clients can build the two same external_aad for decrypting and verifying this multicast notification and the following ones.

9. Intermediaries

This section specifies how the approach presented in Section 2 and Section 3 works when a proxy is used between the clients and the server. In addition to what specified in Section 5.7 of [RFC7252] and Section 5 of [RFC7641], the following applies.

A client sends its original observation request to the proxy. If the proxy is not already registered at the server for that target resource, the proxy forwards the observation request to the server, hence registering itself as an observer. If the server has an ongoing group observation for the target resource or decides to start one, the server considers the proxy as taking part in the group observation, and replies to the proxy with an informative response.

Upon receiving an informative response, the proxy performs as specified for the client in Section 3, with the peculiarity that "consuming" the last notification (if present) means populating its cache.

In particular, by using the information retrieved from the informative response, the proxy configures an observation of the

target resource at the origin server, acting as a client directly taking part in the group observation.

As a consequence, the proxy will listen to the IP multicast address and port number indicated by the server in the informative response, as 'cli_addr' and 'cli_port' element of 'req_info' under the 'tp_info' parameter, respectively (see Section 2.2.1.1). Furthermore, multicast notifications will match the phantom request stored at the proxy, based on the Token value specified in the 'token' element of 'req_info' under the 'tp_info' parameter in the informative response.

Then, the proxy performs the following actions.

- o If the 'last_notif' field is not present, the proxy responds to the client with an Empty Acknowledgement (if indicated by the message type, and if it has not already done so).
- o If the 'last_notif' field is present, the proxy rebuilds the latest multicast notification, as defined in Section 3. Then, the proxy responds to the client, by forwarding back the latest multicast notification.

When responding to an observation request from a client, the proxy also adds that client (and its Token) to the list of its registered observers for the target resource, next to the older observations.

Upon receiving a multicast notification from the server, the proxy forwards it back separately to each observer client over unicast. Note that the notification forwarded back to a certain client has the same Token value of the original observation request sent by that client to the proxy.

Note that the proxy configures the observation of the target resource at the server only once, when receiving the informative response associated to a (newly started) group observation for that target resource.

After that, when receiving an observation request from a following new client to be added to the same group observation, the proxy does not take any further action with the server. Instead, the proxy responds to the client either with the latest multicast notification if available from its cache, or with an Empty Acknowledgement otherwise, as defined above.

An example is provided in Appendix E.

In the general case with a chain of two or more proxies, every proxy in the chain takes the role of client with the (next hop towards the) origin server. Note that the proxy adjacent to the origin server is the only one in the chain that receives informative responses and listens to an IP multicast address to receive notifications for the group observation. Furthermore, every proxy in the chain takes the role of server with the (previous hop towards the) origin client.

10. Intermediaries Together with End-to-End Security

As defined in Section 7, Group OSCORE can be used to protect multicast notifications end-to-end between the origin server and the clients. In such a case, additional actions are required when also the informative responses from the origin server are protected specifically end-to-end, by using OSCORE or Group OSCORE.

In fact, the proxy adjacent to the origin server is not able to access the encrypted payload of such informative responses. Hence, the proxy cannot retrieve the 'ph_req' and 'tp_info' parameters necessary to correctly receive multicast notifications and forward them back to the clients.

Then, differently from what defined in Section 9, each proxy receiving an informative response simply forwards it back to the client that has sent the corresponding observation request. Note that the proxy does not even realize the message to be an actual informative response, since the outer Code field is set to 2.05 (Content).

Upon receiving the informative response, the client does not configure an observation of the target resource. Instead, the client performs a new observe registration request, by transmitting the rebuilt phantom request as intended to reach the proxy adjacent to the origin server. In particular, the client includes the new Listen-To-Multicast-Responses CoAP option defined in Section 10.1, to provide that proxy with the transport-specific information required for receiving multicast notifications for the group observation.

Details on the additional message exchange and processing are defined in Section 10.2.

10.1. The Listen-To-Multicast-Responses Option

To allow the proxy to listen to the multicast notifications sent by the server, a new CoAP option is introduced. This option **MUST** be supported by clients interested to take part in group observations through intermediaries, and by proxies that collect multicast notifications and forward them back to the observer clients.

The option is called Listen-To-Multicast-Responses and is intended only for requests. As summarized in Figure 8, the option is critical and proxy-unsafe.

No.	C	U	N	R	Name	Format	Len.	Default
TBD	x	x	-		Listen-To-Multicast-Responses	(*)	3-1024	(none)

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

(*) See below.

Figure 8: Listen-To-Multicast-Responses

The Listen-To-Multicast-Responses option includes the serialization of a CBOR array. This specifies transport-specific message information required for listening to the multicast notifications of a group observation, and intended to the proxy adjacent to the origin server sending those notifications. In particular, the serialized CBOR array has the same format specified in Section 2.2.1 for the 'tp_info' parameter of the informative response (see Section 2.2).

The Listen-To-Multicast-Responses option is of class U for OSCORE [RFC8613][I-D.ietf-core-oscore-groupcomm].

10.2. Message Processing

Compared to Section 9, the following additions apply when informative responses are protected end-to-end between the origin server and the clients.

After the origin server sends an informative response, each proxy simply forwards it back to the (previous hop towards the) origin client that has sent the observation request.

Once received the informative response, the origin client proceeds in a different way than in Section 7.3.1:

- o The client performs all the additional decryption and verification steps of Section 7.3.1 on the phantom request specified in the 'ph_req' parameter and on the last notification specified in the 'last_notif' parameter (if present).
- o The client builds a ticket request (see Appendix B of [I-D.amsuess-core-cachable-oscore]), as intended to reach the

proxy adjacent to the origin server. The ticket request is formatted as follows.

- * The Token is chosen as the client sees fit. In fact, there is no reason for this Token to be the same as the phantom request's.
- * The Code field, the outer CoAP options and the encrypted payload (containing inner options, AEAD tag etc.) are the same of the phantom request used for the group observation. That is, they are as specified in the 'ph_req' parameter of the received informative response.
- * An outer Observe option is included and set to 0 (Register). This will usually be set in the phantom request already.
- * The outer options Proxy-Scheme, Uri-Host and Uri-Port are included, and set to the same values they had in the original registration request sent by the client.
- * The new option Listen-To-Multicast-Responses is included as an outer option. The value is set to the serialization of the CBOR array specified by the 'tp_info' parameter of the informative response.

Note that, except for transport-specific information such as the Token and Message ID values, every different client participating to the same group observation (hence rebuilding the same phantom request) will build the same ticket request.

Note also that, identically to the phantom request, the ticket request is still protected with Group OSCORE, i.e. it has the same OSCORE option, encrypted payload and counter signature.

Then, the client sends the ticket request to the next hop towards the origin server. Every proxy in the chain forwards the ticket request to the next hop towards the origin server, until the last proxy in the chain is reached. This last proxy, adjacent to the origin server, proceeds as follows.

- o The proxy MUST NOT further forward the ticket request to the origin server.
- o The proxy removes the Proxy-Scheme, Uri-Host and Uri-Port options from the ticket request.

- o The proxy removes the Listen-To-Multicast-Responses option from the ticket request, and extracts the conveyed transport-specific information.
- o The proxy rebuilds the phantom request associated to the group observation, by using the ticket request as directly providing the required transport-independent information. This includes the outer Code field, the outer CoAP options and the encrypted payload concatenated with the counter signature.
- o The proxy configures an observation of the target resource at the origin server, acting as a client directly taking part in the group observation. To this end, the proxy uses the rebuilt phantom request and the transport-specific information retrieved from the Listen-To-Multicast-Responses Option. The particular way to achieve this is implementation specific.

After that, the proxy will listen to the IP multicast address and port number indicated in the Listen-To-Multicast-Responses option, as 'cli_addr' and 'cli_port' element of the serialized CBOR array, respectively. Furthermore, multicast notifications will match the phantom request stored at the proxy, based on the Token value specified in the 'token' element of the serialized CBOR array in the Listen-To-Multicast-Responses option.

An example is provided in Appendix F.

11. Informative Response Parameters

This specification defines a number of fields used in the informative response message defined in Section 2.2.

The table below summarizes them and specifies the CBOR key to use instead of the full descriptive name. Note that the media type application/informative-response+cbor MUST be used when these fields are transported.

Name	CBOR Key	CBOR Type	Reference
tp_info	1	array	Section 2.2
ph_req	2	byte string	Section 2.2
last_notif	3	byte string	Section 2.2
join_uri	4	text string	Section 7.1
sec_gp	5	text string	Section 7.1
as_uri	6	text string	Section 7.1
cs_alg	7	int / text string	Section 7.1
cs_crv	8	int / text string	Section 7.1
cs_kty	9	int / text string	Section 7.1
cs_kenc	10	int	Section 7.1
alg	11	int / text string	Section 7.1
hkdf	12	int / text string	Section 7.1
gp_material	13	map	Appendix C
srv_pub_key	14	byte string	Appendix C
srv_identifier	15	byte string	Appendix C
exp	16	uint	Appendix C

Figure 9: CBOR abbreviations for informative response parameters

12. Transport Protocol Information

This specification defines some values of transport protocol identifiers to use within the 'tp_info' parameter of the informative response message defined in Section 2.2 of this specification.

According to the encoding specified in Section 2.2.1, these values are used for the 'tp_id' element of 'srv_addr', under the 'tp_info' parameter.

The table below summarizes them, specifies the integer value to use instead of the full descriptive name, and provides the corresponding full set of information elements to include in the 'tp_info' parameter.

Transport Protocol	Description	Value	Srv Addr	Req Info	Reference
Reserved	This value is reserved	0			[This document]
UDP	UDP is used as per RFC7252	1	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]

Figure 10: Transport protocol information

13. Security Considerations

The same security considerations from [RFC7252][RFC7641][I-D.ietf-core-groupcomm-bis][RFC8613][I-D.ietf-core-oscore-groupcomm] hold for this document.

If multicast notifications are protected using Group OSCORE, the original registration requests and related unicast (notification) responses MUST also be secured, including and especially the informative responses from the server. This prevents on-path active adversaries from altering the conveyed IP multicast address and serialized phantom registration request. Thus, it ensures secure binding between every multicast notification for a same observed resource and the phantom registration request that started the group observation of that resource.

To this end, clients and servers SHOULD use OSCORE or Group OSCORE, so ensuring that the secure binding above is enforced end-to-end between the server and each observing client.

13.1. Listen-To-Multicast-Responses Option

The CoAP option Listen-To-Multicast-Responses defined in Section 10.1 is of class U for OSCORE and Group OSCORE [RFC8613][I-D.ietf-core-oscore-groupcomm].

This allows the proxy adjacent to the origin server to access the option value conveyed in a ticket request (see Section 10.2), and to retrieve from it the transport-specific information about a phantom

request. By doing so, the proxy becomes able to configure an observation of the target resource and to receive multicast notifications matching to the phantom request.

Any proxy in the chain, as well as further possible intermediaries or on-path active adversaries, are thus able to remove the option or alter its content, before the ticket request reaches the proxy adjacent to the origin server.

Removing the option would result in the proxy adjacent to the origin server to not configure the group observation, if that has not happened yet. In such a case, the proxy would not receive the corresponding multicast notifications to be forwarded back to the clients.

Altering the option content would result in the proxy adjacent to the origin server to incorrectly configure a group observation (e.g., by indicating a wrong multicast IP address) hence preventing the correct reception of multicast notifications and their forwarding to the clients; or to configure bogus group observations that are currently not active on the origin server.

In order to prevent what described above, the ticket requests conveying the Listen-To-Multicast-Responses option can be additionally protected hop-by-hop.

14. IANA Considerations

This document has the following actions for IANA.

14.1. Media Type Registrations

This specification registers the media type 'application/informative-response+cbor' for error messages as informative response defined in Section 2.2 of this specification, when carrying parameters encoded in CBOR. This registration follows the procedures specified in [RFC6838].

- o Type name: application
- o Subtype name: informative-response+cbor
- o Required parameters: N/A
- o Optional parameters: N/A
- o Encoding considerations: Must be encoded as a CBOR map containing the parameters defined in Section 2.2 of [this document].

- o Security considerations: See Section 13 of [this document].
- o Interoperability considerations: N/A
- o Published specification: [this document]
- o Applications that use this media type: The type is used by CoAP servers and clients that support error messages as informative response defined in Section 2.2 of [this document].
- o Fragment identifier considerations: N/A
- o Additional information: N/A
- o Person & email address to contact for further information: iesg@ietf.org [1]
- o Intended usage: COMMON
- o Restrictions on usage: None
- o Author: Marco Tiloca marco.tiloca@ri.se [2]
- o Change controller: IESG

14.2. CoAP Content-Formats Registry

IANA is asked to add the following entry to the "CoAP Content-Formats" sub-registry defined in Section 12.3 of [RFC7252], within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Media Type: application/informative-response+cbor

Encoding: -

ID: TBD

Reference: [this document]

14.3. Informative Response Parameters Registry

This specification establishes the "Informative Response Parameters" IANA registry. The registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 14.6.

The columns of this registry are:

- o Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.
- o CBOR Key: This is the value used as CBOR key of the item. These values MUST be unique. The value can be a positive integer, a negative integer, or a string.
- o CBOR Type: This contains the CBOR type of the item, or a pointer to the registry that defines its type, when that depends on another item.
- o Reference: This contains a pointer to the public specification for the item.

This registry has been initially populated by the values in Section 11. The "Reference" column for all of these entries refers to sections of this document.

14.4. CoAP Transport Information Registry

This specification defines the subregistry "CoAP Transport Information" within the "CoRE Parameters" registry. The registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 14.6.

The columns of this registry are:

- o Transport Protocol: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.
- o Description: Text giving an overview of the transport protocol referred by this item.
- o Value: CBOR abbreviation for the transport protocol referred by this item. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.
- o Server Addr: List of elements providing addressing information of the server.

- o Req Info: List of elements providing transport-specific information related to a pertinent CoAP request. Optional elements are prepended by '?'.
- o Reference: This contains a pointer to the public specification for the item.

This registry has been initially populated by the values in Section 12. The "Reference" column for all of these entries refers to sections of this document.

14.5. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry defined in [RFC7252] within the "CoRE Parameters" registry.

Number	Name	Reference
TBD	Multicast-Response-Feedback-Divider	[This document]
TBD	Listen-To-Multicast-Responses	[This document]

14.6. Expert Review Instructions

The IANA registries established in this document are defined as expert review. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- o Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments, code points in other ranges should not be assigned for testing.
- o Specifications are required for the standards track range of point assignment. Specifications should exist for specification required ranges, but early assignment before a specification is available is considered to be permissible. Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way.

When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.

- o Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

15. References

15.1. Normative References

[COSE.Algorithms]

IANA, "COSE Algorithms",
<<https://www.iana.org/assignments/cose/cose.xhtml#algorithms>>.

[COSE.Elliptic.Curves]

IANA, "COSE Elliptic Curves",
<<https://www.iana.org/assignments/cose/cose.xhtml#elliptic-curves>>.

[COSE.Key.Types]

IANA, "COSE Key Types",
<<https://www.iana.org/assignments/cose/cose.xhtml#key-type>>.

[I-D.ietf-ace-key-groupcomm-oscore]

Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-10 (work in progress), February 2021.

[I-D.ietf-ace-oscore-profile]

Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-16 (work in progress), January 2021.

[I-D.ietf-core-groupcomm-bis]

Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-ietf-core-groupcomm-bis-03 (work in progress), February 2021.

- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-11 (work in progress), February 2021.
- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", draft-ietf-cose-rfc8152bis-algs-12 (work in progress), September 2020.
- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", draft-ietf-cose-rfc8152bis-struct-15 (work in progress), February 2021.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

- [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>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

15.2. Informative References

- [I-D.amsuess-core-cachable-oscore]
Amsuess, C. and M. Tiloca, "Cachable OSCORE", draft-amsuess-core-cachable-oscore-01 (work in progress), February 2021.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-37 (work in progress), February 2021.
- [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-09 (work in progress), September 2019.

- [I-D.ietf-core-coral]
Hartke, K., "The Constrained RESTful Application Language (CoRAL)", draft-ietf-core-coral-03 (work in progress), March 2020.
- [I-D.ietf-core-resource-directory]
Amsuess, C., Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-26 (work in progress), November 2020.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-41 (work in progress), February 2021.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-08 (work in progress), February 2021.
- [MOBICOM99]
Ni, S., Tseng, Y., Chen, Y., and J. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network", Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking , August 1999,
<<https://people.eecs.berkeley.edu/~culler/cs294-f03/papers/bcast-storm.pdf>>.
- [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>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.

15.3. URIs

[1] <mailto:iesg@ietf.org>

[2] <mailto:marco.tiloca@ri.se>

Appendix A. Different Sources for Group Observation Data

While the clients usually receive the phantom registration request and other information related to the group observation through an Informative Response, the same data can be made available through different services, such as the following ones.

A.1. Topic Discovery in Publish-Subscribe Settings

In a Publish-Subscribe scenario ([I-D.ietf-core-coap-pubsub]), a group observation can be discovered along with topic metadata. For instance, a discovery step can make the following metadata available.

This example assumes a CoRAL namespace [I-D.ietf-core-coral], that contains properties analogous to those in the content-format application/informative-response+cbor.

Request:

```
GET </ps/topics?rt=oic.r.temperature>
Accept: CoRAL
```

Response:

```
2.05 Content
Content-Format: CoRAL

rdf:type <http://example.org/pubsub/topic-list>
topic </ps/topics/1234> {
  tp_info [1, h"7b", h"20010db80100..0001", 5683,
           h"ff35003020010db8..1234", 5683],
  ph_req h"0160..",
  last_notif h"256105.."
}
```

Figure 11: Group observation discovery in a Pub-Sub scenario

With this information from the topic discovery step, the client can already set up its multicast address and start receiving multicast notifications.

In heavily asymmetric networks like municipal notification services, discovery and notifications do not necessarily need to use the same network link. For example, a departure monitor could use its (costly and usually-off) cellular uplink to discover the topics it needs to update its display to, and then listen on a LoRA-WAN interface for receiving the actual multicast notifications.

A.2. Introspection at the Multicast Notification Sender

For network debugging purposes, it can be useful to query a server that sends multicast responses as matching a phantom registration request.

Such an interface is left for other documents to specify on demand. As an example, a possible interface can be as follows, and rely on the already known Token value of intercepted multicast notifications, associated to a phantom registration request.

Request:

```
GET </.well-known/core/mc-sender?token=6464>
```

Response:

2.05 Content

Content-Format: application/informative-response+cbor

```
{
  'tp_info': [1, h"7b", h"20010db80100..0001", 5683,
              h"ff35003020010db8..1234", 5683],
  'ph_req': h"0160..",
  'last_notif' : h"256105.."
}
```

Figure 12: Group observation discovery with server introspection

For example, a network sniffer could offer sending such a request when unknown multicast notifications are seen on a network. Consequently, it can associate those notifications with a URI, or decrypt them, if member of the correct OSCORE group.

Appendix B. Pseudo-Code for Rough Counting of Clients

This appendix provides a description in pseudo-code of the two algorithms used for the rough counting of active observers, as defined in Section 6.

In particular, Appendix B.1 describes the algorithm for the client side, while Appendix B.2 describes an optimized version for constrained clients. Finally, Appendix B.3 describes the algorithm for the server side.

B.1. Client Side

```
input:  int Q, // Value of the MRFD option
        int LEISURE_TIME, // DEFAULT_LEISURE from RFC 7252,
                          // unless overridden
```

```
output: None
```

```
int RAND_MIN = 0;
int RAND_MAX = (2**Q) - 1;
int I = randomInteger(RAND_MIN, RAND_MAX);

if (I == 0) {
    float fraction = randomFloat(0, 1);

    Timer t = new Timer();
    t.setAndStart(fraction * LEISURE_TIME);
    while(!t.isExpired());

    Request req = new Request();
    // Initialize as NON and with maximum
    // No-Response settings, set options ...

    Option opt = new Option(OBSERVE);
    opt.set(0);
    req.setOption(opt);

    opt = new Option(MRFD);
    req.setOption(opt);

    req.send(SRV_ADDR, SRV_PORT);
}
```

B.2. Client Side - Optimized Version

```
input:  int Q, // Value of the MRFD option
        int LEISURE_TIME, // DEFAULT_LEISURE from RFC 7252,
                          // unless overridden

output: None

const unsigned int UINT_BIT = CHAR_BIT * sizeof(unsigned int);

if (respond_to(Q) == true) {
    float fraction = randomFloat(0, 1);

    Timer t = new Timer();
    t.setAndStart(fraction * LEISURE_TIME);
    while(!t.isExpired());

    Request req = new Request();
    // Initialize as NON and with maximum
    // No-Response settings, set options ...

    Option opt = new Option(OBSERVE);
    opt.set(0);
    req.setOption(opt);

    opt = new Option(MRFD);
    req.setOption(opt);

    req.send(SRV_ADDR, SRV_PORT);
}

bool respond_to(int Q) {
    while (Q >= UINT_BIT) {
        if (rand() != 0) return false;
        Q -= UINT_BIT;
    }
    unsigned int mask = ~((~0u) << Q);
    unsigned int masked = mask & rand();
    return masked == 0;
}
```

B.3. Server Side

```
input:  int COUNT, // Current observer counter
        int M, // Desired number of confirmations
        int MAX_CONFIRMATION_WAIT,
        Response notification, // Multicast notification to send
```

```
output: int NEW_COUNT // Updated observer counter

int D = 4; // Dampener value
int RETRY_NEXT_THRESHOLD = 4;
float CANCEL_THRESHOLD = 0.2;

int N = max(COUNT, 1);
int Q = max(ceil(log2(N / M)), 0);
Option opt = new Option(MRFD);
opt.set(Q);

notification.setOption(opt);
<Finalize the notification message>
notification.send(GRP_ADDR, GRP_PORT);

Timer t = new Timer();
t.setAndStart(MAX_CONFIRMATION_WAIT); // Time t1
while(!t.isExpired());

// Time t2

int R = <number of requests to the target resource
        between t1 and t2, with the MRFD option>;

int E = R * (2**Q);

// Determine after how many multicast notifications
// the next count update will be performed
if ((R == 0) || (max(E/N, N/E) > RETRY_NEXT_THRESHOLD)) {
    <Next count update with the next multicast notification>
}
else {
    <Next count update after 10 multicast notifications>
}

// Compute the new count estimate
int COUNT_PRIME = <current value of the observer counter>;
int NEW_COUNT = COUNT_PRIME + ((E - N) / D);

// Determine whether to cancel the group observation
if (NEW_COUNT < CANCEL_THRESHOLD) {
    <Cancel the group observation>;
    return 0;
}

return NEW_COUNT;
```

Appendix C. OSCORE Group Self-Managed by the Server

For simple settings, where no pre-arranged group with suitable memberships is available, the server can be responsible to setup and manage the OSCORE group used to protect the group observation.

In such a case, a client would implicitly request to join the OSCORE group when sending the observe registration request to the server. When replying, the server includes the group keying material and related information in the informative response (see Section 2.2).

Additionally to what defined in Section 2, the CBOR map in the informative response payload contains the following fields, whose CBOR labels are defined in Section 11.

- o 'gp_material': this element is a CBOR map, which includes what the client needs in order to set up the Group OSCORE Security Context.

This parameter has as value a subset of the Group_OSCORE_Input_Material object, which is defined in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore] and extends the OSCORE_Input_Material object encoded in CBOR as defined in Section 3.2.1 of [I-D.ietf-ace-oscore-profile].

In particular, the following elements of the Group_OSCORE_Input_Material object are included, using the same CBOR labels from the OSCORE Security Context Parameters Registry, as in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore].

- * 'ms', 'contextId', 'cs_alg', 'cs_params', 'cs_key_params', 'cs_key_enc'. These elements MUST be included.

- * 'hkdf', 'alg', 'salt'. These elements MAY be included.

The following elements of the Group_OSCORE_Input_Material object MUST NOT be included.

- * 'group_senderId', 'ecdh_alg', 'ecdh_params', 'ecdh_key_params'.

- o 'srv_pub_key': this element is a CBOR byte string, which wraps the serialization of the public key that the server uses in the OSCORE group. If the public key of the server is encoded as a COSE_Key (see the 'cs_key_enc' element of the 'gp_material' parameter), it includes 'kid' specifying the Sender ID that the server has in the OSCORE group.
- o 'srv_identifier': this element MUST be present only if 'srv_pub_key' is also present and, at the same time, the used

encoding for the public key of the server does not allow to specify a Sender ID within the associated public key. Otherwise, it MUST NOT be present. If present, this element is a CBOR byte string, with value the Sender ID that the server has in the OSCORE group.

- o 'exp': with value the expiration time of the keying material of the OSCORE group specified in the 'gp_material' parameter, encoded as a CBOR unsigned integer. This field contains a numeric value representing the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds, analogous to what specified for NumericDate in Section 2 of [RFC7519].

A client receiving an informative response uses the information above to set up the Group OSCORE Security Context, as described in Section 2 of [I-D.ietf-core-oscore-groupcomm]. Note that the client does not obtain a Sender ID of its own, hence it installs a Security Context that a "silent server" would, i.e. without Sender Context. From then on, the client uses the received keying material to process the incoming multicast notifications from the server.

The server complies with the following points.

- o The server MUST NOT self-manage OSCORE groups and provide the related keying material in the informative response for any other purpose than the protection of group observations, as defined in this document.

The server MAY use the same self-managed OSCORE group to protect the phantom request and the multicast notifications of multiple group observations it hosts.

- o The server MUST NOT provide in the informative response the keying material of other OSCORE groups it is or has been a member of.

After the time indicated in the 'exp' field:

- o The server MUST stop using the keying material and MUST cancel the group observations for which that keying material is used (see Section 2.5). If the server creates a new group observation as a replacement or follow-up using the same OSCORE group:
 - * The server MUST update the Master Secret.
 - * The server MUST update the ID Context (Gid). Consistently with Section 2.3 of [I-D.ietf-core-oscore-groupcomm], the server

MUST assign an ID Context that it has never assigned before in the OSCORE group.

- * The server MAY update the Master Salt.
- o The client MUST stop using the keying material and MAY re-register the observation at the server.

Before the key material has expired, the server can send a multicast response with response code 5.03 (Service Unavailable) to the observing clients, protected with the current key material. In particular, this is an informative response (see Section 2.2) and contains the abovementioned parameters for the next group keying material to be used. Alternatively, the server can simply cancel the group observation (see Section 2.5), which results in the eventual re-registration of the clients that are still interested in the group observation.

Applications requiring backward security and forward security are REQUIRED to use an actual group joining process (usually through a dedicated Group Manager), e.g. the ACE joining procedure defined in [I-D.ietf-ace-key-groupcomm-oscure]. The server can facilitate the clients by providing them information about the OSCORE group to join, as described in Section 7.1.

Appendix D. Phantom Request as Deterministic Request

In some settings, the server can assume that all the approaching clients already have the exact phantom observation request to use.

For instance, the clients can be pre-configured with the phantom observation request, or they may be expected to retrieve it through dedicated means (see Appendix A), before sending an observe registration request to the server.

If Group OSCORE is used to protect the group observation (see Section 7), and the OSCORE group supports the concept of Deterministic Client [I-D.amsuess-core-cachable-oscure], then the server and each client in the OSCORE group can independently protect the phantom observation request possibly available as plain CoAP message. To this end, they use the approach defined in Section 2 of [I-D.amsuess-core-cachable-oscure] to compute a protected deterministic request, against which the protected multicast notifications will match for the group observation in question.

Note that, if the optimization defined in Appendix C is also used, the error informative response from the server has to include additional information, i.e. the Sender ID of the Deterministic

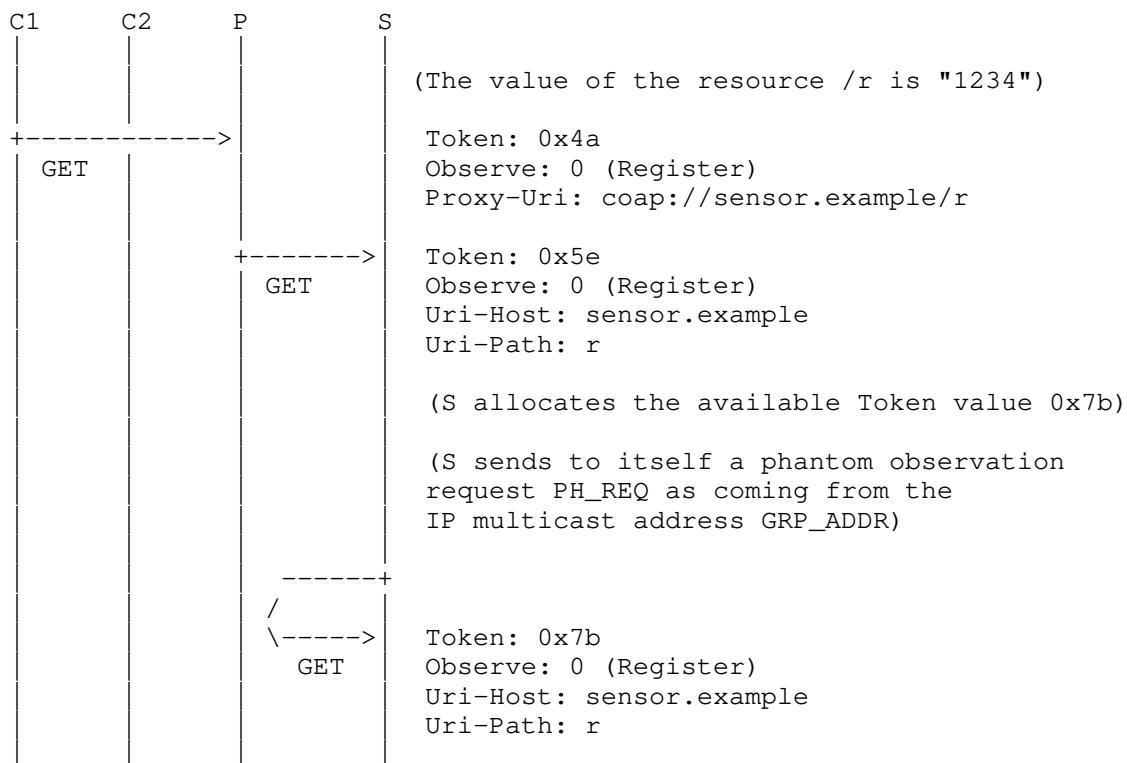
Client in the OSCORE group, and the hash algorithm used to compute the deterministic request (see Section 2.3.1 of [I-D.amsuess-core-cachable-oscore]).

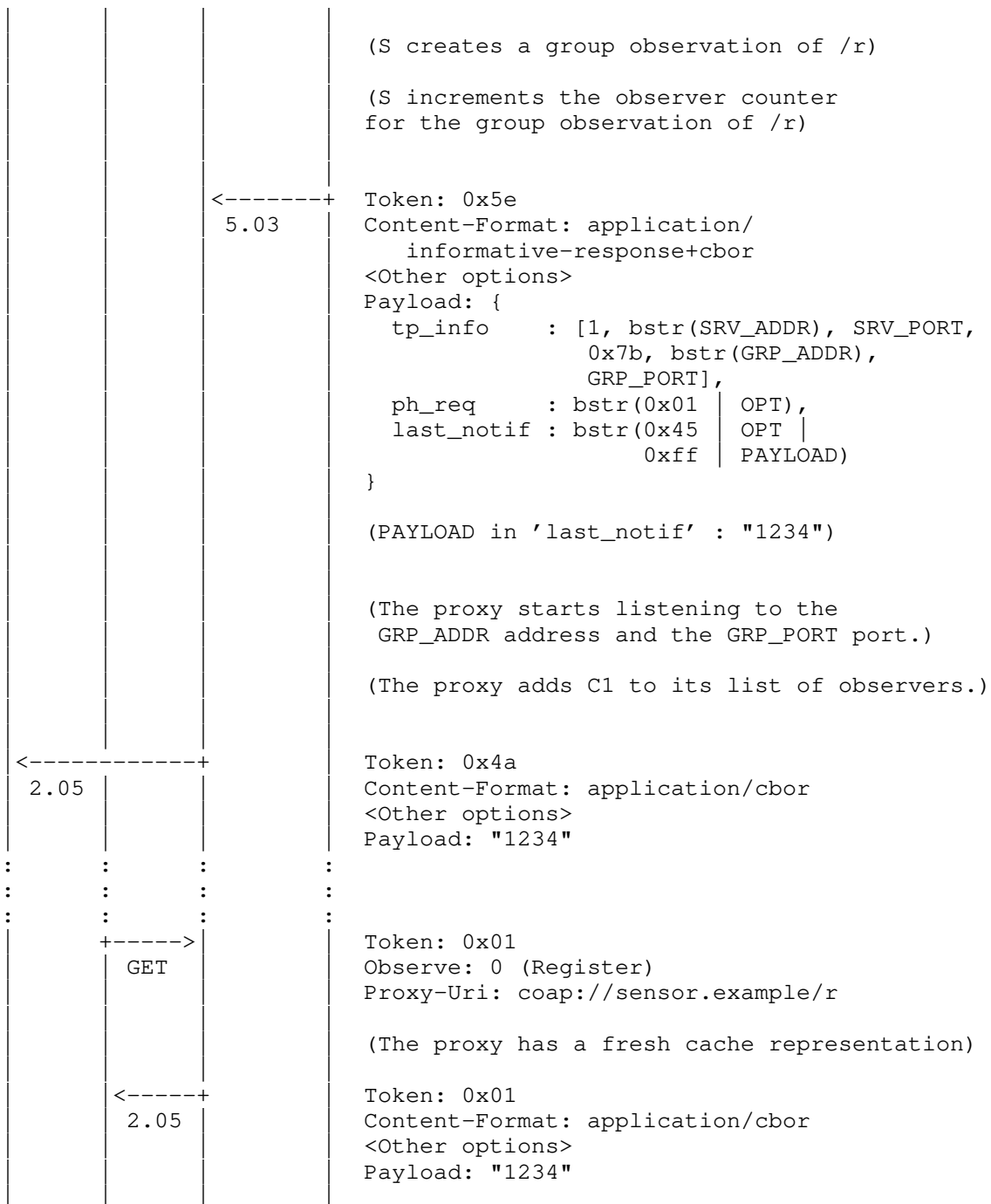
This optimization allows the server to not provide the same full phantom observation request to each client in the error informative response (see Section 2.2). That is, the informative response does not need to include the 'ph_req' parameter, but only the 'tp_info' parameter specifying the transport-specific information and (optionally) the 'last_notif' parameter specifying the latest sent multicast notification.

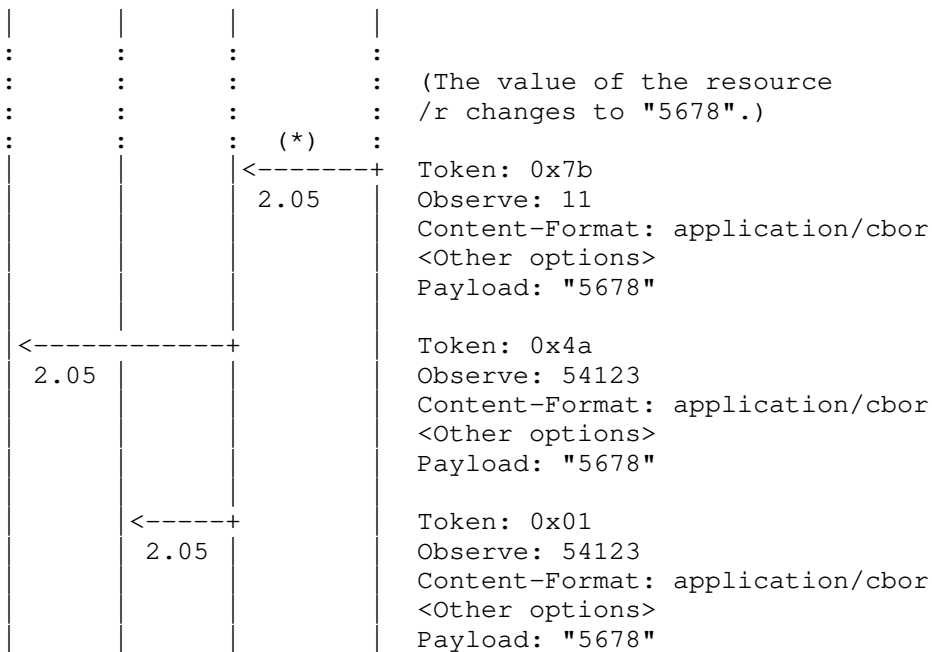
Appendix E. Example with a Proxy

This section provides an example when a proxy P is used between the clients and the server. The same assumptions and notation used in Section 5 are used for this example. In addition, the proxy has address PRX_ADDR and listens to the port number PRX_PORT.

Unless explicitly indicated, all messages transmitted on the wire are sent over unicast.







(*) Sent over IP multicast to GROUP_ADDR:GROUP_PORT

Figure 13: Example of group observation with a proxy

Note that the proxy has all the information to understand the observation request from C2, and can immediately start to serve the still fresh values.

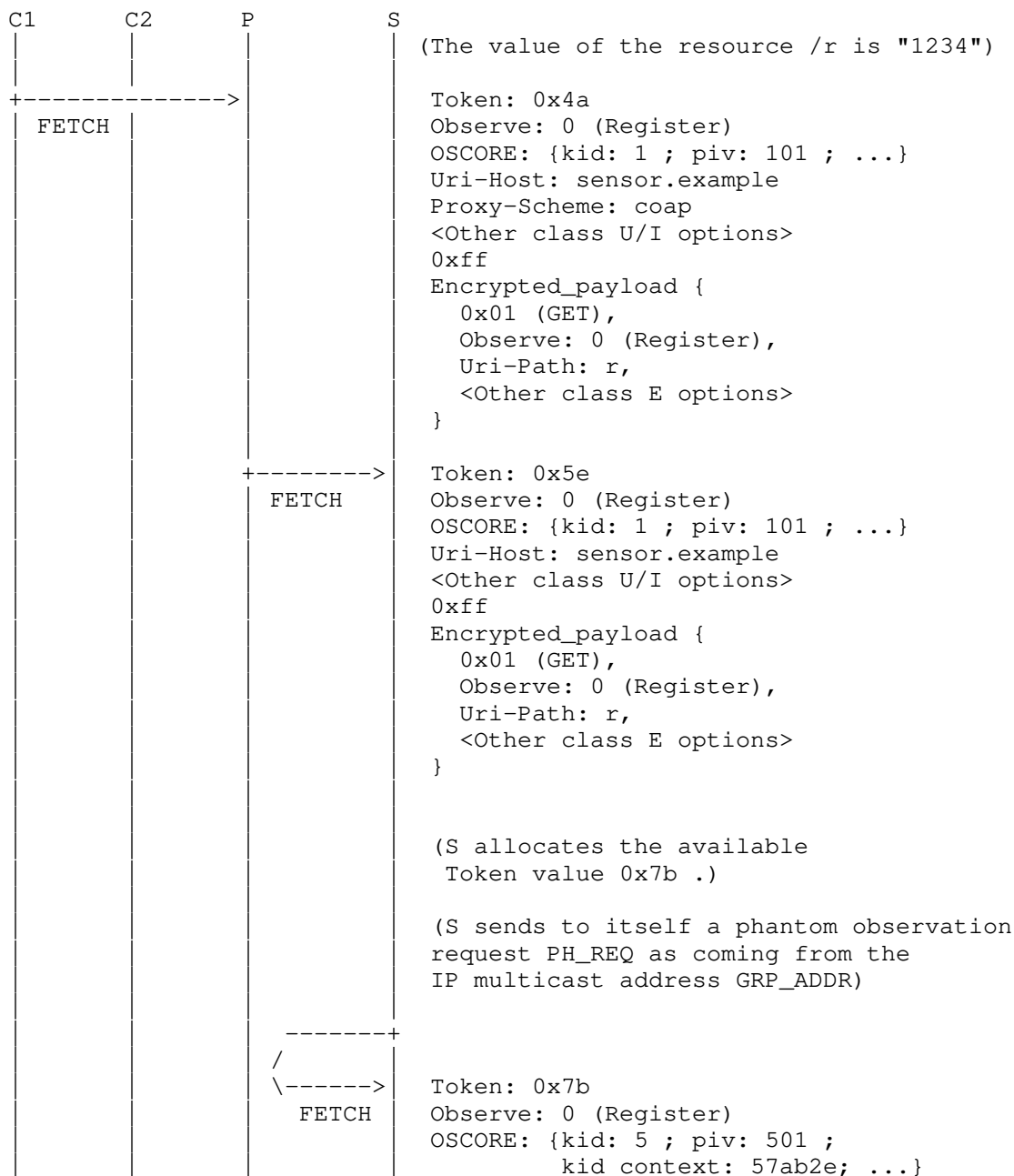
This behavior is mandated by Section 5 of [RFC7641], i.e. the proxy registers itself only once with the next hop and fans out the notifications it receives to all registered clients.

Appendix F. Example with a Proxy and Group OSCORE

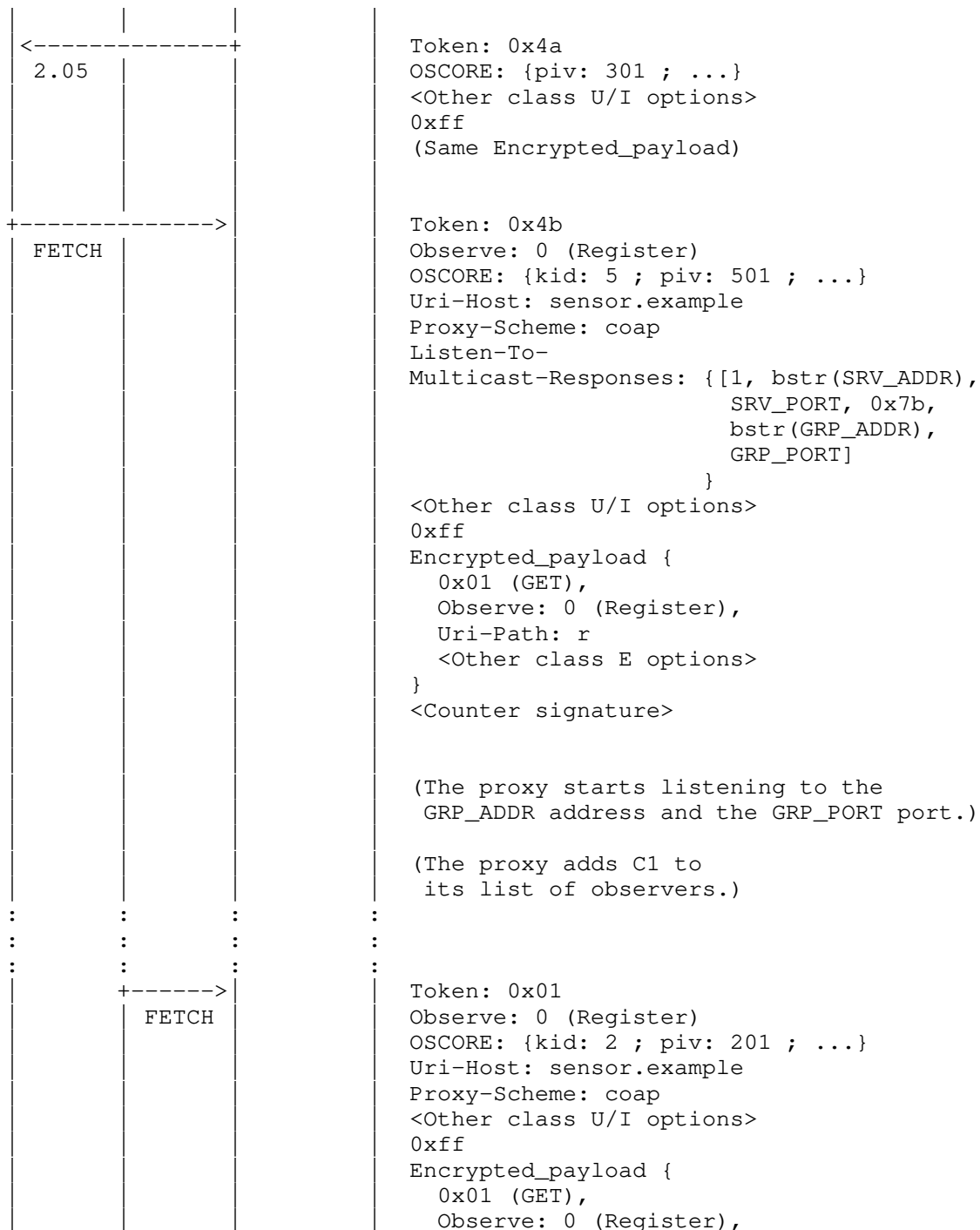
This section provides an example when a proxy P is used between the clients and the server, and Group OSCORE is used to protect multicast notifications end-to-end between the server and the clients.

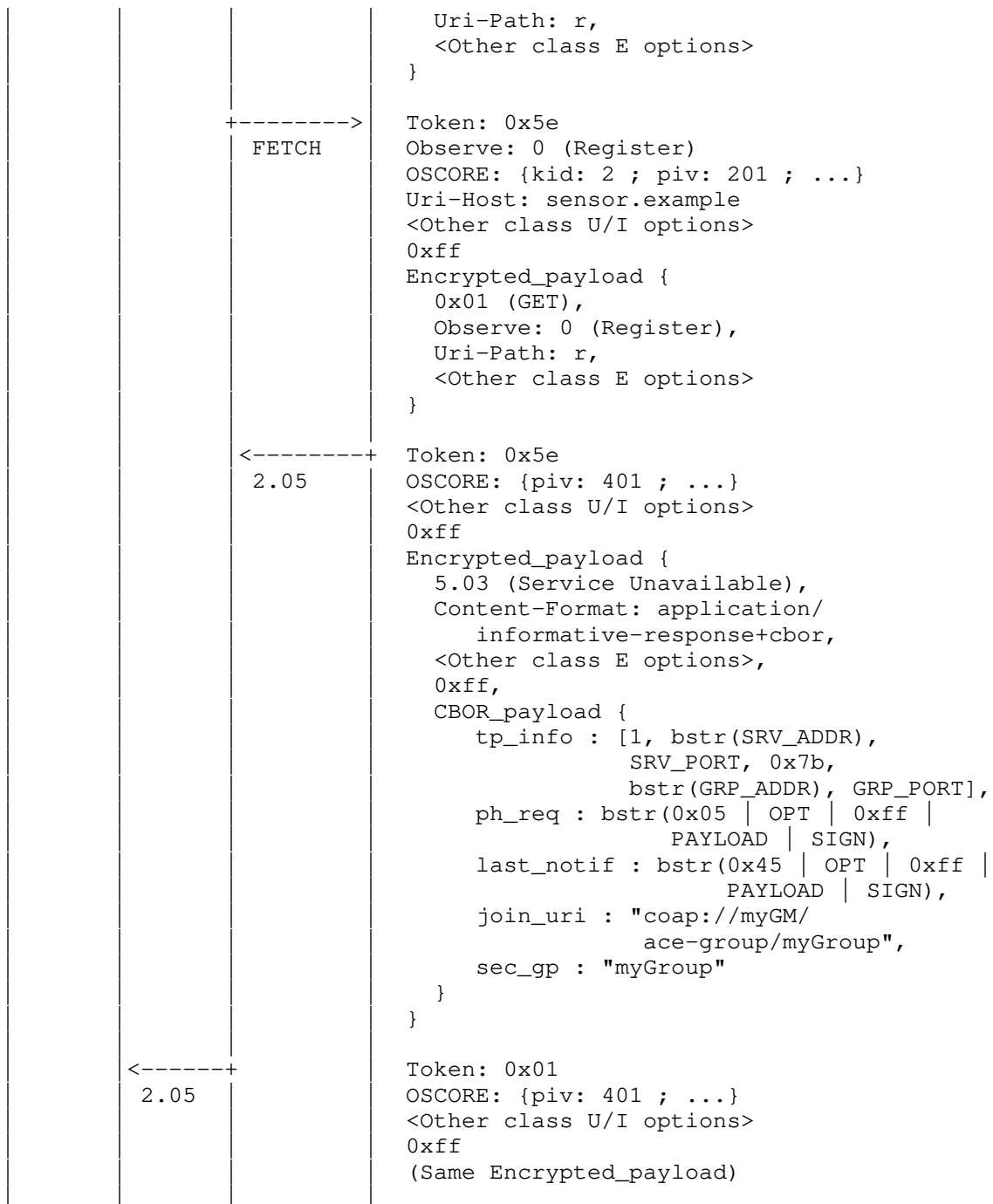
The same assumptions and notation used in Section 8 are used for this example. In addition, the proxy has address PRX_ADDR and listens to the port number PRX_PORT.

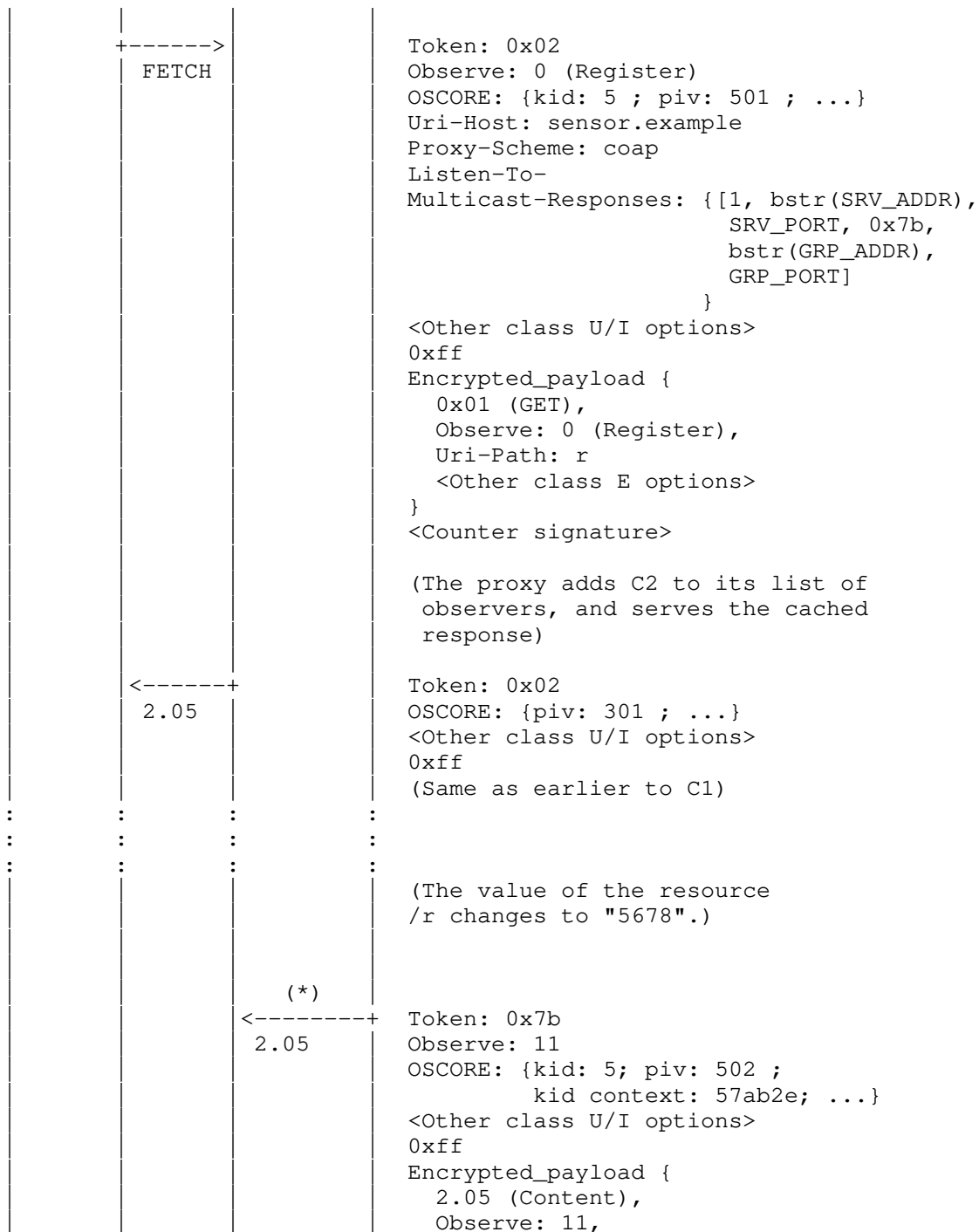
Unless explicitly indicated, all messages transmitted on the wire are sent over unicast and protected with OSCORE end-to-end between a client and the server.

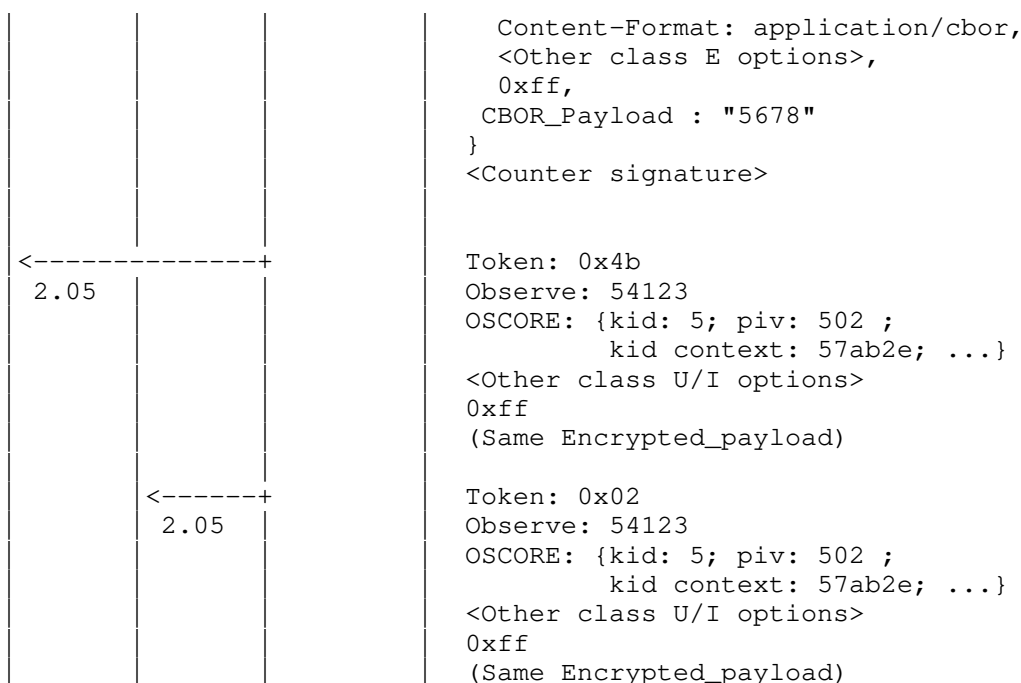


			<pre> Uri-Host: sensor.example <Other class U/I options> 0xff Encrypted_payload { 0x01 (GET), Observe: 0 (Register), Uri-Path: r <Other class E options> } <Counter signature> (S steps SN_5 in the Group OSCORE Security Context : SN_5 <== 502) (S creates a group observation of /r) (S increments the observer counter for the group observation of /r) <-----+ 2.05 Token: 0x5e OSCORE: {piv: 301 ; ...} <Other class U/I options> 0xff Encrypted_payload { 5.03 (Service Unavailable), Content-Format: application/ informative-response+cbor, <Other class E options>, 0xff, CBOR_payload { tp_info : [1, bstr(SRV_ADDR), SRV_PORT, 0x7b, bstr(GRP_ADDR), GRP_PORT], ph_req : bstr(0x05 OPT 0xff PAYLOAD SIGN), last_notif : bstr(0x45 OPT 0xff PAYLOAD SIGN), join_uri : "coap://myGM/ ace-group/myGroup", sec_gp : "myGroup" } } </pre>
--	--	--	---









(*) Sent over IP multicast to GROUP_ADDR:GROUP_PORT and protected with Group OSCORE end-to-end between the server and the clients.

Figure 14: Example of group observation with a proxy and Group OSCORE

Unlike in the unprotected example in Appendix E, the proxy does not have all the information to perform request deduplication, and can only recognize the identical request once the client sends the ticket request.

Acknowledgments

The authors sincerely thank Carsten Bormann, Klaus Hartke, Jaime Jimenez, John Mattsson, Ludwig Seitz, Jim Schaad and Goeran Selander for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Rikard Hoeglund
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: rikard.hoglund@ri.se

Christian Amsuess
Hollandstr. 12/4
Vienna 1020
Austria

Email: christian@amsuess.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

M. Tiloca
RISE AB
C. Amsuess

P. van der Stok
Consultant
November 02, 2020

Discovery of OSCORE Groups with the CoRE Resource Directory
draft-tiloca-core-oscore-discovery-07

Abstract

Group communication over the Constrained Application Protocol (CoAP) can be secured by means of Group Object Security for Constrained RESTful Environments (Group OSCORE). At deployment time, devices may not know the exact security groups to join, the respective Group Manager, or other information required to perform the joining process. This document describes how a CoAP endpoint can use descriptions and links of resources registered at the CoRE Resource Directory to discover security groups and to acquire information for joining them through the respective Group Manager. A given security group may protect multiple application groups, which are separately announced in the Resource Directory as sets of endpoints sharing a pool of resources. This approach is consistent with, but not limited to, the joining of security groups based on the ACE framework for Authentication and Authorization in constrained 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 May 6, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	5
2. Registration of Group Manager Endpoints	6
2.1. Parameters	6
2.2. Relation Link to Authorization Server	9
2.3. Registration Example	9
2.3.1. Example in Link Format	10
2.3.2. Example in CoRAL	10
3. Addition and Update of Security Groups	11
3.1. Addition Example	11
3.1.1. Example in Link Format	12
3.1.2. Example in CoRAL	12
4. Discovery of Security Groups	14
4.1. Discovery Example #1	15
4.1.1. Example in Link Format	15
4.1.2. Example in CoRAL	16
4.2. Discovery Example #2	18
4.2.1. Example in Link Format	18
4.2.2. Example in CoRAL	19
5. Use Case Example With Full Discovery	20
6. Security Considerations	24
7. IANA Considerations	24
8. References	24
8.1. Normative References	24
8.2. Informative References	26
Appendix A. Use Case Example With Full Discovery (CoRAL)	27
Acknowledgments	31
Authors' Addresses	32

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports group communication over IP multicast [I-D.ietf-core-groupcomm-bis] to improve efficiency and latency of communication and reduce bandwidth requirements. A set of CoAP endpoints constitutes an application group by sharing a common pool of resources, that can be efficiently accessed through group communication. The members of an application group may be members of a security group, thus sharing a common set of keying material to secure group communication.

The security protocol Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm] builds on OSCORE [RFC8613] and protects CoAP messages end-to-end in group communication contexts through CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs]. An application group may rely on one or more security groups, and a same security group may be used by multiple application groups at the same time.

A CoAP endpoint relies on a Group Manager (GM) to join a security group and get the group keying material. The joining process in [I-D.ietf-ace-key-groupcomm-oscore] is based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz], with the joining endpoint and the GM acting as ACE Client and Resource Server, respectively. That is, the joining endpoint accesses the group-membership resource exported by the GM and associated with the security group to join.

Typically, devices store a static X509 IDevID certificate installed at manufacturing time [I-D.ietf-anima-bootstrapping-keyinfra]. This is used at deployment time during an enrollment process that provides the devices with an Operational Certificate, possibly updated during the device lifetime. Operational Certificates may specify information to join security groups, especially a reference to the group-membership resources to access at the respective GMs.

However, it is usually impossible to provide such precise information to freshly deployed devices, as part of their (early) Operational Certificate. This can be due to a number of reasons: (1) the security group(s) to join and the responsible GM(s) are generally unknown at manufacturing time; (2) a security group of interest is created, or the responsible GM is deployed, only after the device is enrolled and fully operative in the network; (3) information related to existing security groups or to their GMs has changed. This requires a method for CoAP endpoints to dynamically discover security

groups and their GM, and to retrieve relevant information about deployed groups.

To this end, CoAP endpoints can use descriptions and links of group-membership resources at GMs, to discover security groups and retrieve the information required for joining them. With the discovery process of security groups expressed in terms of links to resources, the remaining problem is the discovery of those links. The CoRE Resource Directory (RD) [I-D.ietf-core-resource-directory] allows such discovery in an efficient way, and it is expected to be used in many setups that would benefit of security group discovery.

This specification builds on this approach and describes how CoAP endpoints can use the RD to perform the link discovery steps, in order to discover security groups and retrieve the information required to join them through their GM. In short, the GM registers as an endpoint with the RD. The resulting registration resource includes one link per security group under that GM, specifying the path to the related group-membership resource to access for joining that group.

Additional descriptive information about the security group is stored with the registered link. In a RD based on Link Format [RFC6690] as defined in [I-D.ietf-core-resource-directory], this information is specified as target attributes of the registered link, and includes the identifiers of the application groups which use that security group. This enables a lookup of those application groups at the RD, where they are separately announced by a Commissioning Tool (see Appendix A of [I-D.ietf-core-resource-directory]).

When querying the RD for security groups, a CoAP endpoint can use CoAP observation [RFC7641]. This results in automatic notifications on the creation of new security groups or the update of existing groups. Thus, it facilitates the early deployment of CoAP endpoints, i.e. even before the GM is deployed and security groups are created.

Interaction examples are provided in Link Format, as well as in the Constrained RESTful Application Language CoRAL [I-D.ietf-core-coral] with reference to a CoRAL-based RD [I-D.hartke-t2trg-coral-reef]. While all the CoRAL examples use the CoRAL textual serialization format, the CBOR [I-D.ietf-cbor-7049bis] or JSON [RFC8259] binary serialization format is used when sending such messages on the wire.

The approach in this document is consistent with, but not limited to, the joining of security groups defined in [I-D.ietf-ace-key-groupcomm-oscore].

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.

This specification requires readers to be familiar with the terms and concepts discussed in [I-D.ietf-core-resource-directory] and [RFC6690], as well as in [I-D.ietf-core-coral]. Readers should also be familiar with the terms and concepts discussed in [RFC7252][I-D.ietf-core-groupcomm-bis], [I-D.ietf-core-oscore-groupcomm] and [I-D.ietf-ace-key-groupcomm-oscore].

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

Consistently with the definitions from Section 2.1 of [I-D.ietf-core-groupcomm-bis], this document also refers to the following terminology.

- o CoAP group: a set of CoAP endpoints all configured to receive CoAP multicast messages sent to the group's associated IP multicast address and UDP port. An endpoint may be a member of multiple CoAP groups by subscribing to multiple IP multicast addresses.
- o Security group: a set of CoAP endpoints that share the same security material, and use it to protect and verify exchanged messages. A CoAP endpoint may be a member of multiple security groups. There can be a one-to-one or a one-to-many relation between security groups and CoAP groups.

This document especially considers a security group to be an OSCORE group, where all members share one OSCORE Security Context to protect group communication with Group OSCORE [I-D.ietf-core-oscore-groupcomm]. However, the approach defined in this document can be used to support the discovery of different security groups than OSCORE groups.

- o Application group: a set of CoAP endpoints that share a common set of resources. An endpoint may be a member of multiple application groups. An application group can be associated with one or more security groups, and multiple application groups can use the same security group. Application groups are announced in the RD by a Commissioning Tool, according to the RD-Groups usage pattern (see Appendix A of [I-D.ietf-core-resource-directory]).

2. Registration of Group Manager Endpoints

During deployment, a Group Manager (GM) can find the CoRE Resource Directory (RD) as described in Section 4 of [I-D.ietf-core-resource-directory].

Afterwards, the GM registers as an endpoint with the RD, as described in Section 5 of [I-D.ietf-core-resource-directory]. The GM SHOULD NOT use the Simple Registration approach described in Section 5.1 of [I-D.ietf-core-resource-directory].

When registering with the RD, the GM also registers the links to all the group-membership resources it has at that point in time, i.e. one for each of its security groups.

In the registration request, each link to a group-membership resource has as target the URI of that resource at the GM. Also, it specifies a number of descriptive parameters as defined in Section 2.1.

2.1. Parameters

For each registered link to a group-membership resource at a GM, the following parameters are specified together with the link.

In the RD defined in [I-D.ietf-core-resource-directory] and based on Link Format, each parameter is specified in a target attribute with the same name.

In a RD based on CoRAL, such as the one defined in [I-D.hartke-t2trg-coral-reef], each parameter is specified in a nested element with the same name.

- o 'rt', specifying the resource type of the group-membership resource at the Group Manager, with value "core.osc.gm" registered in Section 21.11 of [I-D.ietf-ace-key-groupcomm-oscore].
- o 'if', specifying the interface description for accessing the group-membership resource at the Group Manager, with value "ace.group" registered in Section 8.10 of [I-D.ietf-ace-key-groupcomm].
- o 'sec-gp', specifying the name of the security group of interest, as a stable and invariant identifier, such as the group name used in [I-D.ietf-ace-key-groupcomm-oscore]. This parameter MUST specify a single value.
- o 'app-gp', specifying the name(s) of the application group(s) associated to the security group of interest indicated by 'sec-

gp'. This parameter MUST occur once for each application group, and MUST specify only a single application group.

When a security group is created at the GM, the names of the application groups using it are also specified as part of the security group configuration (see [I-D.ietf-ace-oscore-gm-admin]). Thus, when registering the links to its group-membership resource, the GM is aware of the application groups and their names.

If a different entity than the GM registers the security groups to the RD, e.g. a Commissioning Tool, this entity has to also be aware of the application groups and their names to specify. To this end, it can obtain them from the GM or from the Administrator that created the security groups at the GM (see [I-D.ietf-ace-oscore-gm-admin]).

Optionally, the following parameters can also be specified.

- o 'cs_alg', specifying the algorithm used to countersign messages in the security group. If present, this parameter MUST specify a single value encoded as a text string, which is taken from the 'Value' column of the "COSE Algorithms" Registry [COSE.Algorithms].
- o 'cs_alg_crv', specifying the elliptic curve (if applicable) for the algorithm used to countersign messages in the security group. If present, this parameter MUST specify a single value encoded as a text string, which is taken from the 'Value' column of the "COSE Elliptic Curves" Registry [COSE.Elliptic.Curves].
- o 'cs_key_kty', specifying the key type of countersignature keys used to countersign messages in the security group. If present, this parameter MUST specify a single value encoded as a text string, which is taken from the 'Value' column of the "COSE Key Types" Registry [COSE.Key.Types].
- o 'cs_key_crv', specifying the elliptic curve (if applicable) of countersignature keys used to countersign messages in the security group. If present, this parameter MUST specify a single value encoded as a text string, which is taken from the 'Value' column of the "COSE Elliptic Curves" Registry defined in [COSE.Elliptic.Curves].
- o 'cs_kenc', specifying the encoding of the public keys used in the security group. If present, this parameter MUST specify a single value encoded as a text string. This specification explicitly admits the signaling of COSE Keys [I-D.ietf-cose-rfc8152bis-struct] as encoding for public keys,

which is indicated with "1", as taken from the 'Confirmation Key' column of the "CWT Confirmation Method" Registry defined in [RFC8747]. Future specifications may define additional values for this parameter.

- o 'alg', specifying the AEAD algorithm used in the security group. If present, this parameter MUST specify a single value encoded as a text string, which is taken from the 'Value' column of the "COSE Algorithms" Registry [COSE.Algorithms].
- o 'hkdf', specifying the HKDF algorithm used in the security group. If present, this parameter MUST specify a single value encoded as a text string, which is taken from the 'Value' column of the "COSE Algorithms" Registry defined in [COSE.Algorithms].

Note that the values registered in the COSE Registries [COSE.Algorithms][COSE.Elliptic.Curves][COSE.Key.Types] are strongly typed. On the contrary, Link Format is weakly typed and thus does not distinguish between, for instance, the string value "-10" and the integer value -10.

Thus, in RDs that return responses in Link Format, string values which look like an integer are not supported. Therefore, such values MUST NOT be advertised through the corresponding parameters above.

A CoAP endpoint that queries the RD to discover security groups and their group-membership resource to access (see Section 4) would benefit from the information above as follows.

- o The values of 'cs_alg', 'cs_alg_crv', 'cs_key_kty', 'cs_key_crv' and 'cs_kenc' related to a group-membership resource provide an early knowledge of the format and encoding of public keys used in the security group. Thus, the CoAP endpoint does not need to ask the GM for this information as a preliminary step before the joining process, or to perform a trial-and-error joining exchange with the GM. Hence, the CoAP endpoint is able to provide the GM with its own public key in the correct expected format and encoding at the very first step of the joining process.
- o The values of 'cs_alg', 'alg' and 'hkdf' related to a group-membership resource provide an early knowledge of the algorithms used in the security group. Thus, the CoAP endpoint is able to decide whether to actually proceed with the joining process, depending on its support for the indicated algorithms.

2.2. Relation Link to Authorization Server

For each registered link to a group-membership resource, the GM MAY additionally specify the link to the ACE Authorization Server (AS) [I-D.ietf-ace-oauth-authz] associated to the GM, and issuing authorization credentials to join the security group as described in [I-D.ietf-ace-key-groupcomm-oscore].

The link to the AS has as target the URI of the resource where to send an authorization request to.

In the RD defined in [I-D.ietf-core-resource-directory] and based on Link Format, the link to the AS is separately registered with the RD, and includes the following parameters as target attributes.

- o 'rel', with value "authorization_server".
- o 'anchor', with value the target of the link to the group-membership resource at the GM.

In a RD based on CoRAL, such as the one defined in [I-D.hartke-t2trg-coral-reef], this is mapped (as describe there) to a link from the registration resource to the AS, using the <http://www.iana.org/assignments/relation/authorization_server> link relation type.

2.3. Registration Example

The example below shows a GM with endpoint name "gm1" and address 2001:db8::ab that registers with the RD.

The GM specifies the value of the 'sec-gp' parameter for accessing the security group with name "feedca570000", and used by the application group with name "group1" specified with the value of the 'app-gp' parameter. The countersignature algorithm used in the security group is EdDSA, with elliptic curve Ed25519 and keys of type OKP. Public keys used in the security group are encoded as COSE Keys [I-D.ietf-cose-rfc8152bis-struct].

In addition, the GM specifies the link to the ACE Authorization Server associated to the GM, to which a CoAP endpoint should send an Authorization Request for joining the corresponding security group (see [I-D.ietf-ace-key-groupcomm-oscore]).

2.3.1. Example in Link Format

Request: GM -> RD

Req: POST coap://rd.example.com/rd?ep=gml

Content-Format: 40

Payload:

```
</ace-group/feedca570000>;ct=41;rt="core.osc.gm";if="ace.group";
    sec-gp="feedca570000";app-gp="group1";
    cs_alg="-8";cs_alg_crv="6";
    cs_key_kty="1";cs_key_crv=6";
    cs_kenc="1",
<coap://as.example.com/token>;
    rel="authorization-server";
    anchor="coap://[2001:db8::ab]/ace-group/feedca570000"
```

Response: RD -> GM

Res: 2.01 Created

Location-Path: /rd/4521

2.3.2. Example in CoRAL

Request: GM -> RD

Req: POST coap://rd.example.com/rd?ep=gml

Content-Format: TBD123456 (application/coral+cbor)

Payload:

```
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>
#using iana = <http://www.iana.org/assignments/relation/>

#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedca570000> {
    reef:rt "core.osc.gm"
    reef:if "ace.group"
    sec-gp "feedca570000"
    app-gp "group1"
    cs_alg -8
    cs_alg_crv 6
    cs_key_kty 1
    cs_key_crv 6
    cs_kenc 1
    iana:authorization-server <coap://as.example.com/token>
}
```

Response: RD -> GM

Res: 2.01 Created
Location-Path: /rd/4521

3. Addition and Update of Security Groups

The GM is responsible to refresh the registration of all its group-membership resources in the RD. This means that the GM has to update the registration within its lifetime as per Section 5.3.1 of [I-D.ietf-core-resource-directory], and has to change the content of the registration when a group-membership resource is added/removed, or if its parameters have to be changed, such as in the following cases.

- o The GM creates a new security group and starts exporting the related group-membership resource.
- o The GM dismisses a security group and stops exporting the related group-membership resource.
- o Information related to an existing security group changes, e.g. the list of associated application groups.

To perform an update of its registrations, the GM can re-register with the RD and fully specify all links to its group-membership resources.

Alternatively, the GM can perform a PATCH/iPATCH [RFC8132] request to the RD, as per Section 5.3.3 of [I-D.ietf-core-resource-directory]. This requires new media-types to be defined in future standards, to apply a new document as a patch to an existing stored document.

3.1. Addition Example

The example below shows how the GM from Section 2 re-registers with the RD. When doing so, it specifies:

- o The same previous group-membership resource associated to the security group with name "feedca570000".
- o An additional group-membership resource associated to the security group with name "ech0ech00000" and used by the application group "group2".
- o A third group-membership resource associated with the security group with name "abcdef120000" and used by two application groups, namely "group3" and "group4".

Furthermore, the GM relates the same Authorization Server also to the security groups "ech0ech00000" and "abcdef120000".

3.1.1. Example in Link Format

Request: GM -> RD

Req: POST coap://rd.example.com/rd?ep=gml

Content-Format: 40

Payload:

```
</ace-group/feedca570000>;ct=41;rt="core.osc.gm";if="ace.group";
    sec-gp="feedca570000";app-gp="group1";
    cs_alg="-8";cs_alg_crv="6";
    cs_key_kty="1";cs_key_crv="6";
    cs_kenc="1",
</ace-group/ech0ech00000>;ct=41;rt="core.osc.gm";if="ace.group";
    sec-gp="ech0ech00000";app-gp="group2";
    cs_alg="-8";cs_alg_crv="6";
    cs_key_kty="1";cs_key_crv="6";
    cs_kenc="1",
</ace-group/abcdef120000>;ct=41;rt="core.osc.gm";if="ace.group";
    sec-gp="abcdef120000";app-gp="group3";
    app-gp="group4";cs_alg="-8";
    cs_alg_crv="6";cs_key_kty="1";
    cs_key_crv="6";cs_kenc="1",
<coap://as.example.com/token>;
    rel="authorization-server";
    anchor="coap://[2001:db8::ab]/ace-group/feedca570000",
<coap://as.example.com/token>;
    rel="authorization-server";
    anchor="coap://[2001:db8::ab]/ace-group/ech0ech00000",
<coap://as.example.com/token>;
    rel="authorization-server";
    anchor="coap://[2001:db8::ab]/ace-group/abcdef120000"
```

Response: RD -> GM

Res: 2.04 Changed

Location-Path: /rd/4521

3.1.2. Example in CoRAL

Request: GM -> RD

Req: POST coap://rd.example.com/rd?ep=gml
 Content-Format: TBD123456 (application/coral+cbor)

Payload:

```
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>
#using iana = <http://www.iana.org/assignments/relation/>

#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedca570000> {
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "feedca570000"
  app-gp "group1"
  cs_alg -8
  cs_alg_crv 6
  cs_key_kty 1
  cs_key_crv 6
  cs_kenc 1
  iana:authorization-server <coap://as.example.com/token>
}
reef:rd-item </ace-group/ech0ech000000> {
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "ech0ech000000"
  app-gp "group2"
  cs_alg -8
  cs_alg_crv 6
  cs_key_kty 1
  cs_key_crv 6
  cs_kenc 1
  iana:authorization-server <coap://as.example.com/token>
}
reef:rd-item </ace-group/abcdef120000> {
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "abcdef120000"
  app-gp "group3"
  app-gp "group4"
  cs_alg -8
  cs_alg_crv 6
  cs_key_kty 1
  cs_key_crv 6
  cs_kenc 1
  iana:authorization-server <coap://as.example.com/token>
}
```

Response: RD -> GM

Res: 2.04 Changed
Location-Path: /rd/4521

4. Discovery of Security Groups

A CoAP endpoint that wants to join a security group, hereafter called the joining node, might not have all the necessary information at deployment time. Also, it might want to know about possible new security groups created afterwards by the respective Group Managers.

To this end, the joining node can perform a resource lookup at the RD as per Section 6.1 of [I-D.ietf-core-resource-directory], to retrieve the missing pieces of information needed to join the security group(s) of interest. The joining node can find the RD as described in Section 4 of [I-D.ietf-core-resource-directory].

The joining node uses the following parameter value for the lookup filtering.

- o 'rt' = "core.osc.gm", specifying the resource type of the group-membership resource at the Group Manager, with value "core.osc.gm" registered in Section 21.11 of [I-D.ietf-ace-key-groupcomm-oscore].

The joining node may additionally consider the following parameters for the lookup filtering, depending on the information it has already available.

- o 'sec-gp', specifying the name of the security group of interest. This parameter MUST specify a single value.
- o 'ep', specifying the registered endpoint of the GM.
- o 'app-gp', specifying the name(s) of the application group(s) associated with the security group of interest. This parameter MAY be included multiple times, and each occurrence MUST specify the name of one application group.
- o 'if', specifying the interface description for accessing the group-membership resource at the Group Manager, with value "ace.group" registered in Section 8.10 of [I-D.ietf-ace-key-groupcomm].

The response from the RD may include links to a group-membership resource specifying multiple application groups, as all using the same security group. In this case, the joining node is already expected to know the exact application group of interest.

Furthermore, the response from the RD may include the links to different group-membership resources, all specifying a same application group of interest for the joining node, if the corresponding security groups are all used by that application group.

In this case, application policies on the joining node should define how to determine the exact security group to join (see Section 2.1 of [I-D.ietf-core-groupcomm-bis]). For example, different security groups can reflect different security algorithms to use. Hence, a client application can take into account what the joining node supports and prefers, when selecting one particular security group among the indicated ones, while a server application would need to join all of them. Later on, the joining node will be anyway able to join only security groups for which it is actually authorized to be a member (see [I-D.ietf-ace-key-groupcomm-oscore]).

Note that, with RD-based discovery, including the 'app-gp' parameter multiple times would result in finding only the group-membership resource that serves all the specified application groups, i.e. not any group-membership resource that serves either. Therefore, a joining node needs to perform N separate queries with different values for 'app-gp', in order to safely discover the (different) group-membership resource(s) serving the N application groups.

4.1. Discovery Example #1

Consistently with the examples in Section 2 and Section 3, the examples below consider a joining node that wants to join the security group associated with the application group "group1", but that does not know the name of the security group, the responsible GM and the group-membership resource to access.

4.1.1. Example in Link Format

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res
?rt=core.osc.gm&app-gp=group1

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/ace-group/feedca570000>;rt="core.osc.gm";
  if="ace.group";sec-gp="feedca570000";app-gp="group1";
  cs_alg="-8";cs_alg_crv="6";cs_key_kty="1";cs_key_crv=6;
  cs_kenc="1";anchor="coap://[2001:db8::ab]"
```

By performing the separate resource lookup below, the joining node can retrieve the link to the ACE Authorization Server associated to the GM, where to send an Authorization Request for joining the corresponding security group (see [I-D.ietf-ace-key-groupcomm-oscore]).

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res
?rel="authorization-server"&
anchor="coap://[2001:db8::ab]/ace-group/feedca570000"

Response: RD -> Joining node

Res: 2.05 Content
Payload:
<coap://as.example.com/token>

To retrieve the multicast IP address of the CoAP group used by the application group "group1", the joining node performs an endpoint lookup as shown below. The following assumes that the application group "group1" had been previously registered as per Appendix A of [I-D.ietf-core-resource-directory], with ff35:30:2001:db8::23 as multicast IP address of the associated CoAP group.

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/ep
?et=core.rd-group&ep=group1

Response: RD -> Joining node

Res: 2.05 Content
Payload:
</rd/501>;ep="group1";et="core.rd-group";
base="coap://[ff35:30:2001:db8::23]"

4.1.2. Example in CoRAL

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res
?rt=core.osc.gm&app-gp=group1
Accept: TBD123456 (application/coral+cbor)

Response: RD -> Joining node

Res: 2.05 Content

Content-Format: TBD123456 (application/coral+cbor)

Payload:

#using <http://coreapps.org/core.oscore-discovery#>

#using reef = <http://coreapps.org/reef#>

#using iana = <http://www.iana.org/assignments/relation/>

#base <coap://[2001:db8::ab]/>

reef:rd-item </ace-group/feedca570000> {

 reef:rt "core.osc.gm"

 reef:if "ace.group"

 sec-gp "feedca570000"

 app-gp "group1"

 cs_alg -8

 cs_alg_crv 6

 cs_key_kty 1

 cs_key_crv 6

 cs_kenc 1

 iana:authorization-server <coap://as.example.com/token>

}

To retrieve the multicast IP address of the CoAP group used by the application group "group1", the joining node performs an endpoint lookup as shown below. The following assumes that the application group "group1" had been previously registered, with ff35:30:2001:db8::23 as multicast IP address of the associated CoAP group.

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/ep
 ?et=core.rd-group&ep=group1

Accept: TBD123456 (application/coral+cbor)

Response: RD -> Joining node

```
Res: 2.05 Content
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>

reef:rd-unit <./rd/501> {
  reef:ep="group1"
  reef:et="core.rd-group"
  reef:base <coap://[ff35:30:2001:db8::23]>
}
```

4.2. Discovery Example #2

Consistently with the examples in Section 2 and Section 3, the examples below consider a joining node that wants to join the security group with name "feedca570000", but that does not know the responsible GM, the group-membership resource to access, and the associated application groups.

The examples also show how the joining node uses CoAP observation [RFC7641], in order to be notified of possible changes to the parameters of the group-membership resource. This is also useful to handle the case where the security group of interest has not been created yet, so that the joining node can receive the requested information when it becomes available.

4.2.1. Example in Link Format

Request: Joining node -> RD

```
Req: GET coap://rd.example.com/rd-lookup/res
     ?rt=core.osc.gm&sec-gp=feedca570000
Observe: 0
```

Response: RD -> Joining node

```
Res: 2.05 Content
Observe: 24
Payload:
<coap://[2001:db8::ab]/ace-group/feedca570000>;rt="core.osc.gm";
  if="ace.group";sec-gp="feedca570000";app-gp="group1";
  cs_alg="-8";cs_alg_crv="6";cs_key_kty="1";cs_key_crv="6";
  cs_kenc="1";anchor="coap://[2001:db8::ab]"
```

Depending on the search criteria, the joining node performing the resource lookup can get large responses. This can happen, for

instance, when the lookup request targets all the group-membership resources at a specified GM, or all the group-membership resources of all the registered GMs, as in the example below.

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res?rt=core.osc.gm

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/ace-group/feedca570000>;rt="core.osc.gm";
  if="ace.group";sec-gp="feedca570000";app-gp="group1";
  cs_alg="-8";cs_alg_crv="6";cs_key_kty="1";cs_key_crv=6";
  cs_kenc="1";anchor="coap://[2001:db8::ab]",
<coap://[2001:db8::ab]/ace-group/ech0ech00000>;rt="core.osc.gm";
  if="ace.group";sec-gp="ech0ech00000";app-gp="group2";
  cs_alg="-8";cs_alg_crv="6";cs_key_kty="1";cs_key_crv=6";
  cs_kenc="1";anchor="coap://[2001:db8::ab]",
<coap://[2001:db8::ab]/ace-group/abcdef120000>;rt="core.osc.gm";
  if="ace.group";sec-gp="abcdef120000";app-gp="group3";
  app-gp="group4";cs_alg="-8";cs_alg_crv="6";cs_key_kty="1";
  cs_key_crv=6";cs_kenc="1";anchor="coap://[2001:db8::ab]"
```

Therefore, it is RECOMMENDED that a joining node which performs a resource lookup with the CoAP Observe option specifies the value of the parameter 'sec-gp' in its GET request sent to the RD.

4.2.2. Example in CoRAL

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res
?rt=core.osc.gm&sec-gp=feedca570000

Accept: TBD123456 (application/coral+cbor)

Observe: 0

Response: RD -> Joining node

```
Res: 2.05 Content
Observe: 24
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>
#using iana = <http://www.iana.org/assignments/relation/>

#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedca570000> {
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "feedca570000"
  app-gp "group1"
  cs_alg -8
  cs_alg_crv 6
  cs_key_kty 1
  cs_key_crv 6
  cs_kenc 1
  iana:authorization-server <coap://as.example.com/token>
}
```

5. Use Case Example With Full Discovery

In this section, the discovery of security groups is described to support the installation process of a lighting installation in an office building. The described process is a simplified version of one of many processes.

The process described in this section is intended as an example and does not have any particular ambition to serve as recommendation or best practice to adopt. That is, it shows a possible workflow involving a Commissioning Tool (CT) used in a certain way, while it is not meant to prescribe how the workflow should necessarily be.

Assume the existence of four luminaires that are members of two application groups. In the first application group, the four luminaires receive presence messages and light intensity messages from sensors or their proxy. In the second application group, the four luminaires and several other pieces of equipment receive building state schedules.

Each of the two application groups is associated to a different security group and to a different CoAP group with its own dedicated multicast IP address.

The Fairhair Alliance describes how a new device is accepted and commissioned in the network [Fairhair], by means of its certificate stored during the manufacturing process. When commissioning the new device in the installation network, the new device gets a new identity defined by a newly allocated certificate, following the BRSKI specification.

Section 7.3 of [I-D.ietf-core-resource-directory] describes how the CT assigns an endpoint name based on the CN field, (CN=ACME) and the serial number of the certificate (serial number = 123x, with $3 < x < 8$). Corresponding ep-names ACME-1234, ACME-1235, ACME-1236 and ACME-1237 are also assumed.

It is common practice that locations in the building are specified according to a coordinate system. After the acceptance of the luminaires into the installation network, the coordinate of each device is communicated to the CT. This can be done manually or automatically.

The mapping between location and ep-name is calculated by the CT. For instance, on the basis of grouping criteria, the CT assigns: i) application group "grp_R2-4-015" to the four luminaires; and ii) application group "grp_schedule" to all schedule requiring devices. Also, the device with ep name ACME-123x has been assigned IP address: [2001:db8:4::x]. The RD is assigned IP address: [2001:db8:4:ff]. The used multicast addresses are: [ff05::5:1] and [ff05::5:2].

The following assumes that each device is pre-configured with the name of the two application groups it belongs to. Additional mechanisms can be defined in the RD, for supporting devices to discover the application groups they belong to.

Appendix A provides this same use case example in CoRAL.

*** **

The CT defines the application group "grp_R2-4-015", with resource /light and base address [ff05::5:1], as follows.

Request: CT -> RD

```
Req: POST coap://[2001:db8:4::ff]/rd
    ?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::5:1]
Content-Format: 40
Payload:
</light>;rt="oic.d.light"
```

Response: RD -> CT

```
Res: 2.01 Created
Location-Path: /rd/501
```

Also, the CT defines a second application group "grp_schedule", with resource /schedule and base address [ff05::5:2], as follows.

Request: CT \rightarrow RD

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=grp_schedule&et=core.rd-group&base=coap://[ff05::5:2]
Content-Format: 40
Payload:
</schedule>;rt="oic.r.time.period"
```

Response: RD \rightarrow CT

```
Res: 2.01 Created
Location-Path: /rd/502
```

*** **

Finally, the CT defines the corresponding security groups. In particular, assuming a Group Manager responsible for both security groups and with address [2001:db8::ab], the CT specifies:

Request: CT \rightarrow RD

```

Req: POST coap://[2001:db8:4::ff]/rd
    ?ep=gml&base=coap://[2001:db8::ab]
Content-Format: 40
Payload:
</ace-group/feedca570000>;ct=41;rt="core.osc.gm";if="ace.group";
    sec-gp="feedca570000";
    app-gp="grp_R2-4-015",
</ace-group/feedsc590000>;ct=41;rt="core.osc.gm";if="ace.group";
    sec-gp="feedsc590000";
    app-gp="grp_schedule"

```

Response: RD \rightarrow CT

Res: 2.01 Created
Location-Path: /rd/4521

*** **

The device with IP address [2001:db8:4::x] can retrieve the multicast IP address of the CoAP group used by the application group "grp_R2-4-015", by performing an endpoint lookup as shown below.

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
?et=core.rd-group&ep=grp_R2-4-015

Response: RD -> Joining node

Res: 2.05 Content
Content-Format: 40
Payload:
</rd/501>;ep="grp_R2-4-015";et="core.rd-group";
base="coap://[ff05::5:1]"

Similarly, to retrieve the multicast IP address of the CoAP group used by the application group "grp_schedule", the device performs an endpoint lookup as shown below.

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
?et=core.rd-group&ep=grp_schedule

Response: RD -> Joining node

Res: 2.05 Content
Content-Format: 40
Payload:
</rd/502>;ep="grp_schedule";et="core.rd-group";
base="coap://[ff05::5:2]"

*** *** *** *** *** *** *** *** *** *** *** *** *** *** ***

Consequently, the device learns the security groups it has to join. In particular, it does the following for app-gp="grp_R2-4-015".

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
?rt=core.osc.gm&app-gp=grp_R2-4-015

Response: RD -> Joining Node

Res: 2.05 Content
Content-Format: 40
Payload:
<coap://[2001:db8::ab]/ace-group/feedca570000>;
rt="core.osc.gm";if="ace.group";sec-gp="feedca570000";
app-gp="grp_R2-4-015";anchor="coap://[2001:db8::ab]"

Similarly, the device does the following for app-gp="grp_schedule".

Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
 ?rt=core.osc.gm&app-gp=grp_schedule

Response: RD -> Joining Node

Res: 2.05 Content
Content-Format: 40
Payload:

```
<coap://[2001:db8::ab]/ace-group/feedsc590000>;  
  rt="core.osc.gm";if="ace.group";sec-gp="feedsc590000";  
  app-gp="grp_schedule";anchor="coap://[2001:db8::ab]"
```

*** **

After this last discovery step, the device can ask permission to join the security groups, and effectively join them through the Group Manager, e.g. according to [I-D.ietf-ace-key-groupcomm-oscore].

6. Security Considerations

The security considerations as described in Section 8 of [I-D.ietf-core-resource-directory] apply here as well.

7. IANA Considerations

This document has no actions for IANA.

8. References

8.1. Normative References

[COSE.Algorithms]

IANA, "COSE Algorithms",
<<https://www.iana.org/assignments/cose/cose.xhtml#algorithms>>.

[COSE.Elliptic.Curves]

IANA, "COSE Elliptic Curves",
<<https://www.iana.org/assignments/cose/cose.xhtml#elliptic-curves>>.

[COSE.Key.Types]

IANA, "COSE Key Types",
<<https://www.iana.org/assignments/cose/cose.xhtml#key-type>>.

- [I-D.ietf-core-coral]
Hartke, K., "The Constrained RESTful Application Language (CoRAL)", draft-ietf-core-coral-03 (work in progress), March 2020.
- [I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-ietf-core-groupcomm-bis-02 (work in progress), November 2020.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-10 (work in progress), November 2020.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Kostner, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-25 (work in progress), July 2020.
- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", draft-ietf-cose-rfc8152bis-algs-12 (work in progress), September 2020.
- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", draft-ietf-cose-rfc8152bis-struct-14 (work in progress), September 2020.
- [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>>.
- [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>>.
- [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>>.

- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.

8.2. Informative References

- [Fairhair]
FairHair Alliance, "Security Architecture for the Internet of Things (IoT) in Commercial Buildings", White Paper, ed. Piotr Polak , March 2018, <https://openconnectivity.org/wp-content/uploads/2019/11/fairhair_security_wp_march-2018.pdf>.
- [I-D.hartke-t2trg-coral-reef]
Hartke, K., "Resource Discovery in Constrained RESTful Environments (CoRE) using the Constrained RESTful Application Language (CoRAL)", draft-hartke-t2trg-coral-reef-04 (work in progress), May 2020.
- [I-D.ietf-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-ietf-ace-key-groupcomm-10 (work in progress), November 2020.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-09 (work in progress), November 2020.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-35 (work in progress), June 2020.
- [I-D.ietf-ace-oscore-gm-admin]
Tiloca, M., Hoeglund, R., Stok, P., Palombini, F., and K. Hartke, "Admin Interface for the OSCORE Group Manager", draft-ietf-ace-oscore-gm-admin-01 (work in progress), November 2020.
- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-44 (work in progress), September 2020.

- [I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work in progress), September 2020.
- [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>>.
- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

Appendix A. Use Case Example With Full Discovery (CoRAL)

This section provides the same use case example of Section 5, but specified in CoRAL [I-D.ietf-core-coral].

*** **

The CT defines the application group "grp_R2-4-015", with resource /light and base address [ff05::5:1], as follows.

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using reef = <http://coreapps.org/reef#>

```
#base <coap://[ff05::5:1]/>
reef:ep "grp_R2-4-015"
reef:et "core.rd-group"
reef:rd-item </light> {
    reef:rt "oic.d.light"
}
```

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/501

Also, the CT defines a second application group "grp_schedule", with resource /schedule and base address [ff05::5:2], as follows.

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd?ep=grp_schedule&et=core.rd-group
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using reef = <http://coreapps.org/reef#>

```
#base <coap://[ff05::5:2]/>
reef:rd-item </schedule> {
    reef:rt "oic.r.time.period"
}
```

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/502

*** **

Finally, the CT defines the corresponding security groups. In particular, assuming a Group Manager responsible for both security groups and with address [2001:db8::ab], the CT specifies:

Request: CT -> RD

```
Req: POST coap://[2001:db8:4::ff]/rd?ep=gml
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>

#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedca570000> {
    reef:ct 41
    reef:rt "core.osc.gm"
    reef:if "ace.group"
    sec-gp "feedca570000"
    app-gp "grp_R2-4-015"
}
reef:rd-item </ace-group/feedsc590000> {
    reef:ct 41
    reef:rt "core.osc.gm"
    reef:if "ace.group"
    sec-gp "feedsc590000"
    app-gp "grp_schedule"
}
```

```
Res: 2.05 Content
Content-Format: TBD123456 (application/coral+cbor)
```

```
Payload:
#using reef = <http://coreapps.org/reef#>
```

```
#base <coap://[2001:db8:4::ff]/rd/>
reef:rd-unit <501> {
  reef:ep "grp_R2-4-015"
  reef:et "core.rd-group"
  reef:base <coap://[ff05::5:1]/>
}
```

Similarly, to retrieve the multicast IP address of the CoAP group used by the application group "grp_schedule", the device performs an endpoint lookup as shown below.

Request: Joining node -> RD

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
      ?et=core.rd-group&ep=grp_schedule
```

Response: RD -> Joining node

```
Res: 2.05 Content
Content-Format: TBD123456 (application/coral+cbor)
```

```
Payload:
#using reef = <http://coreapps.org/reef#>
```

```
#base <coap://[2001:db8:4::ff]/rd/>
reef:rd-unit <501> {
  reef:ep "grp_schedule"
  reef:et "core.rd-group"
  reef:base <coap://[ff05::5:2]/>
}
```

*** **

Consequently, the device learns the security groups it has to join. In particular, it does the following for app-gp="grp_R2-4-015".

Request: Joining node -> RD

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
      ?rt=core.osc.gm&app-gp=grp_R2-4-015
```

Response: RD -> Joining Node

```
Res: 2.05 Content
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>

#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedca570000> {
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "feedca570000"
  app-gp "grp_R2-4-015"
}
```

Similarly, the device does the following for app-gp="grp_schedule".

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
      ?rt=core.osc.gm&app-gp=grp_schedule
```

Response: RD -> Joining Node

```
Res: 2.05 Content
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>

#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedsc590000> {
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "feedsc590000"
  app-gp "grp_schedule"
}
```

*** **

After this last discovery step, the device can ask permission to join the security groups, and effectively join them through the Group Manager, e.g. according to [I-D.ietf-ace-key-groupcomm-oscore].

Acknowledgments

The authors sincerely thank Carsten Bormann, Klaus Hartke, Jaime Jimenez, Francesca Palombini, Dave Robin and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; by the H2020 project SIFIS-Home (Grant agreement 952652); and by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Christian Amsuess
Hollandstr. 12/4
Vienna 1020
Austria

Email: christian@amsuess.com

Peter van der Stok
Consultant

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2022

M. Tiloca
RISE AB
C. Amsuess

P. van der Stok
Consultant
7 March 2022

Discovery of OSCORE Groups with the CoRE Resource Directory
draft-tiloca-core-oscore-discovery-11

Abstract

Group communication over the Constrained Application Protocol (CoAP) can be secured by means of Group Object Security for Constrained RESTful Environments (Group OSCORE). At deployment time, devices may not know the exact security groups to join, the respective Group Manager, or other information required to perform the joining process. This document describes how a CoAP endpoint can use descriptions and links of resources registered at the CoRE Resource Directory to discover security groups and to acquire information for joining them through the respective Group Manager. A given security group may protect multiple application groups, which are separately announced in the Resource Directory as sets of endpoints sharing a pool of resources. This approach is consistent with, but not limited to, the joining of security groups based on the ACE framework for Authentication and Authorization in constrained environments.

Discussion Venues

This note is to be removed before publishing as an RFC.

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

Source for this draft and an issue tracker can be found at <https://gitlab.com/crimson84/draft-tiloca-core-oscore-discovery>.

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	5
2. Registration of Group Manager Endpoints	6
2.1. Parameters	7
2.2. Relation Link to Authorization Server	12
2.3. Registration Example	12
2.3.1. Example in Link Format	13
2.3.2. Example in CoRAL	13
3. Addition and Update of Security Groups	14
3.1. Addition Example	14
3.1.1. Example in Link Format	15
3.1.2. Example in CoRAL	15
4. Discovery of Security Groups	17
4.1. Discovery Example #1	18
4.1.1. Example in Link Format	19
4.1.2. Example in CoRAL	20
4.2. Discovery Example #2	21
4.2.1. Example in Link Format	21
4.2.2. Example in CoRAL	22
5. Use Case Example With Full Discovery	23

6. Security Considerations	28
7. IANA Considerations	28
8. References	28
8.1. Normative References	28
8.2. Informative References	29
Appendix A. Use Case Example With Full Discovery (CoRAL)	32
Acknowledgments	36
Authors' Addresses	36

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports group communication over IP multicast [I-D.ietf-core-groupcomm-bis] to improve efficiency and latency of communication and reduce bandwidth requirements. A set of CoAP endpoints constitutes an application group by sharing a common pool of resources, that can be efficiently accessed through group communication. The members of an application group may be members of a security group, thus sharing a common set of keying material to secure group communication.

The security protocol Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm] builds on OSCORE [RFC8613] and protects CoAP messages end-to-end in group communication contexts through CBOR Object Signing and Encryption (COSE)

[I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs]. An application group may rely on one or more security groups, and a same security group may be used by multiple application groups at the same time.

A CoAP endpoint relies on a Group Manager (GM) to join a security group and get the group keying material. The joining process in [I-D.ietf-ace-key-groupcomm-oscore] is based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz], with the joining endpoint and the GM acting as ACE Client and Resource Server, respectively. That is, the joining endpoint accesses the group-membership resource exported by the GM and associated with the security group to join.

Typically, devices store a static X509 IDevID certificate installed at manufacturing time [RFC8995]. This is used at deployment time during an enrollment process that provides the devices with an Operational Certificate, possibly updated during the device lifetime. Operational Certificates may specify information to join security groups, especially a reference to the group-membership resources to access at the respective GMs.

However, it is usually impossible to provide such precise information to freshly deployed devices, as part of their (early) Operational Certificate. This can be due to a number of reasons: (1) the security group(s) to join and the responsible GM(s) are generally unknown at manufacturing time; (2) a security group of interest is created, or the responsible GM is deployed, only after the device is enrolled and fully operative in the network; (3) information related to existing security groups or to their GMs has changed. This requires a method for CoAP endpoints to dynamically discover security groups and their GM, and to retrieve relevant information about deployed groups.

To this end, CoAP endpoints can use descriptions and links of group-membership resources at GMs, to discover security groups and retrieve the information required for joining them. With the discovery process of security groups expressed in terms of links to resources, the remaining problem is the discovery of those links. The CoRE Resource Directory (RD) [I-D.ietf-core-resource-directory] allows such discovery in an efficient way, and it is expected to be used in many setups that would benefit of security group discovery.

This specification builds on this approach and describes how CoAP endpoints can use the RD to perform the link discovery steps, in order to discover security groups and retrieve the information required to join them through their GM. In short, the GM registers as an endpoint with the RD. The resulting registration resource includes one link per security group under that GM, specifying the path to the related group-membership resource to access for joining that group.

Additional descriptive information about the security group is stored with the registered link. In a RD based on Link Format [RFC6690] as defined in [I-D.ietf-core-resource-directory], this information is specified as target attributes of the registered link, and includes the identifiers of the application groups which use that security group. This enables a lookup of those application groups at the RD, where they are separately announced by a Commissioning Tool (see Appendix A of [I-D.ietf-core-resource-directory]).

When querying the RD for security groups, a CoAP endpoint can use CoAP observation [RFC7641]. This results in automatic notifications on the creation of new security groups or the update of existing groups. Thus, it facilitates the early deployment of CoAP endpoints, i.e., even before the GM is deployed and security groups are created.

Interaction examples are provided in Link Format, as well as in the Constrained RESTful Application Language CoRAL [I-D.ietf-core-coral] with reference to a CoRAL-based RD [I-D.hartke-t2trg-coral-reef]. While all the CoRAL examples show the CoRAL textual serialization format, its binary serialization format is used on the wire.

[NOTE:

The reported CoRAL examples are based on the textual representation used until version -03 of [I-D.ietf-core-coral]. These will be revised to use the CBOR diagnostic notation instead.

]

The approach in this document is consistent with, but not limited to, the joining of security groups defined in [I-D.ietf-ace-key-groupcomm-oscore].

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.

This specification requires readers to be familiar with the terms and concepts discussed in [I-D.ietf-core-resource-directory] and [RFC6690], as well as in [I-D.ietf-core-coral]. Readers should also be familiar with the terms and concepts discussed in [RFC7252] [I-D.ietf-core-groupcomm-bis], [I-D.ietf-core-oscore-groupcomm] and [I-D.ietf-ace-key-groupcomm-oscore].

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

Consistently with the definitions from Section 2.1 of [I-D.ietf-core-groupcomm-bis], this document also refers to the following terminology.

- * CoAP group: a set of CoAP endpoints all configured to receive CoAP multicast messages sent to the group's associated IP multicast address and UDP port. An endpoint may be a member of multiple CoAP groups by subscribing to multiple IP multicast addresses.

- * **Security group:** a set of CoAP endpoints that share the same security material, and use it to protect and verify exchanged messages. A CoAP endpoint may be a member of multiple security groups. There can be a one-to-one or a one-to-many relation between security groups and CoAP groups.

This document especially considers a security group to be an OSCORE group, where all members share one OSCORE Security Context to protect group communication with Group OSCORE [I-D.ietf-core-oscore-groupcomm]. However, the approach defined in this document can be used to support the discovery of different security groups than OSCORE groups.

- * **Application group:** a set of CoAP endpoints that share a common set of resources. An endpoint may be a member of multiple application groups. An application group can be associated with one or more security groups, and multiple application groups can use the same security group. Application groups are announced in the RD by a Commissioning Tool, according to the RD-Groups usage pattern (see Appendix A of [I-D.ietf-core-resource-directory]).

Like [I-D.ietf-core-oscore-groupcomm], this document also uses the following term.

- * **Authentication credential:** set of information associated with an entity, including that entity's public key and parameters associated with the public key. Examples of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claims Sets (CCSs) [RFC8392], X.509 certificates [RFC7925] and C509 certificates [I-D.ietf-cose-cbor-encoded-cert].

2. Registration of Group Manager Endpoints

During deployment, a Group Manager (GM) can find the CoRE Resource Directory (RD) as described in Section 4 of [I-D.ietf-core-resource-directory].

Afterwards, the GM registers as an endpoint with the RD, as described in Section 5 of [I-D.ietf-core-resource-directory]. The GM SHOULD NOT use the Simple Registration approach described in Section 5.1 of [I-D.ietf-core-resource-directory].

When registering with the RD, the GM also registers the links to all the group-membership resources it has at that point in time, i.e., one for each of its security groups.

In the registration request, each link to a group-membership resource has as target the URI of that resource at the GM. Also, it specifies a number of descriptive parameters as defined in Section 2.1.

Furthermore, the GM MAY additionally register the link to its resource implementing the ACE authorization information endpoint (see Section 5.10.1 of [I-D.ietf-ace-oauth-authz]). A joining node can provide the GM with its own access token by sending it in a request targeting that resource, thus proving to be authorized to join certain security groups (see Section 6.1 of [I-D.ietf-ace-key-groupcomm-oscore]). In such a case, the link MUST include the parameter 'rt' with value "ace.ai", defined in Section 8.2 of [I-D.ietf-ace-oauth-authz].

2.1. Parameters

For each registered link to a group-membership resource at a GM, the following parameters are specified together with the link.

In the RD defined in [I-D.ietf-core-resource-directory] and based on Link Format, each parameter is specified in a target attribute with the same name.

In a RD based on CoRAL, such as the one defined in [I-D.hartke-t2trg-coral-reef], each parameter is specified in a nested element with the same name.

- * 'ct', specifying the content format used with the group-membership resource at the Group Manager, with value "application/ace-groupcomm+cbor" registered in Section 8.2 of [I-D.ietf-ace-key-groupcomm].

Note: The examples in this document use the provisional value 65000 from the range "Reserved for Experimental Use" of the "CoAP Content-Formats" registry, within the "CoRE Parameters" registry.

- * 'rt', specifying the resource type of the group-membership resource at the Group Manager, with value "core.osc.gm" registered in Section 25.11 of [I-D.ietf-ace-key-groupcomm-oscore].
- * 'if', specifying the interface description for accessing the group-membership resource at the Group Manager, with value "ace.group" registered in Section 8.10 of [I-D.ietf-ace-key-groupcomm].

- * 'sec-gp', specifying the name of the security group of interest, as a stable and invariant identifier, such as the group name used in [I-D.ietf-ace-key-groupcomm-oscore]. This parameter MUST specify a single value.
- * 'app-gp', specifying the name(s) of the application group(s) associated with the security group of interest indicated by 'sec-gp'. This parameter MUST occur once for each application group, and MUST specify only a single application group.

When a security group is created at the GM, the names of the application groups using it are also specified as part of the security group configuration (see [I-D.ietf-ace-oscore-gm-admin]). Thus, when registering the links to its group-membership resource, the GM is aware of the application groups and their names.

If a different entity than the GM registers the security groups to the RD, e.g., a Commissioning Tool, this entity has to also be aware of the application groups and their names to specify. To this end, it can obtain them from the GM or from the Administrator that created the security groups at the GM (see [I-D.ietf-ace-oscore-gm-admin]).

Optionally, the following parameters can also be specified. If Link Format is used, the value of each of these parameters is encoded as a text string.

- * 'hkdf', specifying the HKDF Algorithm used in the security group. If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Algorithms" Registry [COSE.Algorithms].
- * 'cred_fmt', specifying the format of the authentication credentials used in the security group. If present, this parameter MUST specify a single value, which is taken from the 'Label' column of the "COSE Header Parameters" Registry [COSE.Header.Parameters]. Acceptable values denote a format that MUST explicitly provide the public key as well as the comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable).

At the time of writing this specification, acceptable formats of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claim Sets (CCSs) [RFC8392], X.509 certificates [RFC7925] and C509 certificates [I-D.ietf-cose-chor-encoded-cert]. Further formats may be available in the future, and would be acceptable to use as long as they comply with the criteria defined above.

[As to CWTs and unprotected CWT claim sets, there is a pending registration requested by draft-ietf-lake-edhoc.]

[As to C509 certificates, there is a pending registration requested by draft-ietf-cose-cbor-encoded-cert.]

- * 'sign_enc_alg', specifying the encryption algorithm used to encrypt messages in the security group, when these are also signed, e.g., as in the group mode of Group OSCORE (see Section 8 of [I-D.ietf-core-oscore-groupcomm]). If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Algorithms" Registry [COSE.Algorithms].
- * 'sign_alg', specifying the algorithm used to sign messages in the security group. If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Algorithms" Registry [COSE.Algorithms].
- * 'sign_alg_crv', specifying the elliptic curve (if applicable) for the algorithm used to sign messages in the security group. If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Elliptic Curves" Registry [COSE.Elliptic.Curves].
- * 'sign_key_kty', specifying the key type of countersignature keys used to sign messages in the security group. If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Key Types" Registry [COSE.Key.Types].
- * 'sign_key_crv', specifying the elliptic curve (if applicable) of countersignature keys used to sign messages in the security group. If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Elliptic Curves" Registry [COSE.Elliptic.Curves].
- * 'alg', specifying the encryption algorithm used to encrypt messages in the security group, when these are encrypted with pairwise encryption keys, e.g., as in the pairwise mode of Group OSCORE (see Section 9 of [I-D.ietf-core-oscore-groupcomm]). If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Algorithms" Registry [COSE.Algorithms].

- * `'ecdh_alg'`, specifying the ECDH algorithm used to derive pairwise encryption keys in the security group, e.g., as for the pairwise mode of Group OSCORE (see Section 2.4 of [I-D.ietf-core-oscore-groupcomm]). If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Algorithms" Registry [COSE.Algorithms].
- * `'ecdh_alg_crv'`, specifying the elliptic curve for the ECDH algorithm used to derive pairwise encryption keys in the security group. If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Elliptic Curves" Registry [COSE.Elliptic.Curves].
- * `'ecdh_key_kty'`, specifying the key type of keys used with an ECDH algorithm to derive pairwise encryption keys in the security group. If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Key Types" Registry [COSE.Key.Types].
- * `'ecdh_key_crv'`, specifying the elliptic curve of keys used with an ECDH algorithm to derive pairwise encryption keys in the security group. If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Elliptic Curves" Registry [COSE.Elliptic.Curves].
- * `'det_hash_alg'`, specifying the hash algorithm used in the security group when producing deterministic requests, as defined in [I-D.amsuess-core-cachable-oscore]. If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "COSE Algorithms" Registry [COSE.Algorithms]. This parameter MUST NOT be present if the security group does not use deterministic requests.
- * `'rekeying_scheme'`, specifying the rekeying scheme used in the security group for distributing new group keying material to the group members. If present, this parameter MUST specify a single value, which is taken from the 'Value' column of the "ACE Groupcomm Rekeying Schemes" registry defined in Section 11.14 of [I-D.ietf-ace-key-groupcomm].

If the security group does not recur to message signing, then the parameters `'sign_enc_alg'`, `'sign_alg'`, `'sign_alg_crv'`, `'sign_key_kty'` and `'sign_key_crv'` MUST NOT be present. For instance, this is the case for a security group that uses Group OSCORE and uses only the pairwise mode (see Section 9 of [I-D.ietf-core-oscore-groupcomm]).

If the security group does not recur to message encryption through pairwise encryption keys, then the parameters 'alg', 'ecdh_alg', 'ecdh_alg_crv', 'ecdh_key_kty' and 'ecdh_key_crv' MUST NOT be present. For instance, this is the case for a security group that uses Group OSCORE and uses only the group mode see Section 8 of [I-D.ietf-core-oscore-groupcomm]).

Note that the values registered in the COSE Registries [COSE.Algorithms][COSE.Elliptic.Curves][COSE.Key.Types] are strongly typed. On the contrary, Link Format is weakly typed and thus does not distinguish between, for instance, the string value "-10" and the integer value -10.

Thus, in RDs that return responses in Link Format, string values which look like an integer are not supported. Therefore, such values MUST NOT be advertised through the corresponding parameters above.

A CoAP endpoint that queries the RD to discover security groups and their group-membership resource to access (see Section 4) would benefit from the information above as follows.

- * The values of 'cred_fmt', 'sign_alg', 'sign_alg_crv', 'sign_key_kty', 'sign_key_crv', 'ecdh_alg', 'ecdh_alg_crv', 'ecdh_key_kty' and 'ecdh_key_crv' related to a group-membership resource provide an early knowledge of the format of authentication credentials as well as of the type of public keys used in the security group.

Thus, the CoAP endpoint does not need to ask the GM for this information as a preliminary step before the joining process, or to perform a trial-and-error joining exchange with the GM. Hence, at the very first step of the joining process, the CoAP endpoint is able to provide the GM with its own authentication credential in the correct expected format and including a public key of the correct expected type.

- * The values of 'hkdf', 'sign_enc_alg', 'sign_alg', 'alg' and 'ecdh_alg' related to a group-membership resource provide an early knowledge of the algorithms used in the security group.

Thus, the CoAP endpoint is able to decide whether to actually proceed with the joining process, depending on its support for the indicated algorithms.

2.2. Relation Link to Authorization Server

For each registered link to a group-membership resource, the GM MAY additionally specify the link to the ACE Authorization Server (AS) [I-D.ietf-ace-oauth-authz] associated with the GM, and issuing authorization credentials to join the security group as described in [I-D.ietf-ace-key-groupcomm-oscore].

The link to the AS has as target the URI of the resource where to send an authorization request to.

In the RD defined in [I-D.ietf-core-resource-directory] and based on Link Format, the link to the AS is separately registered with the RD, and includes the following parameters as target attributes.

- * 'rel', with value "authorization_server".
- * 'anchor', with value the target of the link to the group-membership resource at the GM.

In a RD based on CoRAL, such as the one defined in [I-D.hartke-t2trg-coral-reef], this is mapped (as describe there) to a link from the registration resource to the AS, using the <http://www.iana.org/assignments/relation/authorization_server> link relation type.

2.3. Registration Example

The example below shows a GM with endpoint name "gm1" and address 2001:db8::ab that registers with the RD.

The GM specifies the value of the 'sec-gp' parameter for accessing the security group with name "feedca570000", and used by the application group with name "group1" specified with the value of the 'app-gp' parameter.

The countersignature algorithm used in the security group is EdDSA (-8), with elliptic curve Ed25519 (6). Authentication credentials used in the security group are X.509 certificates [RFC7925], which is signaled through the COSE Header Parameter "x5chain" (33). The ECDH algorithm used in the security group is ECDH-SS + HKDF-256 (-27), with elliptic curve X25519 (4).

In addition, the GM specifies the link to the ACE Authorization Server associated with the GM, to which a CoAP endpoint should send an Authorization Request for joining the corresponding security group (see [I-D.ietf-ace-key-groupcomm-oscore]).

2.3.1. Example in Link Format

Request: GM -> RD

Req: POST coap://rd.example.com/rd?ep=gml

Content-Format: 40

Payload:

```
</ace-group/feedca570000>;ct=65000;rt="core.osc.gm";if="ace.group";
    sec-gp="feedca570000";app-gp="group1";
    cred_fmt="33";sign_enc_alg="10";
    sign_alg="-8";sign_alg_crv="6";
    alg="10";ecdh_alg="-27";ecdh_alg_crv="4",
<coap://as.example.com/token>;rel="authorization-server";
    anchor="coap://[2001:db8::ab]/ace-group/feedca570000"
```

Response: RD -> GM

Res: 2.01 Created

Location-Path: /rd/4521

2.3.2. Example in CoRAL

Request: GM -> RD

Req: POST coap://rd.example.com/rd?ep=gml

Content-Format: TBD123456 (application/coral+cbor)

Payload:

#using <http://coreapps.org/core.oscore-discovery#>

#using reef = <http://coreapps.org/reef#>

#using iana = <http://www.iana.org/assignments/relation/>

#base <coap://[2001:db8::ab]/>

reef:rd-item </ace-group/feedca570000> {

 reef:ct 65000

 reef:rt "core.osc.gm"

 reef:if "ace.group"

 sec-gp "feedca570000"

 app-gp "group1"

 cred_fmt 33

 sign_enc_alg 10

 sign_alg -8

 sign_alg_crv 6

 alg 10

 ecdh_alg -27

 ecdh_alg_crv 4

 iana:authorization-server <coap://as.example.com/token>

}

Response: RD -> GM

Res: 2.01 Created

Location-Path: /rd/4521

3. Addition and Update of Security Groups

The GM is responsible to refresh the registration of all its group-membership resources in the RD. This means that the GM has to update the registration within its lifetime as per Section 5.3.1 of [I-D.ietf-core-resource-directory], and has to change the content of the registration when a group-membership resource is added/removed, or if its parameters have to be changed, such as in the following cases.

- * The GM creates a new security group and starts exporting the related group-membership resource.
- * The GM dismisses a security group and stops exporting the related group-membership resource.
- * Information related to an existing security group changes, e.g., the list of associated application groups.

To perform an update of its registrations, the GM can re-register with the RD and fully specify all links to its group-membership resources.

Alternatively, the GM can perform a PATCH/iPATCH [RFC8132] request to the RD, as per Section 5.3.3 of [I-D.ietf-core-resource-directory]. This requires new media-types to be defined in future standards, to apply a new document as a patch to an existing stored document.

3.1. Addition Example

The example below shows how the GM from Section 2 re-registers with the RD. When doing so, it specifies:

- * The same previous group-membership resource associated with the security group with name "feedca570000".
- * An additional group-membership resource associated with the security group with name "ech0ech00000" and used by the application group "group2".
- * A third group-membership resource associated with the security group with name "abcdef120000" and used by two application groups, namely "group3" and "group4".

Furthermore, the GM relates the same Authorization Server also to the security groups "ech0ech00000" and "abcdef120000".

3.1.1. Example in Link Format

Request: GM -> RD

```
Req: POST coap://rd.example.com/rd?ep=gml
Content-Format: 40
Payload:
</ace-group/feedca570000>;ct=65000;rt="core.osc.gm";if="ace.group";
    sec-gp="feedca570000";app-gp="group1";
    cred_fmt="33";sign_enc_alg="10";
    sign_alg="-8";sign_alg_crv="6";
    alg="10";ecdh_alg="-27";ecdh_alg_crv="4",
</ace-group/ech0ech00000>;ct=65000;rt="core.osc.gm";if="ace.group";
    sec-gp="ech0ech00000";app-gp="group2";
    cred_fmt="33";sign_enc_alg="10";
    sign_alg="-8";sign_alg_crv="6";
    alg="10";ecdh_alg="-27";ecdh_alg_crv="4",
</ace-group/abcdef120000>;ct=65000;rt="core.osc.gm";if="ace.group";
    sec-gp="abcdef120000";app-gp="group3";
    app-gp="group4";cred_fmt="33";
    sign_enc_alg="10";sign_alg="-8";
    sign_alg_crv="6";alg="10";ecdh_alg="-27";
    ecdh_alg_crv="4",
<coap://as.example.com/token>;rel="authorization-server";
    anchor="coap://[2001:db8::ab]/ace-group/feedca570000",
<coap://as.example.com/token>;rel="authorization-server";
    anchor="coap://[2001:db8::ab]/ace-group/ech0ech00000",
<coap://as.example.com/token>;rel="authorization-server";
    anchor="coap://[2001:db8::ab]/ace-group/abcdef120000"
```

Response: RD -> GM

```
Res: 2.04 Changed
Location-Path: /rd/4521
```

3.1.2. Example in CoRAL

Request: GM -> RD

```
Req: POST coap://rd.example.com/rd?ep=gml
Content-Format: TBD123456 (application/coral+cbor)
```

```
Payload:
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>
```

```
#using iana = <http://www.iana.org/assignments/relation/>

#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedca570000> {
  reef:ct 65000
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "feedca570000"
  app-gp "group1"
  cred_fmt 33
  sign_enc_alg 10
  sign_alg -8
  sign_alg_crv 6
  alg 10
  ecdh_alg -27
  ecdh_alg_crv 4
  iana:authorization-server <coap://as.example.com/token>
}
reef:rd-item </ace-group/ech0ech00000> {
  reef:ct 65000
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "ech0ech00000"
  app-gp "group2"
  cred_fmt 33
  sign_enc_alg 10
  sign_alg -8
  sign_alg_crv 6
  alg 10
  ecdh_alg -27
  ecdh_alg_crv 4
  iana:authorization-server <coap://as.example.com/token>
}
reef:rd-item </ace-group/abcdef120000> {
  reef:ct 65000
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "abcdef120000"
  app-gp "group3"
  app-gp "group4"
  cred_fmt 33
  sign_enc_alg 10
  sign_alg -8
  sign_alg_crv 6
  alg 10
  ecdh_alg -27
  ecdh_alg_crv 4
  iana:authorization-server <coap://as.example.com/token>
}
```

```
}
```

```
Response: RD -> GM
```

```
Res: 2.04 Changed
```

```
Location-Path: /rd/4521
```

4. Discovery of Security Groups

A CoAP endpoint that wants to join a security group, hereafter called the joining node, might not have all the necessary information at deployment time. Also, it might want to know about possible new security groups created afterwards by the respective Group Managers.

To this end, the joining node can perform a resource lookup at the RD as per Section 6.1 of [I-D.ietf-core-resource-directory], to retrieve the missing pieces of information needed to join the security group(s) of interest. The joining node can find the RD as described in Section 4 of [I-D.ietf-core-resource-directory].

The joining node uses the following parameter value for the lookup filtering.

- * 'rt' = "core.osc.gm", specifying the resource type of the group-membership resource at the Group Manager, with value "core.osc.gm" registered in Section 25.11 of [I-D.ietf-ace-key-groupcomm-oscore].

The joining node may additionally consider the following parameters for the lookup filtering, depending on the information it has already available.

- * 'sec-gp', specifying the name of the security group of interest. This parameter MUST specify a single value.
- * 'ep', specifying the registered endpoint of the GM.
- * 'app-gp', specifying the name(s) of the application group(s) associated with the security group of interest. This parameter MAY be included multiple times, and each occurrence MUST specify the name of one application group.
- * 'if', specifying the interface description for accessing the group-membership resource at the Group Manager, with value "ace.group" registered in Section 8.10 of [I-D.ietf-ace-key-groupcomm].

The response from the RD may include links to a group-membership resource specifying multiple application groups, as all using the same security group. In this case, the joining node is already expected to know the exact application group of interest.

Furthermore, the response from the RD may include the links to different group-membership resources, all specifying a same application group of interest for the joining node, if the corresponding security groups are all used by that application group.

In this case, application policies on the joining node should define how to determine the exact security group to join (see Section 2.1 of [I-D.ietf-core-groupcomm-bis]). For example, different security groups can reflect different security algorithms to use. Hence, a client application can take into account what the joining node supports and prefers, when selecting one particular security group among the indicated ones, while a server application would need to join all of them. Later on, the joining node will be anyway able to join only security groups for which it is actually authorized to be a member (see [I-D.ietf-ace-key-groupcomm-oscore]).

Note that, with RD-based discovery, including the 'app-gp' parameter multiple times would result in finding only the group-membership resource that serves all the specified application groups, i.e., not any group-membership resource that serves either. Therefore, a joining node needs to perform N separate queries with different values for 'app-gp', in order to safely discover the (different) group-membership resource(s) serving the N application groups.

The discovery of security groups as defined in this document is applicable and useful to other CoAP endpoints than the actual joining nodes. In particular, other entities can be interested to discover and interact with the group-membership resource at the Group Manager. These include entities acting as signature checkers, e.g., intermediary gateways, that do not join a security group, but can retrieve authentication credentials of group members from the Group Manager, in order to verify counter signatures of group messages (see Section 3 of [I-D.ietf-core-oscore-groupcomm]).

4.1. Discovery Example #1

Consistently with the examples in Section 2 and Section 3, the examples below consider a joining node that wants to join the security group associated with the application group "group1", but that does not know the name of the security group, the responsible GM and the group-membership resource to access.

4.1.1. Example in Link Format

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res
?rt=core.osc.gm&app-gp=group1

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/ace-group/feedca570000>;ct=65000;  
rt="core.osc.gm";if="ace.group";sec-gp="feedca570000";  
app-gp="group1";cred_fmt="33";sign_enc_alg="10";  
sign_alg="-8";sign_alg_crv="6";alg="10";  
ecdh_alg="-27";ecdh_alg_crv="4"
```

By performing the separate resource lookup below, the joining node can retrieve the link to the ACE Authorization Server associated with the GM, where to send an Authorization Request for joining the corresponding security group (see [I-D.ietf-ace-key-groupcomm-oscore]).

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res
?rel=authorization-server&
anchor=coap://[2001:db8::ab]/ace-group/feedca570000

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<coap://as.example.com/token>;rel=authorization-server;  
anchor="coap://[2001:db8::ab]/ace-group/feedca570000"
```

To retrieve the multicast IP address of the CoAP group used by the application group "group1", the joining node performs an endpoint lookup as shown below. The following assumes that the application group "group1" had been previously registered as per Appendix A of [I-D.ietf-core-resource-directory], with ff35:30:2001:db8::23 as multicast IP address of the associated CoAP group.

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/ep
?et=core.rd-group&ep=group1

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
</rd/501>;ep="group1";et="core.rd-group";  
base="coap://[ff35:30:2001:db8::23]";rt="core.rd-ep"
```

4.1.2. Example in CoRAL

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res

?rt=core.osc.gm&app-gp=group1

Accept: TBD123456 (application/coral+cbor)

Response: RD -> Joining node

Res: 2.05 Content

Content-Format: TBD123456 (application/coral+cbor)

Payload:

#using <http://coreapps.org/core.oscore-discovery#>

#using reef = <http://coreapps.org/reef#>

#using iana = <http://www.iana.org/assignments/relation/>

#base <coap://[2001:db8::ab]/>

reef:rd-item </ace-group/feedca570000> {

reef:ct 65000

reef:rt "core.osc.gm"

reef:if "ace.group"

sec-gp "feedca570000"

app-gp "group1"

cred_fmt 33

sign_enc_alg 10

sign_alg -8

sign_alg_crv 6

alg 10

ecdh_alg -27

ecdh_alg_crv 4

iana:authorization-server <coap://as.example.com/token>

}

To retrieve the multicast IP address of the CoAP group used by the application group "group1", the joining node performs an endpoint lookup as shown below. The following assumes that the application group "group1" had been previously registered, with ff35:30:2001:db8::23 as multicast IP address of the associated CoAP group.

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/ep
 ?et=core.rd-group&ep=group1
Accept: TBD123456 (application/coral+cbor)

Response: RD -> Joining node

Res: 2.05 Content
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>

```
reef:rd-unit <./rd/501> {  
  reef:ep "group1"  
  reef:et "core.rd-group"  
  reef:base <coap://[ff35:30:2001:db8::23]>  
  reef:rt "core.rd-ep"  
}
```

4.2. Discovery Example #2

Consistently with the examples in Section 2 and Section 3, the examples below consider a joining node that wants to join the security group with name "feedca570000", but that does not know the responsible GM, the group-membership resource to access, and the associated application groups.

The examples also show how the joining node uses CoAP observation [RFC7641], in order to be notified of possible changes to the parameters of the group-membership resource. This is also useful to handle the case where the security group of interest has not been created yet, so that the joining node can receive the requested information when it becomes available.

4.2.1. Example in Link Format

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res
 ?rt=core.osc.gm&sec-gp=feedca570000
Observe: 0

Response: RD -> Joining node

Res: 2.05 Content

Observe: 24

Payload:

```
<coap://[2001:db8::ab]/ace-group/feedca570000>;ct=65000;
  rt="core.osc.gm";if="ace.group";sec-gp="feedca570000";
  app-gp="group1";cred_fmt="33";sign_enc_alg="10";
  sign_alg="-8";sign_alg_crv="6";alg="10";
  ecdh_alg="-27";ecdh_alg_crv="4"
```

Depending on the search criteria, the joining node performing the resource lookup can get large responses. This can happen, for instance, when the lookup request targets all the group-membership resources at a specified GM, or all the group-membership resources of all the registered GMs, as in the example below.

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res?rt=core.osc.gm

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/ace-group/feedca570000>;ct=65000;
  rt="core.osc.gm";if="ace.group";sec-gp="feedca570000";
  app-gp="group1";cred_fmt="33";sign_enc_alg="10";
  sign_alg="-8";sign_alg_crv="6";alg="10";ecdh_alg="-27";
  ecdh_alg_crv="4",
<coap://[2001:db8::ab]/ace-group/ech0ech00000>;ct=65000;
  rt="core.osc.gm";if="ace.group";sec-gp="ech0ech00000";
  app-gp="group2";cred_fmt="33";sign_enc_alg="10";
  sign_alg="-8";sign_alg_crv="6";alg="10";ecdh_alg="-27";
  ecdh_alg_crv="4",
<coap://[2001:db8::ab]/ace-group/abcdef120000>;ct=65000;
  rt="core.osc.gm";if="ace.group";sec-gp="abcdef120000";
  app-gp="group3";app-gp="group4";cred_fmt="33";
  sign_enc_alg="10";sign_alg="-8";sign_alg_crv="6";alg="10";
  ecdh_alg="-27";ecdh_alg_crv="4"
```

Therefore, it is RECOMMENDED that a joining node which performs a resource lookup with the CoAP Observe option specifies the value of the parameter 'sec-gp' in its GET request sent to the RD.

4.2.2. Example in CoRAL

Request: Joining node -> RD

```
Req: GET coap://rd.example.com/rd-lookup/res
     ?rt=core.osc.gm&sec-gp=feedca570000
Accept: TBD123456 (application/coral+cbor)
Observe: 0
```

Response: RD -> Joining node

```
Res: 2.05 Content
Observe: 24
Content-Format: TBD123456 (application/coral+cbor)
```

```
Payload:
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>
#using iana = <http://www.iana.org/assignments/relation/>
```

```
#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedca570000> {
  reef:ct 65000
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "feedca570000"
  app-gp "group1"
  cred_fmt 33
  sign_enc_alg 10
  sign_alg -8
  sign_alg_crv 6
  alg 10
  ecdh_alg -27
  ecdh_alg_crv 4
  iana:authorization-server <coap://as.example.com/token>
}
```

5. Use Case Example With Full Discovery

In this section, the discovery of security groups is described to support the installation process of a lighting installation in an office building. The described process is a simplified version of one of many processes.

The process described in this section is intended as an example and does not have any particular ambition to serve as recommendation or best practice to adopt. That is, it shows a possible workflow involving a Commissioning Tool (CT) used in a certain way, while it is not meant to prescribe how the workflow should necessarily be.

Assume the existence of four luminaires that are members of two application groups. In the first application group, the four luminaires receive presence messages and light intensity messages from sensors or their proxy. In the second application group, the four luminaires and several other pieces of equipment receive building state schedules.

Each of the two application groups is associated with a different security group and to a different CoAP group with its own dedicated multicast IP address.

The Fairhair Alliance describes how a new device is accepted and commissioned in the network [Fairhair], by means of its certificate stored during the manufacturing process. When commissioning the new device in the installation network, the new device gets a new identity defined by a newly allocated certificate, following the BRSKI specification.

Section 7.3 of [I-D.ietf-core-resource-directory] describes how the CT assigns an endpoint name based on the CN field, (CN=ACME) and the serial number of the certificate (serial number = 123x, with $3 < x < 8$). Corresponding ep-names ACME-1234, ACME-1235, ACME-1236 and ACME-1237 are also assumed.

It is common practice that locations in the building are specified according to a coordinate system. After the acceptance of the luminaires into the installation network, the coordinate of each device is communicated to the CT. This can be done manually or automatically.

The mapping between location and ep-name is calculated by the CT. For instance, on the basis of grouping criteria, the CT assigns: i) application group "grp_R2-4-015" to the four luminaires; and ii) application group "grp_schedule" to all schedule requiring devices. Also, the device with ep name ACME-123x has been assigned IP address: [2001:db8:4::x]. The RD is assigned IP address: [2001:db8:4:ff]. The used multicast addresses are: [ff05::5:1] and [ff05::5:2].

The following assumes that each device is pre-configured with the name of the two application groups it belongs to. Additional mechanisms can be defined in the RD, for supporting devices to discover the application groups they belong to.

Appendix A provides this same use case example in CoRAL.

*** **

The CT defines the application group "grp_R2-4-015", with resource /light and base address [ff05::5:1], as follows.

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd
 ?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::5:1]
Content-Format: 40
Payload:
</light>;rt="oic.d.light"

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/501

Also, the CT defines a second application group "grp_schedule", with resource /schedule and base address [ff05::5:2], as follows.

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd
 ?ep=grp_schedule&et=core.rd-group&base=coap://[ff05::5:2]
Content-Format: 40
Payload:
</schedule>;rt="oic.r.time.period"

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/502

*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***

Finally, the CT defines the corresponding security groups. In particular, assuming a Group Manager responsible for both security groups and with address [2001:db8::ab], the CT specifies:

Request: CT -> RD

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=gml&base=coap://[2001:db8::ab]
Content-Format: 40
Payload:
</ace-group/feedca570000>;ct=65000;rt="core.osc.gm";if="ace.group";
      sec-gp="feedca570000";
      app-gp="grp_R2-4-015",
</ace-group/feedsc590000>;ct=65000;rt="core.osc.gm";if="ace.group";
      sec-gp="feedsc590000";
      app-gp="grp_schedule"
```

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/4521

*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***

The device with IP address [2001:db8:4::x] can retrieve the multicast IP address of the CoAP group used by the application group "grp_R2-4-015", by performing an endpoint lookup as shown below.

Request: Joining node -> RD

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
      ?et=core.rd-group&ep=grp_R2-4-015
```

Response: RD -> Joining node

```
Res: 2.05 Content
Content-Format: 40
Payload:
</rd/501>;ep="grp_R2-4-015";et="core.rd-group";
      base="coap://[ff05::5:1]";rt="core.rd-ep"
```

Similarly, to retrieve the multicast IP address of the CoAP group used by the application group "grp_schedule", the device performs an endpoint lookup as shown below.

Request: Joining node -> RD

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
      ?et=core.rd-group&ep=grp_schedule
```

Response: RD -> Joining node

```
Res: 2.05 Content
Content-Format: 40
Payload:
</rd/502>;ep="grp_schedule";et="core.rd-group";
      base="coap://[ff05::5:2]";rt="core.rd-ep"
```

*** **

Consequently, the device learns the security groups it has to join. In particular, it does the following for app-gp="grp_R2-4-015".

Request: Joining node -> RD

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
      ?rt=core.osc.gm&app-gp=grp_R2-4-015
```

Response: RD -> Joining Node

```
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://[2001:db8::ab]/ace-group/feedca570000>;ct=65000;
      rt="core.osc.gm";if="ace.group";sec-gp="feedca570000";
      app-gp="grp_R2-4-015"
```

Similarly, the device does the following for app-gp="grp_schedule".

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
      ?rt=core.osc.gm&app-gp=grp_schedule
```

Response: RD -> Joining Node

```
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://[2001:db8::ab]/ace-group/feedsc590000>;ct=65000;
      rt="core.osc.gm";if="ace.group";sec-gp="feedsc590000";
      app-gp="grp_schedule"
```

*** **

After this last discovery step, the device can ask permission to join the security groups, and effectively join them through the Group Manager, e.g., according to [I-D.ietf-ace-key-groupcomm-oscore].

6. Security Considerations

The security considerations as described in Section 8 of [I-D.ietf-core-resource-directory] apply here as well.

7. IANA Considerations

This document has no actions for IANA.

8. References

8.1. Normative References

- [COSE.Algorithms]
IANA, "COSE Algorithms",
<<https://www.iana.org/assignments/cose/cose.xhtml#algorithms>>.
- [COSE.Elliptic.Curves]
IANA, "COSE Elliptic Curves",
<<https://www.iana.org/assignments/cose/cose.xhtml#elliptic-curves>>.
- [COSE.Header.Parameters]
IANA, "COSE Header Parameters",
<<https://www.iana.org/assignments/cose/cose.xhtml#header-parameters>>.
- [COSE.Key.Types]
IANA, "COSE Key Types",
<<https://www.iana.org/assignments/cose/cose.xhtml#key-type>>.
- [I-D.ietf-core-coral]
Amsüss, C. and T. Fossati, "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-ietf-core-coral-04, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-coral-04.txt>>.
- [I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-06, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-06.txt>>.

- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P.,
and J. Park, "Group OSCORE - Secure Group Communication
for CoAP", Work in Progress, Internet-Draft, draft-ietf-
core-oscore-groupcomm-14, 7 March 2022,
<[https://www.ietf.org/archive/id/draft-ietf-core-oscore-
groupcomm-14.txt](https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-14.txt)>.
- [I-D.ietf-core-resource-directory]
Amsüss, C., Shelby, Z., Kostér, M., Bormann, C., and P. V.
D. Stok, "CoRE Resource Directory", Work in Progress,
Internet-Draft, draft-ietf-core-resource-directory-28, 7
March 2021, <[https://www.ietf.org/archive/id/draft-ietf-
core-resource-directory-28.txt](https://www.ietf.org/archive/id/draft-ietf-core-resource-directory-28.txt)>.
- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE):
Initial Algorithms", Work in Progress, Internet-Draft,
draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020,
<[https://www.ietf.org/archive/id/draft-ietf-cose-
rfc8152bis-algs-12.txt](https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt)>.
- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE):
Structures and Process", Work in Progress, Internet-Draft,
draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021,
<[https://www.ietf.org/archive/id/draft-ietf-cose-
rfc8152bis-struct-15.txt](https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt)>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [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>>.
- [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

[Fairhair] FairHair Alliance, "Security Architecture for the Internet of Things (IoT) in Commercial Buildings", White Paper, ed. Piotr Polak , March 2018, <https://openconnectivity.org/wp-content/uploads/2019/11/fairhair_security_wp_march-2018.pdf>.

[I-D.amsuess-core-cachable-oscore]
Amsüss, C. and M. Tiloca, "Cacheable OSCORE", Work in Progress, Internet-Draft, draft-amsuess-core-cachable-oscore-04, 6 March 2022, <<https://www.ietf.org/archive/id/draft-amsuess-core-cachable-oscore-04.txt>>.

[I-D.hartke-t2trg-coral-reef]
Hartke, K., "Resource Discovery in Constrained RESTful Environments (CoRE) using the Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-hartke-t2trg-coral-reef-04, 9 May 2020, <<https://www.ietf.org/archive/id/draft-hartke-t2trg-coral-reef-04.txt>>.

[I-D.ietf-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-15, 23 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-15.txt>>.

[I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-13, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-13.txt>>.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-46, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>>.

- [I-D.ietf-ace-oscore-gm-admin]
Tiloca, M., Höglund, R., Stok, P. V. D., and F. Palombini,
"Admin Interface for the OSCORE Group Manager", Work in
Progress, Internet-Draft, draft-ietf-ace-oscore-gm-admin-
05, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-gm-admin-05.txt>>.
- [I-D.ietf-cose-cbor-encoded-cert]
Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and
M. Furuheid, "CBOR Encoded X.509 Certificates (C509
Certificates)", Work in Progress, Internet-Draft, draft-
ietf-cose-cbor-encoded-cert-03, 10 January 2022,
<[https://www.ietf.org/archive/id/draft-ietf-cose-cbor-
encoded-cert-03.txt](https://www.ietf.org/archive/id/draft-ietf-cose-cbor-encoded-cert-03.txt)>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig,
"CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392,
May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz,
"Object Security for Constrained RESTful Environments
(OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019,
<<https://www.rfc-editor.org/info/rfc8613>>.

[RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M.,
 and K. Watsen, "Bootstrapping Remote Secure Key
 Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995,
 May 2021, <<https://www.rfc-editor.org/info/rfc8995>>.

Appendix A. Use Case Example With Full Discovery (CoRAL)

This section provides the same use case example of Section 5, but
specified in CoRAL [I-D.ietf-core-coral].

*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***

The CT defines the application group "grp_R2-4-015", with resource
/light and base address [ff05::5:1], as follows.

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using reef = <<http://coreapps.org/reef#>>

```
#base <coap://[ff05::5:1]/>
reef:ep "grp_R2-4-015"
reef:et "core.rd-group"
reef:rd-item </light> {
    reef:rt "oic.d.light"
}
```

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/501

Also, the CT defines a second application group "grp_schedule", with
resource /schedule and base address [ff05::5:2], as follows.

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd?ep=grp_schedule&et=core.rd-group
Content-Format: TBD123456 (application/coral+cbor)

Payload:

#using reef = <http://coreapps.org/reef#>

```
#base <coap://[ff05::5:2]/>
reef:rd-item </schedule> {
  reef:rt "oic.r.time.period"
}
```

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/502

*** **

Finally, the CT defines the corresponding security groups. In particular, assuming a Group Manager responsible for both security groups and with address [2001:db8::ab], the CT specifies:

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd?ep=gml
Content-Format: TBD123456 (application/coral+cbor)

Payload:

#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>

```
#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedca570000> {
  reef:ct 65000
  reef:ct 41
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "feedca570000"
  app-gp "grp_R2-4-015"
}
reef:rd-item </ace-group/feedsc590000> {
  reef:ct 65000
  reef:ct 41
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "feedsc590000"
  app-gp "grp_schedule"
}
```

Response: RD \rightarrow CT

Res: 2.01 Created
Location-Path: /rd/4521

*** **

The device with IP address [2001:db8:4::x] can retrieve the multicast IP address of the CoAP group used by the application group "grp_R2-4-015", by performing an endpoint lookup as shown below.

Request: Joining node -> RD

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
      ?et=core.rd-group&ep=grp_R2-4-015
```

Response: RD -> Joining node

```
Res: 2.05 Content
Content-Format: TBD123456 (application/coral+cbor)
```

```
Payload:
#using reef = <http://coreapps.org/reef#>
```

```
#base <coap://[2001:db8:4::ff]/rd/>
reef:rd-unit <501> {
    reef:ep "grp_R2-4-015"
    reef:et "core.rd-group"
    reef:base <coap://[ff05::5:1]/>
    reef:rt "core.rd-ep"
}
```

Similarly, to retrieve the multicast IP address of the CoAP group used by the application group "grp_schedule", the device performs an endpoint lookup as shown below.

Request: Joining node -> RD

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
      ?et=core.rd-group&ep=grp_schedule
```

Response: RD -> Joining node

Res: 2.05 Content
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using reef = <http://coreapps.org/reef#>

```
#base <coap://[2001:db8:4::ff]/rd/>
reef:rd-unit <502> {
  reef:ep "grp_schedule"
  reef:et "core.rd-group"
  reef:base <coap://[ff05::5:2]/>
  reef:rt "core.rd-ep"
}
```

*** **

Consequently, the device learns the security groups it has to join.
In particular, it does the following for app-gp="grp_R2-4-015".

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
?rt=core.osc.gm&app-gp=grp_R2-4-015

Response: RD -> Joining Node

Res: 2.05 Content
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>

```
#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedca570000> {
  reef:ct 65000
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "feedca570000"
  app-gp "grp_R2-4-015"
}
```

Similarly, the device does the following for app-gp="grp_schedule".

Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
?rt=core.osc.gm&app-gp=grp_schedule

Response: RD -> Joining Node

```
Res: 2.05 Content
Content-Format: TBD123456 (application/coral+cbor)

Payload:
#using <http://coreapps.org/core.oscore-discovery#>
#using reef = <http://coreapps.org/reef#>

#base <coap://[2001:db8::ab]/>
reef:rd-item </ace-group/feedsc590000> {
  reef:ct 65000
  reef:rt "core.osc.gm"
  reef:if "ace.group"
  sec-gp "feedsc590000"
  app-gp "grp_schedule"
}
```

*** **

After this last discovery step, the device can ask permission to join the security groups, and effectively join them through the Group Manager, e.g., according to [I-D.ietf-ace-key-groupcomm-oscore].

Acknowledgments

The authors sincerely thank Carsten Bormann, Klaus Hartke, Jaime Jimenez, Francesca Palombini, Dave Robin and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; by the H2020 project SIFIS-Home (Grant agreement 952652); and by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

Christian Amsuess
Hollandstr. 12/4
1020 Vienna
Austria
Email: christian@amsuess.com

Peter van der Stok
Consultant
Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: stokcons@bbhmail.nl