CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 26 August 2021
22 February 2021

C. Amsüss

M. Tiloca
RISE AB

Cacheable OSCORE
draft-amsuess-core-cachable-oscore-01

Abstract

Group communication with the Constrained Application Protocol (CoAP) can be secured end-to-end using Group Object Security for Constrained RESTful Environments (Group OSCORE), also across untrusted intermediary proxies. However, this sidesteps the proxies' abilities to cache responses from the origin server(s). This specification restores cachability of protected responses at proxies, by introducing consensus requests which any client in a group can send to one server or multiple servers in the same group.

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/.

Source for this draft and an issue tracker can be found at https://gitlab.com/chrysn/core-cachable-oscore/.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Use cases 4
  1.2. Terminology 4
2. Deterministic Requests 5
  2.1. Design Considerations 6
  2.2. Request-Hash 7
  2.3. Use of Deterministic Requests 8
    2.3.1. Pre-Conditions 8
    2.3.2. Client Processing of Deterministic Request 9
    2.3.3. Server Processing of Deterministic Request 10
    2.3.4. Response to a Deterministic Request 12
    2.3.5. Deterministic Requests to Multiple Servers 13
3. Security Considerations 14
4. IANA Considerations 15
  4.1. CoAP Option Numbers Registry 15
5. References 15
  5.1. Normative References 15
  5.2. Informative References 17
Appendix A. Change log 18
Appendix B. Padding 18
  B.1. Definition of the Padding Option 18
  B.2. Using and processing the Padding option 19
Appendix C. Simple Cachability using Ticket Requests 19
Appendix D. Application for more efficient end-to-end protected
        multicast notifications 20
Appendix E. Open questions 21
Appendix F. Unsorted further ideas 21
Acknowledgments 21
Authors' Addresses 21

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports also group communication, for instance over UDP and IP multicast [I-D.ietf-core-groupcomm-bis]. In a group communication environment, exchanged messages can be secured end-to-end by using Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm].

Requests and responses protected with the group mode of Group OSCORE can be read by all group members, i.e. not only by the intended recipient(s), thus achieving group-level confidentiality.

This allows a trusted intermediary proxy which is also a member of the OSCORE group to populate its cache with responses from origin servers. Later on, the proxy can possibly reply to a request in the group with a response from its cache, if recognized as an eligible server by the client.

However, an untrusted proxy which is not member of the OSCORE group only sees protected responses as opaque, uncacheable ciphertext. In particular, different clients in the group that originate a same plain CoAP request would send different protected requests, as a result of their Group OSCORE processing. Such protected requests cannot yield a cache hit at the proxy, which makes the whole caching of protected responses pointless.

This document addresses this complication and enables cachability of protected responses, also for proxies that are not members of the OSCORE group and are unaware of OSCORE in general. To this end, it builds on the concept of "consensus request" initially considered in [I-D.tiloca-core-observe-multicast-notifications], and defines "deterministic request" as a convenient incarnation of such concept.

Intuitively, given a GET or FETCH plain CoAP request, all clients wishing to send that request are able to deterministically compute the same protected request, using a variation on the pairwise mode of Group OSCORE. It follows that cache hits become possible at the proxy, which can thus serve clients in the group from its cache. Like in [I-D.tiloca-core-observe-multicast-notifications], this requires that clients and servers are already members of a suitable OSCORE group.

Cachability of protected responses is useful also in applications where several clients wish to retrieve the same object. Some security properties of OSCORE are dispensed with to gain other desirable properties.

1.1. Use cases

When firmware updates are delivered using CoAP, many similar devices fetch large representations at the same time. Collecting them at a proxy not only keeps the traffic low, but also lets the clients ride single file to hide their numbers[SW:EPIV].

When fanning out multicast data delivery, deterministic requests allow for a more efficient setup (Appendix D).

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.

Readers are expected to be familiar with terms and concepts of CoAP [RFC7252] and its method FETCH [RFC8132], group communication for CoAP [I-D.ietf-core-groupcomm-bis], COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs], OSCORE [RFC8613], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This document introduces the following new terms.

* Consensus Request: a Group OSCORE request that can be used repeatedly to access a particular resource, hosted at one or more servers in the OSCORE group.

  A Consensus Request has all the properties relevant to caching, but its transport dependent properties (e.g. Token or Message ID) are not defined. Thus, different requests on the wire can both be said to "be the same Consensus Request" even if they have different Tokens or client addresses.

  The Consensus Request is the reference for request-response binding. Hence, if it does not generate a Consensus Request by itself, the client has still to be able to read and verify any obtained Consensus Request, before using it to verify a bound response.

* Deterministic Client: a fictitious member of an OSCORE group, having no Sender Sequence Number, no asymmetric key pair, and no Recipient Context.

The Group Manager sets up the Deterministic Client, and assigns it a unique Sender ID as for other group members. Furthermore, the Deterministic Client has only the minimum common set of privileges shared by all group members.

* Deterministic Request: a Consensus Request generated by the Deterministic Client. The use of Deterministic Requests is defined in Section 2.

* Ticket Request: a Consensus Request generated by the server itself.

  This term is not used in the main document, but is useful in comparison with other applications of consensus requests that are generated in a different way than as a Deterministic Request. The prototypical Ticket Request is the Phantom Request defined in [I-D.tiloca-core-observe-multicast-notifications].

  In Appendix C, the term is used bridge the gap to that draft.

2. Deterministic Requests

This section defines a method for clients starting from a same plain CoAP request to independently arrive at a same Deterministic Request protected with Group OSCORE.

While the first client sending the Deterministic Request actually reaches the origin server, the response can be cached by an intermediary proxy. Later on, a different client with the same plain CoAP request would send the same Deterministic Request, which will be served from the proxy's cache.

Clients build the unprotected Deterministic Request in a way which is as much reproducible as possible. This document does not set out full guidelines for minimizing the variation, but considered starting points are:

* Set the inner Observe option to 0 if the requested resource is described as observable, even if no observation is intended (and no outer Observe is set). Thus, both observing and non-observing requsts can be aggregated into a single request, that is upstreamed as an observation at latest when any observing request reaches the proxy.

* Avoid setting the ETag option in requests on a whim. Only set it when there was a recent response with that ETag. When obtaining later blocks, do not send the known-stale ETag.

* In block-wise transfer, maximally sized large inner blocks (szx=6) should be selected. This serves not only to align the clients on consistent cache entries, but also helps amortize the additional data transferred in the per-message signatures.

  Outer block-wise transfer can then be used if these messages excede a hop's efficiently usable MTU size.

  (If BERT [RFC8323] is usable with OSCORE, its use is fine as well; in that case, the server picks a consistent block size for all clients anyway).

* If padding (see Appendix B) is used to limit an adversary's ability to deduce requests' content from their length, the length requests are padded to should be agreed on among all users of a security context.

These only serve to ensure that cache entries are utilized; failure to follow them has no more severe consequences than decreasing the utility of a cache.

2.1. Design Considerations

The hard part is arriving at a consensus pair (key, nonce) to be used with the AEAD cipher for encrypting the Deterministic Request, while also avoiding reuse of the same (key, nonce) pair across different requests.

Diversity can conceptually be enforced by applying a cryptographic hash function to the complete input of the encryption operation over the plain CoAP request (i.e., the AAD and the plaintext of the COSE object), and then using the result as source of uniqueness. Any non-malleable cryptographically secure hash of sufficient length to make collisions sufficiently unlikely is suitable for this purpose.

A tempting possibility is to use a fixed key, and use the hash as a deterministic AEAD nonce for each Deterministic Request throught the Partial IV component (see Section 5.2 of [RFC8613]). However, the 40 bit available for the Partial IV are by far insufficient to ensure that the deterministic nonce is not reused across different Deterministic Requests. Even if the full deterministic AEAD nonce could be set, the sizes used by common algorithms would still be too small.

As a consequence, the proposed method takes the opposite approach, by considering a fixed deterministic AEAD nonce, while generating a different deterministic encryption key for each Deterministic Request. That is, the hash computed over the plain CoAP request is taken as input to the key generation. As an advantage, this approach does not require to transport the computed hash in the OSCORE option.

[Note: This has a further positive side effect arising with version -11 of Group OSCORE. That is, since the full encoded OSCORE option is part of the AAD, it avoids a circular dependency from feeding the AAD into the hash computation, which in turn needs crude workarounds like building the full AAD twice, or zeroing out the hash-to-be.]

2.2. Request-Hash

In order to transport the hash of the plain CoAP request, a new CoAP option is defined, which MUST be supported by clients and servers that support Deterministic Requests.

The option is called Request-Hash. As summarized in Figure 1, the Request-Hash option is elective, safe to forward, part of the cache key and repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD1				x	Request-Hash	opaque	any	(none)

Figure 1: Request-Hash Option

The Request-Hash option is identical in all its properties to the Request-Tag option defined in [I-D.ietf-core-echo-request-tag], with the following exceptions:

* It may be arbitrarily long.

  Implementations can limit its length to that of the longest output of the supported hash functions.

* A proxy MAY use any fresh cached response from the selected server to respond to a request with the same Request-Hash (or possibly even if the new request's Request-Hash is a prefix of the cached one).

This is a potential future optimization which is not mentioned anywhere else yet, and allows clients to elide all other options and payload if it has reason to believe that it can produce a cache hit with the abbreviated request alone.

* When used with a Deterministic Request, this option is created at message protection time by the client, and used before message unprotection by the server. Therefore, when used in a Deterministic Request, this option is treated as Class U for OSCORE [RFC8613]. Other uses of this option can put it into different classes for OSCORE the processing.

2.3. Use of Deterministic Requests

This section defines how a Deterministic Request is built on the client side and then processed on the server side.

2.3.1. Pre-Conditions

The use of Deterministic Requests in an OSCORE group requires that the interested group members are aware of the Deterministic Client in the group. In particular, they need to know:

* The Sender ID of the Deterministic Client, to be used as 'kid' parameter for the Deterministic Requests. This allows all group members to compute the Sender Key of the Deterministic Client.

* The hash algorithm to use for computing the hash of a plain CoAP request, when producing the associated Deterministic Request.

* Optionally, a creation timestamp associated to the Deterministic Client. This is aligned with the Group Manager that might replace the current Deterministic Client with a new one with a different Sender ID, e.g. to enforce freshness indications without rekeying the whole group.

Group members have to obtain this information from the Group Manager. A group member can do that, for instance, when obtaining the group key material upon joining the OSCORE group, or later on as an active member by sending a request to a dedicated resource at the Group Manager. In either case, information on the latest Deterministic Client is returned.

The Group Manager defined in [I-D.ietf-ace-key-groupcomm-oscore] can be easily extended to support the provisioning of information about the Deterministic Client; no such extension has been drafted as of the publication of this draft.

2.3.2. Client Processing of Deterministic Request

In order to build a Deterministic Request, the client protects the plain CoAP request using the pairwise mode of Group OSCORE (see Section 9 of [I-D.ietf-core-oscore-groupcomm]), with the following alterations.

1. When preparing the OSCORE option, the AAD and the AEAD nonce:

   * The used Sender ID is the Deterministic Client's Sender ID.

   * The used Partial IV is 0, hence it does not need to be set in the OSCORE option.

2. The client uses the hash function indicated for the Deterministic Client, and computes a hash H over the following input: the Sender Key of the Deterministic Client, concatenated with the AAD from step 1, concatenated with the COSE plaintext.

   Note that the payload of the plain CoAP request (if any) is not self-delimiting, and thus hash functions are limited to non-malleable ones.

3. The client derives the Pairwise Sender Key K as defined in Section 2.3.1 of [I-D.ietf-core-oscore-groupcomm], with the following differences:

   * The Sender Key of the Deterministic Client is used as first argument of the HKDF.

   * The hash H from step 2 is used as second argument of the HKDF, i.e. as "Shared Secret" computable by all the group members.

     An actual Diffie-Hellman secret cannot be obtained, as there is no public key associated with the deterministic client.

   * The Sender ID of the Deterministic Client is used as value for the 'id' element of the 'info' parameter used as third argument of the HKDF.

4. The client includes a Request-Hash option in the request to protect, with value set to the hash H from Step 2.

5. The client updates the value of the 'request_kid' field in the AAD, and sets it to the hash H from step 2.

(This step is still under active debate: While setting it like that makes the request and response AADs consistent, it is also means that implementations which build the AAD in memory need to do so twice).

1. The client protects the request using the pairwise mode of Group OSCORE as defined in Section 9.3 of [I-D.ietf-core-oscore-groupcomm], using the AEAD nonce from step 1, the AEAD encryption key from step 3, and the finalized AAD from step 5.

2. The client sets FETCH as the outer code of the protected request to make it usable for a proxy's cache, even if no observation is requested [RFC7641].

The result is the Deterministic Request to be sent.

Since the encryption key K is derived using material from the whole plain CoAP request, this (key, nonce) pair is only used for this very message, which is deterministically encrypted unless there is a hash collision between two Deterministic Requests.

The deterministic encryption requires the used AEAD algorithm to be deterministic in itself. This is the case for all the AEAD algorithms currently registered with COSE in [COSE.Algorithms]. For future algorithms, a flag in the COSE registry is to be added.

Note that, while the process defined above is based on the pairwise mode of Group OSCORE, no information about the server takes part to the key derivation or is included in the AAD. This is intentional, since it allows for sending a deterministic request to multiple servers at once (see Section 2.3.5). On the other hand, it requires later checks at the client when verifying a response to a Deterministic Request (see Section 2.3.4).

2.3.3. Server Processing of Deterministic Request

Upon receiving a Deterministic Request, a server performs the following actions.

A server that does not support Deterministic Requests would not be able to create the necessary Recipient Context, and thus will fail decrypting the request.

1. If not already available, the server retrieves the information about the Deterministic Client from the Group Manager, and derives the Sender Key of the Deterministic Client.

2. The server actually recognizes the request to be a Deterministic Request, due to the presence of the Request-Hash option and to the 'kid' parameter of the OSCORE option set to the Sender ID of the Deterministic Client.

   If the 'kid' parameter of the OSCORE option specifies a different Sender ID than the one of the Deterministic Client, the server MUST NOT take the following steps, and instead processes the request as per Section 9.4 of [I-D.ietf-core-oscore-groupcomm].

3. The server retrieves the hash H from the Request-Hash option.

4. The server derives a Recipient Context for processing the Deterministic Request. In particular:

   * The Recipient ID is the Sender ID of the Deterministic Client.

   * The Recipient Key is derived as the key K in step 3 of Section 2.3.2, with the hash H retrieved at the previous step.

5. The server verifies the request using the pairwise mode of Group OSCORE, as defined in Section 9.4 of [I-D.ietf-core-oscore-groupcomm], using the Recipient Context from step 4, with the following differences.

   * The server sets the value of the 'request_kid' field in the AAD to be the hash H from step 3.

   * The server does not perform replay checks against a Replay Window (see below).

In case of successful verification, the server MUST also perform the following actions, before possibly delivering the request to the application.

* Starting from the recovered plain CoAP request, the server MUST recompute the same hash that the client computed at step 2 of Section 2.3.2.

  If the recomputed hash value differs from the value retrieved from the Request-Hash option at step 3, the server MUST treat the request as invalid and MAY reply with an unprotected 4.00 (Bad Request) error response. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the string "Decryption failed".

This prevents an attacker that guessed a valid authentication tag for a given Request-Hash value to poison caches with incorrect responses.

* The server MUST verify that the unprotected request is safe to be processed in the REST sense, i.e. that it has no side effects. If verification fails, the server MUST discard the message and SHOULD reply with a protected 4.01 (Unauthorized) error response.

  Note that some CoAP implementations may not be able to prevent that an application produces side effects from a safe request. This may incur checking whether the particular resource handler is explicitly marked as eligible for processing deterministic requests. An implementation may also have a configured list of requests that are known to be side effect free, or even a pre-built list of valid hashes for all sensible requests for them, and reject any other request.

  These checks replace the otherwise present requirement that the server needs to check the Replay Window of the Recipient Context (see step 5 above), which is inapplicable with the Recipient Context derived at step 4 from the value of the Request-Hash option. The reasoning is analogous to the one in [I-D.amsuess-lwig-oscore] to treat the potential replay as answerable, if the handled request is side effect free.

2.3.4. Response to a Deterministic Request

Both when protecting and unprotecting the response, the 'request_kid' field of the external AAD is replaced with the Request-Hash value. This creates the request-response binding ensuring that no mismatched responses can be successfully unprotected.

[Note: Mismatching this with the actual request's 'request_kid' (that stays the Deterministic Client's Sender ID) is ugly, but also the only way to avoid any zeroing/rebuilding of the AAD.]

[Suggestion for any OSCORE v2: avoid request details in the request's AAD as individual elements. Rather than having 'request_kid', 'request_piv' and (in Group OSCORE) 'request_kid_context' as separate fields, they can better be something more pluggable.]

When preparing the response, the server performs the following actions.

* The server sets a non-zero Max-Age option, thus making the Deterministic Request usable for the proxy cache.

* The server MUST protect the response using the group mode of Group OSCORE, as defined in Section 8.3 of [I-D.ietf-core-oscore-groupcomm]. This is required to ensure that the client can verify source authentication of the response, since the "pairwise" key used for the Deterministic Request is actually shared among all the group members.

  In particular, the server sets the value of the 'request_kid' field in the AAD to be the hash H retrieved from the Request-Hash option of the Deterministic Request (see step 3 in Section 2.3.3).

* The server MUST use its own Sender Sequence Number as Partial IV to protect the response, and include it as Partial IV in the OSCORE option of the response. This is required since the server does not perform replay protection on the Deterministic Request (see Section 2.3.4).

* The server uses 2.05 (Content) as outer code even though it is not necessarily an Observe notification [RFC7641], in order to make the response cacheable.

Upon receiving the response, the client performs the following actions.

* In case the response includes a 'kid' in the OSCORE option and unless responses from multiple servers are expected (see Section 2.3.5), the client MUST verify it to be exactly the 'kid' of the server to which the Deterministic Request was sent.

* The client verifies the response using the group mode of Group OSCORE, as defined in Section 8.4 of [I-D.ietf-core-oscore-groupcomm]. In particular, the client verifies the counter signature in the response, based on the 'kid' of the server it sent the request to.

2.3.5. Deterministic Requests to Multiple Servers

A Deterministic Request _can_ be sent to a CoAP group, e.g. over UDP and IP multicast [I-D.ietf-core-groupcomm-bis], thus targeting multiple servers at once.

To simplify key derivation, such a Deterministic Request is still created in the same way as a one-to-one request and still protected with the pairwise mode of Group OSCORE, as defined in Section 2.3.2.

[Note: If it was protected with the group mode, the request hash would need to be fed into the group key derivation just for this corner case. Furthermore, there would need to be a signature from the absent public key.]

When a server receives a request from the Deterministic Client as addressed to a CoAP group, the server MUST include its own Sender ID in the response, as 'kid' parameter of the OSCORE option.

Although it is normally optional for the server to include its Sender ID when replying to a request protected in pairwise mode, it is required in this case for allowing the client to retrieve the Recipient Context associated to the server originating the response.

3. Security Considerations

The same security considerations from [RFC7252][I-D.ietf-core-groupcomm-bis][RFC8613][I-D.ietf-core-oscore-groupcomm] hold for this document.

Compared to Group OSCORE, deterministic requests dispense with some of OSCORE's security properties by just so much as to make caching possible:

* Receiving a response to a deterministic request does not mean that the response was generated after the request was sent.

  It still contains two freshness statements, though:

  - It is more recent than any other response from the same group member that has a smaller sequence number.

  - It is more recent than the original creation of the deterministic security context.

* Request confidentiality is limited.

  An intermediary can determine that two requests from different clients are identical, and associate the different responses generated for them. Padding is suggested for responses where necessary.

* Source authentication for requests is lost.

  Instead, the server must verify that the request (precisely: its handler) is side effect free. The distinct semantics of the CoAP request codes can help the server make that assessment.

[More on the verification of the Deterministic Request]

4. IANA Considerations

This document has the following actions for IANA.

4.1. 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	Request-Hash	[[this document]]
TBD2	Padding	[[this document]]

Figure 2: CoAP Option Numbers

[

For the Request-Hash option, the number suggested to IANA is 548.

For the Padding option, the option number is picked to be the highest number in the Experts Review range; the high option number allows it to follow practically all other options, and thus to be set when the final unpadded message length including all options is known. Therefore, the number suggested to IANA is 64988.

Applications that make use of the "Experimental use" range and want to preserve that property are invited to pick the largest suitable experimental number (65532)

Note that unless other high options are used, this means that padding a message adds an overhead of at least 3 bytes, i.e. 1 byte for option delta/length and two more bytes of extended option delta. This is considered acceptable overhead, given that the application has already chosen to prefer the privacy gains of padding over wire transfer length.

]

5. References

5.1. 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-02, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-groupcomm-bis-02.txt>.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-10, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-oscore-groupcomm-10.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-14, 24 September 2020, <http://www.ietf.org/internet-drafts/draft-ietf-cose-rfc8152bis-struct-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, <http://www.ietf.org/internet-drafts/draft-ietf-cose-rfc8152bis-algs-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>.

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

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

[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>.

[COSE.Algorithms] IANA, "COSE Algorithms", <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>.

5.2. Informative References

[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>.

[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-11, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-echo-request-tag-11.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-09, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-key-groupcomm-oscore-09.txt>.

[I-D.amsuess-lwig-oscore] Amsuess, C., "OSCORE Implementation Guidance", Work in Progress, Internet-Draft, draft-amsuess-lwig-oscore-00, 29 April 2020, <http://www.ietf.org/internet-drafts/draft-amsuess-lwig-oscore-00.txt>.

[I-D.tiloca-core-observe-multicast-notifications] Tiloca, M., Hoeglund, R., Amsuess, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-tiloca-core-observe-multicast-notifications-04, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-tiloca-core-observe-multicast-notifications-04.txt>.

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

Appendix A. Change log

Since -00:

* More precise specification of the hashing (guided by first implementations)

* Focus shifted to deterministic requests (where it should have been in the first place; all the build-up of Token Requests was moved to a motivating appendix)

* Aligned with draft-tiloca-core-observe-responses-multicast-05 (not submitted at the time of submission)

* List the security properties lost compared to OSCORE

Appendix B. Padding

As discussed in Section 3, information can be leaked by the length of a response or, in different contexts, of a request.

In order to hide such information and mitigate the impact on privacy, the following Padding option is defined, to allow increasing a message's length without changing its meaning.

The option can be used with any CoAP transport, but is especially useful with OSCORE as that does not provide any padding of its own.

Before choosing to pad a message by using the Padding option, application designers should consider whether they can arrange for common message variants to have the same length by picking a suitable content representation; the canonical example here is expressing "yes" and "no" with "y" and "n", respectively.

B.1. Definition of the Padding Option

As summarized in Figure 3, the Padding option is elective, safe to forward and not part of the cache key; these follow from the usage instructions. The option may be repeated, as that may be the only way to achieve a certain total length for the padded message.

No.	C	U	N	R	Name	Format	Length	Default
TBD2			x	x	Padding	opaque	any	(none)

Figure 3: Padding Option

B.2. Using and processing the Padding option

A client may set the Padding option, specifying any content of any length as its value.

A server MUST ignore the option.

Proxies are free to keep the Padding option on a message, to remove it or to add further padding of their own.

Appendix C. Simple Cachability using Ticket Requests

Building on the concept of Phantom Requests and Informative Responses defined in [I-D.tiloca-core-observe-multicast-notifications], basic caching is already possible without building a Deterministic Request.

This appendix is not provided for application (for it is only efficient when dealing with very large representations and no OSCORE inner Block-Wise mode, which is inefficient for other reasons, and for observations which are already well covered in [I-D.tiloca-core-observe-multicast-notifications]). It is more provided as a "mental exercise" for the authors and interested readers to bridge the gap between these documents.

That is, instead of replying to a client with a regular response, a server can send an Informative Response, defined as a protected 5.03 (Service Unavailable) error message. The payload of the Informative Response contains the Phantom Request, which is a Ticket Request in this document's broader terminology.

Unlike a Deterministic Request, a Phantom Request is protected in Group Mode. Instead of verifying a hash, the client can see from the signature that this was indeed the request the server is answering. The client also verifies that the request URI is identical between the original request and the Ticket Request.

The remaining exchange largely plays out like in [I-D.tiloca-core-observe-multicast-notifications]'s "Example with a Proxy and Group OSCORE": The client sends the Phantom Request to the proxy (but, lacking a "tp_info", without a Listen-To-Multicast-Responses option), which forwards it to the server for lack of the option.

The server then produces a regular response and includes a non-zero Max-Age option as an outer CoAP option. Note that there is no point in including in an inner Max-Age option, as the client could not pin it in time.

When a second, different client later asks for the same resource at the same server, its new request uses a different 'kid' and 'Partial IV' than the first client's. Thus, the new request produces a cache miss at the proxy and is forwarded to the server, which responds with the same Ticket Request provided to the first client. After that, when the second client sends the Ticket Request, a cache hit at the proxy will be produced, and the Ticket Request can be served from the proxy's cache.

When multiple proxies are in use, or the response has expired from the proxy's cache, the server receives the Ticket Request multiple times. It is a matter of perspective whether the server treats that as an acceptable replay (given that this whole mechansim only makes sense on requests free of side effects), or whether it is conceptualized as having an internal proxy where the request produces a cache hit.

Appendix D. Application for more efficient end-to-end protected multicast notifications

Comparing the "Example with a Proxy" and the "Example with a Proxy and Group OSCORE" in [I-D.tiloca-core-observe-multicast-notifications] shows that with OSCORE, more requests than without need to hit the server. This is because every client originally protects their request individually and thus needs a custom response served to send the Phantom Request as a Ticket Request.

If the clients send their deterministic requests in a deterministic way, and the server uses these requests as Ticket Requests as well, then there is no need for a Phantom Request to be sent back to the client.

Instead, the server can send an unprotected Informative Response very much like in the example without OSCORE, setting the proxy up and giving the latest response along the way.

The proxy can thus be configured by the server with the first request, and has an active observation and a fresh cache entry in time for the second client to arrive.

Appendix E. Open questions

* Is "deterministic encryption" something worthwhile to consider in COSE?

  COSE would probably specify something more elaborate for the KDF (the current KDF round is the pairwise mode's; COSE would probably run through KDF with a KDF context structure).

  COSE would give a header parameter name to the Request-Hash (which for the purpose of OSCORE deterministic requests would put back into Request-Hash by extending the option compression function across the two options).

  Conceptually, they should align well, and the implementation changes are likely limited to how the KDF is run.

* An unprotection failure from a mismatched hash will not be part of the ideally constant-time code paths that otherwise lead to AEAD unprotect failures. Is that a problem?

  After all, it does tell the attacker that they did succeed in producing a valid MAC (it's just not doing it any good, because this key is only used for deterministic requests and thus also needs to pass the Request-Hash check).

Appendix F. Unsorted further ideas

* All or none of the deterministic requests should have an inner observe option. Preferably none - that makes messages shorter, and clients need to ignore that option either way when checking whether a Consensus Request matches their intended request.

Acknowledgments

The authors sincerely thank Jim Schaad for his 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

Christian Amsüss
Austria

Email: christian@amsuess.com

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

Email: marco.tiloca@ri.se

CoRE   C. Amsüss
Internet-Draft   22 February 2021
Intended status: Experimental
Expires: 26 August 2021

CoRE Resource Directory Extensions
draft-amsuess-core-resource-directory-extensions-05

Abstract

A collection of extensions to the Resource Directory [I-D.ietf-core-resource-directory] that can stand on their own, and have no clear future in specification yet.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
2. Reverse Proxy requests 3
  2.1. Discovery 3
  2.2. Registration 3
    2.2.1. Registration updates 4
    2.2.2. Proxy behavior 5
    2.2.3. On-Demand proxying 5
    2.2.4. Multiple upstreams 5
    2.2.5. Examples 6
    2.2.6. Notes on stability and maturity 7
    2.2.7. Security considerations 7
3. Infinite lifetime 7
  3.1. Example 8
4. Lookup across link relations 8
  4.1. Example 9
5. Lifetime Age 10
6. Zone identifier introspection 10
  6.1. Example 11
7. Proxying multicast requests 11
  7.1. Example 12
8. Opportunistic RD 12
  8.1. Applications 15
9. References 15
  9.1. Normative References 15
  9.2. Informative References 16
Appendix A. Change log 17
Appendix B. Acknowledgements 18
Author's Address 18

1. Introduction

This document pools some extensions to the Resource Directory [I-D.ietf-core-resource-directory] that might be useful but have no place in the original document.

They might become individual documents for IETF submission, simple registrations in the RD Parameter Registry at IANA, or grow into a shape where they can be submitted as a collection of tools.

At its current state, this draft is a collection of ideas.

[This document is being developed at https://gitlab.com/chrysn/resource-directory-extensions (https://gitlab.com/chrysn/resource-directory-extensions).]

2. Reverse Proxy requests

When a registrant registers at a Resource Directory, it might not have a suitable address it can use as a base address. Typical reasons include being inside a NAT without control over port forwarding, or only being able to open outgoing connections (as program running inside a web browser utilizing CoAP over WebSocket [RFC8323] might be).

[I-D.ietf-core-resource-directory] suggests (in the Cellular M2M use case) that proxy access to such endpoints can be provided, it gives no concrete mechanism to do that; this is such a mechanism.

This mechanism is intended to be a last-resort option to provide connectivity. Where possible, direct connections are preferred. Before registering for proxying, the registrant should attempt to obtain a publicly available port, for example using PCP ([RFC6887]).

The same mechanism can also be employed by clients that want to conceal their network address from its clients.

A deployed application where this is implicitly done is LwM2M [citation missing]. Notable differences are that the protocol used between the client and the proxying RD is not CoAP but application specific, and that the RD (depending on some configuration) eagerly populates its proxy caches by sending requests and starting observations at registration time.

2.1. Discovery

An RD that provides proxying functionality advertises it by announcing the additional resource type "TBD1" on its directory resource.

2.2. Registration

A client passes the "proxy=yes" or "proxy=ondemand" query parameter in addition to (but typically instead of) a "base" query parameter.

A server that receives a "proxy=yes" query parameter in a registration (or receives "proxy=ondemand" and decides it needs to proxy) MUST come up with a "Proxy URL" on which it accepts requests, and which it uses as a Registration Base URI for lookups on the present registration.

The Proxy URL SHOULD have no path component, as acting as a reverse proxy in such a scenario means that any relative references in all representations that are proxied must be recognized and possibly rewritten.

The RD MAY mint several alternative Registration Base URIs using different protocols to make the proxied content available; [I-D.silverajan-core-coap-protocol-negotiation] can be used to advertise them.

The registrant is not informed of the chosen public name by the RD.

This mechanism is applicable to all transports that can be used to register. If proxying is active, the restrictions on when the base parameter needs to be present ([I-D.ietf-core-resource-directory] Registration template) are relaxed: The base parameter may also be absent if the connection originates from an ephemeral port, as long as the underlying protocol supports role reversal, and link-local IPv6 addresses may be used without any concerns of expressibility.

If the client uses the role reversal rule relaxation, both it and the server keep that connection open for as long as it wants to be reachable. When the connection terminates, the RD SHOULD treat the registration as having timed out (even if its lifetime has not been exceeded) and MAY eventually remove the registration. It is yet to be decided whether the RD's announced ability to do proxying should imply that infinite lifetimes are necessarily supported for such registrations; at least, it is RECOMMENDED.

2.2.1. Registration updates

The "proxy" query parameter can not be changed or repeated in a registration update; RD servers MUST answer 4.00 Bad Request to any registration update that has a "proxy" query parameter.

As always, registration updates can explicitly or implicitly update the Registration Base URI. In proxied registrations, those changes are not propagated to lookup, but do change the forwarding address of the proxy.

For example, if a registration is established over TCP, an update can come along in a new TCP connection. Starting then, proxied requests are forwarded along that new connection.

2.2.2. Proxy behavior

The RD operates as a reverse-proxy as described in [RFC7252] Section 5.7.3 at the announced Proxy URL(s), where it decides based on the requested host and port to which registrant endpoint to forward the request.

The address the incoming request are forwarded to is the base address of the registration. If an explicit "base" paremter is given, the RD will forward requests to that location. Otherwise, it forwards to the registration's source address (which is the implied base parameter).

When an implicit base is used, the requests forwarded by the RD to the EP contain no Uri-Host option. EPs that want to run multiple parallel registrations (especially gateway-like devices) can either open up separate connections, or use an additional (to-be-specified) mechanism to set the "virtual host name" for that registration in a separate argument.

2.2.3. On-Demand proxying

If an endpoint is deployed in an unknown network, it might not know whether it is behind a NAT that would require it to configure an explicit base address, and ask the RD to assist by proxying if necessary by registering with the "proxy=ondemand" query parameter.

A server receiving that SHOULD use a different IP address to try to access the registrant's .well-known/core file using a GET request under the Registration Base URI. If that succeeds, it may assume that no NAT is present, and ignore the proxying request. Otherwise, it configures proxying as if "proxy=yes" were requested.

Note that this is only a heuristic [and not tested in deployments yet].

2.2.4. Multiple upstreams

When a proxying RD is operating behind a router that has uplinks with multiple provisioning domains (see [RFC7556]) or a similar setup, it MAY mint multiple addresses that are reachable on the respective provisioning domains. When possible, it is preferred to keep the number of URIs handed out low (avoiding URI aliasing); this can be achieved by announcing both the proxy's public addresses under the same wildcard name.

If RDs are announced by the uplinks using RDAO, the proxy may use the methods of [I-D.amsuess-core-rd-replication] to distribute its registrations to all the announced upstream RDs.

In such setups, the router can forward the upstream RDs using the PvD option ([RFC8801]) to PvD-aware hosts and only announce the local RD to PvD-unaware ones (which then forwards their registrations). It can be expected that PvD-aware endpoints are capable of registering with multiple RDs simultaneously.

2.2.5. Examples

2.2.5.1. Registration through a firewall

```
Req from [2001:db8:42::9876]:5683:
POST coap://rd.example.net/rd?ep=node9876&proxy=ondemand
</some-resource>;rt="example.x"

Req from other-address.rd.example.net:
GET coap://[2001:db8:42::9876]/.well-known/core

Request blocked by stateful firewall around [2001:db8:42::]

RD decides that proxying is necessary

Res: 2.04 Created
Location: /reg/abcd
```

Later, lookup of that registration might say:

```
Req: GET coap://rd.example.net/lookup/res?rt=example.x

Res: 2.05 Content
<coap://node987.rd.example.net/some-resource>;rt="example.x
```

A request to that resource will end up at an IP address of the RD, which will forward it using its the IP and port on which the registrant had registered as source port, thus reaching the registrant through the stateful firewall.

2.2.5.2. Registration from a browser context

```
Req: POST coaps+ws://rd.example.net/rd?ep=node1234&proxy=yes
</gyroscope>;rt="core.s"

Res: 2.04 Created
Location: /reg/123
```

The gyroscope can now not only be looked up in the RD, but also be reached:

```
Req: GET coap://rd.example.net/lookup/res?rt=core.s

Res: 2.05 Content
<coap://[2001:db8:1::1]:10123/gyroscope>;rt="core.s"
```

In this example, the RD has chosen to do port-based rather than host-based virtual hosting and announces its literal IP address as that allows clients to not send the lengthy Uri-Host option with all requests.

2.2.6. Notes on stability and maturity

Using this with UDP can be quite fragile; the author only draws on own experience that this can work across cell-phone NATs and does not claim that this will work over generic firewalls.

[It may make sense to have the example as TCP right away.]

2.2.7. Security considerations

An RD MAY impose additional restrictions on which endpoints can register for proxying, and thus respond 4.01 Unauthorized to request that would pass had they not requested proxying.

Attackers could do third party registrations with an attacked device’s address as base URI, though the RD would probably not amplify any attacks in that case.

The RD MUST NOT reveal the address at which it reaches the registrant except for adaequately authenticated and authorized debugging purposes, as that address could reveal sensitive location data the registrant may wish to hide by using a proxy.

Usual caveats for proxies apply.

3. Infinite lifetime

An RD can indicate support for infinite lifetimes by adding the resoruce type "TBD2" to its list of resource types.

A registrant that wishes to keep its registration alive indefinitely can set the lifetime value as "lt=inf".

Registrations with infinite lifetimes never time out. Unlike regular registrations, they are not "soft state"; the registrant can expect the RD to persist the registrations across network changes, reboots, softare updates and that like.

Typical use cases for infinite life times are:

* Commissioning tools (CTs) that do not return to the deployment site, and thus can not refresh the soft state

* Proxy registrations whose lifetime is limited by a connection that is kept alive

3.1. Example

Had the example of Section 2.2.5.2 discovered support for infinite lifetimes during lookup like this:

```
Req: GET coaps+ws://rd.example.net/.well-known/coer?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd TBD1 TBD2";ct=40
```

it could register like that:

```
Req: POST coaps+ws://rd.example.net/rd?ep=node1234&proxy=yes&lt=inf
</gyroscope>;rt="core.s"

Res: 2.04 Created
Location: /reg/123
```

and never need to update the registration for as long as the websocket connection is open.

(When it gets terminated, it could try renewing the registration, but needs to be prepared for the RD to already have removed the original registration.)

4. Lookup across link relations

Resource lookup occasionally needs execute multiple queries to follow links.

An RD server (or any other server that supports [RFC6690] compatible lookup), can announce support for following links in resource lookups by announcing support for the TBD3 interface type on its resource lookup.

A client can the query that server to not only provide the matched links, but also links that are reachable over relations given in "follow" query parameters.

4.1. Example

Assume a node presents the following data in its <.well-known/core> resource (and submitted the same to the RD):

```
</temp>;if="core.s";rt="example.temperature",
</t-prot>;rel="calibration-protocol";anchor="/temp",
<http://vendor.example.com/temp9000>;rel="describedby";anchor="/temp",
</hum>;if="core.s";rt="example.humidity",
</h-prot>;rel="calibration-protocol";anchor="/hum",
```

A lookup client can, in one query, find the temperature sensor and its relevant metadata:

```
Req: GET /rd-lookup/res?rt=example.temperature&follow=calibration-protocol&follow
=describedby

<coap://node1/temp>;if="core.s";rt="example.temperature";anchor="coap://node1",
<coap://node1/t-prot>;rel="calibration-protocol";anchor="coap://node1/temp",
<http://vendor.example.com/temp9000>;rel="describedby";anchor="coap://node1/temp"
,
```

[There is a better example (https://github.com/ace-wg/ace-oauth/issues/120#issuecomment-407997786) in an earlier stage of [I-D.tiloca-core-oscore-discovery]]

Given the likelihood of a CoRAL based successor to [RFC6690], this lookup variant might easily be superseeded by a CoRAL FETCH format; it might look like this there:

```
Req: FETCH /reef-lookup
Content-Format: application/template-coral+cbor
Payload:
#using core = <...>
#using reef = <...>
reef:content ?x {
  core:rt "example.temperature"
  calibration-protocol ?y {
    core:describedby ?z
  }
}

Res: 2.01 Content
Content-Format: aplication/coral+cbor
Payload:
reef:content <coap://node1/temp> {
    core:rt "example.temperature"
    calibration-protocol <coap://node1/t-prot> {
      core:describedby <http://vendor.example.com/temp9000>
    }
}
```

5. Lifetime Age

This extension is described in [I-D.amsuess-core-rd-replication] Section 5.2.

The "provenance" extension in Section 5.1 of the same document should probably be expressed differently to avoid using non-target link attributes.

6. Zone identifier introspection

The 'split-horizon' mechanism introduced in [I-D.ietf-core-resource-directory] (-19) (that registrations with link-local bases can only be read from the zone they registered on) reduces the usability of the endpoint lookup interface for debugging purposes.

To allow an administrator to read out the "show-zone-id" query parameter for endpoint and resource lookup is introduced.

A Resource Directory that understands this parameter MUST NOT limit lookup results to registrations from the lookup's zone, and MUST use [RFC6874] zone identifiers to annotate which zone those registrations are valid on.

The RD MUST limit such requests to authenticated and authorized debugging requests, as registrants may rely on the RD to keep their presence secret from other links.

6.1. Example

```
Req: GET /rd-lookup/ep?show-zone-id&et=printer

Res: 2.05 Content
</reg/1>;base="coap://[2001:db8::1]";et=printer;ep="bigprinter",
</reg/2>;base="coap://[fe80::99%wlan0]";et=printer;ep="localprinter-1234",
</reg/3>;base="coap://[fe80::99%eth2]";et=printer;ep="localprinter-5678",
```

7. Proxying multicast requests

Multicast requests are hard to forward at a proxy: Even if a media type is used in which multiple responses can be aggregated transparently, the proxy can not reliably know when all responses have come in. [RFC7390] Section 2.9 destribes the difficulties in more detail.

Note that [I-D.tiloca-core-groupcomm-proxy] provides a mechanism that _does_ allow the forwarding of multicast requests. It is yet to be determined what the respective pros and cons are. Conversely, that lookup mechanism may also serve as an alternative to resource lookup on an RD.

A proxy MAY expose an interface compatible with the RD lookup interface, which SHOULD be advertised by a link to it that indicates the resource types core.rd-lookup-res and TBD4.

The proxy sends multicast requests to All CoAP Nodes ([RFC7252] Section 12.8) requesting their .well-known/core files either eagerly (ie. in regular intervals independent of queries) or on demand (in which case it SHOULD limit the results by applying [RFC6690] query filtering; if it has received multiple query parameters it should forward the one it deems most likely to limit the results, as .well-known/core only supports a single query parameter).

In comparison to classical RD operation, this RD behaves roughly as if it had received a simple registration with a All CoAP Nodes address as the source address, if such behavior were specified. The individual registrations that result from this neither have an explicit registration resource nor an explicit endpoint name; given that the endpoint lookup interface is not present on such proxies, neither can be queried.

Clients that would intend to do run a multicast discovery operation behind the proxy can then instead query that resource lookup interface. They SHOULD use observation on lookups, as an on-demand implementation MAY return the first result before others have arrived, or MAY even return an empty link set immediately.

7.1. Example

```
Req: GET coap+ws://gateway.example.com/.well-known/core?rt=TBD4

Res: 2.05 Content
</discover>;rt="core.rd-lookup-res TBD4";ct=40

Req: GET coap+ws://gateway.example.com/discover?rt=core.s
Observe: 0

Res: 2.05 Content
Observe: 0
Content-Format: 40
(empty payload)
```

At the same time, the proxy sends out multicast requests on its interfaces:

```
Req: GET coap://ff05::fd/.well-known/core?rt=core.s

Res (from [2001:db8::1]:5683): 2.05 Content
</temp>;ct="0 112";rt="core.s"

Res (from [2001:db8::2]:5683): 2.05 Content
</light>;ct="0 112";rt="core.s"
```

upon receipt of which it sends out a notification to the websocket client:

```
Res: 2.05 Content
Observe: 1
Content-Format: 40
<coap://[2001:db8::1]/temp>;ct="0 112";rt="core.s";anchor="coap://[2001:db8::1]",
<coap://[2001:db8::2]/light>;ct="0 112";rt="core.s";anchro="coap://[2001:db8::2]"
```

8. Opportunistic RD

An application that wants to advertise its resources in Resource Directory can find itself in a network that has no RD deployed. It may be able to start an RD on its own to fill in that gap until an explicitly configured one gets installed.

This bears the risk of having competing RDs on the same network, where resources registered at one can not be discovered on the other. To mitigate that, such Opportunistic Resource Directories should follow those steps:

* The RD chooses its own Opportunistic Capability value. That integer number is an estimate of number of target attributes it expects to be able to store, where in absence of better estimates one would assume that a registration contains 16 links, and each links contains three target attributes each with an eight byte key and a 16 byte value.

  The Opportunistic Capability value is advertised as a TBD5-cap= target attribute on the registration resource.

* The RD chooses its own Opportunistic Tie-Break value. That integer number needs no other properties than being likely to be different even for two instances of the same device being started; numeric forms of MAC addresses or random numbers make good candidates.

  The Opportunistic Capability value is advertised as a TBD5-tie= target attribute on the registration resource.

* The Opportunistic RD, before advertising its resources, performs RD discovery itself, using at least all the discovery paths it may become discoverable on itself.

* If the Opportunistic RD finds no other RD, or if the RD it finds is less capable than itself, it can start advertising itself as a Resource Directory.

  An RD is called more capable than another if its TBD5-cap value is greater than the other's, or if its TBD5-tie value is greater than the other's if the former results in a tie. Absent or unparsable attributes are considered greater than any present attribute.

  In case an RD observes a tie even after evaluating the tie breaker, it may change its Opportunistic Tie-Break value if that was picked randomly, or reevaluate its life choices if it uses its own MAC address.

* A running Opportunistic RD needs to perform discovery for other RDs repeatedly. If it discovers a more capable RD, it stops advertising its own resources. It should continue to serve lookup requests, but refuse any new registration or registration updates (which will trigger the registrant endpoints to look for a new RD).

An inactive Opportunistic RD will be notified of the highter capability RD's shutdown by the expiry of whatever it may be started to advertise that was now advertised there; see below for possible improvements.

* An RD that discovers an Opportunistic RD of lower capability may speed up the transition process by (not mutually exclusive) two ways

  - It can register its own (registration) resource(s) into the lower capability directory. That RD can take that as having discovered a higher capability RD and stop advertising.

  - It can expose resources and registrations of the lower capability directory using the methods described in [I-D.amsuess-core-rd-replication].

* An Opportunistic RD that yields to a more capable RD may ease the transition by attempting to register its active registrations at the more capable RD, taking the role of a CT. The lifetimes picked for those must not exceed the remaining lifetime of its registrations, and it must not renew those registrations.

Future iterations of this document may want to cut down on the possibilities listed above.

Some ideas are around for making the shutdown of a commissioned or otherwise high-capability RD more graceful, but they still have some problems

* Setting a commissioned or high-capability RD's Capability to zero in preparation of the shutdown may create loops in any distributed lookups.

* Asking the lower capability RD to register its registration resource (even though not otherwise advertised) at the higher capability RD still creates a situation where clients may find two RDs running simultaneously, and we can't expect clients to make any decisions based on TBD5 values.

* Asking the higher capability RD to register its registration resource with the lower capability RD contradicts the current recommendation for the passive Opportunistic RD to not accept registrations / renewals. Also, the deployed RD may not know that Opportunistic RDs are a thing.

* Advertising an almost-but-not-quite rt= value on passive registration resources may be an option, but needs to be thought through thoroughly.

Installations of Opportunistic RDs are at special risk of resource exhaustion because they are not sized with their actual deployment in mind, but rely on defaults set by the application that starts the RD. Opportunistic RDs should only be started if the application's administrator can be informed in a timely fashion when the RD's resources are nearing exhaustion; guidance towards installing a more capable RD on the network should be provided in that case.

8.1. Applications

* Group managers using [I-D.tiloca-core-oscore-discovery] can ship its own low-priority Opportunistic RD to announce its join resources. This provides benefits over announcing them on multicast discovery if the network can efficiently route requests to the All CoAP Resource Directories multicast address (so group members get a response back from an early focused request to all RDs rather than falling back to multicasting All CoAP Nodes for "?rt=osc.j&...."), or if discovery of the group's multicast address is used.

* Administrative tools that try to provide a broad overview of a network's CoAP devices could offer to open an Opportunistic RD if they find no active RD on the network (but should ask the user in interactive scenarios).

  That allows them to see devices that newly join the network quickly (by observing their own or the found RD), rather than relying periodic multicasts.

9. References

9.1. Normative References

[I-D.amsuess-core-rd-replication]
Amsuess, C., "Resource Directory Replication", Work in Progress, Internet-Draft, draft-amsuess-core-rd-replication-02, 11 March 2019, <http://www.ietf.org/internet-drafts/draft-amsuess-core-rd-replication-02.txt>.

[I-D.ietf-core-resource-directory]
Amsuess, C., Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", Work in Progress, Internet-Draft, draft-ietf-core-resource-directory-26, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-resource-directory-26.txt>.

[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>.

[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>.

9.2. Informative References

[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>.

[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-02, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-tiloca-core-groupcomm-proxy-02.txt>.

[I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-07, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-tiloca-core-oscore-discovery-07.txt>.

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

[RFC6887] Wing, D., Ed., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, DOI 10.17487/RFC6887, April 2013, <https://www.rfc-editor.org/info/rfc6887>.

[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>.

[RFC7556] Anipko, D., Ed., "Multiple Provisioning Domain Architecture", RFC 7556, DOI 10.17487/RFC7556, June 2015, <https://www.rfc-editor.org/info/rfc7556>.

[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>.

[RFC8801] Pfister, P., Vyncke, É., Pauly, T., Schinazi, D., and W. Shao, "Discovering Provisioning Domain Names and Data", RFC 8801, DOI 10.17487/RFC8801, July 2020, <https://www.rfc-editor.org/info/rfc8801>.

Appendix A. Change log

Since -04:

* Minor adjustments:

  - Mention LwM2M and how it is already doing RD proxying.

  - Tie proxying in with infinite lifetimes.

  - Remove note on not being able to switch protocols: RDs that support some future protocol negotiation can do that.

  - Point out that there is no Uri-Host from the RD proxy to the EP, but there could be.

  - Infinite lifetimes: Take up CTs more explicitly from RD discussion.

  - Start exploring interactions with groupcomm-proxy.

Since -03:

* Added interaction with PvD (Provisioning Domains)

Since -02:

* Added abstract

* Added example of CoRAL FETCH to Lookup across link relations section

Since -01:

* Added section on Opportunistic RDs

Since -00:

* Add multicast proxy usage pattern

* ondemand proxying: Probing queries must be sent from a different address

* proxying: Point to RFC7252 to describe how the actual proxying happens

* proxying: Describe this as a last-resort options and suggest attempting PCP first

Appendix B. Acknowledgements

[Reviews from: Jaime Jimenez]

Author's Address

Christian Amsüss
Hollandstr. 12/4
1020
Austria

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

TODO Working Group C. Gündoan
Internet-Draft HAW Hamburg
Intended status: Informational C. Amsüss
Expires: 26 August 2021
TC. Schmidt
HAW Hamburg
M. Waehlisch
link-lab & FU Berlin
22 February 2021

A Data-centric Deployment Option for CoAP
draft-gundogan-core-icncoap-00

Abstract

The information-centric networking (ICN) paradigm offers replication of autonomously verifiable content throughout a network, in which content is bound to names instead of hosts. This has proven beneficial in particular for the constrained IoT. Several approaches, the most prominent of which being Content-Centric Networking (CCNx) and Named-Data Networking (NDN), propose access to named content directly on the network layer. Independently, the CoRe WG developed mechanisms that support autonomous content processing, on-path caching, and content object security using CoAP proxies and OSCORE.

This document describes a data-centric deployment option using standard CoAP features to replicate information-centric properties and benefits to the host-centric IoT world.

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://github.com/inetrg/draft-core-icncoap.

Status of This Memo

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 26 August 2021.

Copyright Notice

Table of Contents

1. Introduction 3
2. Requirements Language 3
3. Data-centric Deployment Option for CoAP 3
  3.1. Stateful Forwarding 3
  3.2. Content Caching 4
  3.3. Corrective Actions 4
4. Security Considerations 5
5. IANA Considerations 5
6. References 5
  6.1. Normative References 5
  6.2. Informative References 6
Acknowledgments 6
Authors' Addresses 6

1. Introduction

Information-Centric Networking (ICN) introduced the idea to turn named content objects into first class citizens of the Internet ecosystem. This paradigm gave rise to (i) a decoupling of content from hosts and the ability of ubiquitous content caching without content delivery networks (CDNs), and (ii) serverless routing on names without the DNS infrastructure; (iii) Named Data Networking (NDN) additionally abandoned network endpoint addresses in favor of a stateful forwarding fabric. These properties enable an asynchronous, hop-wise content fetching, which prevents forwarding of unsolicited data. The latter significantly reduces the attack surface of (Distributed) Denial-of-Service (DDoS).

All three constituents make ICN appealing to the (constrained) Internet of Things (IoT) as infrastructural burdens and common DDoS threats stand in the way of a lean and efficient inter-networking for embedded devices. Early experimental work [NDN-IOT] shows that NDN can successfully operate on very constrained nodes with noticeable resource savings compared to IP. In addition, short-term in-network caching proved valuable for increasing reliability in low-power lossy networks with nodes frequently at sleep as common at the IoT edge.

The deployment option described in this document replicates these information-centric properties using standard CoAP features. Recent experimental evaluations [OBJECTSEC][ICN-COAP] in a testbed with real IoT hardware demonstrate promising results.

2. Requirements Language

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.

3. Data-centric Deployment Option for CoAP

3.1. Stateful Forwarding

In the data-centric deployment, all IoT devices act as CoAP proxies with enabled caching functionality. A forwarding information base (FIB) on the application-layer describes a mapping of resource names to next-hop CoAP proxies. This mapping list is compiled statically, or is dynamically discovered in the network; future document iterations will further elaborate on this topic.

Within the IoT stub network, requests traverse multiple proxies, install forwarding state, and build return paths for corresponding responses. The use of IPv6 link-local addresses between each proxy hop is encouraged for a better 6LoWPAN compressibility. Responses return on symmetrical request paths, which consequently consumes existing forwarding state.

3.2. Content Caching

A deployment of proxy nodes on each hop enables a hop-wise caching just as performed by CCNx [RFC8569] and NDN. Responses replicate on a request path following a cache decision and cache replacement strategy. A simple and lightweight approach is to _cache everywhere_ and replace _least recently used_ (LRU) content.

OSCORE enables content object security for CoAP and allows for transmitting autonomously verifiable content similar to CCNx and NDN. Further details on cachable OSCORE messages is recorded in [I-D.draft-amsuess-core-cachable-oscore-00].

3.3. Corrective Actions

In contrast to end-to-end retransmissions for standard CoAP deployments, the data-centric setup performs hop-wise retransmissions in the event of message timeouts. Confirmable messages arm message timers on each proxy node.

Figure 1 illustrates the default retransmission behavior: each subsequent packet traverses the full request path to recover a lost message.

```
         Initial request:

         ,-------,  Request   ,-------,  Request   ,-------,
         |client |------------|router |----------->|server |
         |       |  x---------|       |------------|       |
         '-------'  Response  '-------'  Response  '-------'



         Request retransmission:

         ,-------,  Request   ,-------,  Request   ,-------,
         |client |------------|router |----------->|server |
         |       |<-----------|       |------------|       |
         '-------'  Response  '-------'  Response  '-------'
```

Figure 1: End-to-end recovery of lost packets.

Figure 2 demonstrates the shortening of request paths for subsequent request retransmissions due to the on-path caching functionality.

```
Initial request:

,-------,  Request   ,-------,  Request   ,-------,
| Proxy |----------->| Proxy |----------->| Proxy |
|(cache)|  x---------|(cache)|<-----------|(cache)|
'-------'  Response  '-------'  Response  '-------'



Request retransmission:

,-------,  Request   ,-------,            ,-------,
| Proxy |----------->| Proxy |            | Proxy |
|(cache)|<-----------|(cache)|            |(cache)|
'-------'  Response  '-------'            '-------'
```

Figure 2: Hop-wise recovery of lost packets with on-path caching.

Proxy nodes aggregate requests and suppress the forwarding procedure, if they already maintain an on-going request with the same cache key.

4. Security Considerations

TODO Security

5. IANA Considerations

This document has no IANA actions.

6. References

6.1. Normative References

[I-D.draft-amsuess-core-cachable-oscore-00]
Amsuess, C. and M. Tiloca, "Cachable OSCORE", Work in Progress, Internet-Draft, draft-amsuess-core-cachable-oscore-00, 13 July 2020, <http://www.ietf.org/internet-drafts/draft-amsuess-core-cachable-oscore-00.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>.

[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>.

6.2. Informative References

[ICN-COAP] Gündoan, C., Amsüss, C., Schmidt, TC., and M. Waehlisch, "Toward a RESTful Information-Centric Web of Things: A Deeper Look at Data Orientation in CoAP", Proceedings of 7th ACM ICN, DOI 10.1145/3405656.3418718, 2020, <https://dl.acm.org/doi/10.1145/3405656.3418718>.

[NDN-IOT] Gündoan, C., Kietzmann, P., Lenders, M., Petersen, H., Schmidt, TC., and M. Waehlisch, "NDN, CoAP, and MQTT: a comparative measurement study in the IoT", Proceedings of 5th ACM ICN, DOI 10.1145/3267955.3267967, 2018, <https://dl.acm.org/doi/10.1145/3267955.3267967>.

[OBJECTSEC] Gündoan, C., Amsüss, C., Schmidt, TC., and M. Waehlisch, "IoT Content Object Security with OSCORE and NDN: A First Experimental Comparison", Proceedings of 19th IFIP Networking, 2020, <https://ieeexplore.ieee.org/document/9142731>.

[RFC8569] Mosko, M., Solis, I., and C. Wood, "Content-Centric Networking (CCNx) Semantics", RFC 8569, DOI 10.17487/RFC8569, July 2019, <https://www.rfc-editor.org/info/rfc8569>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Cenk Gündoan
HAW Hamburg

Email: cenk.guendogan@haw-hamburg.de

Christian Amsüss

Email: christian@amsuess.com

Thomas C. Schmidt
HAW Hamburg

Email: t.schmidt@haw-hamburg.de

Matthias Waehlisch
link-lab & FU Berlin

Email: m.waehlisch@haw-hamburg.de

CoRE Working Group R. Hoeglund
Internet-Draft M. Tiloca
Updates: 8613 (if approved) RISE AB
Intended status: Standards Track February 19, 2021
Expires: August 23, 2021

AEAD Key Usage Limits in OSCORE
draft-hoeglund-core-oscore-key-limits-00

Abstract

Object Security for Constrained RESTful Environments (OSCORE) uses AEAD algorithms to ensure confidentiality and integrity of exchanged messages. Due to known issues allowing forgery attacks against AEAD algorithms, limits should be followed on the number of times a specific key is used for encryption or decryption. This document defines how two peers using OSCORE must take these limits into account and what steps they must take to preserve the security of their communications. Therefore, this document updates RFC8613.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
  1.1. Terminology 3
2. Problem Overview 3
  2.1. Limits for 'q' and 'v' 4
3. Additional Information in the Security Context 4
4. OSCORE Messages Processing 5
  4.1. Protecting a Request or a Response 5
  4.2. Verifying a Request or a Response 6
5. Methods for Rekeying OSCORE 6
6. Security Considerations 7
7. IANA Considerations 7
Acknowledgments 7
9. References 8
  9.1. Normative References 8
  9.2. Informative References 8
Authors' Addresses 9

1. Introduction

Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] provides end-to-end protection of CoAP [RFC7252] messages at the application-layer, ensuring message confidentiality and integrity, replay protection, as well as binding of response to request between a sender and a recipient.

In particular, OSCORE uses AEAD algorithms to provide confidentiality and integrity of messages exchanged between two peers. Due to known issues allowing forgery attacks against AEAD algorithms, limits should be followed on the number of times a specific key is used to perform encryption or decryption [I-D.irtf-cfrg-aead-limits].

Should these limits be exceeded, an adversary may break the security properties of the AEAD algorithm, such as message confidentiality and integrity, e.g. by performing a message forgery attack. The original OSCORE specification [RFC8613] does not consider such limits.

This document updates [RFC8613] and defines when a peer must stop using an OSCORE Security Context shared with another peer, due to the reached key usage limits. When this happens, the two peers have to establish a new Security Context with new keying material, in order to continue their secure communication with 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.

Readers are expected to be familiar with the terms and concepts related to the CoAP [RFC7252] and OSCORE [RFC8613] protocols.

2. Problem Overview

The OSCORE security protocol [RFC8613] uses AEAD algorithms to provide integrity and confidentiality of messages, as exchanged between two peers sharing an OSCORE Security Context.

When processing messages with OSCORE, each peer should follow specific limits as to the number of times it uses a specific key. This applies separately to the Sender Key used to encrypt outgoing messages, and to the Recipient Key used to decrypt and verify incoming protected messages.

Exceeding these limits may allow an adversary to break the security properties of the AEAD algorithm, such as message confidentiality and integrity, e.g. by performing a message forgery attack.

The following refers to the two parameters 'q' and 'v' introduced in [I-D.irtf-cfrg-aead-limits], to use when deploying an AEAD algorithm.

- 'q': this parameter has as value the number of messages protected with a specific key, i.e. the number of times the AEAD algorithm has been invoked to encrypt data with that key.

- 'v': this parameter has as value the number of alleged forgery attempts that have been made against a specific key, i.e. the amount of failed decryptions that has been done with the AEAD algorithm for that key.

When a peer uses OSCORE:

- The key used to protect outgoing messages is its Sender Key, in its Sender Context.

- The key used to decrypt and verify incoming messages is its Recipient Key, in its Recipient Context.

Both keys are derived as part of the establishment of the OSCORE Security Context, as defined in Section 3.2 of [RFC8613].

As mentioned above, exceeding specific limits for the 'q' or 'v' value can weaken the security properties of the AEAD algorithm used, thus compromising secure communication requirements.

Therefore, in order to preserve the security of the used AEAD algorithm, OSCORE has to observe limits for the 'q' and 'v' values, throughout the lifetime of the used AEAD keys.

2.1. Limits for 'q' and 'v'

Recommendations for setting limits for the maximum 'q' and 'v' value are defined in [I-D.irtf-cfrg-aead-limits].

In particular, Figure 1 shows the limits given for AES-CCM-16-64-128, which is the mandatory to implement AEAD algorithm for OSCORE.

```
              q <= sqrt((p * 2^126) / l^2)

              v * 2^64 + (2l * (v + q))^2 <= p * 2^128
```

Figure 1: AES-CCM-16-64-128 limits

Considering the values p_q = 2^-60 and p_v = 2^-57 defined in [I-D.ietf-tls-dtls13], as well as l=1024, this gives the following values for the limits of 'q' and 'v'.

```
q <= sqrt(((2^-60) * 2^126) / 1024^2)

q <= 2^23

v * 2^64 + (2*1024 * (v + 2^23))^2 <= 2^-57 * 2^128

v <= 112
```

3. Additional Information in the Security Context

In addition to what defined in Section 3.1 of [RFC8613], the OSCORE Security Context MUST also include the following information.

The Sender Context is extended to include the following parameters.

- 'count_q': a non-negative integer counter, keeping track of the current 'q' value for the Sender Key. At any time, 'count_q' has as value the number of messages that have been encrypted using the

Sender Key. The value of 'count_q' is set to 0 when establishing the Sender Context.

- 'limit_q': a non-negative integer, which specifies the highest value that 'count_q' is allowed to reach, before stopping using the Sender Key to process outgoing messages.

  The value of 'limit_q' depends on the AEAD algorithm specified in the Common Context, considering the properties of that algorithm. The value of 'limit_q' is determined according to Section 3.

The Recipient Context is extended to include the following parameters.

- 'count_v': a non-negative integer counter, keeping track of the current 'v' value for the Recipient Key. At any time, 'count_v' has as value the number of failed decryptions occurred on incoming messages using the Recipient Key. The value of 'count_v' is set to 0 when establishing the Recipient Context.

- 'limit_v': a non-negative integer, which specifies the highest value that 'count_v' is allowed to reach, before stopping using the Recipient Key to process incoming messages.

  The value of 'limit_v' depends on the AEAD algorithm specified in the Common Context, considering the properties of that algorithm. The value of 'limit_v' is determined according to Section 3.

4. OSCORE Messages Processing

In order to keep track of the 'q' and 'v' values and ensure that AEAD keys are not used beyond reaching their limits, the processing of OSCORE messages is extended as defined in this section.

In particular, the processing of OSCORE messages follows the steps outlined in Section 8 of [RFC8613], with the additions defined below.

4.1. Protecting a Request or a Response

Before encrypting the COSE object using the Sender Key, the 'count_q' counter MUST be incremented.

If 'count_q' exceeds the 'limit_q' limit, the message processing MUST be aborted. From then on, the Sender Key MUST NOT be used to encrypt further messages.

4.2. Verifying a Request or a Response

If the decryption and verification of the COSE object using the Recipient Key fails, the 'count_v' counter MUST be incremented.

After 'count_v' has exceeded the 'limit_v' limit, incoming messages MUST NOT be decrypted and verified using the Recipient Key, and their processing MUST be aborted.

5. Methods for Rekeying OSCORE

Before the limit of 'q' or 'v' has been reached for an OSCORE Security Context, the two peers have to establish a new OSCORE Security Context, in order to continue using OSCORE for secure communication.

In practice, the two peers have to establish new Sender and Recipient Keys, as the keys actually used by the AEAD algorithm. When this happens, both peers reset their 'count_q' and 'count_v' values to 0 (see Section 3).

Currently, a number of ways exist to accomplish this.

- The two peers can run the procedure defined in Appendix B.2 of [RFC8613]. That is, the two peers exchange three or four messages, protected with temporary Security Contexts adding randomness to the ID Context.

  As a result, the two peers establish a new OSCORE Security Context with new ID Context, Sender Key and Recipient Key, while keeping the same OSCORE Master Secret and OSCORE Master Salt from the old OSCORE Security Context.

  This procedure does not require any additional components to what OSCORE already provides, and it does not provide perfect forward secrecy.

- The two peers can run the OSCORE profile [I-D.ietf-ace-oscore-profile] of the Authentication and Authorization for Constrained Environments (ACE) Framework [I-D.ietf-ace-oauth-authz].

  When a CoAP client uploads an Access Token to a CoAP server as an access credential, the two peers also exchange two nonces. Then, the two peers use the two nonces together with information provided by the ACE Authorization Server that issued the Access Token, in order to derive an OSCORE Security Context.

This procedure does not provide perfect forward secrecy.

- o The two peers can run the EDHOC key exchange protocol based on Diffie-Hellman and defined in [I-D.ietf-lake-edhoc], in order to establish a pseudo-random key in a mutually authenticated way.

  Then, the two peers can use the established pseudo-random key to derive external application keys. This allows the two peers to securely derive especially an OSCORE Master Secret and an OSCORE Master Salt, from which an OSCORE Security Context can be established.

  This procedure additionally provides perfect forward secrecy.

Manually updating the OSCORE Security Context at the two peers should be a last resort option, and it might often be not practical or feasible.

It is RECOMMENDED that the peer initiating the rekeying procedure starts it before reaching the 'q' or 'v' limits. Otherwise, the AEAD keys possibly to be used during the rekeying procedure itself may already be or become invalid before the rekeying is completed, which may prevent a successful establishment of the new OSCORE Security Context altogether.

6. Security Considerations

This document mainly covers security considerations about using AEAD keys in OSCORE and their usage limits, in addition to the security considerations of [RFC8613].

Depending on the specific rekeying procedure used to establish a new OSCORE Security Context, the related security considerations also apply.

TODO: Add more considerations.

7. IANA Considerations

This document has no actions for IANA.

Acknowledgments

The authors sincerely thank Christian Amsuess, John Mattsson and Goeran Selander for the initial discussions that allowed shaping this document.

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).

9. References

9.1. Normative References

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

[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>.

9.2. Informative References

[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-36 (work in progress), November 2020.

[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-15 (work in progress), January 2021.

[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.

[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-dtls13-40 (work in progress), January 2021.

[I-D.irtf-cfrg-aead-limits]
Guenther, F., Thomson, M., and C. Wood, "Usage Limits on AEAD Algorithms", draft-irtf-cfrg-aead-limits-01 (work in progress), September 2020.

Authors' Addresses

Rikard Hoeglund
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: rikard.hoglund@ri.se

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

Email: marco.tiloca@ri.se

CoRE M. Veillette, Ed.
Internet-Draft Trilliant Networks Inc.
Intended status: Standards Track P. van der Stok, Ed.
Expires: July 21, 2021 consultant
A. Pelov
Acklio
A. Bierman
YumaWorks
I. Petrov, Ed.
Acklio
January 17, 2021

CoAP Management Interface (CORECONF)
draft-ietf-core-comi-11

Abstract

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

Note

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

Status of This Memo

This Internet-Draft will expire on July 21, 2021.

Table of Contents

1. Introduction 3
1.1. Terminology 4
2. CORECONF Architecture 5
2.1. Major differences between RESTCONF and CORECONF 6
2.1.1. Differences due to CoAP and its efficient usage . . . 6
2.1.2. Differences due to the use of CBOR 7
2.2. Compression of YANG identifiers 7
2.3. Instance-identifier 8
2.4. Media-Types 8
2.5. Unified datastore 10
3. Example syntax 11
4. CoAP Interface 11
4.1. Using the 'k' query parameter 13
4.2. Data Retrieval 15
4.2.1. Using the 'c' query parameter 15
4.2.2. Using the 'd' query parameter 16
4.2.3. GET 16
4.2.4. FETCH 19
4.3. Data Editing 20
4.3.1. Data Ordering 20
4.3.2. POST 20
4.3.3. PUT 21
4.3.4. iPATCH 22
4.3.5. DELETE 23
4.4. Full datastore access 24
4.4.1. Full datastore examples 24
4.5. Event stream 25
4.5.1. Notify Examples 26
4.5.2. The 'f' query parameter 27

4.6. RPC statements 27
4.6.1. RPC Example 28
5. Use of Block-wise Transfers 30
6. Application Discovery 30
6.1. YANG library 30
6.2. Resource Discovery 31
6.2.1. Datastore Resource Discovery 31
6.2.2. Data node Resource Discovery 32
6.2.3. Event stream Resource Discovery 32
7. Error Handling 33
8. Security Considerations 36
9. IANA Considerations 37
9.1. Resource Type (rt=) Link Target Attribute Values Registry 37
9.2. CoAP Content-Formats Registry 37
9.3. Media Types Registry 38
9.4. YANG Namespace Registration 39
10. Acknowledgments 39
11. References 40
11.1. Normative References 40
11.2. Informative References 41
Appendix A. ietf-coreconf YANG module 42
Appendix B. ietf-coreconf .sid file 48
Authors' Addresses 51

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy, smart city, and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. Messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

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

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

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

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

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "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 following terms are defined in the YANG data modeling language [RFC7950]: action, anydata, anyxml, client, container, data model, data node, identity, instance identifier, leaf, leaf-list, list, module, RPC, schema node, server, submodule.

The following terms are defined in [RFC6241]: configuration data, datastore, state data

The following term is defined in [I-D.ietf-core-sid]: YANG schema item identifier (YANG SID, often shorten to simply SID).

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

The following terms are defined in this document:

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

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

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

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

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

single instance identifier: Handle used to identify a specific data node which can be instantiated only once. This includes data nodes defined at the root of a YANG module and data nodes defined within a container. This excludes data nodes defined within a list or any children of these data nodes.

instance-identifier: List instance identifier or single instance identifier.

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

2. CORECONF Architecture

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

```
+------------------------------------------------------------------+
|                 SMIv2 specification (optional) (2)               |
+------------------------------------------------------------------+
                               |
                               V
+------------------------------------------------------------------+
|                      YANG specification  (1)                     |
+------------------------------------------------------------------+
         |                                            |
Client   V                                  Server      V
+----------------+                          +------------------------+
|        Request |--> CoAP request(3) -->| Indication             |
|        Confirm |<-- CoAP response(3)<--| Response          (4)  |
|                |                          |                        |
|                |<==== Security (7) ===>|+----------------------+|
+----------------+                          || Datastore(s)    (5) ||
                                            |+----------------------+|
                                            |+----------------------+|
                                            || Event stream(s) (6) ||
                                            |+----------------------+|
                                            +------------------------+
```

Figure 1: Abstract CORECONF architecture

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

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

(2) SMIv2 specification: Optional part that consists of a named module which, specifies a set of variables and "conceptual tables". There is an algorithm to translate SMIv2 specifications to YANG specifications.

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

(4) Request, Indication, Response, Confirm: Processes performed by the CORECONF clients and servers.

(5) Datastore: A resource used to access configuration data, state data, RPCs, and actions. A CORECONF server may support a single unified datastore or multiple datastores as those defined by Network Management Datastore Architecture (NMDA) [RFC8342].

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

(7) Security: The server MUST prevent unauthorized users from reading or writing any CORECONF resources. CORECONF relies on security protocols such as DTLS [RFC6347] or OSCORE [RFC8613] to secure CoAP communications.

2.1. Major differences between RESTCONF and CORECONF

CORECONF is a RESTful protocol for small devices where saving bytes to transport a message is very important. Contrary to RESTCONF, many design decisions are motivated by the saving of bytes. Consequently, CORECONF is not a RESTCONF over CoAP protocol, but differs more significantly from RESTCONF.

2.1.1. Differences due to CoAP and its efficient usage

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

o CORECONF uses the methods FETCH and iPATCH to access multiple data nodes. RESTCONF uses instead the HTTP method PATCH and the HTTP method GET with the "fields" Query parameter.

o RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not supported by CoAP.

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

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

2.1.2. Differences due to the use of CBOR

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

- o CORECONF also differ in the handling of default values, only 'report-all' and 'trim' options are supported.

2.2. Compression of YANG identifiers

In the YANG specification, items are identified with a name string. In order to significantly reduce the size of identifiers used in CORECONF, numeric identifiers called YANG Schema Item iDentifier (YANG SID or simply SID) are used instead.

When used in a URI, SIDs are encoded using base64 encoding of the SID bytes. The base64 encoding is using the URL and Filename safe alphabet as defined by [RFC4648] section 5, without padding. The last 6 bits encoded is always aligned with the least significant 6 bits of the SID represented using an unsigned integer. 'A' characters (value 0) at the start of the resulting string are removed. See Figure 2 for complete illustration.

```
SID in base64 = URLsafeChar[SID >> 60 & 0x3F] |
                 URLsafeChar[SID >> 54 & 0x3F] |
                 URLsafeChar[SID >> 48 & 0x3F] |
                 URLsafeChar[SID >> 42 & 0x3F] |
                 URLsafeChar[SID >> 36 & 0x3F] |
                 URLsafeChar[SID >> 30 & 0x3F] |
                 URLsafeChar[SID >> 24 & 0x3F] |
                 URLsafeChar[SID >> 18 & 0x3F] |
                 URLsafeChar[SID >> 12 & 0x3F] |
                 URLsafeChar[SID >> 6 & 0x3F] |
                 URLsafeChar[SID & 0x3F]
```

Figure 2

For example, SID 1721 is encoded as follow.

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

The resulting base64 representation of SID 1721 is the two-character string "a5".

2.3. Instance-identifier

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

When part of a payload, instance-identifiers are encoded in CBOR based on the rules defined in [I-D.ietf-core-yang-cbor] section 6.13.1. When part of a URI, the SID is appended to the URI of the targeted datastore, the keys are specified using the 'k' query parameter as defined in Section 4.1.

2.4. Media-Types

CORECONF uses Media-Types based on the YANG to CBOR mapping specified in [I-D.ietf-core-yang-cbor].

The following Media-Type is used as defined in [I-D.ietf-core-sid].

- application/yang-data+cbor; id=sid

The following new Media-Types are defined in this document:

application/yang-identifiers+cbor: This Media-Type represents a CBOR YANG document containing a list of instance-identifier used to target specific data node instances within a datastore.

FORMAT: CBOR array of instance-identifier

The message payload of Media-Type 'application/yang-identifiers+cbor' is encoded using a CBOR array. Each entry of this CBOR array contain an instance-identifier encoded as defined in [I-D.ietf-core-yang-cbor] section 6.13.1.

application/yang-instances+cbor: This Media-Type represents a CBOR YANG document containing a list of data node instances. Each data node instance is identified by its associated instance-identifier.

FORMAT: CBOR array of CBOR map of instance-identifier, instance-value

The message payload of Media-Type 'application/yang-instances+cbor' is encoded using a CBOR array. Each entry within this CBOR array contains a CBOR map carrying an instance-identifier and associated instance-value. Instance-identifiers are encoded using the rules defined in [I-D.ietf-core-yang-cbor] section 6.13.1, instance-values are encoded using the rules defined in [I-D.ietf-core-yang-cbor] section 4.

When present in an iPATCH request payload, this Media-Type carry a list of data node instances to be replaced, created, or deleted. For each data node instance D, for which the instance-identifier is the same as a data node instance I, in the targeted datastore resource: the value of D replaces the value of I. When the value of D is null, the data node instance I is removed. When the targeted datastore resource does not contain a data node instance with the same instance-identifier as D, a new instance is created with the same instance-identifier and value as D.

The different Media-Type usages are summarized in the table below:

Method	Resource	Media-Type
GET response	data node	application/yang-data+cbor; id=sid
PUT request	data node	application/yang-data+cbor; id=sid
POST request	data node	application/yang-data+cbor; id=sid
DELETE	data node	n/a
GET response	datastore	application/yang-data+cbor; id=sid
PUT request	datastore	application/yang-data+cbor; id=sid
POST request	datastore	application/yang-data+cbor; id=sid
FETCH request	datastore	application/yang-identifiers+cbor
FETCH response	datastore	application/yang-instances+cbor
iPATCH request	datastore	application/yang-instances+cbor
GET response	event stream	application/yang-instances+cbor
POST request	rpc, action	application/yang-data+cbor; id=sid
POST response	rpc, action	application/yang-data+cbor; id=sid

2.5. Unified datastore

CORECONF supports a simple datastore model consisting of a single unified datastore. This datastore provides access to both configuration and operational data. Configuration updates performed on this datastore are reflected immediately or with a minimal delay as operational data.

Alternatively, CORECONF servers MAY implement a more complex datastore model such as the Network Management Datastore Architecture (NMDA) as defined by [RFC8342]. Each datastore supported is implemented as a datastore resource.

Characteristics of the unified datastore are summarized in the table below:

Name	Value
Name	unified
YANG modules	all modules
YANG nodes	all data nodes ("config true" and "config false")
Access	read-write
How applied	changes applied in place immediately or with a minimal delay
Protocols	CORECONF
Defined in	"ietf-coreconf"

3. Example syntax

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

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

4. CoAP Interface

This note specifies a Management Interface. CoAP endpoints that implement the CORECONF management protocol, support at least one discoverable management resource of resource type (rt): core.c.ds. The path of the discoverable management resource is left to implementers to select (see Section 6).

The mapping of YANG data node instances to CORECONF resources is as follows. Every data node of the YANG modules loaded in the CORECONF server represents a sub-resource of the datastore resource (e.g. /c/YANGSID). When multiple instances of a list exist, instance selection is possible as described in Section 4.1, Section 4.2.3.1, and Section 4.2.4.

CORECONF also supports event stream resources used to observe notification instances. Event stream resources can be discovered using resource type (rt): core.c.ev.

The description of the CORECONF management interface is shown in the table below:

CoAP resource	Example path	rt
Datastore resource	/c	core.c.ds
Data node resource	/c/YANGSID	core.c.dn
Default event steam resource	/s	core.c.ev

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

The methods used by CORECONF are:

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

There is at most one instance of the 'k' query parameter for YANG list element selection for the GET, PUT, POST, and DELETE methods. Having multiple instances of that query parameter shall be treated as an error.

Query parameter	Description
k	Select an instance within YANG list(s)

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

4.1. Using the 'k' query parameter

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

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

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

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

In this table:

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

o The boolean values false and true are represented as the single-character strings "0" and "1" respectively.

o The method urlSafeBase64() is used to convert a binary string to base64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5, without padding. For example, urlSafeBase64(0xF956A13C) return the six-character string "-VahPA".

o The method CBORencode() is used to convert a YANG value to CBOR as specified in [I-D.ietf-core-yang-cbor] section 6.

The resulting key strings are joined using commas between every two consecutive key values to produce the value of the ’k’ parameter.

4.2. Data Retrieval

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

There are two additional query parameters for the GET and FETCH methods.

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

4.2.1. Using the 'c' query parameter

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

The allowed values are:

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

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

If this query parameter is not present, the default value is "a" (the quotes are added for readability, but they are not part of the payload).

4.2.2. Using the 'd' query parameter

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

The allowed values are:

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

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

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

> The server MUST NOT return the child resource if d=t
>
> The server MUST return the child resource if d=a.

If this query parameter is not present, the default value is "t" (the quotes are added for readability, but they are not part of the payload).

4.2.3. GET

A request to read the value of a data node instance is sent with a CoAP GET message. The URI is set to the data node resource requested, the 'k' query parameter is added if any of the parents of the requested data node is a list node.

```
FORMAT:
  GET <data node resource> ['k' Uri-Query option]

  2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value
```

The returned payload contains the CBOR encoding of the requested instance-value.

4.2.3.1. GET Examples

Using, for example, the current-datetime leaf from module ietf-system [RFC7317], a request is sent to retrieve the value of 'system-state/clock/current-datetime'. The SID of 'system-state/clock/current-datetime' is 1723, encoded in base64 according to Section 2.2, yields a7. The response to the request returns the CBOR map with the key set to the SID of the requested data node (i.e. 1723) and the value encoded using a 'text string' as defined in [I-D.ietf-core-yang-cbor] section 6.4. The datastore resource path /c is an example location discovered with a request similar to Figure 4.

```
REQ: GET </c/a7>

RES: 2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
{
  1723 : "2014-10-26T12:16:31Z"
}
```

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

```
REQ: GET </c/a5>

RES: 2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
{
  1721 : {
    2 : "2014-10-26T12:16:51Z",    / current-datetime (SID 1723) /
    1 : "2014-10-21T03:00:00Z"     / boot-datetime (SID 1722) /
  }
}
```

Figure 3

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

```
REQ: GET </c/X9>

RES: 2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
{
  1533 : [
    {
      4 : "eth0",                  / name  (SID 1537) /
      1 : "Ethernet adaptor",      / description (SID 1534) /
      5 : 1880,                    / type, (SID 1538) identity /
                                   / ethernetCsmacd (SID 1880) /
      2 : true                     / enabled (SID 1535) /
    },
    {
      4 : "eth1",                  / name (SID 1537) /
      1 : "Ethernet adaptor",      / description (SID 1534) /
      5 : 1880,                    / type, (SID 1538) identity /
                                   / ethernetCsmacd (SID 1880) /
      2 : false                    / enabled (SID 1535) /
    }
  ]
}
```

To retrieve a specific instance within the /interfaces/interface YANG list, the CORECONF client adds the key of the targeted instance in its CoAP request using the ’k’ query parameter. The return payload containing the instance requested is encoded using a CBOR array as specified by [I-D.ietf-core-yang-cbor] section 4.4.1 containing the requested instance.

```
REQ: GET </c/X9?k=eth0>

RES: 2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
{
  1533 : [
    {
      4 : "eth0",                  / name  (SID 1537) /
      1 : "Ethernet adaptor",      / description (SID 1534) /
      5 : 1880,                    / type, (SID 1538) identity /
                                   / ethernetCsmacd (SID 1880) /
      2 : true                     / enabled (SID 1535) /
    }
  ]
}
```

It is equally possible to select a leaf of a specific instance of a list. The example below requests the description leaf (SID 1534, base64: X-) within the interface list corresponding to the interface name "eth0". The returned value is encoded in CBOR based on the rules specified by [I-D.ietf-core-yang-cbor] section 6.4.

```
REQ: GET </c/X-?k=eth0>

RES: 2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
{
  1534 : "Ethernet adaptor"
}
```

4.2.4. FETCH

The FETCH is used to retrieve multiple instance-values. The FETCH request payload contains the list of instance-identifier of the data node instances requested.

The return response payload contains a list of data node instance-values in the same order as requested. A CBOR null is returned for each data node requested by the client, not supported by the server or not currently instantiated.

For compactness, indexes of the list instance identifiers returned by the FETCH response SHOULD be elided, only the SID is provided. This approach may also help reducing implementations complexity since the format of each entry within the CBOR array of the FETCH response is identical to the format of the corresponding GET response.

```
FORMAT:
  FETCH <datastore resource>
        (Content-Format: application/yang-identifiers+cbor)
  CBOR array of instance-identifier

  2.05 Content (Content-Format: application/yang-instances+cbor)
  CBOR array of CBOR map of SID, instance-value
```

4.2.4.1. FETCH examples

This example uses the current-datetime leaf from module ietf-system [RFC7317] and the interface list from module ietf-interfaces [RFC8343]. In this example the value of current-datetime (SID 1723) and the interface list (SID 1533) instance identified with name="eth0" are queried.

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

RES: 2.05 Content (Content-Format: application/yang-instances+cbor)
[
  {
    1723 : "2014-10-26T12:16:31Z" / current-datetime (SID 1723) /
  },
  {
    1533 : {
       4 : "eth0",                / name (SID 1537) /
       1 : "Ethernet adaptor",    / description (SID 1534) /
       5 : 1880,                  / type (SID 1538), identity /
                                  / ethernetCsmacd (SID 1880) /
       2 : true,                  / enabled (SID 1535) /
      11 : 3                      / oper-status (SID 1544), value is testing /
    }
  }
]
```

4.3. Data Editing

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

4.3.1. Data Ordering

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

4.3.2. POST

The CoAP POST operation is used in CORECONF for the creation of data node resources and the invocation of "ACTION" and "RPC" resources. Refer to Section 4.6 for details on "ACTION" and "RPC" resources.

A request to create a data node instance is sent with a CoAP POST message. The URI specifies the data node resource of the instance to be created. In the case of a list instance, keys MUST be present in the payload.

```
FORMAT:
  POST <data node resource>
       (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

  2.01 Created
```

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

4.3.2.1. Post example

The example uses the interface list from module ietf-interfaces [RFC8343]. This example creates a new list instance within the interface list (SID = 1533), while assuming the datastore resource is hosted on the CoAP server with DNS name example.com and with path /ds. The path /ds is an example location that is assumed to have been discovered using request similar to Figure 4.

```
REQ: POST <coap://example.com/ds/X9>
     (Content-Format: application/yang-data+cbor; id=sid)
{
  1533 : [
    {
      4 : "eth5",               / name (SID 1537) /
      1 : "Ethernet adaptor",   / description (SID 1534) /
      5 : 1880,                 / type (SID 1538), identity /
                                / ethernetCsmacd (SID 1880) /
      2 : true                  / enabled (SID 1535) /
    }
  ]
}

RES: 2.01 Created
```

4.3.3. PUT

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

```
FORMAT:
  PUT <data node resource> ['k' Uri-Query option]
       (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

  2.01 Created
```

4.3.3.1. PUT example

The example uses the interface list from module ietf-interfaces [RFC8343]. This example updates the instance of the list interface (SID = 1533) with key name="eth0". The example location /c is an example location that is discovered using a request similar to Figure 4.

```
REQ: PUT </c/X9?k=eth0>
     (Content-Format: application/yang-data+cbor; id=sid)
{
  1533 : [
    {
      4 : "eth0",              / name (SID 1537) /
      1 : "Ethernet adaptor",  / description (SID 1534) /
      5 : 1880,                / type (SID 1538), identity /
                               / ethernetCsmacd (SID 1880) /
      2 : true                 / enabled (SID 1535) /
    }
  ]
}

RES:  2.04 Changed
```

4.3.4. iPATCH

One or multiple data node instances are replaced with the idempotent CoAP iPATCH method [RFC8132].

There are no query parameters for the iPATCH method.

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

```
FORMAT:
  iPATCH <datastore resource>
         (Content-Format: application/yang-instances+cbor)
  CBOR array of CBOR map of instance-identifier, instance-value

  2.04 Changed
```

4.3.4.1. iPATCH example

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

- o Set "/system/ntp/enabled" (SID 1755) to true.

- o Remove the server "tac.nrc.ca" from the "/system/ntp/server" (SID 1756) list.

- o Add/set the server "NTP Pool server 2" to the list "/system/ntp/server" (SID 1756).

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

RES: 2.04 Changed
```

4.3.5. DELETE

A data node resource is deleted with the DELETE method.

```
FORMAT:
  Delete <data node resource> ['k' Uri-Query option]

  2.02 Deleted
```

4.3.5.1. DELETE example

This example uses the interface list from module ietf-interfaces [RFC8343]. This example deletes an instance of the interface list (SID = 1533):

```
REQ:   DELETE </c/X9?k=eth0>

RES:   2.02 Deleted
```

4.4. Full datastore access

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

```
FORMAT:
  GET <datastore resource>

  2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

FORMAT:
  PUT <datastore resource>
       (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

  2.04 Changed

FORMAT:
  POST <datastore resource>
        (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

  2.01 Created

FORMAT:
  DELETE <datastore resource>

  2.02 Deleted
```

The content of the CBOR map represents the complete datastore of the server at the GET indication of after a successful processing of a PUT or POST request.

4.4.1. Full datastore examples

The example uses the interface list from module ietf-interfaces [RFC8343] and the clock container from module ietf-system [RFC7317]. We assume that the datastore contains two modules ietf-system (SID 1700) and ietf-interfaces (SID 1500); they contain the 'interface' list (SID 1533) with one instance and the 'clock' container (SID 1721). After invocation of GET, a CBOR map with data nodes from these two modules is returned:

```
REQ:  GET </c>

RES: 2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
{
  1721 : {                       / Clock (SID 1721) /
    2: "2016-10-26T12:16:31Z",   / current-datetime (SID 1723) /
    1: "2014-10-05T09:00:00Z"    / boot-datetime (SID 1722) /
  },
  1533 : [
    {                              / interface (SID 1533) /
       4 : "eth0",                 / name (SID 1537) /
       1 : "Ethernet adaptor",     / description (SID 1534) /
       5 : 1880,                   / type (SID 1538), identity: /
                                   / ethernetCsmacd (SID 1880) /
       2 : true,                   / enabled (SID 1535) /
      11 : 3                       / oper-status (SID 1544), value is testing /
    }
  ]
}
```

4.5. Event stream

Event notification is an essential function for the management of servers. CORECONF allows notifications specified in YANG [RFC5277] to be reported to a list of clients. The path for the default event stream can be discovered as described in Section 4. The server MAY support additional event stream resources to address different notification needs.

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

Each response payload carries one or multiple notifications. The number of notifications reported, and the conditions used to remove notifications from the reported list are left to implementers. When multiple notifications are reported, they MUST be ordered starting from the newest notification at index zero. Note that this could lead to notifications being sent multiple times, which increases the probability for the client to receive them, but it might potentially lead to messages that exceed the MTU of a single CoAP packet. If such cases could arise, implementers should make sure appropriate fragmentation is available - for example the one described in Section 5.

The format of notification without any content is a null value. The format of single notification is defined in [I-D.ietf-core-yang-cbor] section 4.2.1. For multiple notifications the format is an array where each element is a single notification as described in [I-D.ietf-core-yang-cbor] section 4.2.1.

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

  2.05 Content (Content-Format: application/yang-instances+cbor)
  CBOR array of CBOR map of instance-identifier, instance-value
```

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

An example implementation is:

> Every time an event is generated, the generated notification instance is appended to the chosen stream(s). After an aggregation period, which may be limited by the maximum number of notifications supported, the content of the instance is sent to all clients observing the modified stream.

4.5.1. Notify Examples

Let suppose the server generates the example-port-fault event as defined below.

```
module example-port {
  ...
  notification example-port-fault {   // SID 60010
    description
      "Event generated if a hardware fault is detected";
    leaf port-name {                  // SID 60011
      type string;
    }
    leaf port-fault {                 // SID 60012
      type string;
    }
  }
}
```

In this example the default event stream resource path /s is an example location discovered with a request similar to Figure 5. By executing a GET with Observe 0 on the default event stream resource the client receives the following response:

```
REQ:  GET </s> Observe(0)

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

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

4.5.2. The ’f’ query parameter

The ’f’ (filter) option is used to indicate which subset of all possible notifications is of interest. If not present, all notifications supported by the event stream are reported.

When not supported by a CORECONF server, this option shall be ignored, all events notifications are reported independently of the presence and content of the ’f’ (filter) option.

When present, this option contains a comma-separated list of notification SIDs. For example, the following request returns notifications 60010 and 60020.

```
REQ:  GET </s?f=60010,60020> Observe(0)
```

4.6. RPC statements

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

The request payload contains the values assigned to the input container when specified. The response payload contains the values of the output container when specified. Both the input and output containers are encoded in CBOR using the rules defined in [I-D.ietf-core-yang-cbor] section 4.2.1.

The returned success response code is 2.05 Content.

```
FORMAT:
  POST <data node resource> ['k' Uri-Query option]
       (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

  2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value
```

4.6.1. RPC Example

The example is based on the YANG action "reset" as defined in [RFC7950] section 7.15.3 and annotated below with SIDs.

```
   module example-server-farm {
     yang-version 1.1;
     namespace "urn:example:server-farm";
     prefix "sfarm";

     import ietf-yang-types {
       prefix "yang";
     }

     list server {                           // SID 60000
       key name;
       leaf name {                           // SID 60001
         type string;
       }
       action reset {                        // SID 60002
         input {
           leaf reset-at {                   // SID 60003
             type yang:date-and-time;
             mandatory true;
            }
          }
          output {
            leaf reset-finished-at {         // SID 60004
              type yang:date-and-time;
              mandatory true;
            }
          }
        }
      }
    }
```

This example invokes the 'reset' action (SID 60002, base64: Opq), of the server instance with name equal to "myserver".

```
 REQ:  POST </c/Opq?k=myserver>
       (Content-Format: application/yang-data+cbor; id=sid)
 {
   60002 : {
     1 : "2016-02-08T14:10:08Z09:00" / reset-at (SID 60003) /
   }
 }

 RES:  2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
 {
   60002 : {
     2 : "2016-02-08T14:10:08Z09:18" / reset-finished-at (SID 60004)/
   }
 }
```

5. Use of Block-wise Transfers

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

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

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

6. Application Discovery

Two application discovery mechanisms are supported by CORECONF, the YANG library data model as defined by [I-D.ietf-core-yang-library] and the CORE resource discovery [RFC6690]. Implementers may choose to implement one or the other or both.

6.1. YANG library

The YANG library data model [I-D.ietf-core-yang-library] provides a high-level description of the resources available. The YANG library contains the list of modules, features, and deviations supported by the CORECONF server. From this information, CORECONF clients can infer the list of data nodes supported and the interaction model to be used to access them. This module also contains the list of datastores implemented.

As described in [RFC6690], the location of the YANG library can be found by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.yl". Upon success, the return payload will contain the root resource of the YANG library module.

The following example assumes that the SID of the YANG library is 2351 (kv encoded as specified in Section 2.2) and that the server uses /c as datastore resource path.

```
REQ: GET </.well-known/core?rt=core.c.yl>

RES: 2.05 Content (Content-Format: application/link-format)
</c/kv>;rt="core.c.yl"
```

6.2. Resource Discovery

As some CoAP interfaces and services might not support the YANG library interface and still be interested to discover resources that are available, implementations MAY choose to support discovery of all available resources using "/.well-known/core" as defined by [RFC6690].

6.2.1. Datastore Resource Discovery

The presence and location of (path to) each datastore implemented by the CORECONF server can be discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.ds".

Upon success, the return payload contains the list of datastore resources.

Each datastore returned is further qualified using the "ds" Link-Format attribute. This attribute is set to the SID assigned to the datastore identity. When a unified datastore is implemented, the ds attribute is set to 1029 as specified in Appendix B. For other examples of datastores, see the Network Management Datastore Architecture (NMDA) [RFC7950].

```
link-extension    = ( "ds" "=" sid ) )
                      ; SID assigned to the datastore identity
sid               = 1*DIGIT
```

The following example assumes that the server uses /c as datastore resource path.

```
REQ: GET </.well-known/core?rt=core.c.ds>

RES: 2.05 Content (Content-Format: application/link-format)
</c>; rt="core.c.ds";ds=1029
```

Figure 4

6.2.2. Data node Resource Discovery

If implemented, the presence and location of (path to) each data node implemented by the CORECONF server are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.dn".

Upon success, the return payload contains the SID assigned to each data node and their location.

The example below shows the discovery of the presence and location of data nodes. Data nodes '/ietf-system:system-state/clock/boot-datetime' (SID 1722) and '/ietf-system:system-state/clock/current-datetime' (SID 1723) are returned. The example assumes that the server uses /c as datastore resource path.

```
REQ: GET </.well-known/core?rt=core.c.dn>

RES: 2.05 Content (Content-Format: application/link-format)
</c/a6>;rt="core.c.dn",
</c/a7>;rt="core.c.dn"
```

Without additional filtering, the list of data nodes may become prohibitively long. If this is the case implementations SHOULD support a way to obtain all links using multiple GET requests (for example through some form of pagination).

6.2.3. Event stream Resource Discovery

The presence and location of (path to) each event stream implemented by the CORECONF server are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.es".

Upon success, the return payload contains the list of event stream resources.

The following example assumes that the server uses /s as the default event stream resource.

```
REQ: GET </.well-known/core?rt=core.c.es>

RES: 2.05 Content (Content-Format: application/link-format)
</s>;rt="core.c.es"
```

Figure 5

7. Error Handling

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

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

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

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

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

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

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

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

- o Error 4.13 (Request Entity Too Large) may be returned by the CORECONF server during a block transfer request, see [RFC7959] for more details.

o Error 4.15 (Unsupported Content-Format) is returned by the CORECONF server when the Content-Format used in the request does not match those specified in section Section 2.4.

The CORECONF server MUST also enforce the different constraints associated with the YANG data models implemented. These constraints are described in [RFC7950] section 8. These errors are reported using the CoAP error code 4.00 (Bad Request) and may have the following error container as payload. The YANG definition and associated .sid file are available in Appendix A and Appendix B. The error container is encoded using the encoding rules of a YANG data template as defined in [I-D.ietf-core-yang-cbor] section 5.

```
+--rw error!
   +--rw error-tag             identityref
   +--rw error-app-tag?        identityref
   +--rw error-data-node?      instance-identifier
   +--rw error-message?        string
```

The following 'error-tag' and 'error-app-tag' are defined by the ietf-coreconf YANG module, these tags are implemented as YANG identity and can be extended as needed.

o error-tag 'operation-failed' is returned by the CORECONF server when the operation request cannot be processed successfully.

   * error-app-tag 'malformed-message' is returned by the CORECONF server when the payload received from the CORECONF client does not contain a well-formed CBOR content as defined in [RFC7049] section 3.3 or does not comply with the CBOR structure defined within this document.

   * error-app-tag 'data-not-unique' is returned by the CORECONF server when the validation of the 'unique' constraint of a list or leaf-list fails.

   * error-app-tag 'too-many-elements' is returned by the CORECONF server when the validation of the 'max-elements' constraint of a list or leaf-list fails.

   * error-app-tag 'too-few-elements' is returned by the CORECONF server when the validation of the 'min-elements' constraint of a list or leaf-list fails.

   * error-app-tag 'must-violation' is returned by the CORECONF server when the restrictions imposed by a 'must' statement are violated.

   * error-app-tag 'duplicate' is returned by the CORECONF server when a client tries to create a duplicate list or leaf-list entry.

- error-tag 'invalid-value' is returned by the CORECONF server when the CORECONF client tries to update or create a leaf with a value encoded using an invalid CBOR datatype or if the 'range', 'length', 'pattern' or 'require-instance' constrain is not fulfilled.

   * error-app-tag 'invalid-datatype' is returned by the CORECONF server when CBOR encoding does not follow the rules set by the YANG Build-In type or when the value is incompatible with it (e.g. a value greater than 127 for an int8, undefined enumeration).

   * error-app-tag 'not-in-range' is returned by the CORECONF server when the validation of the 'range' property fails.

   * error-app-tag 'invalid-length' is returned by the CORECONF server when the validation of the 'length' property fails.

   * error-app-tag 'pattern-test-failed' is returned by the CORECONF server when the validation of the 'pattern' property fails.

- error-tag 'missing-element' is returned by the CORECONF server when the operation requested by a CORECONF client fails to comply with the 'mandatory' constraint defined. The 'mandatory' constraint is enforced for leafs and choices, unless the node or any of its ancestors have a 'when' condition or 'if-feature' expression that evaluates to 'false'.

   * error-app-tag 'missing-key' is returned by the CORECONF server to further qualify a missing-element error. This error is returned when the CORECONF client tries to create or list instance, without all the 'key' specified or when the CORECONF client tries to delete a leaf listed as a 'key'.

   * error-app-tag 'missing-input-parameter' is returned by the CORECONF server when the input parameters of an RPC or action are incomplete.

- error-tag 'unknown-element' is returned by the CORECONF server when the CORECONF client tries to access a data node of a YANG module not supported, of a data node associated with an 'if-feature' expression evaluated to 'false' or to a 'when' condition evaluated to 'false'.

o error-tag 'bad-element' is returned by the CORECONF server when the CORECONF client tries to create data nodes for more than one case in a choice.

o error-tag 'data-missing' is returned by the CORECONF server when a data node required to accept the request is not present.

   * error-app-tag 'instance-required' is returned by the CORECONF server when a leaf of type 'instance-identifier' or 'leafref' marked with require-instance set to 'true' refers to an instance that does not exist.

   * error-app-tag 'missing-choice' is returned by the CORECONF server when no nodes exist in a mandatory choice.

o error-tag 'error' is returned by the CORECONF server when an unspecified error has occurred.

For example, the CORECONF server might return the following error.

```
RES:  4.00 Bad Request (Content-Format: application/yang-data+cbor; id=sid)
{
  1024 : {
    4 : 1011,        / error-tag (SID 1028) /
                     /   = invalid-value (SID 1011) /
    1 : 1018,        / error-app-tag (SID 1025) /
                     /   = not-in-range (SID 1018) /
    2 : 1740,        / error-data-node (SID 1026) /
                     /   = timezone-utc-offset (SID 1740) /
    3 : "maximum value exceeded" / error-message (SID 1027) /
  }
}
```

8. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. CORECONF re-uses the security mechanisms already available to CoAP, this includes DTLS [RFC6347] and OSCORE [RFC8613] for protected access to resources, as well as suitable authentication and authorization mechanisms, for example those defined in ACE OAuth [I-D.ietf-ace-oauth-authz].

All the security considerations of [RFC7252], [RFC7959], [RFC8132] and [RFC7641] apply to this document as well. The use of NoSec DTLS, when OSCORE is not used, is NOT RECOMMENDED.

In addition, mechanisms for authentication and authorization may need to be selected if not provided with the CoAP security mode.

As [I-D.ietf-core-yang-cbor] and [RFC4648] are used for payload and SID encoding, the security considerations of those documents also need to be well-understood.

9. IANA Considerations

9.1. Resource Type (rt=) Link Target Attribute Values Registry

This document adds the following resource type to the "Resource Type (rt=) Link Target Attribute Values", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Value	Description	Reference
core.c.ds	YANG datastore	RFC XXXX
core.c.dn	YANG data node	RFC XXXX
core.c.yl	YANG module library	RFC XXXX
core.c.es	YANG event stream	RFC XXXX

// RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

9.2. CoAP Content-Formats Registry

This document adds the following Content-Format to the "CoAP Content-Formats", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Media Type	Content Coding	ID	Reference
application/yang-identifiers+cbor		TBD2	RFC XXXX
application/yang-instances+cbor		TBD3	RFC XXXX

// RFC Ed.: replace TBD1, TBD2 and TBD3 with assigned IDs and remove this note. // RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

9.3. Media Types Registry

This document adds the following media types to the "Media Types" registry.

Name	Template	Reference
yang-identifiers+cbor	application/ yang-identifiers+cbor	RFC XXXX
yang-instances+cbor	application/ yang-instances+cbor	RFC XXXX

Each of these media types share the following information:

- Subtype name: <as listed in table>
- Required parameters: N/A
- Optional parameters: N/A
- Encoding considerations: binary
- Security considerations: See the Security Considerations section of RFC XXXX
- Interoperability considerations: N/A
- Published specification: RFC XXXX
- Applications that use this media type: CORECONF
- Fragment identifier considerations: N/A
- Additional information:

* Deprecated alias names for this type: N/A
* Magic number(s): N/A
* File extension(s): N/A
* Macintosh file type code(s): N/A

- o Person & email address to contact for further information: iesg&ietf.org

- o Intended usage: COMMON

- o Restrictions on usage: N/A

- o Author: Michel Veillette, ietf&augustcellars.com

- o Change Controller: IESG

- o Provisional registration? No

// RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

9.4. YANG Namespace Registration

This document registers the following XML namespace URN in the "IETF XML Registry", following the format defined in [RFC3688]:

URI: please assign urn:ietf:params:xml:ns:yang:ietf-coreconf

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

Reference: RFC XXXX

// RFC Ed.: please replace XXXX with RFC number and remove this note

10. Acknowledgments

We are very grateful to Bert Greevenbosch who was one of the original authors of the CORECONF specification.

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR.

The draft has benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Klaus Hartke, Michael van Hartskamp, Tanguy Ropitault, Juergen Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

11. References

11.1. Normative References

[I-D.ietf-core-sid]
Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item iDentifier (YANG SID)", draft-ietf-core-sid-14 (work in progress), July 2020.

[I-D.ietf-core-yang-cbor]
Veillette, M., Petrov, I., and A. Pelov, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-13 (work in progress), July 2020.

[I-D.ietf-core-yang-library]
Veillette, M. and I. Petrov, "Constrained YANG Module Library", draft-ietf-core-yang-library-03 (work in progress), January 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>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://www.rfc-editor.org/info/rfc5277>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <https://www.rfc-editor.org/info/rfc6243>.

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

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

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

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

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

[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>.

11.2. Informative References

[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-36 (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>.

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

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[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. ietf-coreconf YANG module

```
<CODE BEGINS> file "ietf-coreconf@2019-03-28.yang"
module ietf-coreconf {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-coreconf";
  prefix coreconf;

  import ietf-datastores {
    prefix ds;
  }

  import ietf-restconf {
    prefix rc;
    description
      "This import statement is required to access
       the yang-data extension defined in RFC 8040.";
    reference "RFC 8040: RESTCONF Protocol";
  }

  organization
    "IETF Core Working Group";

  contact
    "Michel Veillette
     <mailto:michel.veillette@trilliantinc.com>

     Alexander Pelov
     <mailto:alexander@ackl.io>
```

```
     Peter van der Stok
     <mailto:consultancy@vanderstok.org>

     Andy Bierman
     <mailto:andy@yumaworks.com>";

  description
    "This module contains the different definitions required
     by the CORECONF protocol.

     Copyright (c) 2019 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Simplified BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).

     This version of this YANG module is part of RFC XXXX;
     see the RFC itself for full legal notices.";

  revision 2019-03-28 {
     description
      "Initial revision.";
    reference
      "[I-D.ietf-core-comi] CoAP Management Interface";
  }

  identity unified {
    base ds:datastore;
    description
      "Identifier of the unified configuration and operational
       state datastore.";
  }

  identity error-tag {
    description
      "Base identity for error-tag.";
  }

  identity operation-failed {
    base error-tag;
    description
      "Returned by the CORECONF server when the operation request
       can't be processed successfully.";
  }
```

```
  identity invalid-value {
    base error-tag;
    description
      "Returned by the CORECONF server when the CORECONF client tries to
       update or create a leaf with a value encoded using an
       invalid CBOR datatype or if the 'range', 'length',
       'pattern' or 'require-instance' constrain is not
       fulfilled.";
  }

  identity missing-element {
    base error-tag;
    description
      "Returned by the CORECONF server when the operation requested
       by a CORECONF client fails to comply with the 'mandatory'
       constraint defined. The 'mandatory' constraint is
       enforced for leafs and choices, unless the node or any of
       its ancestors have a 'when' condition or 'if-feature'
       expression that evaluates to 'false'.";
  }

  identity unknown-element {
    base error-tag;
    description
      "Returned by the CORECONF server when the CORECONF client tries to
       access a data node of a YANG module not supported, of a
       data node associated with an 'if-feature' expression
       evaluated to 'false' or to a 'when' condition evaluated
       to 'false'.";
  }

  identity bad-element {
    base error-tag;
    description
      "Returned by the CORECONF server when the CORECONF client tries to
       create data nodes for more than one case in a choice.";
  }

  identity data-missing {
    base error-tag;
    description
      "Returned by the CORECONF server when a data node required to
       accept the request is not present.";
  }

  identity error {
    base error-tag;
    description
```

```
      "Returned by the CORECONF server when an unspecified error has
      occurred.";
  }

  identity error-app-tag {
    description
      "Base identity for error-app-tag.";
  }

  identity malformed-message {
    base error-app-tag;
    description
      "Returned by the CORECONF server when the payload received
       from the CORECONF client don't contain a well-formed CBOR
       content as defined in [RFC7049] section 3.3 or don't
       comply with the CBOR structure defined within this
       document.";
  }

  identity data-not-unique {
    base error-app-tag;
    description
      "Returned by the CORECONF server when the validation of the
       'unique' constraint of a list or leaf-list fails.";
  }

  identity too-many-elements {
    base error-app-tag;
    description
      "Returned by the CORECONF server when the validation of the
       'max-elements' constraint of a list or leaf-list fails.";
  }

  identity too-few-elements {
    base error-app-tag;
    description
      "Returned by the CORECONF server when the validation of the
       'min-elements' constraint of a list or leaf-list fails.";
  }

  identity must-violation {
    base error-app-tag;
    description
      "Returned by the CORECONF server when the restrictions
       imposed by a 'must' statement are violated.";
  }

  identity duplicate {
```

```
    base error-app-tag;
    description
      "Returned by the CORECONF server when a client tries to create
       a duplicate list or leaf-list entry.";
  }

  identity invalid-datatype {
    base error-app-tag;
    description
      "Returned by the CORECONF server when CBOR encoding is
       incorect or when the value encoded is incompatible with
       the YANG Built-In type. (e.g. value greater than 127
       for an int8, undefined enumeration).";
  }

  identity not-in-range {
    base error-app-tag;
    description
      "Returned by the CORECONF server when the validation of the
       'range' property fails.";
  }

  identity invalid-length {
    base error-app-tag;
    description
      "Returned by the CORECONF server when the validation of the
       'length' property fails.";
  }

  identity pattern-test-failed {
    base error-app-tag;
    description
      "Returned by the CORECONF server when the validation of the
       'pattern' property fails.";
  }

  identity missing-key {
    base error-app-tag;
    description
      "Returned by the CORECONF server to further qualify a
       missing-element error. This error is returned when the
       CORECONF client tries to create or list instance, without all
       the 'key' specified or when the CORECONF client tries to
       delete a leaf listed as a 'key'.";
  }

  identity missing-input-parameter {
    base error-app-tag;
```

```
    description
      "Returned by the CORECONF server when the input parameters
       of a RPC or action are incomplete.";
  }

  identity instance-required {
    base error-app-tag;
    description
      "Returned by the CORECONF server when a leaf of type
       'instance-identifier' or 'leafref' marked with
       require-instance set to 'true' refers to an instance
       that does not exist.";
  }

  identity missing-choice {
    base error-app-tag;
    description
      "Returned by the CORECONF server when no nodes exist in a
       mandatory choice.";
  }

  rc:yang-data coreconf-error {
    container error {
      description
        "Optional payload of a 4.00 Bad Request CoAP error.";

      leaf error-tag {
        type identityref {
          base error-tag;
        }
        mandatory true;
        description
          "The enumerated error-tag.";
      }

      leaf error-app-tag {
        type identityref {
          base error-app-tag;
        }
        description
          "The application-specific error-tag.";
      }

      leaf error-data-node {
        type instance-identifier;
        description
          "When the error reported is caused by a specific data node,
           this leaf identifies the data node in error.";
```

```
      }

      leaf error-message {
        type string;
        description
          "A message describing the error.";
      }
    }
  }
}
<CODE ENDS>
```

Appendix B. ietf-coreconf .sid file


```
   {
     "assignment-ranges": [
       {
         "entry-point": 1000,
         "size": 100
       }
     ],
     "module-name": "ietf-coreconf",
     "module-revision": "2019-03-28",
     "items": [
       {
         "namespace": "module",
         "identifier": "ietf-coreconf",
         "sid": 1000
       },
       {
         "namespace": "identity",
         "identifier": "bad-element",
         "sid": 1001
       },
       {
         "namespace": "identity",
         "identifier": "data-missing",
         "sid": 1002
       },
       {
         "namespace": "identity",
         "identifier": "data-not-unique",
         "sid": 1003
       },
       {
         "namespace": "identity",
         "identifier": "duplicate",
         "sid": 1004
```

```
        },
        {
          "namespace": "identity",
          "identifier": "error",
          "sid": 1005
        },
        {
          "namespace": "identity",
          "identifier": "error-app-tag",
          "sid": 1006
        },
        {
          "namespace": "identity",
          "identifier": "error-tag",
          "sid": 1007
        },
        {
          "namespace": "identity",
          "identifier": "instance-required",
          "sid": 1008
        },
        {
          "namespace": "identity",
          "identifier": "invalid-datatype",
          "sid": 1009
        },
        {
          "namespace": "identity",
          "identifier": "invalid-length",
          "sid": 1010
        },
        {
          "namespace": "identity",
          "identifier": "invalid-value",
          "sid": 1011
        },
        {
          "namespace": "identity",
          "identifier": "malformed-message",
          "sid": 1012
        },
        {
          "namespace": "identity",
          "identifier": "missing-choice",
          "sid": 1013
        },
        {
          "namespace": "identity",
```

```
          "identifier": "missing-element",
          "sid": 1014
        },
        {
          "namespace": "identity",
          "identifier": "missing-input-parameter",
          "sid": 1015
        },
        {
          "namespace": "identity",
          "identifier": "missing-key",
          "sid": 1016
        },
        {
          "namespace": "identity",
          "identifier": "must-violation",
          "sid": 1017
        },
        {
          "namespace": "identity",
          "identifier": "not-in-range",
          "sid": 1018
        },
        {
          "namespace": "identity",
          "identifier": "operation-failed",
          "sid": 1019
        },
        {
          "namespace": "identity",
          "identifier": "pattern-test-failed",
          "sid": 1020
        },
        {
          "namespace": "identity",
          "identifier": "too-few-elements",
          "sid": 1021
        },
        {
          "namespace": "identity",
          "identifier": "too-many-elements",
          "sid": 1022
        },
        {
          "namespace": "identity",
          "identifier": "unified",
          "sid": 1029
        },
```

```
        {
          "namespace": "identity",
          "identifier": "unknown-element",
          "sid": 1023
        },
        {
          "namespace": "data",
          "identifier": "/ietf-coreconf:error",
          "sid": 1024
        },
        {
          "namespace": "data",
          "identifier": "/ietf-coreconf:error/error-app-tag",
          "sid": 1025
        },
        {
          "namespace": "data",
          "identifier": "/ietf-coreconf:error/error-data-node",
          "sid": 1026
        },
        {
          "namespace": "data",
          "identifier": "/ietf-coreconf:error/error-message",
          "sid": 1027
        },
        {
          "namespace": "data",
          "identifier": "/ietf-coreconf:error/error-tag",
          "sid": 1028
        }
      ]
    }
```


Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Email: michel.veillette@trilliant.com

Peter van der Stok (editor)
consultant

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

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

Email: andy@yumaworks.com

Ivaylo Petrov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: ivaylo@ackl.io

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: August 26, 2021

M. Koster
SmartThings
B. Silverajan, Ed.
Tampere University

February 22, 2021

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-13

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 and Control 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

Copyright Notice

Table of Contents

1. Introduction 3
2. Terminology 3
3. Conditional Attributes 4
3.1. Conditional Notification Attributes 4
3.1.1. Greater Than (gt) 5
3.1.2. Less Than (lt) 5
3.1.3. Change Step (st) 6
3.1.4. Notification Band (band) 6
3.1.5. Edge (edge) 7
3.2. Conditional Control Attributes 7
3.2.1. Minimum Period (pmin) 8
3.2.2. Maximum Period (pmax) 8
3.2.3. Minimum Evaluation Period (epmin) 9
3.2.4. Maximum Evaluation Period (epmax) 9
3.2.5. Confirmable Notification (con) 9
3.3. Server processing of Conditional Attributes 9
4. Link Bindings 10
4.1. The "bind" attribute and Binding Methods 11
4.1.1. Polling 12
4.1.2. Observe 12
4.1.3. Push 13
4.1.4. Execute 13
4.2. Link Relation 13
5. Binding Table 14
6. Implementation Considerations 15
7. Security Considerations 16
8. IANA Considerations 16
8.1. Resource Type value 'core.bnd' 16
8.2. Link Relation Type 17
9. Acknowledgements 17
10. Contributors 17
11. Changelog 18
12. References 21
12.1. Normative References 21
12.2. Informative References 21
Appendix A. Examples 22
A.1. Minimum Period (pmin) example 22
A.2. Maximum Period (pmax) example 22

A.3. Greater Than (gt) example 24
A.4. Greater Than (gt) and Period Max (pmax) example 24
Authors' Addresses 25

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 and Control 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], [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.

Notification Band: A resource value range that results in state sychronization. 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 Attributes

This specification defines conditional attributes, which provide for fine-grained control of notification and state synchronization when using CoRE Observe [RFC7641] or Link Bindings (see Section 4). 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 attributes.

Note: In this draft, we assume that there are finite quantization effects in the internal or external updates to the value representing the state of a resource; specifically, that a resource state may be updated at any time with any valid value. We therefore avoid any continuous-time assumptions in the description of the conditional 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. Conditional Notification Attributes

Conditional Notification Attributes define the conditions that trigger a notification. 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.

The set of Conditional Notification Attributes defined here allow a client to control how often a client is interested in receiving notifications and how much a value should change for the new representation state to be interesting. One or more Conditional Notification Attributes MAY be included as query parameters in an Observe request.

Conditional Notification Attributes are defined below:

Attribute	Parameter	Value
Greater Than	gt	xs:decimal
Less Than	lt	xs:decimal
Change Step	st	xs:decimal (>0)
Notification Band	band	xs:boolean
Edge	edge	xs:boolean

Table 1: Conditional Notification Attributes

3.1.1. 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 fpr 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.

3.1.2. 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 samples value crosses the specified lower limit value, relative to the last reported value, and the time fpr 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.

3.1.3. Change Step (st)

When present, the change step indicates how much the value representing a resource state SHOULD change before triggering a notification, compared to the old state. Upon reception of a query including the st attribute, the current resource state representing the most recently sampled value is reported, and then set as the last reported value (last_rep_v). When a subsequent sampled value or update of the resource state differs from the last reported state 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 new resource state 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 resource states represented with a scalar numeric value.

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

3.1.4. 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, in monitoring the temperature of machinery, whilst the temperature is in the normal operating range, only periodic updates are needed. However as the temperature moves to more abnormal ranges more frequent state updates may be sent to clients.

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.

3.1.5. Edge (edge)

When present, the Edge attribute indicates interest for receiving notifications of either the falling edge or the rising edge transition of a boolean resource state. When the value of the Edge attribute is 0, the server notifies the client each time a resource state changes from True to False. When the value of the Edge attribute is 1, the server notifies the client each time a resource state changes from False to True.

The Edge attribute can only be supported on resources with a boolean value.

3.2. Conditional Control Attributes

Conditional Control Attributes define the time intervals between consecutive notifications as well as the cadence of the measurement of the conditions that trigger a notification. Conditional Control Attributes can be used to configure the internal server-driven sampling process for performing measurements of the conditions of a resource. One or more Conditional Control Attributes MAY be included as query parameters in an Observe request.

Conditional Control Attributes are defined below:

Attribute	Parameter	Value
Minimum Period (s)	pmin	xs:decimal (>0)
Maximum Period (s)	pmax	xs:decimal (>0)
Minimum Evaluation Period (s)	epmin	xs:decimal (>0)
Maximum Evaluation Period (s)	epmax	xs:decimal (>0)
Confirmable Notification	con	xs:boolean

Table 2: Conditional Control Attributes

3.2.1. Minimum Period (pmin)

When present, the minimum period indicates the minimum time, in seconds, between two consecutive notifications (whether or not the resource state 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 update the resource state with the last sampled value that occured 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.

3.2.2. Maximum Period (pmax)

When present, the maximum period indicates the maximum time, in seconds, between two consecutive notifications (whether or not the resource state 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, or equal to, the minimum period parameter (if present) otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.2.3. Minimum Evaluation Period (epmin)

When present, the minimum evaluation period indicates the minimum time, in seconds, the client recommends to the server to wait between two consecutive measurements of the conditions of a resource since the client has no interest in the server doing more frequent measurements. When the minimum evaluation period expires after the previous measurement, the server MAY immediately perform a new measurement. In the absence of this parameter, the minimum evaluation period is not defined and thus not used by the server. The server MAY use pmin, if defined, as a guidance on the desired measurement cadence. The minimum evaluation period MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.2.4. Maximum Evaluation Period (epmax)

When present, the maximum evaluation period indicates the maximum time, in seconds, the server MAY wait between two consecutive measurements of the conditions of a resource. When the maximum evaluation period expires after the previous measurement, the server MUST immediately perform a new measurement. In the absence of this parameter, the maximum evaluation period is not defined and thus not used by the server. The maximum evaluation period MUST be greater than zero and MUST be greater than the minimum evaluation period parameter (if present) otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.2.5. Confirmable Notification (con)

When present with a value of 1 in a query, the con attribute indicates a notification MUST be confirmable, i.e., the server MUST send the notification in a confirmable CoAP message, to request an acknowledgement from the client. When present with a value of 0 in a query, the con attribute indicates a notification can be confirmable or non-confirmable, i.e., it can be sent in a confirmable or a non-confirmable CoAP message.

3.3. Server processing of Conditional Attributes

Conditional Notification Attributes and Conditional Control Attributes 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 3: 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 4: 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 5: 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 rquest 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 pmax and pmin are equal, the expected behaviour is that notifications will be sent every (pmin == pmax) seconds. However, these notifications can only be fulfilled by the server on a best effort basis. Because pmin and pmax are designed as acceptable tolerance bounds for sending state updates, a query from an interested client containing equal pmin and pmax values must not be seen as a hard real-time scheduling contract between the client and the server.

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 or the configuration of epmin and epmax values for 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.

When a server has multiple observations with different measurement cadences as defined by the epmin and epmax values, the server MAY evaluate all observations when performing the measurement of any one observation.

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 Ocak 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. Alan Soloway contributed text leading to the inclusion of epmin and epmax. David Navarro proposed allowing for pmax to be equal to pmin.

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-13

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

draft-ietf-core-dynlink-12

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

draft-ietf-core-dynlink-11

- 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 conlict 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 the 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:

- This initial version is based on the text regarding the dynamic linking functionality in I.D.ietf-core-interfaces-05.

- The WADL description has been dropped in favour of a thorough textual description of the REST API.

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

```
           Observed   CLIENT  SERVER     Actual
       t   State         |      |         State
           ____________  |      |  ____________
       1                 |      |
       2    unknown      |      |     18.5 Cel
       3                 +----->|                  Header: GET
       4                 | GET  |                   Token: 0x4a
       5                 |      |                Uri-Path: temperature
       6                 |      |               Uri-Query: pmin="10"
       7                 |      |                 Observe: 0 (register)
       8                 |      |
       9   ____________  |<-----+                  Header: 2.05
      10                 | 2.05 |                   Token: 0x4a
      11    18.5 Cel     |      |                 Observe: 9
      12                 |      |                 Payload: "18.5 Cel"
      13                 |      |  ____________
      14                 |      |
      15                 |      |     23 Cel
      16                 |      |
      17                 |      |
      18                 |      |
      19                 |      |  ____________
      20   ____________  |<-----+                  Header: 2.05
      21                 | 2.05 |     26 Cel        Token: 0x4a
      22    26 Cel       |      |                 Observe: 20
      23                 |      |                 Payload: "26 Cel"
      24                 |      |
      25                 |      |
```

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

```
           Observed   CLIENT  SERVER     Actual
       t   State         |      |         State
           ____________  |      |  ____________
       1                 |      |
```

```
       2    unknown      |      |     18.5 Cel
       3                 +----->|                  Header: GET
       4                 | GET  |                   Token: 0x4a
       5                 |      |                Uri-Path: temperature
       6                 |      |               Uri-Query: pmax="20"
       7                 |      |                 Observe: 0 (register)
       8                 |      |
       9   ____________  |<-----+                  Header: 2.05
      10                 | 2.05 |                   Token: 0x4a
      11    18.5 Cel     |      |                 Observe: 9
      12                 |      |                 Payload: "18.5 Cel"
      13                 |      |
      14                 |      |
      15                 |      |  ____________
      16   ____________  |<-----+                  Header: 2.05
      17                 | 2.05 |     23 Cel        Token: 0x4a
      18    23 Cel       |      |                 Observe: 16
      19                 |      |                 Payload: "23 Cel"
      20                 |      |
      21                 |      |
      22                 |      |
      23                 |      |
      24                 |      |
      25                 |      |
      26                 |      |
      27                 |      |
      28                 |      |
      29                 |      |
      30                 |      |
      31                 |      |
      32                 |      |
      33                 |      |
      34                 |      |
      35                 |      |
      36                 |      |  ____________
      37   ____________  |<-----+                  Header: 2.05
      38                 | 2.05 |     23 Cel        Token: 0x4a
      39    23 Cel       |      |                 Observe: 37
      40                 |      |                 Payload: "23 Cel"
      41                 |      |
      42                 |      |
```

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

```
        Observed   CLIENT  SERVER     Actual
    t   State         |       |         State
        ____________  |       |  ____________
    1                 |       |
    2    unknown      |       |     18.5 Cel
    3                 +------>|                  Header: GET
    4                 | GET   |                   Token: 0x4a
    5                 |       |                Uri-Path: temperature
    6                 |       |               Uri-Query: gt=25
    7                 |       |                 Observe: 0 (register)
    8                 |       |
    9   ____________  |<------+                  Header: 2.05
   10                 | 2.05  |                   Token: 0x4a
   11    18.5 Cel     |       |                 Observe: 9
   12                 |       |                 Payload: "18.5 Cel"
   13                 |       |
   14                 |       |
   15                 |       |  ____________
   16   ____________  |<------+                  Header: 2.05
   17                 | 2.05  |     26 Cel        Token: 0x4a
   18    26 Cel       |       |                 Observe: 16
   29                 |       |                 Payload: "26 Cel"
   20                 |       |
   21                 |       |
```

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

```
        Observed   CLIENT  SERVER     Actual
    t   State         |       |         State
        ____________  |       |  ____________
    1                 |       |
    2    unknown      |       |     18.5 Cel
    3                 +------>|                  Header: GET
    4                 | GET   |                   Token: 0x4a
    5                 |       |                Uri-Path: temperature
    6                 |       |         Uri-Query: pmax=20;gt=25
    7                 |       |                 Observe: 0 (register)
    8                 |       |
    9   ____________  |<------+                  Header: 2.05
   10                 | 2.05  |                   Token: 0x4a
   11    18.5 Cel     |       |                 Observe: 9
   12                 |       |                 Payload: "18.5 Cel"
```

```
   13                 |      |
   14                 |      |
   15                 |      |
   16                 |      |
   17                 |      |
   18                 |      |
   19                 |      |
   20                 |      |
   21                 |      |
   22                 |      |
   23                 |      |
   24                 |      |
   25                 |      |
   26                 |      |
   27                 |      |
   28                 |      |
   29                 |      |  _____________
   30   _____________ |<-----+                  Header: 2.05
   31                 | 2.05 |     23 Cel        Token: 0x4a
   32    23 Cel       |      |                 Observe: 30
   33                 |      |                 Payload: "23 Cel"
   34                 |      |
   35                 |      |
   36                 |      |  _____________
   37   _____________ |<-----+                  Header: 2.05
   38                 | 2.05 |     26 Cel        Token: 0x4a
   39    26 Cel       |      |                 Observe: 37
   40                 |      |                 Payload: "26 Cel"
   41                 |      |
   42                 |      |
```

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@smartthings.com

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

Email: bilhanan.silverajan@tuni.fi

CoRE Working Group E. Dijk
Internet-Draft IoTconsultancy.nl
Obsoletes: 7390 (if approved) C. Wang
Updates: 7252, 7641 (if approved) InterDigital
Intended status: Standards Track M. Tiloca
Expires: August 26, 2021 RISE AB
February 22, 2021

Group Communication for the Constrained Application Protocol (CoAP)
draft-ietf-core-groupcomm-bis-03

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 devices or networks. This document replaces RFC7390, while it updates RFC7252 and RFC7641.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
 1.1. Scope 4
 1.2. Terminology 5
2. Group Definition and Group Configuration 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
3. CoAP Usage in Group Communication 13
 3.1. Request/Response Model 13
 3.2. Caching 16
  3.2.1. Freshness Model 17
  3.2.2. Validation Model 17
 3.3. Port and URI Path Selection 21
 3.4. Proxy Operation 22
  3.4.1. Forward-Proxies 22
  3.4.2. Reverse-Proxies 24
  3.4.3. Caching 25
 3.5. Congestion Control 30
 3.6. Observing Resources 31
 3.7. Block-Wise Transfer 33
 3.8. Transport 34
  3.8.1. UDP/IPv6 Multicast Transport 34
  3.8.2. UDP/IPv4 Multicast Transport 34
  3.8.3. 6LoWPAN 34
 3.9. Interworking with Other Protocols 35
  3.9.1. MLD/MLDv2/IGMP/IGMPv3 35
  3.9.2. RPL 36
  3.9.3. MPL 36
4. Unsecured Group Communication 37
5. Secured Group Communication using Group OSCORE 37
 5.1. Secure Group Maintenance 39

5.2. Caching of Responses at Proxies 40
5.2.1. Using Deterministic Requests to Achieve Cachability . 40
5.2.2. Validation of Responses 41
6. Security Considerations 41
6.1. CoAP NoSec Mode 41
6.2. Group OSCORE 42
6.2.1. Group Key Management 42
6.2.2. Source Authentication 43
6.2.3. Countering Attacks 43
6.3. Replay of Non-Confirmable Messages 45
6.4. Use of CoAP No-Response Option 45
6.5. 6LoWPAN 46
6.6. Wi-Fi 46
6.7. Monitoring 47
6.7.1. General Monitoring 47
6.7.2. Pervasive Monitoring 47
7. IANA Considerations 48
7.1. CoAP Option Numbers Registry 48
8. References 48
8.1. Normative References 48
8.2. Informative References 51
Appendix A. Use Cases 53
A.1. Discovery 53
A.1.1. Distributed Device Discovery 53
A.1.2. Distributed Service Discovery 54
A.1.3. Directory Discovery 54
A.2. Operational Phase 55
A.2.1. Actuator Group Control 55
A.2.2. Device Group Status Request 55
A.2.3. Network-wide Query 55
A.2.4. Network-wide / Group Notification 56
A.3. Software Update 56
Appendix B. Document Updates 56
B.1. Version -02 to -03 56
B.2. Version -01 to -02 57
B.3. Version -00 to -01 57
Acknowledgments 57
Authors' Addresses 58

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-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; cachability of responses to group requests at proxies; a response validation model for responses to group requests; and the use of Group OSCORE [I-D.ietf-core-oscore-groupcomm] to achieve security for CoAP group communication. Finally, this document also updates [RFC7641], by defining the multicast usage of the CoAP Observe Option 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. 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 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, for example as a name in the URI path. 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 nor at the CoAP 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. 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.

```
    +------------------------+                 +---------------------+
    |   Application group    |                 |     CoAP group      |
    |........................|                 |.....................|
    |                        |                 |                     |
    | [ - group name ]       +-----------------+ - IP mcast address  |
    | [ - group identifier ] |  1...N       1  | - UDP port          |
    |                        |                 |                     |
    |                        |                 |                     |
    +-------------+----------+                 +---------+----------+
                  |  1...N                               |  1...N
                  |                                      |
                  |                                      |
                  |                                      |  1...N
                  |                           +----------+------------+
                  |                           |   Security group      |
                  |                           |.......................|
                  |                           |                       |
                  \---------------------------+ - Security group name |
                                       1...N  | - Security material   |
                                              |                       |
                                              +-----------------------+
```

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 on which all the shown resources are hosted for each server.

```
  ________________________________     _________________________________
 /                                \   /                                 \
|       +---------------------+    | |  +---------------------+          |
|       | Application Group 1 |    | |  | Application Group 3 |  Cli2    |
|       |                     |    | |  |                     |          |
| Cli1  | Srv1   Srv2   Srv3  |    | |  | Srv5   Srv6         |  Cli4    |
|       | /resA  /resA  /resA |    | |  | /resC  /resC        |          |
| Cli2  +---------------------+    | |  | /resD  /resD        |          |
|                                  | |  +---------------------+          |
| Cli3     Security Group 1        | |                                   |
|                                  | |        Security Group 2           |
|       +---------------------+    |  \_________________________________/
|       | Application Group 2 |    |
|       |                     |    |
|       | Srv1   Srv4         |    |
|       | /resB  /resB        |    |
|       +---------------------+    |
 \________________________________/
```

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

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 (RD), 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 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

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 5), 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 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 5.1.

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 3.6) and block-wise transfer (see Section 3.7).

How CoAP group messages are carried over various transport layers is the subject of Section 3.8. Finally, Section 3.9 covers the interworking of CoAP group communication with other protocols that may operate in the same network.

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; 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 the CoAP group request - but the server MAY suppress the 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 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 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 payload or CoAP response options, such as an application-level unique ID associated to 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 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, although 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 for different multicast requests MUST be greater than:

```
MIN_TOKEN_REUSE_TIME = (NON_LIFETIME + MAX_LATENCY +
                        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 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 for the multicast case and considering a reuse time greater than MIN_TOKEN_REUSE_TIME. If an alternative security protocol for CoAP group communication is defined in the future which does not ensure secure binding between requests and responses, a client MUST follow the Token processing requirements 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, 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 over multicast 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 over multicast 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 part of the request URI with the transport-layer source address of the cached response message.

o 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, as discussed in Section 3.4.3.

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

3.2.1. Freshness Model

For caching at endpoints, the same freshness model relying on the Max-Age Option as defined in Section 5.6.1 of [RFC7252] applies.

For caching at proxies, the freshness model defined in Section 3.4.3 of this specification applies.

3.2.2. Validation Model

Section 5.6.2 of [RFC7252] defines a model to "validate" or "revalidate" responses stored in cache, hence enabling the suppression of responses that the client already has.

This relies on the ETag Option defined in Section 5.10.6 of [RFC7252], with its usage limited to exchanges between a CoAP client and one CoAP server. That is, Section 8.2.1 of [RFC7252] explicitly forbids using an ETag Option in requests sent over multicast, and leaves a mechanism to suppress responses for that case for further study.

This section provides such a model to "validate" or "revalidate" responses that the client already has cached. In particular, the group request can indicate entity-tag values separately for each CoAP server from which the client wishes to get a response revalidation, together with addressing information identifying that server.

To this end, this specification defines the new Multi-ETag Option. Operations related to this validation model and using the new option are defined in Section 3.2.2.2 for the client side, and in Section 3.2.2.3 for the server side.

The Multi-ETag Option has the properties summarized in Figure 3, which extends Table 4 of [RFC7252]. The Multi-ETag Option is elective, safe to forward, part of the cache key, and repeatable.

The option is intended only for group requests, as directly sent to a CoAP group or to a CoAP proxy that forwards it to the CoAP group (see Section 3.4).

No.	C	U	N	R	Name	Format	Length	Default
TBD1				x	Multi-ETag	(*)	any	(none)

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

(*) See below.

Figure 3: The Multi-ETag Option.

The Multi-ETag Option has the same properties of the ETag Option defined in Section 5.10.6 of [RFC7252], but it differs in the format and length, as well as having a different reason for its repeatability.

Each occurrence of the Multi-ETag Option targets exactly one of the servers in the CoAP group, from which the client wishes to get a response revalidation. The option value is set to a CBOR sequence [RFC8742] composed of (1+M) elements, where:

- The first element specifies the addressing information of the corresponding server, encoded as defined in Section 3.2.2.1.

  This mirrors the format of the Multicast-Signaling option defined in Section 3 of [I-D.tiloca-core-groupcomm-proxy]. Thus, in the presence of a forward proxy supporting the mechanism defined in [I-D.tiloca-core-groupcomm-proxy], the client can seamlessly use the server addressing information obtained from the proxy, when this forwards back a response to a group request from that server.

- The following M elements are CBOR byte strings, each of which has as value an entity-tag value that the client wants to try against the corresponding server.

  The entity-tag values included in the Multi-ETag Option are subject to the same considerations for the entity-tag values used in an ETag Option (see Section 5.10.6 of [RFC7252]).

The Multi-ETag Option is of class E in terms of OSCORE processing (see Section 4.1 of [RFC8613]).

3.2.2.1. Encoding of Server Addressing Information

The first element of the CBOR sequence in the Multi-ETag Option value is set to the byte serialization of the CBOR array 'tp_info' defined in Section 2.2.1 of [I-D.tiloca-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'.

When the Multi-ETag Option is used in group requests transported over UDP as in this specification, the 'tp_info' array includes the following elements, encoded as defined in Section 2.2.1.1 of [I-D.tiloca-core-observe-multicast-notifications].

- 'tp_id': the CBOR integer with value 1 ("UDP"), from the "Value" column of the "Transport Protocol Identifiers" Registry defined in Section 14.4 of [I-D.tiloca-core-observe-multicast-notifications]

- 'srv_host': 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)".

- 'srv_port': as a CBOR unsigned integer or the CBOR simple value Null. If it is a CBOR integer, it has as value the destination port number where to send individual requests intended to the server. This element MAY be present. If not included, the default port number 5683 is assumed.

The CDDL notation [RFC8610] provided below describes the 'tp_info' CBOR array using the format 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
]
```

3.2.2.2. Processing on the Client Side

Similar to what is defined in Section 5.6.2 of [RFC7252], the client may have one or more stored responses for a GET or FETCH group request sent to the CoAP group, but cannot use any of them (e.g. because they are not fresh).

In that case, the client can send a GET or FETCH group request, in order to give the origin servers an opportunity both to select a stored response to be used, and to update its freshness. As in [RFC7252], this process is known as "validating" or "revalidating" the stored response.

When sending such a group request, the endpoint SHOULD include one Multi-ETag Option for each server it wishes to revalidate the corresponding response with. As defined in Section 3.2.2, the Multi-ETag Option can include multiple entity-tag values, each applicable to a stored response from the corresponding server for that group request.

Specifically, in the same GET or FETCH group request:

- The client MUST NOT include one or more ETag Option(s) together with one or more Multi-ETag Option(s).

- The client MUST include only one Multi-ETag Option for each server it wishes to get a response revalidation from.

- The client SHOULD limit the number of Multi-ETag Options, hence limiting the number of servers as intended target of the revalidation process, and SHOULD rather spread revalidation with different sets of servers over different group requests. Also, the client SHOULD limit the number of entity-tag values specified in each Multi-ETag Option, preferably indicating only one entity-tag value.

  This allows for limiting the overall size of the group request. As a guideline, the server addressing information can be 9-24 bytes in size, while each entity-tag value can be 1-8 bytes in size. Thus, a single Multi-ETag Option can be up to (24 + 8 * M) bytes in size, where M is the number of entity-tag values it includes.

A 2.03 (Valid) response indicates that the stored response identified by the entity-tag given in the response's ETag Option can be reused, after updating the stored response as described in Section 5.9.1.3 of [RFC7252]. So the client can determine if any one of the stored representations from that server is current, without need to transfer the current resource representation again.

Any other Response Code indicates that none of the stored responses from that server, identified in the Multi-ETag Option of the group request, are suitable. Instead, such response SHOULD be used to satisfy the request and MAY replace the stored response.

3.2.2.3. Processing on the Server Side

If a GET or FETCH request includes both one or more ETag Options together with one or more Multi-ETag Options, then the server MUST ignore all the included ETag and Multi-ETag Options.

The server MUST ignore any Multi-ETag Option which is malformed, or included in a request that is neither GET nor FETCH, or which specifies addressing information not matching with its own endpoint address.

The server considers only its pertaining Multi-ETag Option, i.e. specifying addressing information associated to its own endpoint. The server MUST ignore any pertaining Multi-ETag Option that occurs more than once.

If the pertaining Multi-ETag Option specifies the CBOR simple value Null for the 'srv_port' element of 'tp_info' (see Section 3.2.2.1), the server MUST assume the default port number 5683.

Then, the server can issue a 2.03 (Valid) response in place of a 2.05 (Content) response, if one of the entity-tag values from the pertaining Multi-ETag Option is the entity-tag for the current resource representation, i.e. it is valid. The 2.03 (Valid) response echoes this specific entity-tag within an ETag Option included in the response.

The inclusion of an ETag Option in a response works as defined in Section 5.6.10.1 of [RFC7252].

3.3. 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 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 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. 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.

Port 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 5683 MUST be supported (see Section 7.1 of [RFC7252]) for the "All CoAP Nodes" multicast group as detailed in Section 3.8.

3.4. Proxy Operation

This section defines how proxies operate in a group communication scenario. In particular, Section 3.4.1 defines operations of forward-proxies, Section 3.4.2 defines operations of reverse-proxies, and Section 3.4.3 defines operations of proxies that employ a cache for responses to group requests.

3.4.1. Forward-Proxies

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.

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 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 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.

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:

o 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.

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 allow-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 identify the CoAP servers by other means such as application-specific information in the response payload.

3.4.2. 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 the request has to be forwarded to a group of servers over IP 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 forwards a request to a group of servers over IP multicast, the same considerations as defined in Section 5.7.3 of [RFC7252] hold, with the following additions:

- o The three issues and limitations defined in Section 3.4.1 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) 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 3.4.1 which applies also to this case.

3.4.3. Caching

A proxy that supports forwarding of group requests and that employs a cache maintains the following two types of cache entry.

- The first type, "individual" cache entry, is associated to one server and stores one response from that server, regardless whether it is a response to a unicast request or to a group request.

A hit to this entry would be produced by a matching request intended to that server, i.e. to the corresponding unicast URI.

When the response is a response to a unicast request to the server, the unicast URI is the same target URI used for the request.

When the response is a response to a group request to the CoAP group, the unicast URI is obtained by replacing the authority part of the group URI in the group request with the transport-layer source address and port number of the response message.

- o The second type, "aggregated" cache entry, is associated to the CoAP group, and stores all the responses that: the proxy has received as a response to a group request to that group; and that have been also forwarded back to the client that sent the group request.

  A hit to this entry would be produced by a matching group request intended to the CoAP group, i.e. to the corresponding group URI.

When forwarding a group request to a CoAP group using the request's group URI and processing the responses, the proxy handles its cache entries as follows. The same applies if the proxy spontaneously re-sends a group request to the CoAP group, in order to refresh an aggregated cache entry after its expiration or invalidation.

1. For each response to the group request which is received and also forwarded back to the client:

   * The proxy creates or refreshes the individual cache entry associated to the origin server and for that response. That is, the response is stored in the individual cache entry, and the lifetime of the cache entry is set to the lifetime of the response, as indicated by the Max-Age Option if present, or as the default value of 60 seconds otherwise (see Section 5.6.1 of [RFC7252]). This cache entry becomes immediately usable to serve requests from clients.

   * The proxy adds the response to a temporary list L.

2. After stopping to forward the received responses back to the client:

  * The proxy creates an aggregated cache entry associated to the group for that group request, if not existing yet. In case of an existing entry to be refreshed, the proxy deletes all the responses stored in the entry.

  * The proxy stores all the responses from the list L in the aggregated cache entry.

  * The proxy sets the lifetime of the cache entry to the smallest lifetime among all the responses stored in the entry, determined in the same way as defined in step 1 above.

  * The proxy sets the aggregated cache entry as usable to serve group requests from clients.

When forwarding a request to an individual server using the associated unicast URI and processing its response, the proxy handles its cache entries as follows. The same applies if the proxy spontaneously re-sends a unicast request to a single server, in order to refresh an individual cache entry after its expiration or invalidation.

1. The proxy creates or refreshes the individual cache entry associated to the origin server and for that response. That is, the response is stored in the cache entry, and the lifetime of the cache entry is set to the lifetime of the response, as indicated by the Max-Age Option if present, or as the default value of 60 seconds otherwise (see Section 5.6.1 of [RFC7252]). This cache entry becomes immediately usable to serve requests from clients.

2. The proxy checks whether it has a non-expired and valid aggregated cache entry, such that a hit would be produced by a group request analogous to the forwarded unicast request.

   That is, such group request would be intended to the group URI of the CoAP group associated to the aggregated cache entry, rather than intended to the unicast URI of the forwarded request.

3. If an aggregated cache entry is found at the previous step:

   * The proxy stores the received response in the aggregated cache entry, possibly replacing an already stored instance of that response from that origin server.

   * The proxy sets as new lifetime of the aggregated cache entry the minimum value between the current lifetime of the cache entry and the lifetime of the just-stored response, as

indicated by the Max-Age Option if present, or as the default value of 60 seconds otherwise (see Section 5.6.1 of [RFC7252]).

Note that a proxy embedded in a router can monitor network control messages, hence learning when a new server has joined a CoAP group and is listening to the multicast IP address of that CoAP group. This information could be used to guide the proxy in refreshing an aggregated cache entry, by sending a request to the CoAP group over the group URI before the entry expires, and thus storing also a response from the newly joined server.

Following the expiration or invalidation of a cache entry, as well as if wishing to refresh a cache entry, the proxy can directly interact with the servers in the CoAP group. To this end, it takes the role of a CoAP client as defined in Section 3.2. In particular, the proxy can perform revalidation of responses to group requests by using the Multi-ETag Option, as defined in Section 3.2.2.

As further discussed in Section 5.2, additional means are required to enable cachability of responses at the proxy when communications in the group are secured with Group OSCORE [I-D.ietf-core-oscore-groupcomm].

3.4.3.1. Validation of Responses to a Group Request

A client can revalidate the full set of responses to a group request from the corresponding aggregated cache entry at the proxy. To this end, this specification 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 Forward-Proxy, as well as for the associated responses retrieved from the corresponding aggregated cache entry at the proxy.

No.	C	U	N	R	Name	Format	Length	Default
TBD2				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]).

When providing 2.05 (Content) responses to a GET or FETCH group request from an aggregated cache entry, the proxy can include one Group-ETag Option, specifying the current entity-tag value associated to that cache entry. Each of such responses MUST NOT include more than one Group-ETag Option.

If the proxy supports this form of response revalidation, it MUST update the current entity-tag value associated to an aggregated cache entry, every time a response is added to that cache entry or replaces an already included response.

When sending a GET or FETCH group request to the proxy, to be forwarded to a CoAP group, the client can include one or more Group-ETag Option(s). Each option specifies one entity-tag value, as applicable to the aggregated cache entry for that group request.

In case the group request hits an aggregated cache entry and its current entity-tag value matches with one of the entity-tag value(s) specified in the Group-ETag option(s), then 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 aggregated cache entry.

That is, the 2.03 (Valid) response from the proxy indicates that the stored responses idenfied 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 set of representations from the aggregated cache entry at the proxy is current, without needing to transfer any of them again.

Note that, if a client triggers the proxy to perform forwarding of a group request (i.e., there is no hit of an aggregated cache entry), this will result in a new aggregated cache entry created at the proxy. Then, the client cannot obtain an entity-tag value through a Group-ETag Option in any of the responses forwarded back by the proxy.

In fact, the proxy will only have an assigned entity-tag value to provide after all responses have been forwarded back to that client, which is the moment that the new aggregated cache entry is eventually created. However, when follow-up group requests from the same client or different clients are served from this aggregated cache entry, the proxy can include a Group-ETag Option in each returned response, specifying the current entity-tag for the aggregated cache entry.

3.5. 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:

- 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]).

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

- A server in a constrained network SHOULD only support group communication 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 3.8.3) is used on the constrained network.

- 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].

- 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 3.7.

- 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.

3.6. 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. Consistent 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 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 MUST use a Message ID that differs from the initial multicast message.

Furthermore, consistent 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 (as a new entry) 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 occassionally 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 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 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 3.4 apply. The method defined in [I-D.tiloca-core-groupcomm-proxy] enables group communication including resource observation through proxies and addresses those limitations.

3.7. 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 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.

3.8. 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.

3.8.1. 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 3.9.2 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 3.8.3) 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].

3.8.2. 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].

3.8.3. 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 3.8.1) in order to discover only CoAP servers on the local 6LoWPAN network.

3.9. Interworking with Other Protocols

3.9.1. 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 3.9.2) or MPL (see Section 3.9.3) 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.

3.9.2. 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.

3.9.3. 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). Methods to configure which multicast groups are to be propagated into the MPL Domain could be:

- o Manual configuration on each ingress MPL Forwarder.

- o 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.

- o 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.

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. Before using this mode of operation, the security implications (Section 6.1) must be well understood.

5. 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 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 funtionalities. 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, 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 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 respective Group Managers by using the method described in [I-D.tiloca-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 on the group's UDP port.

- o All further CoAP endpoints configured only as CoAP clients, that send (multicast) CoAP requests to the CoAP group.

5.1. Secure Group Maintenance

As part of group maintenance operations (see Section 2.2.4), additional key management operations are required for an OSCORE group, depending on the security requirements of the application (see Section 6.2). Specifically:

- 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].

5.2. Caching of Responses at Proxies

When using Group OSCORE 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.

5.2.1. Using Deterministic Requests to Achieve Cachability

For application scenarios that require secure group communication, it is still possible to achieve cachability 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 3.4.3 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 Multi-ETag Options.

That is, even though the servers would produce the same plain CoAP responses in reply to the different Deterministic Requests, those will result in different protected responses to each respective

Deterministic Request, and hence in different cache entries at the proxy.

Thus, given a plain group request, a client needs to reuse the same set of Multi-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.

5.2.2. Validation of Responses

When directly interacting with the servers in the CoAP group to refresh its cache entries, the proxy cannot rely on response revalidation anymore. In fact, responses protected with Group OSCORE cannot have 2.03 (Valid) as Outer Code. Response revalidation remains possible end-to-end between the client and the servers in the group, by means of including inner ETag Option(s) or inner Multi-ETag Option(s).

Finally, it is not possible for a client to revalidate responses to a group request from an aggregated cache entry at the proxy, by using the outer Group-ETag Option as defined in Section 3.4.3.1. In fact, that would require the proxy to respond with an unprotected 2.03 (Valid) response potentially. However, success responses have to be protected with Group OSCORE, so cannot have 2.03 (Valid) as Outer Code.

6. Security Considerations

This section provides security considerations for CoAP group communication using 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.

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.

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 10 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, 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 time 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.

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.

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.

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 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 procedure as 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 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 (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).

6.3. 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 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 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.

6.4. 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 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.

6.5. 6LoWPAN

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 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 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. See Section 3.9 for protocols that allow multicast listeners to signal which groups they would like to listen to.

6.6. 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.

6.7. Monitoring

6.7.1. 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).

6.7.2. 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 compared to IP unicast. Also, CoAP traffic is typically used 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, 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 3.5 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 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.

Number	Name	Reference
TBD1	Multi-ETag	[This document]
TBD2	Group-ETag	[This document]

8. References

8.1. 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-12 (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.

[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-05 (work in progress), February 2021.

[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>.

[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>.

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.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-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-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-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-02 (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-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.tiloca-core-groupcomm-proxy]
Tiloca, M. and E. Dijk, "Proxy Operations for CoAP Group Communication", draft-tiloca-core-groupcomm-proxy-03 (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 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 -02 to -03

- o Multiple responses from same server handled at the application.
- o Clarifications about issues with forward-proxies.
- o Operations for reverse-proxies.
- o Caching of responses at proxies.
- o Client-Server response revalidation, with Multi-ETag Option.

- Client-Proxy response revalidation, with the Group-ETag Option.

B.2. 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.

B.3. Version -00 to -01

- Clarifications on group memberships for the different group types.
- Simplified description of Token reusage, 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 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 M. Boucadair
Internet-Draft Orange
Intended status: Standards Track J. Shallow
Expires: August 23, 2021 February 19, 2021

Constrained Application Protocol (CoAP) Block-Wise Transfer Options for Faster Transmission
draft-ietf-core-new-block-07

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

Copyright Notice

Table of Contents

1. Introduction 3
 1.1. Alternative CoAP Block-Wise Transfer Options 3
 1.2. CoAP Response Code (4.08) Usage 5
 1.3. Applicability Scope 5
2. Terminology 6
3. The Q-Block1 and Q-Block2 Options 6
 3.1. Properties of Q-Block1 and Q-Block2 Options 7
 3.2. Structure of Q-Block1 and Q-Block2 Options 9
 3.3. Using the Q-Block1 Option 10
 3.4. Using the Q-Block2 Option 13
 3.5. Using Observe Option 15
 3.6. Using Size1 and Size2 Options 15
 3.7. Using Q-Block1 and Q-Block2 Options Together 16
 3.8. Using Q-Block2 Option With Multicast 16
4. The Use of 4.08 (Request Entity Incomplete) Response Code . . 16
5. The Use of Tokens 17
6. Congestion Control for Unreliable Transports 18
 6.1. Confirmable (CON) 18
 6.2. Non-confirmable (NON) 18
7. Caching Considerations 22
8. HTTP-Mapping Considerations 23
9. Examples with Non-confirmable Messages 23
 9.1. Q-Block1 Option 23
  9.1.1. A Simple Example 24
  9.1.2. Handling MAX_PAYLOADS Limits 24
  9.1.3. Handling MAX_PAYLOADS with Recovery 24
  9.1.4. Handling Recovery with Failure 26
 9.2. Q-Block2 Option 26
  9.2.1. A Simple Example 27
  9.2.2. Handling MAX_PAYLOADS Limits 27
  9.2.3. Handling MAX_PAYLOADS with Recovery 28
  9.2.4. Handling Recovery using M-bit Set 29
 9.3. Q-Block1 and Q-Block2 Options 30
  9.3.1. A Simple Example 30
  9.3.2. Handling MAX_PAYLOADS Limits 31
  9.3.3. Handling Recovery 32
10. IANA Considerations 34
 10.1. New CoAP Options 34
 10.2. New Media Type 35
 10.3. New Content Format 36
 10.4. New CoAP Signaling Option Number 37

11. Security Considerations 37
12. Acknowledgements 38
13. References 38
13.1. Normative References 38
13.2. Informative References 39
Appendix A. Examples with Confirmable Messages 40
A.1. Q-Block1 Option 40
A.2. Q-Block2 Option 42
Appendix B. Examples with Reliable Transports 44
B.1. Q-Block1 Option 44
B.2. Q-Block2 Option 44
Authors' Addresses 45

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. [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 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 where each individual 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 (i.e., 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., [I-D.ietf-dots-telemetry]). As a reminder, [RFC7959] recommends the use of Confirmable (CON) responses to handle potential packet loss. However, such a recommendation does not work with a flooded pipe DDoS situation.

1.1. 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 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 (Section 6.2).

- o To reduce the transmission times for CON transmission of large bodies, NSTART needs to be increased from 1, but this affects congestion control where other parameters need to be tuned (Section 4.7 of [RFC7252]). Such tuning is out of scope of this document.

- 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.

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 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 packet. Even if there is asymmetrical packet loss, a body can still be sent and received by the peer whether the body comprises of 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 informs the CoAP sender endpoint either successful receipt 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 performance 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 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 Confirmable and NON as they are not used. Some examples using a reliable transport are provided in Appendix B.

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 option is not supported by a peer, then transmissions can fall back to using Block1 and Block2 Options.

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].

Q-Block1 and Q-Block2 Options are designed to work in particular with Non-confirmable requests and responses.

The No-Response Option was considered but was abandoned as it does not apply to Q-Block2 responses. A unified solution is defined in the document.

1.2. 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 4 for more details.

1.3. 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 of Non-Confirmable transmission. 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 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. However, using a NoSec security mode may still be inadequate for reasons discussed in Section 11.

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 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. 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, FETCH, PATCH, and iPATCH requests and their responses.

Q-Block2 Option is useful with GET, POST, PUT, FETCH, PATCH, and iPATCH requests and their payload-bearing responses (2.01, 2.02, 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.

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.

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].

Alternatively, with CoAP over reliable transports, Capabilities and Settings Messages (CSMs) can be used to indicate support of Q-Block Options by means of the Q-Block-Wise-Transfer Capability Option (Table 2). The behavior of CoAP peers is similar to the one specified in Section 5.3.2 of [RFC8323], except that it indicates support of Q-Block Options.

#	C	R	Applies to	Name	Format	Length	Base Value
TBA4			CSM	Q-Block-Wise-Transfer	empty	0	(none)

C=Critical, R=Repeatable

Table 2: The Q-Block-Wise-Transfer Capability Option

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.

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 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 both a class E and a class U in terms of OSCORE processing (Table 3). 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 3: 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. Q-Block and Block Options can be mixed across Inner and Outer options as these are handled independently of each other.

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, 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. It is also used to identify a particular payload of a body that needs to be retransmitted. The Request-Tag is opaque, the server still treats it as opaque but the client SHOULD 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 3.6 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 the successful receipt of the body by the server or some of the payloads have not arrived after a timeout and 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 payload of the body is treated as a new request (Section 5).

The client MUST send the payloads with the block numbers increasing, starting from zero, until the body is complete (subject to any congestion control (Section 6)). 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 the resource was created. The token used SHOULD be from the last received payload. The client should then release all of the tokens used for this body.

2.02 (Deleted)

This Response Code indicates successful receipt of the entire body and the resource was deleted when using POST (Section 5.8.2 [RFC7252]). The token used SHOULD be from the last received payload. 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 the resource was updated. The token used SHOULD be from the last received payload. 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 the appropriate representation of the resource is being returned. The token used in the response SHOULD be from the last received payload. If the FETCH request includes the Observe Option, then the server MUST use the same token 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 unless a resource is being observed.

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) in the response have been successfully received. The token used SHOULD be from the last received payload.

A response using this Response Code SHOULD NOT be generated for every received Q-Block1 Option request. It SHOULD only be generated when all the payload requests are Non-confirmable and MAX_PAYLOADS (Section 6.2) payloads have been received by the server.

It SHOULD NOT be generated for CON.

4.00 (Bad Request)

This Response Code MUST be returned if the request does not include both a Request-Tag Option and a Size1 Option but does include a Q-Block1 option.

4.02 (Bad Option)

Either this Response Code (in case of Confirmable request) or a reset message (in case of Non-confirmable request) MUST be returned if the server does not support the Q-Block Options.

4.08 (Request Entity Incomplete)

This Response Code returned without Content-Type "application/missing-blocks+cbor-seq" (Section 10.3) 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 missing payloads using the same Request-Tag, Size1 and Q-Block1 to specify the block NUM, SZX, and M bit as appropriate.

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 used in the response SHOULD be from the last received payload. See Section 4 for further information.

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.

If the server has not received all the payloads of a body, but one or more NON payloads have been received, it SHOULD wait for up to NON_RECEIVE_TIMEOUT (Section 6.2) before sending a 4.08 (Request Entity Incomplete) response. Further considerations related to the transmission timings of 4.08 (Request Entity Incomplete) and 2.31 (Continue) Response Codes are discussed in Section 6.2.

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, it SHOULD "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 SHOULD ignore the payload of the packet, but MUST still respond as if the block was received for the first time.

A server SHOULD only maintain a partial body (missing payloads) for up to NON_PARTIAL_TIMEOUT (Section 6.2).

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 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 set of MAX_PAYLOADS payloads (the latest one having block number NUM-1) has been successfully received and that, upon receipt of this request, the server can continue to send the next set of payloads (the first one 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, the client still treats it as opaque but the server SHOULD 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 3.6 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.

The client SHOULD wait for up to NON_RECEIVE_TIMEOUT (Section 6.2) after the last received payload for NON payloads before issuing a 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. It is permissible to set the M bit to request this and missing blocks from this MAX_PAYLOADS set. Further considerations related to the transmission timing for missing requests are discussed in Section 6.2.

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) to requests not adhering to this behavior.

For Confirmable responses, the client continues to acknowledge each packet. The server acknowledges the initial request using an ACK with the payload, and then sends the subsequent payloads as CON responses. The server will detect failure to send a packet, 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.

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 NON_PARTIAL_TIMEOUT (Section 6.2) or as defined by the Max-Age Option (or its default of 60 seconds (Section 5.6.1 of [RFC7252])), whichever is the less.

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.

It is RECOMMENDED that the server maintains 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.

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 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.

3.5. 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 includes payloads transmitted following receipt of the 'Continue' Q-Block2 Option (Section 3.4) by the server. If a missing payload is requested, then both the request and response MUST NOT include the Observe Option.

3.6. 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.

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.

3.7. 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.

3.8. 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].

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 is encoded as a CBOR Sequence [RFC8742]. It comprises of one or more CBOR encoded [RFC8949] missing block numbers. The missing block numbers MUST be unique in each 4.08 (Request Entity Incomplete) response when created by the server; the client SHOULD drop 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 10.3).

The Concise Data Definition Language [RFC8610] (and see Section 4.1 [RFC8742]) for the data describing these missing blocks is as follows:

```
; A notional 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

The token to use for the response SHOULD be 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 work, 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 to be 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 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.

5. 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: To minimize on the number of tokens that have to be tracked by clients, it is suggested that the bottom 32 bits is kept the same for the same body and the upper 32 bits contains 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].

6. Congestion Control for Unreliable Transports

The transmission of the payloads of a body over an unreliable transport SHOULD 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].

6.1. Confirmable (CON)

Congestion control for CON requests and responses is specified in Section 4.7 of [RFC7252]. For 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 out of scope of this document as it is expected that all requests and responses using Q-Block1 and Q-Block2 will be Non-confirmable.

It is implementation specific as to whether there should be any further requests for missing data as there will have been significant transmission failure as individual payloads will have failed after MAX_TRANSMIT_SPAN.

6.2. Non-confirmable (NON)

This document introduces new parameters MAX_PAYLOADS, NON_TIMEOUT, NON_RECEIVE_TIMEOUT, NON_MAX_RETRANSMIT, NON_PROBING_WAIT, and NON_PARTIAL_TIMEOUT primarily for use with NON (Table 4).

MAX_PAYLOADS should be configurable with a default value of 10. Both CoAP endpoints SHOULD 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].

NON_TIMEOUT is the maximum period of delay between sending sets of MAX_PAYLOADS payloads for the same body. By default, NON_TIMEOUT has the same value as ACK_TIMEOUT (Section 4.8 of [RFC7252]).

NON_RECEIVE_TIMEOUT is the initial maximum 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:

```
Time-to-Wait = NON_RECEIVE_TIMEOUT * (2 ** (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 by at least one second so that the sender of the payloads has the opportunity to start sending the next set of payloads 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 and remove all references to it. 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 calculated when using PROBING_WAIT. NON_PROBING_WAIT has the same value as computed for EXCHANGE_LIFETIME (Section 4.8.2 of [RFC7252]).

NON_PARTIAL_TIMEOUT is used for expiring partially received bodies. NON_PARTIAL_TIMEOUT has the same value as computed for EXCHANGE_LIFETIME (Section 4.8.2 of [RFC7252]).

Parameter Name	Default Value
MAX_PAYLOADS	10
NON_MAX_RETRANSMIT	4
NON_TIMEOUT	2 s
NON_RECEIVE_TIMEOUT	4 s
NON_PROBING_WAIT	247 s
NON_PARTIAL_TIMEOUT	247 s

Table 4: Congestion Control Parameters

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 single body of blocks 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 gap 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, and NON_TIMEOUT can be negotiated between DOTS peers as per [I-D.bosh-dots-quick-blocks].

Each NON 4.08 (Request Entity Incomplete) response is subject to PROBING_RATE.

Each NON GET or FETCH request using Q-Block2 Option is subject 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 NON_TIMEOUT before sending the next set of payloads to manage potential congestion issues.

If the CoAP peer reports at least one payload has not arrived for each body for at least a 24 hour period and it is known that there are no other network issues over that period, then the value of MAX_PAYLOADS can be reduced by 1 at a time (to a minimum of 1) and the situation re-evaluated for another 24 hour period until there is no report of missing payloads under normal operating conditions. The newly derived value for MAX_PAYLOADS should be used for both ends of this particular CoAP peer link. Note that the CoAP peer will not know about the MAX_PAYLOADS change until it is reconfigured. As a consequence of not being reconfigured, the peer may indicate that there are some missing payloads prior to the actual payload being transmitted as all of its MAX_PAYLOADS payloads have not arrived.

The sending of a set of missing payloads of a body is subject to MAX_PAYLOADS set of payloads.

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 set of payloads without any 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 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 payloads to prevent the client unnecessarily delaying. Otherwise 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 this is a repeat for the 2.31 (Continue) response, the server SHOULD send a 4.08 (Request Entity Incomplete) response detailing the missing payloads after the block number that would have been indicated in the 2.31 (Continue). If the repeat request count for a missing payload 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 set of MAX_PAYLOADS payloads before the server times out on waiting for the last of the previous MAX_PAYLOADS payloads. On receipt of the first received payload from the new set of MAX_PAYLOADS payloads, the server SHOULD send a 4.08 (Request Entity Incomplete) Response Code indicating any missing payloads from any previous MAX_PAYLOADS payloads. 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 payloads and then go into another NON_TIMEOUT delay prior to sending the next set of payloads.

For the client receiving NON Q-Block2 responses, it SHOULD send a 'Continue' Q-Block2 request (Section 3.4) for the next set of payloads on receipt of all of the MAX_PAYLOADS payloads 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 set of MAX_PAYLOADS payloads before the client times out on waiting for the last of the previous MAX_PAYLOADS payloads. Upon receipt of the first payload from the new set of MAX_PAYLOADS payloads, the client SHOULD send a request indicating any missing payloads from any previous set of MAX_PAYLOADS payloads. Upon receipt of such request, the server SHOULD send the missing payloads before continuing to send the remainder of the MAX_PAYLOADS payloads and then go into another NON_TIMEOUT delay prior to sending the next set of payloads.

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 after every transmission of MAX_PAYLOADS blocks will be observed. The endpoint receiving the body is still likely to receive the entire body.

7. 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-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 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.

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 (request)
   X<--: Message loss (response)
    ...: Passage of time
```

Figure 2: Notations Used in the Figures

9.1. Q-Block1 Option

9.1.1. 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.

```
        CoAP        CoAP
       Client      Server
         |           |
         +---------->| NON PUT /path M:0x81 T:0xc0 RT=9 QB1:0/1/1024
         +---------->| NON PUT /path M:0x82 T:0xc1 RT=9 QB1:1/1/1024
         +---------->| NON PUT /path M:0x83 T:0xc2 RT=9 QB1:2/1/1024
         +---------->| NON PUT /path M:0x84 T:0xc3 RT=9 QB1:3/0/1024
         |<----------+ NON 2.04 M:0xf1 T:0xc3
         |    ...    |
```

Figure 3: Example of NON Request with Q-Block1 Option (Without Loss)

9.1.2. 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.

```
        CoAP        CoAP
       Client      Server
         |           |
         +---------->| NON PUT /path M:0x01 T:0xf1 RT=10 QB1:0/1/1024
         +---------->| NON PUT /path M:0x02 T:0xf2 RT=10 QB1:1/1/1024
         +---------->| [[Payloads 3 - 9 not detailed]]
         +---------->| NON PUT /path M:0x0a T:0xfa RT=10 QB1:9/1/1024
      [[MAX_PAYLOADS has been reached]]
         |       [[MAX_PAYLOADS blocks receipt acknowledged by server]]
         |<----------+ NON 2.31 M:0x81 T:0xfa
         +---------->| NON PUT /path M:0x0b T:0xfb RT=10 QB1:10/0/1024
         |<----------+ NON 2.04 M:0x82 T:0xfb
         |    ...    |
```

Figure 4: Example of MAX_PAYLOADS NON Request with Q-Block1 Option (Without Loss)

9.1.3. 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.

```
        CoAP        CoAP
       Client      Server
          |          |
          +--------->| NON PUT /path M:0x11 T:0xe1 RT=11 QB1:0/1/1024
          +--->X     | NON PUT /path M:0x12 T:0xe2 RT=11 QB1:1/1/1024
          +--------->| [[Payloads 3 - 8 not detailed]]
          +--------->| NON PUT /path M:0x19 T:0xe9 RT=11 QB1:8/1/1024
          +--->X     | NON PUT /path M:0x1a T:0xea RT=11 QB1:9/1/1024
       [[MAX_PAYLOADS has been reached]]
          |   ...    |
       [[NON_TIMEOUT (client) delay expires]]
          |     [[Client starts sending next set of payloads]]
          +--->X     | NON PUT /path M:0x1b T:0xeb RT=11 QB1:10/1/1024
          +--------->| NON PUT /path M:0x1c T:0xec RT=11 QB1:11/1/1024
          |          |
```

Figure 5: Example of MAX_PAYLOADS NON Request with Q-Block1 Option (With Loss)

On seeing a payload from the next set of payloads, the server realizes that some blocks are missing from the previous MAX_PAYLOADS payloads and asks for the missing blocks in one go (Figure 6). It does so by indicating which blocks from the previous MAX_PAYLOADS payloads have not been received in the data portion of the response. The token used in the response should be the token that was used in the last block number 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 QB1:1/1/1024
         +--------->| NON PUT /path M:0x1e T:0xee RT=11 QB1:9/1/1024
         |     [[Client continues sending next set of payloads]]
         +--------->| NON PUT /path M:0x1f T:0xef RT=11 QB1:12/0/1024
         |   ...    |
      [[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 QB1:10/1/1024
         |<---------+ NON 2.04 M:0x93 T:0xf0
         |   ...    |
```

Figure 6: Example of NON Request with Q-Block1 Option (Blocks Recovery)

9.1.4. 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.

```
        CoAP        CoAP
       Client      Server
          |           |
          +---------->| NON PUT /path M:0x91 T:0xd0 RT=12 QB1:0/1/1024
          +--->X      | NON PUT /path M:0x92 T:0xd1 RT=12 QB1:1/1/1024
          +---------->| NON PUT /path M:0x93 T:0xd2 RT=12 QB1:2/0/1024
          |   ...     |
       [[NON_RECEIVE_TIMEOUT (server) delay expires]]
          |      [[The server realizes a block is missing and asks
          |         for the missing one.  Retry #1]]
          |<----------+ NON 4.08 M:0x01 T:0xd2 [Missing 1]
          |   ...     |
       [[2 * NON_RECEIVE_TIMEOUT (server) delay expires]]
          |      [[The server realizes a block is still missing and asks
          |         for the missing one.  Retry #2]]
          |<----------+ NON 4.08 M:0x02 T:0xd2 [Missing 1]
          |   ...     |
       [[4 * NON_RECEIVE_TIMEOUT (server) delay expires]]
          |      [[The server realizes a block is still missing and asks
          |         for the missing one.  Retry #3]]
          |<----------+ NON 4.08 M:0x03 T:0xd2 [Missing 1]
          |   ...     |
       [[8 * NON_RECEIVE_TIMEOUT (server) delay expires]]
          |      [[The server realizes a block is still missing and asks
          |         for the missing one.  Retry #4]]
          |<----------+ NON 4.08 M:0x04 T:0xd2 [Missing 1]
          |   ...     |
       [[16 * NON_RECEIVE_TIMEOUT (server) delay expires]]
          |      [[NON_MAX_RETRANSMIT exceeded. Server stops requesting
          |        for missing blocks and releases partial body]]
          |   ...     |
```

Figure 7: Example of NON Request with Q-Block1 Option (With Eventual Failure)

9.2. 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.

9.2.1. 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.

```
 CoAP        CoAP
Client      Server
  |          |
  +--------->| NON GET /path M:0x01 T:0xc0 O:0 QB2:0/1/1024
  |<---------+ NON 2.05 M:0xf1 T:0xc0 O:1220 ET=19 QB2:0/1/1024
  |<---------+ NON 2.05 M:0xf2 T:0xc0 O:1220 ET=19 QB2:1/1/1024
  |<---------+ NON 2.05 M:0xf3 T:0xc0 O:1220 ET=19 QB2:2/1/1024
  |<---------+ NON 2.05 M:0xf4 T:0xc0 O:1220 ET=19 QB2:3/0/1024
  |   ...    |
  |     [[Observe triggered]]
  |<---------+ NON 2.05 M:0xf5 T:0xc0 O:1221 ET=20 QB2:0/1/1024
  |<---------+ NON 2.05 M:0xf6 T:0xc0 O:1221 ET=20 QB2:1/1/1024
  |<---------+ NON 2.05 M:0xf7 T:0xc0 O:1221 ET=20 QB2:2/1/1024
  |<---------+ NON 2.05 M:0xf8 T:0xc0 O:1221 ET=20 QB2:3/0/1024
  |   ...    |
```

Figure 8: Example of NON Notifications with Q-Block2 Option (Without Loss)

9.2.2. 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 has been reached]]
         |      [[MAX_PAYLOADS blocks 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 has been reached]]
         |      [[MAX_PAYLOADS blocks 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)

9.2.3. 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.

```
         CoAP        CoAP
        Client      Server
          |    ...    |
          |      [[Observe triggered]]
          |<---------+ NON 2.05 M:0xa1 T:0xf0 O:1236 ET=23 QB2:0/1/1024
          |      X<---+ NON 2.05 M:0xa2 T:0xf0 O:1236 ET=23 QB2:1/1/1024
          |<---------+ [[Payloads 3 - 8 not detailed]]
          |      X<---+ NON 2.05 M:0xaa T:0xf0 O:1236 ET=23 QB2:9/1/1024
       [[MAX_PAYLOADS has been reached]]
          |    ...    |
       [[NON_TIMEOUT (server) delay expires]]
          |      [[Server sends next set of payloads]]
          |<---------+ NON 2.05 M:0xab T:0xf0 O:1236 ET=23 QB2:10/0/1024
          |    ...    |
       [[NON_RECEIVE_TIMEOUT (client) delay expires]]
          |      [[Client realizes blocks are missing and asks for the
          |        missing ones in one go]]
          +--------->| NON GET /path M:0x04 T:0xf3 QB2:1/0/1024\
          |          |                            QB2:9/0/1024
          |      X<---+ NON 2.05 M:0xac T:0xf3 ET=23 QB2:1/1/1024
          |<---------+ NON 2.05 M:0xad T:0xf3 ET=23 QB2:9/1/1024
          |    ...    |
       [[NON_RECEIVE_TIMEOUT (client) delay expires]]
          |      [[Client realizes block is still missing and asks for
          |        missing block]]
          +--------->| NON GET /path M:0x05 T:0xf4 QB2:1/0/1024
          |<---------+ NON 2.05 M:0xae T:0xf4 ET=23 QB2:1/1/1024
       [[Body has been received]]
          |    ...    |
```

Figure 10: Example of NON Notifications with Q-Block2 Option (Blocks Recovery)

9.2.4. 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
       [[MAX_PAYLOADS has been reached]]
          |    ...     |
       [[NON_TIMEOUT (server) delay expires]]
          |      [[Server sends next set of payloads]]
          |      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 has been reached]]
          |      [[MAX_PAYLOADS 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)

9.3. Q-Block1 and Q-Block2 Options

9.3.1. 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)

9.3.2. 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 has been reached]]
      |      [[MAX_PAYLOADS blocks receipt 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 has been reached]]
      |      [[MAX_PAYLOADS blocks 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 has been reached]]
      |      [[MAX_PAYLOADS blocks 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)

9.3.3. Handling Recovery

Consider now a scenario where there are some blocks are lost in transmission as illustrated in Figure 14.

```
      CoAP        CoAP
     Client      Server
       |           |
       +---------->| NON FETCH /path M:0x50 T:0xc0 O:0 RT=31 QB1:0/1/1024
       +--->X      | NON FETCH /path M:0x51 T:0xc1 O:0 RT=31 QB1:1/1/1024
       +--->X      | NON FETCH /path M:0x52 T:0xc2 O:0 RT=31 QB1:2/1/1024
       +---------->| NON FETCH /path M:0x53 T:0xc3 O:0 RT=31 QB1:3/0/1024
       |   ...     |
    [[NON_RECEIVE_TIMEOUT (server) delay expires]]
```

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 be the token that was used in the last block number 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:0xa0 T:0xc3 [Missing 1,2]
       |     [[Client responds with missing payloads]]
       +---------->| NON FETCH /path M:0x54 T:0xc4 O:0 RT=31 QB1:1/1/1024
       +---------->| NON FETCH /path M:0x55 T:0xc5 O:0 RT=31 QB1:2/1/1024
       |     [[Server received FETCH body,
       |       starts transmitting response body]]
       |<----------+ NON 2.05 M:0xa1 T:0xc3 O:1236 ET=23 QB2:0/1/1024
       |     X<----+ NON 2.05 M:0xa2 T:0xc3 O:1236 ET=23 QB2:1/1/1024
       |<----------+ NON 2.05 M:0xa3 T:0xc3 O:1236 ET=23 QB2:2/1/1024
       |     X<----+ NON 2.05 M:0xa4 T:0xc3 O:1236 ET=23 QB2:3/0/1024
       |   ...     |
    [[NON_RECEIVE_TIMEOUT (client) delay expires]]
       |           |
```

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.

```
         CoAP        CoAP
        Client      Server
          |           |
          +---------->| NON FETCH /path M:0x56 T:0xc6 RT=31 QB2:1/0/1024\
          |           |                                      QB2:3/0/1024
          |      [[Server receives FETCH request for missing payloads,
          |        starts transmitting missing blocks]]
          |      X<---+ NON 2.05 M:0xa5 T:0xc6 ET=23 QB2:1/1/1024
          |<----------+ NON 2.05 M:0xa6 T:0xc6 ET=23 QB2:3/0/1024
          |    ...    |
       [[NON_RECEIVE_TIMEOUT (client) delay expires]]
          |      [[Client realizes block is still missing and asks for
          |        missing block]]
          +---------->| NON FETCH /path M:0x57 T:0xc7 RT=31 QB2:1/0/1024
          |      [[Server receives FETCH request for missing payload,
          |        starts transmitting missing block]]
          |<----------+ NON 2.05 M:0xa7 T:0xc7 ET=23 QB2:1/1/1024
       [[Body has been received]]
          |    ...    |
          |      [[Observe triggered]]
          |<----------+ NON 2.05 M:0xa8 T:0xc3 O:1337 ET=24 QB2:0/1/1024
          |      X<---+ NON 2.05 M:0xa9 T:0xc3 O:1337 ET=24 QB2:1/1/1024
          |<----------+ NON 2.05 M:0xaa T:0xc3 O:1337 ET=24 QB2:2/0/1024
       [[NON_RECEIVE_TIMEOUT (client) delay expires]]
          |      [[Client realizes block is still missing and asks for
          |        missing block]]
          +---------->| NON FETCH /path M:0x58 T:0xc8 RT=31 QB2:1/0/1024
          |      [[Server receives FETCH request for missing payload,
          |        starts transmitting missing block]]
          |<----------+ NON 2.05 M:0xa7 T:0xc8 ET=23 QB2:1/1/1024
       [[Body has been received]]
          |    ...    |
```

Figure 16: Example of NON Request with Q-Block1 Option (Client Recovery)

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 5: CoAP Q-Block1 and Q-Block2 Option Numbers

This document suggests 19 (TBA1) and 51 (TBA2) as values to be assigned for the new option numbers.

10.2. New Media Type

This document requests IANA to register the "application/missing-blocks+cbor-seq" media type in the "Media Types" registry [IANA-MediaTypes]:

Type name: application

Subtype name: missing-blocks+cbor-seq

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations Section of [This_Document].

Interoperability considerations: N/A

Published specification: [This_Document]

Applications that use this media type: Data serialization and deserialization.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): N/A
Macintosh file type code(s): N/A

Person & email address to contact for further information: IETF, iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: none

Author: See Authors' Addresses section.

Change controller: IESG

Provisional registration? No

10.3. 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: TBA3
- o Reference: [RFCXXXX]

This document suggests 272 (TBA3) as a value to be assigned for the new content format number.

10.4. New CoAP Signaling Option Number

This document requests IANA to register the following option in the "CoAP Signaling Option Numbers" registry [CSM].

Applies to	Number	Name	Reference
7.01	TBA4	Q-Block-Wise-Transfer	[RFCXXXX]

Table 6: CoAP Signaling Option Codes

This document suggests 8 (TBA4) as a value to be assigned for the new option number.

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. 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 their comments.

Special thanks to Christian Amsuess, Carsten Bormann, and Marco Tiloca 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-11 (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>.

[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>.

[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

[CSM] "CoAP Signaling Option Numbers", <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#signaling-option-numbers>.

[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.K, T., Doron, E., chenmeiling, c., 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., Dukkipati, 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>.

[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

These 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 Q-Block1 Option with a CON request as shown in Figure 17. All the blocks are acknowledged (ACK).

```
         CoAP        CoAP
        Client      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
       [[NSTART(3) limit reached]]
          |<---------+ ACK 0.00 M:0x01
          +--------->| CON PUT /path M:0x04 T:0xf3 RT=10 QB1:3/0/1024
          |<---------+ ACK 0.00 M:0x02
          |<---------+ ACK 0.00 M:0x03
          |<---------+ ACK 2.04 M:0x04
          |           |
```

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.

```
         CoAP        CoAP
        Client      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
       [[NSTART(3) limit reached]]
          |<---------+ ACK 0.00 M:0x05
          +--------->| CON PUT /path M:0x08 T:0xf7 RT=11 QB1:3/1/1024
          |<---------+ ACK 0.00 M:0x08
          |    ...    |
       [[ACK_TIMEOUT (client) for M:0x06 delay expires]]
          |      [[Client retransmits packet]]
          +--------->| CON PUT /path M:0x06 T:0xf5 RT=11 QB1:1/1/1024
       [[ACK_TIMEOUT (client) for M:0x07 delay expires]]
          |      [[Client retransmits packet]]
          +--->X      | CON PUT /path M:0x07 T:0xf6 RT=11 QB1:2/1/1024
          |<---------+ ACK 0.00 M:0x06
          |    ...    |
       [[ACK_TIMEOUT exponential backoff (client) delay expires]]
          |      [[Client retransmits packet]]
          +--->?      | CON PUT /path M:0x07 T:0xf6 RT=11 QB1:2/1/1024
          |    ...    |
       [[Either body transmission failure (acknowledge retry timeout)
          or successfully transmitted.]]
```

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 network transient 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 network transient 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.

```
 CoAP        CoAP
Client      Server
  |           |
  +---------->| PUT /path T:0xf0 RT=10 QB1:0/1/1024
  +---------->| PUT /path T:0xf1 RT=10 QB1:1/1/1024
  +---------->| PUT /path T:0xf2 RT=10 QB1:2/1/1024
  +---------->| PUT /path T:0xf3 RT=10 QB1:3/0/1024
  |<----------+ 2.04
  |           |
```

Figure 20: Example of Reliable Request with Q-Block1 Option

If there is likely to be the possibility of network transient 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.

```
                Client      Server
                  |           |
                  +---------->| GET /path T:0xf0 O:0 QB2:0/1/1024
                  |<----------+ 2.05 T:0xf0 O:1234 ET=21 QB2:0/1/1024
                  |<----------+ 2.05 T:0xf0 O:1234 ET=21 QB2:1/1/1024
                  |<----------+ 2.05 T:0xf0 O:1234 ET=21 QB2:2/1/1024
                  |<----------+ 2.05 T:0xf0 O:1234 ET=21 QB2:3/0/1024
                  |    ...    |
                  |     [[Observe triggered]]
                  |<----------+ 2.05 T:0xf0 O:1235 ET=22 QB2:0/1/1024
                  |<----------+ 2.05 T:0xf0 O:1235 ET=22 QB2:1/1/1024
                  |<----------+ 2.05 T:0xf0 O:1235 ET=22 QB2:2/1/1024
                  |<----------+ 2.05 T:0xf0 O:1235 ET=22 QB2:3/0/1024
                  |    ...    |
```

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.

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 M. Boucadair
Internet-Draft Orange
Intended status: Standards Track J. Shallow
Expires: November 12, 2021 May 11, 2021

Constrained Application Protocol (CoAP) Block-Wise Transfer Options for Faster Transmission
draft-ietf-core-new-block-12

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 enabling faster transmission rates 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

Copyright Notice

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 10
  4.4. Using the Q-Block2 Option 14
  4.5. Using Observe Option 17
  4.6. Using Size1 and Size2 Options 17
  4.7. Using Q-Block1 and Q-Block2 Options Together 17
  4.8. Using Q-Block2 Option With Multicast 17
5. The Use of 4.08 (Request Entity Incomplete) Response Code . . 17
6. The Use of Tokens 19
7. Congestion Control for Unreliable Transports 19
  7.1. Confirmable (CON) 19
  7.2. Non-confirmable (NON) 20
8. Caching Considerations 24
9. HTTP-Mapping Considerations 25
10. Examples with Non-confirmable Messages 25
  10.1. Q-Block1 Option 26
    10.1.1. A Simple Example 26
    10.1.2. Handling MAX_PAYLOADS Limits 26
    10.1.3. Handling MAX_PAYLOADS with Recovery 26
    10.1.4. Handling Recovery with Failure 28
  10.2. Q-Block2 Option 29
    10.2.1. A Simple Example 29
    10.2.2. Handling MAX_PAYLOADS Limits 30
    10.2.3. Handling MAX_PAYLOADS with Recovery 31
    10.2.4. Handling Recovery using M-bit Set 32
  10.3. Q-Block1 and Q-Block2 Options 33
    10.3.1. A Simple Example 33
    10.3.2. Handling MAX_PAYLOADS Limits 34
    10.3.3. Handling Recovery 35
11. Security Considerations 37
12. IANA Considerations 38
  12.1. CoAP Option Numbers Registry 38

12.2. Media Type Registration 38
12.3. CoAP Content-Formats Registry 39
13. References 40
13.1. Normative References 40
13.2. Informative References 41
Appendix A. Examples with Confirmable Messages 42
A.1. Q-Block1 Option 42
A.2. Q-Block2 Option 44
Appendix B. Examples with Reliable Transports 46
B.1. Q-Block1 Option 46
B.2. Q-Block2 Option 46
Acknowledgements 47
Authors' Addresses 47

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 elect 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 at higher rates under network conditions where there may be asymmetrical transient packet loss (e.g., 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 for an acknowledgement (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, they 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 data size, the MAX_PAYLOADS_SET may not comprise all the MAX_PAYLOADS blocks.

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, the faster transmissions 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:

- 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 an intermediate response 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, 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 of Non-Confirmable transmission. Concretely, this mechanism primarily targets applications such as DDoS Open Threat Signaling (DOTS) that cannot use CON responses to handle 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]).

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.

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 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 (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 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.

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 initial 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 unless a resource is being observed.

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 block-wise exchange. However, it is desirable to provide the one used in the last received request.

A response using this Response Code SHOULD NOT be generated for every received Q-Block1 Option request (Section 7.2). 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.

4.00 (Bad Request)

This Response Code MUST be returned if the request does not include a Request-Tag Option or a Size1 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 must 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, Size1, 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.

For Confirmable responses, the client continues to acknowledge each packet. Typically, the server acknowledges the initial request using an ACK with the payload, and then sends the subsequent payloads as CON responses. The server will detect failure to send a packet, 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.

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.

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 data payload of the 4.08 (Request Entity Incomplete) response 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:

```
; A notional 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

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].

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_RECEIVE_TIMEOUT, NON_MAX_RETRANSMIT, NON_PROBING_WAIT, and NON_PARTIAL_TIMEOUT primarily for use with NON (Table 3).

MAX_PAYLOADS should be configurable with a default value of 10. Both CoAP endpoints SHOULD 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 the period of 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_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:

```
Time-to-Wait = NON_RECEIVE_TIMEOUT * (2 ** (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 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 has the same value as EXCHANGE_LIFETIME (Section 4.8.2 of [RFC7252]).

NON_PARTIAL_TIMEOUT is used for expiring partially received bodies. By default, NON_PARTIAL_TIMEOUT has the same value as EXCHANGE_LIFETIME (Section 4.8.2 of [RFC7252]).

Parameter Name	Default Value
MAX_PAYLOADS	10
NON_MAX_RETRANSMIT	4
NON_TIMEOUT	2 s
NON_RECEIVE_TIMEOUT	4 s
NON_PROBING_WAIT	247 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].

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, it is RECOMMENDED that after transmission of every

MAX_PAYLOADS_SET of a single body, a delay is introduced of NON_TIMEOUT before sending the next MAX_PAYLOADS_SET to manage potential congestion issues.

If the CoAP peer reports that at least one payload has not arrived for each body for at least a 24-hour period and it is known that there are no other network issues over that period (e.g., DDoS attacks), then the value of MAX_PAYLOADS can be reduced by 1 at a time (to a minimum of 1) and the situation re-evaluated for another 24-hour period until there is no report of missing payloads under normal operating conditions. Following a period of 24 hours where no packet recovery was required, the value of MAX_PAYLOADS can be increased by 1 (but without exceeding the default value) for a further 24-hour evaluation. The newly derived value for MAX_PAYLOADS should be used for both ends of this particular CoAP peer link. Note that the CoAP peer will not know about the MAX_PAYLOADS change until it is reconfigured. As a consequence of the two peers having different MAX_PAYLOADS values, a peer may continue indicating that there are some missing payloads as all of its MAX_PAYLOADS_SET may not have arrived. How the two peer values for MAX_PAYLOADS are synchronized is out of the scope.

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 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 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 repeat request count for 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 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 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 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.

```
          CoAP        CoAP
         Client      Server
           |           |
           +---------->| NON PUT /path M:0x81 T:0xc0 RT=9 QB1:0/1/1024
           +---------->| NON PUT /path M:0x82 T:0xc1 RT=9 QB1:1/1/1024
           +---------->| NON PUT /path M:0x83 T:0xc2 RT=9 QB1:2/1/1024
           +---------->| NON PUT /path M:0x84 T:0xc3 RT=9 QB1:3/0/1024
           |<----------+ NON 2.04 M:0xf1 T:0xc3
           |    ...    |
```

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.

```
          CoAP        CoAP
         Client      Server
           |           |
           +---------->| NON PUT /path M:0x01 T:0xf1 RT=10 QB1:0/1/1024
           +---------->| NON PUT /path M:0x02 T:0xf2 RT=10 QB1:1/1/1024
           +---------->| [[Payloads 3 - 9 not detailed]]
           +---------->| NON PUT /path M:0x0a T:0xfa RT=10 QB1:9/1/1024
        [[MAX_PAYLOADS_SET has been received]]
           |     [[MAX_PAYLOADS_SET receipt acknowledged by server]]
           |<----------+ NON 2.31 M:0x81 T:0xfa
           +---------->| NON PUT /path M:0x0b T:0xfb RT=10 QB1:10/0/1024
           |<----------+ NON 2.04 M:0x82 T:0xfb
           |    ...    |
```

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.

```
          CoAP        CoAP
         Client      Server
           |           |
           +---------->| NON PUT /path M:0x11 T:0xe1 RT=11 QB1:0/1/1024
           +--->X      | NON PUT /path M:0x12 T:0xe2 RT=11 QB1:1/1/1024
           +---------->| [[Payloads 3 - 8 not detailed]]
           +---------->| NON PUT /path M:0x19 T:0xe9 RT=11 QB1:8/1/1024
           +--->X      | NON PUT /path M:0x1a T:0xea RT=11 QB1:9/1/1024
        [[Some of MAX_PAYLOADS_SET have been received]]
           |   ...     |
        [[NON_TIMEOUT (client) delay expires]]
           |     [[Client starts sending next MAX_PAYLOAD_SET]]
           +--->X      | NON PUT /path M:0x1b T:0xeb RT=11 QB1:10/1/1024
           +---------->| NON PUT /path M:0x1c T:0xec RT=11 QB1:11/1/1024
           |           |
```

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 QB1:1/1/1024
           +--------->| NON PUT /path M:0x1e T:0xee RT=11 QB1:9/1/1024
           |     [[Client continues sending next MAX_PAYLOAD_SET]]
           +--------->| NON PUT /path M:0x1f T:0xef RT=11 QB1:12/0/1024
           |   ...    |
        [[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 QB1:10/1/1024
           |<---------+ 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.

```
          CoAP        CoAP
         Client      Server
           |           |
           +---------->| NON PUT /path M:0x91 T:0xd0 RT=12 QB1:0/1/1024
           +--->X      | NON PUT /path M:0x92 T:0xd1 RT=12 QB1:1/1/1024
           +---------->| NON PUT /path M:0x93 T:0xd2 RT=12 QB1:2/0/1024
           |   ...     |
        [[NON_RECEIVE_TIMEOUT (server) delay expires]]
           |      [[The server realizes a block is missing and asks
           |         for the missing one.  Retry #1]]
           |<----------+ NON 4.08 M:0x01 T:0xd2 [Missing 1]
           |   ...     |
        [[2 * NON_RECEIVE_TIMEOUT (server) delay expires]]
           |      [[The server realizes a block is still missing and asks
           |         for the missing one.  Retry #2]]
           |<----------+ NON 4.08 M:0x02 T:0xd2 [Missing 1]
           |   ...     |
        [[4 * NON_RECEIVE_TIMEOUT (server) delay expires]]
           |      [[The server realizes a block is still missing and asks
           |         for the missing one.  Retry #3]]
           |<----------+ NON 4.08 M:0x03 T:0xd2 [Missing 1]
           |   ...     |
        [[8 * NON_RECEIVE_TIMEOUT (server) delay expires]]
           |      [[The server realizes a block is still missing and asks
           |         for the missing one.  Retry #4]]
           |<----------+ NON 4.08 M:0x04 T:0xd2 [Missing 1]
           |   ...     |
        [[16 * NON_RECEIVE_TIMEOUT (server) delay expires]]
           |      [[NON_MAX_RETRANSMIT exceeded. Server stops requesting
           |        for missing blocks and releases partial body]]
           |   ...     |
```

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.

```
        CoAP        CoAP
       Client      Server
         |           |
         +---------->| NON GET /path M:0x01 T:0xc0 O:0 QB2:0/1/1024
         |<----------+ NON 2.05 M:0xf1 T:0xc0 O:1220 ET=19 QB2:0/1/1024
         |<----------+ NON 2.05 M:0xf2 T:0xc0 O:1220 ET=19 QB2:1/1/1024
         |<----------+ NON 2.05 M:0xf3 T:0xc0 O:1220 ET=19 QB2:2/1/1024
         |<----------+ NON 2.05 M:0xf4 T:0xc0 O:1220 ET=19 QB2:3/0/1024
         |    ...    |
         |     [[Observe triggered]]
         |<----------+ NON 2.05 M:0xf5 T:0xc0 O:1221 ET=20 QB2:0/1/1024
         |<----------+ NON 2.05 M:0xf6 T:0xc0 O:1221 ET=20 QB2:1/1/1024
         |<----------+ NON 2.05 M:0xf7 T:0xc0 O:1221 ET=20 QB2:2/1/1024
         |<----------+ NON 2.05 M:0xf8 T:0xc0 O:1221 ET=20 QB2:3/0/1024
         |    ...    |
```

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.

```
        CoAP        CoAP
       Client      Server
          |    ...    |
          |      [[Observe triggered]]
          |<---------+ NON 2.05 M:0xa1 T:0xf0 O:1236 ET=23 QB2:0/1/1024
          |      X<---+ NON 2.05 M:0xa2 T:0xf0 O:1236 ET=23 QB2:1/1/1024
          |<---------+ [[Payloads 3 - 9 not detailed]]
          |      X<---+ NON 2.05 M:0xaa T:0xf0 O:1236 ET=23 QB2:9/1/1024
       [[Some of MAX_PAYLOADS_SET have been received]]
          |    ...    |
       [[NON_TIMEOUT (server) delay expires]]
          |      [[Server sends next MAX_PAYLOAD_SET]]
          |<---------+ NON 2.05 M:0xab T:0xf0 O:1236 ET=23 QB2:10/0/1024
          |      [[On seeing a payload from the next MAX_PAYLOAD_SET,
          |       Client realizes blocks are missing and asks for the
          |        missing ones in one go]]
          +--------->| NON GET /path M:0x04 T:0xf3 QB2:1/0/1024\
          |          |                            QB2:9/0/1024
          |      X<---+ NON 2.05 M:0xac T:0xf3 ET=23 QB2:1/1/1024
          |<---------+ NON 2.05 M:0xad T:0xf3 ET=23 QB2:9/1/1024
          |    ...    |
       [[NON_RECEIVE_TIMEOUT (client) delay expires]]
          |      [[Client realizes block is still missing and asks for
          |        missing block]]
          +--------->| NON GET /path M:0x05 T:0xf4 QB2:1/0/1024
          |<---------+ NON 2.05 M:0xae T:0xf4 ET=23 QB2:1/1/1024
       [[Body has been received]]
          |    ...    |
```

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 (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.

```
   CoAP        CoAP
  Client      Server
    |           |
    +---------->| NON FETCH /path M:0x50 T:0xc0 O:0 RT=31 QB1:0/1/1024
    +--->X      | NON FETCH /path M:0x51 T:0xc1 O:0 RT=31 QB1:1/1/1024
    +--->X      | NON FETCH /path M:0x52 T:0xc2 O:0 RT=31 QB1:2/1/1024
    +---------->| NON FETCH /path M:0x53 T:0xc3 O:0 RT=31 QB1:3/0/1024
    |   ...     |
 [[NON_RECEIVE_TIMEOUT (server) delay expires]]
```

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.

```
   CoAP        CoAP
  Client      Server
    |           |
    |<----------+ NON 4.08 M:0xa0 T:0xc3 [Missing 1,2]
    |      [[Client responds with missing payloads]]
    +---------->| NON FETCH /path M:0x54 T:0xc4 O:0 RT=31 QB1:1/1/1024
    +---------->| NON FETCH /path M:0x55 T:0xc5 O:0 RT=31 QB1:2/1/1024
    |      [[Server received FETCH body,
    |        starts transmitting response body]]
    |<----------+ NON 2.05 M:0xa1 T:0xc3 O:1236 ET=23 QB2:0/1/1024
    |      X<---+ NON 2.05 M:0xa2 T:0xc3 O:1236 ET=23 QB2:1/1/1024
    |<----------+ NON 2.05 M:0xa3 T:0xc3 O:1236 ET=23 QB2:2/1/1024
    |      X<---+ NON 2.05 M:0xa4 T:0xc3 O:1236 ET=23 QB2:3/0/1024
    |   ...     |
 [[NON_RECEIVE_TIMEOUT (client) delay expires]]
    |           |
```

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.

```
        CoAP        CoAP
       Client      Server
         |           |
         +---------->| NON FETCH /path M:0x56 T:0xc6 RT=31 QB2:1/0/1024\
         |           |                                       QB2:3/0/1024
         |      [[Server receives FETCH request for missing payloads,
         |        starts transmitting missing blocks]]
         |      X<---+ NON 2.05 M:0xa5 T:0xc6 ET=23 QB2:1/1/1024
         |<----------+ NON 2.05 M:0xa6 T:0xc6 ET=23 QB2:3/0/1024
         |    ...    |
       [[NON_RECEIVE_TIMEOUT (client) delay expires]]
         |      [[Client realizes block is still missing and asks for
         |        missing block]]
         +---------->| NON FETCH /path M:0x57 T:0xc7 RT=31 QB2:1/0/1024
         |      [[Server receives FETCH request for missing payload,
         |        starts transmitting missing block]]
         |<----------+ NON 2.05 M:0xa7 T:0xc7 ET=23 QB2:1/1/1024
       [[Body has been received]]
         |    ...    |
         |      [[Observe triggered]]
         |<----------+ NON 2.05 M:0xa8 T:0xc3 O:1337 ET=24 QB2:0/1/1024
         |      X<---+ NON 2.05 M:0xa9 T:0xc3 O:1337 ET=24 QB2:1/1/1024
         |<----------+ NON 2.05 M:0xaa T:0xc3 O:1337 ET=24 QB2:2/0/1024
       [[NON_RECEIVE_TIMEOUT (client) delay expires]]
         |      [[Client realizes block is still missing and asks for
         |        missing block]]
         +---------->| NON FETCH /path M:0x58 T:0xc8 RT=31 QB2:1/0/1024
         |      [[Server receives FETCH request for missing payload,
         |        starts transmitting missing block]]
         |<----------+ NON 2.05 M:0xa7 T:0xc8 ET=24 QB2:1/1/1024
       [[Body has been received]]
         |    ...    |
```

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.

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>.

[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., Dukkipati, 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>.

[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>.

[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).

```
        CoAP        CoAP
       Client      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
      [[NSTART(3) limit reached]]
         |<---------+ ACK 0.00 M:0x01
         +--------->| CON PUT /path M:0x04 T:0xf3 RT=10 QB1:3/0/1024
         |<---------+ ACK 0.00 M:0x02
         |<---------+ ACK 0.00 M:0x03
         |<---------+ ACK 2.04 M:0x04
         |          |
```

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.

```
        CoAP        CoAP
       Client      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
      [[NSTART(3) limit reached]]
         |<---------+ ACK 0.00 M:0x05
         +--------->| CON PUT /path M:0x08 T:0xf7 RT=11 QB1:3/1/1024
         |<---------+ ACK 0.00 M:0x08
         |   ...    |
      [[ACK_TIMEOUT (client) for M:0x06 delay expires]]
         |      [[Client retransmits packet]]
         +--------->| CON PUT /path M:0x06 T:0xf5 RT=11 QB1:1/1/1024
      [[ACK_TIMEOUT (client) for M:0x07 delay expires]]
         |      [[Client retransmits packet]]
         +--->X     | CON PUT /path M:0x07 T:0xf6 RT=11 QB1:2/1/1024
         |<---------+ ACK 0.00 M:0x06
         |   ...    |
      [[ACK_TIMEOUT exponential backoff (client) delay expires]]
         |      [[Client retransmits packet]]
         +--->X     | CON PUT /path M:0x07 T:0xf6 RT=11 QB1:2/1/1024
         |   ...    |
      [[Either body transmission failure (acknowledge retry timeout)
         or successfully transmitted.]]
```

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.

```
        CoAP        CoAP
       Client      Server
         |           |
         +---------->| PUT /path T:0xf0 RT=10 QB1:0/1/1024
         +---------->| PUT /path T:0xf1 RT=10 QB1:1/1/1024
         +---------->| PUT /path T:0xf2 RT=10 QB1:2/1/1024
         +---------->| PUT /path T:0xf3 RT=10 QB1:3/0/1024
         |<----------+ 2.04
         |           |
```

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.

```
               Client      Server
                 |           |
                 +---------->| GET /path T:0xf0 O:0 QB2:0/1/1024
                 |<----------+ 2.05 T:0xf0 O:1234 ET=21 QB2:0/1/1024
                 |<----------+ 2.05 T:0xf0 O:1234 ET=21 QB2:1/1/1024
                 |<----------+ 2.05 T:0xf0 O:1234 ET=21 QB2:2/1/1024
                 |<----------+ 2.05 T:0xf0 O:1234 ET=21 QB2:3/0/1024
                 |    ...    |
                 |     [[Observe triggered]]
                 |<----------+ 2.05 T:0xf0 O:1235 ET=22 QB2:0/1/1024
                 |<----------+ 2.05 T:0xf0 O:1235 ET=22 QB2:1/1/1024
                 |<----------+ 2.05 T:0xf0 O:1235 ET=22 QB2:2/1/1024
                 |<----------+ 2.05 T:0xf0 O:1235 ET=22 QB2:3/0/1024
                 |    ...    |
```

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, and John Scudder 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: August 26, 2021
M. Tiloca
RISE AB
G. Selander
F. Palombini
J. Mattsson
Ericsson AB
J. Park
Universitaet Duisburg-Essen
February 22, 2021

Group OSCORE - Secure Group Communication for CoAP
draft-ietf-core-oscore-groupcomm-11

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

Copyright Notice

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 11
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 14
2.4. Update of Security Context 14
2.4.1. Loss of Mutable Security Context 15
2.4.2. Exhaustion of Sender Sequence Number 16
2.4.3. Retrieving New Security Context Parameters 17
3. The Group Manager 19
3.1. Management of Group Keying Material 20
3.2. Responsibilities of the Group Manager 21
4. The COSE Object 23
4.1. Counter Signature 23
4.2. The 'kid' and 'kid context' parameters 23
4.3. external_aad 23
5. OSCORE Header Compression 25
5.1. Examples of Compressed COSE Objects 26
5.1.1. Examples in Group Mode 26
5.1.2. Examples in Pairwise Mode 27
6. Message Binding, Sequence Numbers, Freshness and Replay Protection 28
6.1. Update of Replay Window 28
6.2. Message Freshness 29
7. Message Reception 29
8. Message Processing in Group Mode 30
8.1. Protecting the Request 31
8.1.1. Supporting Observe 31
8.2. Verifying the Request 32

8.2.1. Supporting Observe 34
8.3. Protecting the Response 34
8.3.1. Supporting Observe 35
8.4. Verifying the Response 35
8.4.1. Supporting Observe 36
9. Message Processing in Pairwise Mode 37
9.1. Pre-Conditions 38
9.2. Main Differences from OSCORE 38
9.3. Protecting the Request 39
9.4. Verifying the Request 39
9.5. Protecting the Response 39
9.6. Verifying the Response 40
10. Security Considerations 40
10.1. Group-level Security 41
10.2. Uniqueness of (key, nonce) 42
10.3. Management of Group Keying Material 42
10.4. Update of Security Context and Key Rotation 43
10.4.1. Late Update on the Sender 43
10.4.2. Late Update on the Recipient 44
10.5. Collision of Group Identifiers 44
10.6. Cross-group Message Injection 45
10.6.1. Attack Description 45
10.6.2. Attack Prevention in Group Mode 46
10.7. Group OSCORE for Unicast Requests 47
10.8. End-to-end Protection 48
10.9. Master Secret 48
10.10. Replay Protection 49
10.11. Message Freshness 49
10.12. Client Aliveness 50
10.13. Cryptographic Considerations 50
10.14. Message Segmentation 51
10.15. Privacy Considerations 51
11. IANA Considerations 52
11.1. OSCORE Flag Bits Registry 52
12. References 52
12.1. Normative References 52
12.2. Informative References 54
Appendix A. Assumptions and Security Objectives 56
A.1. Assumptions 57
A.2. Security Objectives 58
Appendix B. List of Use Cases 59
Appendix C. Example of Group Identifier Format 61
Appendix D. Set-up of New Endpoints 62
Appendix E. Challenge-Response Synchronization 63
Appendix F. No Verification of Signatures in Group Mode 66
Appendix G. Example Values with COSE Capabilities 67
Appendix H. Parameter Extensibility for Future COSE Algorithms . 68
H.1. Counter Signature Parameters 68

H.2. Secret Derivation Parameters 69
H.3. 'par_countersign' in the external_aad 69
Appendix I. Document Updates 71
I.1. Version -10 to -11 71
I.2. Version -09 to -10 72
I.3. Version -08 to -09 72
I.4. Version -07 to -08 73
I.5. Version -06 to -07 75
I.6. Version -05 to -06 75
I.7. Version -04 to -05 76
I.8. Version -03 to -04 76
I.9. Version -02 to -03 77
I.10. Version -01 to -02 78
I.11. Version -00 to -01 79
Acknowledgments 79
Authors' Addresses 80

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 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, 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 the 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:

- 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.

- 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 [RFC8949]; 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). That is, unless otherwise specified, the term group used in this specification 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 3.2.

- 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.

- Group request: CoAP request message sent by a client in the group to all servers in that 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

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):

- 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).

- 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.

- 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.

This format is consistent with every counter signature algorithm currently considered in [I-D.ietf-cose-rfc8152bis-algs], i.e. with algorithms that have only the COSE key type as their COSE capability. Appendix H describes how Counter Signature Parameters can be generalized for possible future registered algorithms having a different set of COSE capabilities.

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.

This format is consistent with every elliptic curve Diffie-Hellman algorithm currently considered in [I-D.ietf-cose-rfc8152bis-algs], i.e. with algorithms that have only the COSE key type as their COSE capability. Appendix H describes how Secret Derivation Parameters can be generalized for possible future registered algorithms having a different set of COSE capabilities.

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 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 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.

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. A newly installed Recipient Context that has required to delete another Recipient Context is initialized with an invalid Replay Window, and accordingly requires the endpoint to take appropriate actions (see Section 2.4.1.2).

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 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 Sender Key    = HKDF(Sender Key, Shared Secret, info, L)
Pairwise Recipient Key = HKDF(Recipient Key, Shared Secret, info, L)
```

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 HKDF algorithm specified by Secret Derivation Algorithm from the Common Context (see Section 2.1.4).

- The Sender Key and private key are from the Sender Context. The Sender Key is used as salt in the HKDF, when deriving the Pairwise Sender Key.

- The Recipient Key and the public key are from the Recipient Context associated to endpoint X. The Recipient Key is used as salt in the HKDF, when deriving the Pairwise Recipient Key.

- 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. The Shared Secret is used as Input Keying Material (IKM) in the HKDF.

- info and L are as defined 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 2.4 and Section 3.1), 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 and MUST accordingly take the actions defined in Section 2.4.1.

2.4.1. Loss of Mutable Security Context

An endpoint may lose its mutable Security Context, e.g. due to a reboot (see Section 2.4.1.1) or to an overflow of Recipient Contexts (see Section 2.4.1.2).

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.

2.4.1.1. 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 2.4.3) and use them to derive a new Sender Context (see Section 2.2). Since this includes a newly derived Sender Key, the 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.

2.4.1.2. 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 to the new Recipient Context.

If it is not configured as silent server, the endpoint MUST either:

- o Retrieve new Security Context parameters from the Group Manager and derive a new Sender Context, as defined in Section 2.4.1.1; or

- o When receiving a first request to process with the new Recipient Context, use the approach specified in Appendix E and based on the Echo Option for CoAP [I-D.ietf-core-echo-request-tag], 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.

2.4.2. 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).

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). The update of the Replay Window in each of the 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 under the current Gid value.

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 2.4.2). An endpoint may request to re-join the group, e.g. because of losing its mutable Security Context (see Section 2.4.1), and 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 3.1).

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.

- 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 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:

- 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 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-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 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. 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.

The Group Manager MUST NOT reassign a Gid value to the same group, and MUST NOT reassign a Sender ID within the same group under the same Gid value.

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 group 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 member of the group, with which it can generate the corresponding Recipient Context. In particular, the requested public key is provided together with the Sender ID of the associated group member. 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, every 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. That is, the Group Manager MUST NOT reassign a Sender ID value within a same group under the same Gid value (see Section 2.4.3.1). Within this restriction, the Group Manager can assign a Sender ID used under an old 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.

Figure 2 overviews the different keying material components, considering their relation and possible reuse across group rekeying.

```
 Components changed in lockstep            * Changing a kid does not
     upon a group rekeying                   need changing the Group ID
 +----------------------------+
 |                            |            * A kid is not reassigned
 |  Master               Group |<--> kid1     under the same Group ID
 |  Secret <---> o <--->  ID  |
 |              ^             |<--> kid2   * Upon changing the Group ID,
 |              |             |              every current kid should
 |              |             |<--> kid3     be preserved for efficient
 |              v             |              key rollover
 |          Master Salt       |  ... ...
 |          (optional)        |            * After changing Group ID, an
 |                            |              unused kid can be assigned
 +----------------------------+
```

Figure 2: Relations among keying material components.

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. 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.

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; and is not reassigned within the same group under the same Gid value, i.e. not 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. 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

When protecting a message in group mode, the 'unprotected' field MUST additionally include the following parameter:

- 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 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, 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, and is defined in this section.

The same external_aad structure is used in group mode and pairwise mode for encryption (see Section 5.3 of [I-D.ietf-cose-rfc8152bis-struct]), as well as in group mode for signing (see Section 4.4 of [I-D.ietf-cose-rfc8152bis-struct]).

In particular, the external_aad includes also the counter signature algorithm and related signature parameters, the value of the 'kid context' in the COSE object of the request, and the OSCORE option of the protected message.

```
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]],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr,
   request_kid_context : bstr,
   OSCORE_option: bstr
]
```

Figure 3: external_aad

Compared with Section 5.4 of [RFC8613], the aad_array has the following differences.

o The 'algorithms' 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.

This format is consistent with every counter signature algorithm currently considered in [I-D.ietf-cose-rfc8152bis-algs], i.e. with algorithms that have only the COSE key type as their COSE capability. Appendix H describes how 'par_countersign' can be generalized for possible future registered algorithms having a different set of COSE capabilities.

- 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 3.1). 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 10.6 when using the group mode, as further explained in Section 10.6.2.

  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 counter signature (when using the group mode). 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.

- 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 specification 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 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, the COSE_CounterSignature0 byte string as described in Section 4 is assumed to be 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: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 counter signature)

o Response with ciphertext = 0x60b035059d9ef5667c5a0710823b, kid = 0x52 and no Partial IV.

* Before compression (88 bytes):

```
[
h'',
{ 4:h'52', 11:h'ca1e ... b3' },
h'60b035059d9ef5667c5a0710823b'
]
```

* After compression (80 bytes):

  Flag byte: 0b00101000 = 0x28 (1 byte)

  Option Value: 0x28 52 (2 bytes)

  Payload: 0x60b035059d9ef5667c5a0710823b ca1e ... b3
  (14 bytes + size of the counter signature)

5.1.2. Examples in Pairwise Mode

o 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)

o 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. 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 2.4.1, 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 to 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 2.4.1).

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.

6.2. 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 2.4.1).

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 Appendix E based on the Echo Option for CoAP [I-D.ietf-core-echo-request-tag], 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 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:

- 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.

- 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.

  As per Section 6.1 of [RFC8613], this holds also when retrieving a Security Context which is valid but not associated to 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 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 predefined, 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 of this specification, 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 Gid as ID Context (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-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.

- 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 4.01 (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 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.

- 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 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. 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. Furthermore, the Replay Window MUST be updated only if both the signature verification and the decryption succeed.

- 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 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 Gid as ID Context (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.

- 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.

- 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 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).

- 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 exactly the one of 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 associated public key. 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 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.

- 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 2.4.3.1), as well as when it installs a new Security Context with a new ID Context (Gid) following a group rekeying (see Section 3.1).

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, 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 3.7 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.

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 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 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 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.

- 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 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.

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.

- o If Observe [RFC7641] is supported, what 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.

- 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 4.01 (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.

- 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 If Observe [RFC7641] is supported, what 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.

- o As discussed in Section 2.4.3.1, the server can obtain a new Sender ID from the Group Manager. 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 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.

- 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.

- If Observe [RFC7641] is supported, what defined in Section 8.4.1 of this document holds.

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 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 counter signature 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 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 under the same Gid value.

- 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 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 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 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 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, thanks to the counter signature also bound to the OSCORE option through the Additional Authenticated Data used in the signing process (see Section 4.3).

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 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 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).

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 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 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.

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. 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 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. 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.

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 9.3, 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 7.2.1 in [I-D.tiloca-core-observe-multicast-notifications].

With particular reference to block-wise transfers [RFC7959], Section 3.7 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, 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 [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 2.4.3, 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.

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 6.1, a Replay Window may be initialized as not valid, following the loss of mutable Security Context Section 2.4.1. In particular, Section 2.4.1.1 and Section 2.4.1.2 define measures that endpoints need to take in such a situation, before resuming to accept incoming messages from other group members.

10.11. Message Freshness

As discussed in Section 6.2, 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 Appendix E and based on the CoAP Echo Option [I-D.ietf-core-echo-request-tag], as a variant of the approach defined in Appendix B.1.2 of [RFC8613] applicable to Group OSCORE.

10.12. Client Aliveness

Building on 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, by using the approach described in Appendix E of this specification.

10.13. 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].

10.14. Message Segmentation

The same considerations from Section 12.7 of [RFC8613] hold for Group OSCORE.

10.15. 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, 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 mechanisms described in Appendix E 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 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 "[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	For using a Group OSCORE Security Context, set to 1 if the message is protected with the group mode	[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-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-cose-countersign]
Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", draft-ietf-cose-countersign-02 (work in progress), December 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-15 (work in progress), February 2021.

[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>.

[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

[Degabriele]
Degabriele, J., Lehmann, A., Paterson, K., Smart, N., and M. Strefler, "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-11 (work in progress), February 2021.

[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-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-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-12 (work in progress), January 2021.

[I-D.ietf-lwig-curve-representations]
Struik, R., "Alternative Elliptic Curve Representations", draft-ietf-lwig-curve-representations-20 (work in progress), February 2021.

[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-05 (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.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-05 (work in progress), February 2021.

[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:

- 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 specification. 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 to 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.

- 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.

- 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.

- 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 (BACSs) 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 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 never reassigns 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 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 (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-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

Appendix E. Challenge-Response Synchronization

This section describes a possible approach that a server endpoint can use to synchronize with Sender Sequence Numbers of client endpoints in the group. In particular, the 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 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 Echo option value SHOULD NOT be reused; when it is reused, it MUST be highly unlikely to have been used with this client recently. 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 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, 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 signature algorithm used in the group supports ECDH (e.g. ECDSA, EdDSA), the client MUST use the pairwise mode of Group OSCORE to protect the request, as described in Section 9.3. Note that, as defined in Section 9, 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 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 preconfigured 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.

- 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 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.

If the verifications above are successful and the Replay Window has not been set yet, the server updates its 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, it 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 [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 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 9.3), 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, 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 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 9.3 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 countersignature 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, 7] // 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-521
(-37) // PS256	[3], [3] // 3: RSA ; 3: RSA
(-38) // PS384	[3], [3] // 3: RSA ; 3: RSA
(-39) // PS512	[3], [3] // 3: RSA ; 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, 4] // 1: OKP ; 1: OKP, 4: X25519
(-27) // ECDH-SS // + HKDF-256	[1], [1, 5] // 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

Appendix H. Parameter Extensibility for Future COSE Algorithms

As defined in Section 8.1 of [I-D.ietf-cose-rfc8152bis-algs], future algorithms can be registered in the "COSE Algorithms" Registry [COSE.Algorithms] as specifying none or multiple COSE capabilities.

To enable the seamless use of such future registered algorithms, this section defines a general, agile format for parameters of the Security Context (see Section 2.1.3 and Section 2.1.5) and for related elements of the external_aad structure (see Section 4.3).

If any of the currently registered COSE algorithms is considered, using this general format yields the same structure defined in this document for the items above, thus ensuring retro-compatibility.

H.1. Counter Signature Parameters

The definition of Counter Signature Parameters in the Common Context (see Section 2.1.3) is generalized as follows.

Counter Signature Parameters is a CBOR array CS_PARAMS including N+1 elements, whose exact structure and value depend on the value of Counter Signature Algorithm.

- The first element, i.e. CS_PARAMS[0], is the array of the N 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 Each following element CS_PARAMS[i], i.e. with index i > 0, is the array of COSE capabilities for the algorithm capability specified in CS_PARAMS[0][i-1].

  For example, if CS_PARAMS[0][0] specifies the key type as capability of the algorithm, then CS_PARAMS[1] 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]).

H.2. Secret Derivation Parameters

The definition of Secret Derivation Parameters in the Common Context (see Section 2.1.5) is generalized as follows.

Secret Derivation Parameters is a CBOR array SD_PARAMS including N+1 elements, whose exact structure and value depend on the value of Secret Derivation Algorithm.

- o The first element, i.e. SD_PARAMS[0], is the array of the N 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 Each following element SD_PARAMS[i], i.e. with index i > 0, is the array of COSE capabilities for the algorithm capability specified in SD_PARAMS[0][i-1].

  For example, if SD_PARAMS[0][0] specifies the key type as capability of the algorithm, then SD_PARAMS[1] 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]).

H.3. 'par_countersign' in the external_aad

The definition of the 'par_countersign' element in the 'algorithms' array of the external_aad structure (see Section 4.3) is generalized as follows.

The 'par_countersign' element takes the CBOR array CS_PARAMS specified by Counter Signature Parameters in the Common Context (see

Section 2.1.3), considering the format generalization in Appendix H. In particular:

- o The first element 'countersign_alg_capab' is the array of COSE capabilities for the countersignature algorithm indicated in 'alg_countersign'. This is CS_PARAMS[0], i.e. the first element of the CBOR array CS_PARAMS specified by Counter Signature Parameters in the Common Context.

- o Each following element 'countersign_capab_i' (i = 1, ..., N) is the array of COSE capabilities for the algorithm capability specified in 'countersign_alg_capab'[i-1]. This algorithm capability is the element CS_PARAMS[0][i-1] of the CBOR array CS_PARAMS specified by Counter Signature Parameters in the Common Context.

  For example, if 'countersign_alg_capab'[i-1] specifies the key type as capability of the algorithm, then 'countersign_capab_i' 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]).

```
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_capab_1,
                                    countersign_capab_2,
                                    ...,
                                    countersign__capab_N]],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr,
   request_kid_context : bstr,
   OSCORE_option: bstr
]

countersign_alg_capab : [c_1 : any, c_2 : any, ..., c_N : any]
```

Figure 6: external_aad with general 'par_countersign'

Appendix I. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

I.1. 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.

I.2. 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 counter signature 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.

I.3. 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.

I.4. 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 (Sectino 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.

I.5. 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.

I.6. 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.

I.7. 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).

I.8. 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 counter signature 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).

- 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).

I.9. 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".

- 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.

I.10. 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.

I.11. 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, 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

John Preuss Mattsson
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: john.mattsson@ericsson.com

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 19 July 2021

A. Keränen
Ericsson
C. Bormann
Universität Bremen TZI
15 January 2021

SenML Data Value Content-Format Indication
draft-ietf-core-senml-data-ct-03

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

Copyright Notice

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 5
7. IANA Considerations 5
8. References 5
  8.1. Normative References 5
  8.2. Informative References 6
Acknowledgements 6
Authors' Addresses 7

1. Introduction

The Sensor Measurement Lists (SenML) media type [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.


```
[
  {"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 [RFC8949]) 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 [RFC8949] 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, using label "ct". The value of this field is a string value, one of:

* a CoAP Content-Format identifier 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 CoRE Link Format [RFC6690] "ct" attribute).

* or a Content-Format-String [I-D.bormann-core-media-content-type-format] 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 value (media type and parameters, if any), separated by a "@" sign. For example: "text/plain; charset=utf-8@deflate". If no "@" sign is present outside the media type parameters, the Content-Coding is not 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. 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. This includes "decompression bombs", compressed data that is crafted to decompress to extremely large data items.

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

[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-02, 18 August 2020, <http://www.ietf.org/internet-drafts/draft-bormann-core-media-content-type-format-02.txt>.

[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>.

[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

[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>.

[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.

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

CoRE C. Bormann
Internet-Draft Universitaet Bremen TZI
Updates: 8428 (if approved) 21 February 2021
Intended status: Standards Track
Expires: 25 August 2021

SenML Features and Versions
draft-ietf-core-senml-versions-02

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

Table of Contents

1. Introduction 2
1.1. Terminology 3
2. Feature Codes and the Version number 3
2.1. Discussion 3
3. Features: Reserved0, Reserved1, Reserved2, Reserved3 4
4. Feature: Secondary Units 4
5. Security Considerations 4
6. IANA Considerations 4
7. References 5
7.1. Normative References 5
7.2. Informative References 6
Acknowledgements 6
Author's Address 6

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].

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: 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.

$$\text{version} = \sum_{fc = 0}^{52} \text{present}(fc) \; 2^{fc}$$

where present(fc) is 1 if the feature with the feature code "fc" is present, 0 otherwise.

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).

Compatibility with the existing SenML version number, 10 decimal (0b1010), requires reserving four of the lower-most bit positions Section 3. There is an upper limit to the range of the integer numbers that can be represented in all SenML representations: practical JSON limits this to 2^(53)-1 [RFC7493]. This means the feature codes 4 to 52 are available, one of which is taken by Section 4, leaving 48 for allocation. (The current version 10 (with all other feature codes unset) can be visualized as "0b00000000000000000000000000000000000000000000000001010".) For a lifetime of this scheme of several decades, approximately two feature codes per year or less should be allocated. (More boutique features can always be communicated by must-understand fields, see Section 4.4 of [RFC8428].)

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 sightly unwieldy, but will be quickly memorized in practice.

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 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. 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 help clarify the document by providing a review.

Author's Address

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

Phone: +49-421-218-63921

Email: cabo@tzi.org

CoRE C. Bormann
Internet-Draft Universitaet Bremen TZI
Updates: 8428 (if approved) 9 May 2021
Intended status: Standards Track
Expires: 10 November 2021

SenML Features and Versions
draft-ietf-core-senml-versions-03

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

Table of Contents

1. Introduction 2
1.1. Terminology 3
2. Feature Codes and the Version number 3
2.1. Discussion 3
2.2. Updating RFC8428 4
3. Features: Reserved0, Reserved1, Reserved2, Reserved3 5
4. Feature: Secondary Units 5
5. Security Considerations 5
6. IANA Considerations 5
7. References 6
7.1. Normative References 6
7.2. Informative References 7
Acknowledgements 7
Author's Address 7

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: 2^64) 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, specifically the sum of, for each feature present, two taken to the power of the feature code of that feature.

```
           __ 52                       fc
version = \        present(fc)  2
          /__ fc = 0
```

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.)

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).

Compatibility with the existing SenML version number, 10 decimal (0b1010), requires reserving four of the least significant bit positions for the base version as described in Section 3. There is an upper limit to the range of the integer numbers that can be represented in all SenML representations: practical JSON limits this to 2^53-1 [RFC7493]. This means the feature codes 4 to 52 are available, one of which is taken by the feature defined in Section 4, leaving 48 for allocation. (The current version 10 (with all other feature codes unset) can be visualized as "0b00000000000000000000000000000000000000000000000001010".) For a lifetime of this scheme of several decades, approximately two feature codes per year or less should be allocated. Note that less generally applicable features can always be communicated via fields labeled with names that end with the "_" character ("must-understand fields"), see Section 4.4 of [RFC8428].)

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 sightly unwieldy, but will be quickly memorized in practice.

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, 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 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 not allocate 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.

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 F. Palombini
Internet-Draft Ericsson
Intended status: Standards Track M. Tiloca
Expires: August 23, 2021 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

Copyright Notice

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.

```
        CoAP Client                                     CoAP Server
             | ------------- EDHOC message_1 ------------> |
             |                                             |
             | <------------ EDHOC message_2 ------------- |
             |                                             |
     EDHOC verification                                    |
             |                                             |
             | ------------- EDHOC message_3 ------------> |
             |                                             |
             |                                 EDHOC verification
             |                                             +
     OSCORE Sec Ctx                              OSCORE Sec Ctx
       Derivation                                   Derivation
             |                                             |
             | ------------- OSCORE Request -------------> |
             |                                             |
             | <------------ OSCORE Response ------------- |
             |                                             |
```

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.

```
         CoAP Client                                      CoAP Server
              | ------------- EDHOC message_1 ------------> |
              |                                             |
              | <------------ EDHOC message_2 ------------- |
              |                                             |
      EDHOC verification                                    |
              +                                             |
        OSCORE Sec Ctx                                      |
          Derivation                                        |
              |                                             |
              | ---- EDHOC message_3 + OSCORE Request ----> |
              |                                             |
              |                                   EDHOC verification
              |                                             +
              |                                      OSCORE Sec Ctx
              |                                        Derivation
              |                                             |
              | <------------ OSCORE Response ------------- |
              |                                             |
```

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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OSCORE option  |   EDHOC option  | other options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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:

- The used OSCORE Partial IV is 0, consistently with the first request protected with the new OSCORE Security Context.

- 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.

- The EDHOC option is registered with CoAP option number 13.

  - OSCORE option value: 0x090020 (3 bytes)

  - EDHOC option value: - (0 bytes)

  - C_R: 0x20 (1 byte)

  - CIPHERTEXT_3: 0x5253c3991999a5ffb86921e99b607c067770e0 (19 bytes)

  - EDHOC message_3: 0x08 5253c3991999a5ffb86921e99b607c067770e0 (20 bytes)

  - OSCORE ciphertext: 0x612f1092f1776f1c1668b3825e (13 bytes)

  From there:

  - 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: August 26, 2021

M. Tiloca
RISE AB
E. Dijk
IoTconsultancy.nl

February 22, 2021

Proxy Operations for CoAP Group Communication
draft-tiloca-core-groupcomm-proxy-03

Abstract

This document specifies the operations performed by a forward-proxy or reverse-proxy, when using the Constrained Application Protocol (CoAP) in group communication scenarios. Such CoAP proxy processes a single request, sent by a CoAP client over unicast, and distributes the request over IP multicast to a group of CoAP servers. It then collects the individual responses from these CoAP servers and sends these responses to the CoAP client such that the client is able to distinguish the responses and their origin servers through addressing information.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Terminology 4
2. The Multicast-Signaling Option 4
3. The Response-Forwarding Option 5
3.1. Encoding of Server Address 7
3.2. Default Values of the Server Port Number 8
4. Requirements and Objectives 8
5. Protocol Description 10
5.1. Request Sending at the Client 10
5.1.1. Request Sending 10
5.1.2. Supporting Observe 11
5.2. Request Processing at the Proxy 11
5.2.1. Request Processing 12
5.2.2. Supporting Observe 13
5.3. Request and Response Processing at the Server 13
5.3.1. Request and Response Processing 13
5.3.2. Supporting Observe 13
5.4. Response Processing at the Proxy 13
5.4.1. Response Processing 13
5.4.2. Supporting Observe 14
5.5. Response Processing at the Client 15
5.5.1. Response Processing 15
5.5.2. Supporting Observe 16
5.6. Example 16
6. Reverse-Proxies 18
6.1. Processing on the Client Side 19
6.2. Processing on the Proxy Side 19
7. Chain of Proxies 19
7.1. Request Processing at the Proxy 20
7.1.1. Supporting Observe 21
7.2. Response Processing at the Proxy 22
7.2.1. Supporting Observe 22
8. Security Considerations 23
8.1. Client Authentication 23
8.2. Multicast-Signaling Option 24
8.3. Response-Forwarding Option 25
9. IANA Considerations 25
9.1. CoAP Option Numbers Registry 25
9.2. CoAP Transport Information Registry 25
10. References 26

10.1. Normative References 26
10.2. Informative References 28
Appendix A. Using OSCORE Between Client and Proxy 28
A.1. Protecting the Request 29
A.2. Verifying the Request 30
A.3. Protecting the Response 30
A.4. Verifying the Response 30
Appendix B. Examples with Reverse-Proxy 31
B.1. Example 1 31
B.2. Example 2 33
Acknowledgments 35
Authors' Addresses 35

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] allows the presence of forward-proxies and reverse-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 3.4 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 3.4 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 per origin server. It also signals 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 multiple, diffent 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.

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-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 in a CoAP request, 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 5.1.1 and Section 5.2.1).

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 the CoAP Observe Option [RFC7641] is used in the same request. Signaling the time period allows the proxy to stop forwarding responses back to the client, that are received from servers after the end of the time period.

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 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	(*)	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 can perform proxied CoAP group communication 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 CoAP 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.tiloca-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 14.4 of [I-D.tiloca-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.tiloca-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. 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, 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.

If not present, 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.

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 tranport 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 9.2), 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 'srv_port' in 'tp_info' specifies the CBOR simple value Null (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 specification defines some values used as transport protocol identifiers, whose respective new entries are included in the "CoAP Transport Information" Registry defined in Section 14.4 of [I-D.tiloca-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 in the 'tp_info' array specifies the CBOR simple value Null, 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'.

- o If 'tp_id' is equal to 2, i.e. CoAP over UDP secured with DTLS, the default port number 5684 as defined in [RFC7252].
- o If 'tp_id' is equal to 3, i.e. CoAP over TCP, the default port number 5683 as defined in [RFC8323].
- o If 'tp_id' is equal to 4, i.e. CoAP over TCP secured with TLS, the default port number 5684 as defined in [RFC8323].
- o If 'tp_id' is equal to 5, i.e. CoAP over WebSockets, the default port number 80 as defined in [RFC8323].
- o 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

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 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 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 in 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 a CoAP Observe Option is used [RFC7641]).

- o OBJ3. The CoAP proxy returns individual unicast responses to the client, each of which matches the original unicast request made 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 at the Client

This section defines the operations that the client performs for sending a request addressed to a group of servers via the CoAP proxy.

5.1.1. Request Sending

The client proceeds according to the following steps.

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 plus the worst-case round-trip delay between the proxy and any one of 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 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.

5.1.2. 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 5.1.1 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

This section defines the operations that the proxy performs when receiving a request addressed 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. 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 proxyied 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 Multicast-Signaling Option with an empty (zero-length) value, specifying that the Multicast-Signaling 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-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.

5.2.2. 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 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 5.2.1 of this specification.

5.3. Request and Response Processing at the Server

This section defines the operations that the server performs 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 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.

5.3.2. 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

This section defines the operations that the proxy performs when receiving a response matching a forwarded group request.

5.4.1. Response Processing

Upon receiving a response matching 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 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 5.1.1). 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.

5.4.2. 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 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, 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 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

This section defines the operations that the client performs when receiving a response matching a request addressed to a group of servers via the CoAP proxy.

5.5.1. Response Processing

Upon receiving from the proxy a response matching 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].

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 end-to-end 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 - in case this is needed - is the same port number specified in the group

URI of the original unicast CoAP group request sent to the proxy (see Section 5.1.1).

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).

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.

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 5.1.1). 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) 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 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 with value the byte serialization of X.

```
   C                          P                      S1           S2
   |                          |                       |            |
   |------------------------->|                       |            |
   | Src: C_ADDR:C_PORT       |                       |            |
   | Dst: P_ADDR:P_PORT       |                       |            |
   | Proxi-URI {              |                       |            |
   |  coap://G_ADDR:G_PORT/r  |                       |            |
   | }                        |                       |            |
   | Multicast-Signaling: 60  |                       |            |
   |                          |                       |            |
   |                          | Src: P_ADDR:P_PORT    |            |
   |                          | Dst: G_ADDR:G_PORT    |            |
   |                          | Uri-Path: /r          |            |
   |                          |---------------+------>|            |
   |                          |                \      |            |
   |                          |                 +----------------->|
   |                          |                       |            |
   |                          | /* t = 0 : P starts   |            |
   |                          | accepting responses   |            |
   |                          | for this request */   |            |
   |                          |                       |            |
   |                          |                       |            |
   |                          |<----------------------|            |
   |                          | Src: S1_ADDR:G_PORT   |            |
   |                          | Dst: P_ADDR:P_PORT    |            |
   |                          |                       |            |
```

```
   |                          |                      |             |
   |<-------------------------|                      |             |
   | Src: P_ADDR:P_PORT       |                      |             |
   | Dst: C_ADDR:C_PORT       |                      |             |
   | Response-Forwarding {    |                      |             |
   |  [1, /*CoAP over UDP*/   |                      |             |
   |   #6.260(bstr(S1_ADDR))  |                      |             |
   |  ]                       |                      |             |
   | }                        |                      |             |
   |                          |<-----------------------------------|
   |                          |               Src: S2_ADDR:S2_PORT |
   |                          |               Dst: P_ADDR:P_PORT   |
   |<-------------------------|                      |             |
   | Src: P_ADDR:P_PORT       |                      |             |
   | Dst: C_ADDR:C_PORT       |                      |             |
   | Response-Forwarding {    |                      |             |
   |  [1, /*CoAP over UDP*/   |                      |             |
   |   #6.260(bstr(S2_ADDR)), |                      |             |
   |   S2_PORT                |                      |             |
   |  ]                       |                      |             |
   | }                        |                      |             |
   |            /* At t = 60, P stops accepting      |             |
   |            responses for this request */        |             |
   |                          |                      |             |
```

Figure 3: Workflow example with a forward-proxy

6. Reverse-Proxies

The use of reverse-proxies in group communication scenarios is defined in Section 3.4.2 of [I-D.ietf-core-groupcomm-bis].

This section clarifies how the Multicast-Signaling Option is effective also in such a context, in order for:

- o The proxy to explictly reveal itself as a reverse-proxy to the client.

- o 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 request.

This practically addresses the addional issues compared to the case with a forward-proxy, as compiled in Section 3.4.2 of [I-D.ietf-core-groupcomm-bis]. A reverse-proxy may also operate without support of the Multicast-Signaling Option, as defined in that section.

Appendix B provides examples with a reverse-proxy.

6.1. Processing on the Client Side

If a client sends a request intended to a group of servers and is aware of actually communicating with a reverse-proxy, then the client MUST perform the steps defined in Section 5.1.1. In particular, this results in a request sent to the proxy including a Multicast-Signaling Option.

The client processes the responses forwarded back by the proxy as defined in Section 5.5.

6.2. Processing on the Proxy Side

If the proxy receives a request and determines that it should forward it to a group of servers over IP multicast, then the proxy MUST perform the steps defined in Section 5.2.

In particular, when such request does not include a Multicast-Signaling Option, the proxy explicitly reveals itself as a reverse-proxy, by sending a 4.00 (Bad Request) response including an Multicast-Signaling Option with empty (zero-length) value.

7. 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 (group) 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 except the first 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 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.

7.1. 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-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 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-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.

- 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-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.

7.1.1. 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].

7.2. 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.

7.2.1. 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.

- o The proxy acts as a client registered with the next hop towards the origin servers, as described earlier in Section 7.1.1.
- 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 7.2.

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].

8. 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 7), 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.

8.1. 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-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 8.2 and

Section 8.3), 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 7), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

8.2. 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 7), 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 7.1).

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 (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 7), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

8.3. 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 (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 7), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

9. IANA Considerations

This document has the following actions for IANA.

9.1. 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]]

9.2. CoAP Transport Information Registry

IANA is asked to enter the following entries to the "CoAP Transport Information" Registry defined in Section 14.4 of [I-D.tiloca-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]

10. References

10.1. Normative References

[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.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-05 (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>.

[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>.

[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>.

10.2. 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-10 (work in progress), February 2021.

[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 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>.

[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. 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.

In particular, a CoAP message is protected with the OSCORE Security Context between the origin client and the proxy, as considering both of them to be terminal endpoints for the exchange in question. This requires that some CoAP options in that message are processed as class E, although originally defined as class U or class I.

This generally applies to all options that the proxy needs to understand and process in its exchange with the origin client. Further options can be added and treated as class U, e.g. related to routing information. The rest of this section hightlights the most relevant CoAP options to consider in this respect.

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 7), 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.

Appendix B. Examples with Reverse-Proxy

The examples in this section refer to the following actors.

- o One origin client C, with address C_ADDR and port number C_PORT.
- o One proxy P, with address P_ADDR and port number P_PORT.
- o 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 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 with value the byte serialization of X.

B.1. Example 1

The example shown in Figure 4 considers a reverse-proxy that stands in for both the whole group of servers and for each of those servers (e.g. acting as a firewall).

In particular:

- o The address 'group1.com' resolves to 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.
- o 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.

Note that this type of reverse-proxy implementation requires the proxy to use (potentially) a large number of distinct IP addresses, so it is not very scalable.

```
   C                              P                      S1           S2
   |                              |                      |             |
   |----------------------------->| /* C is not aware    |             |
   | Src: C_ADDR:C_PORT           | that P is in fact    |             |
   | Dst: group1.com:P_PORT       | a reverse-proxy */   |             |
   | Uri-Path: /r                 |                      |             |
   |                              |                      |             |
   |                              |                      |             |
   |<-----------------------------|                      |             |
   | Src: group1.com:P_PORT       |                      |             |
   | Dst: C_ADDR:C_PORT           |                      |             |
   | 4.00 Bad Request             |                      |             |
   | Multicast-Signaling: (empty) |                      |             |
   | Payload: "Please use         |                      |             |
   |     Multicast-Signaling"     |                      |             |
   |                              |                      |             |
   |----------------------------->|                      |             |
   | Src: C_ADDR:C_PORT           |                      |             |
   | Dst: group1.com:P_PORT       |                      |             |
   | Multicast-Signaling: 60      |                      |             |
   | Uri-Path: /r                 |                      |             |
   |                              |                      |             |
   |                              | Src: P_ADDR:P_PORT   |             |
   |                              | Dst: G_ADDR:G_PORT   |             |
   |                              | Uri-Path: /r         |             |
   |                              |---------------+----->|             |
   |                              |                \     |             |
   |                              |                 +----------------->|
   |                              |                      |             |
   |                              | /* t = 0 : P starts  |             |
   |                              | accepting responses  |             |
   |                              | for this request */  |             |
   |                              |                      |             |
   |                              |                      |             |
   |                              |<---------------------|             |
   |                              | Src: S1_ADDR:S1_PORT |             |
   |                              | Dst: P_ADDR:P_PORT   |             |
   |                              |                      |             |
   |<-----------------------------|                      |             |
   | Src: group1.com:P_PORT       |                      |             |
   | Dst: C_ADDR:C_PORT           |                      |             |
   | Response-Forwarding {        |                      |             |
   |  [3, /*CoAP over TCP*/       |                      |             |
   |   #6.260(bstr(D1_ADDR)),     |                      |             |
   |   D1_PORT                    |                      |             |
   |  ]                           |                      |             |
   | }                            |                      |             |
   |                              |                      |             |
```

```
   |                              |<-----------------------------------|
   |                              |               Src: S2_ADDR:S2_PORT |
   |                              |               Dst: P_ADDR:P_PORT   |
   |                              |                      |             |
   |<-----------------------------|                      |             |
   | Src: group1.com:P_PORT       |                      |             |
   | Dst: C_ADDR:C_PORT           |                      |             |
   | Response-Forwarding {        |                      |             |
   |  [3, /*CoAP over TCP*/       |                      |             |
   |   #6.260(bstr(D2_ADDR)),     |                      |             |
   |   D2_PORT                    |                      |             |
   |  ]                           |                      |             |
   | }                            |                      |             |
   |                              |                      |             |
   |                /* At t = 60, P stops accepting      |             |
   |                responses for this request */        |             |
   |                              |                      |             |
   |                              |                      |             |
   |----------------------------->| /* Request intended  |             |
   | Src: C_ADDR:C_PORT           | only to S1 */        |             |
   | Dst: D1_ADDR:D1_PORT         |                      |             |
   | Uri-Path: /r                 |                      |             |
   |                              |                      |             |
   |                              | Src: P_ADDR:P_PORT   |             |
   |                              | Dst: S1_ADDR:S1_PORT |             |
   |                              | Uri-Path: /r         |             |
   |                              |--------------------->|             |
   |                              |                      |             |
   |                              |                      |             |
   |                              |<---------------------|             |
   |                              | Src: S1_ADDR:S1_PORT |             |
   |                              | Dst: P_ADDR:P_PORT   |             |
   |                              |                      |             |
   |<-----------------------------|                      |             |
   | Src: D1_ADDR:D1_PORT         |                      |             |
   | Dst: C_ADDR:C_PORT           |                      |             |
   |                              |                      |             |
```

Figure 4: Workflow example with reverse-proxy standing in for both the whole group of servers and each individual server

B.2. Example 2

The example shown in Figure 5 builds on the example in Appendix B.1.

However, it considers a reverse-proxy that stands in for only the whole group of servers, but not for each individual server.

The final exchange between C and S1 occurs with CoAP over UDP.

```
   C                              P                      S1           S2
   |                              |                      |             |
   |----------------------------->| /* C is not aware    |             |
   | Src: C_ADDR:C_PORT           | that P is in fact    |             |
   | Dst: group1.com:P_PORT       | a reverse-proxy */   |             |
   | Uri-Path: /r                 |                      |             |
   |                              |                      |             |
   |<-----------------------------|                      |             |
   | Src: group1.com:P_PORT       |                      |             |
   | Dst: C_ADDR:C_PORT           |                      |             |
   | 4.00 Bad Request             |                      |             |
   | Multicast-Signaling: (empty) |                      |             |
   | Payload: "Please use         |                      |             |
   |     Multicast-Signaling"     |                      |             |
   |                              |                      |             |
   |----------------------------->|                      |             |
   | Src: C_ADDR:C_PORT           |                      |             |
   | Dst: group1.com:P_PORT       |                      |             |
   | Multicast-Signaling: 60      |                      |             |
   | Uri-Path: /r                 |                      |             |
   |                              |                      |             |
   |                              | Src: P_ADDR:P_PORT   |             |
   |                              | Dst: G_ADDR:G_PORT   |             |
   |                              | Uri-Path: /r         |             |
   |                              |---------------+----->|             |
   |                              |                \     |             |
   |                              |                 +----------------->|
   |                              |                      |             |
   |                              | /* t = 0 : P starts  |             |
   |                              | accepting responses  |             |
   |                              | for this request */  |             |
   |                              |                      |             |
   |                              |                      |             |
   |                              |<---------------------|             |
   |                              | Src: S1_ADDR:S1_PORT |             |
   |                              | Dst: P_ADDR:P_PORT   |             |
   |                              |                      |             |
   |<-----------------------------|                      |             |
   | Dst: group1.com:P_PORT       |                      |             |
   | Dst: C_ADDR:C_PORT           |                      |             |
   | Response-Forwarding {        |                      |             |
   |  [1, /*CoAP over UDP*/       |                      |             |
   |   #6.260(bstr(S1_ADDR)),     |                      |             |
   |   S1_PORT                    |                      |             |
   |  ]                           |                      |             |
   | }                            |                      |             |
```

```
   |                              |                      |             |
   |                              |<-----------------------------------|
   |                              |               Src: S2_ADDR:S2_PORT |
   |                              |               Dst: P_ADDR:P_PORT   |
   |                              |                      |             |
   |<-----------------------------|                      |             |
   | Dst: group1.com:P_PORT       |                      |             |
   | Dst: C_ADDR:C_PORT           |                      |             |
   | Response-Forwarding {        |                      |             |
   |  [1, /*CoAP over UDP*/       |                      |             |
   |   #6.260(bstr(S2_ADDR)),     |                      |             |
   |   S2_PORT                    |                      |             |
   |  ]                           |                      |             |
   | }                            |                      |             |
   |                              |                      |             |
   |                /* At t = 60, P stops accepting      |             |
   |                responses for this request */        |             |
   |                              |                      |             |
   |                              |                      |             |
   |----------------------------------------------------->|             |
   | Src: C_ADDR:C_PORT           | /* Request intended  |             |
   | Dst: S1.ADDR:S1_PORT         | only to S1 */        |             |
   | Uri-Path: /r                 |                      |             |
   |                              |                      |             |
   |                              |                      |             |
   |<-----------------------------------------------------|             |
   | Src: S1.ADDR:S1_PORT         |                      |             |
   | Dst: C_ADDR:C_PORT           |                      |             |
   |                              |                      |             |
```

Figure 5: 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
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Esko Dijk
IoTconsultancy.nl
________________\
Utrecht
The Netherlands

Email: esko.dijk@iotconsultancy.nl

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

Copyright Notice

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.

- '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.

- '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.

- '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.

- '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:

- The multicast notifications MUST be Non-confirmable.

- 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.

- 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.

- 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:

- 'bstr(X)' denotes a CBOR byte string with value the byte serialization of X, with '|' denoting byte concatenation.

- '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.

- '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 .)|
    |                                                              |
    |                                                              |
    |                                                              |
    |       (S sends to itself a phantom observation request PH_REQ|
    |        as coming from the IP multicast address GRP_ADDR .)   |
    |          ------------------------------------------------    |
    |        /                                                     |
    |        \---------------------------------------------------> |  /r
    |                                           GET                |
    |                                           Token: 0x7b        |
    |                                           Observe: 0 (Register)
    |                                           <Other options>    |
    |                                                              |
    |                       (S creates a group observation of /r .)|
    |                                                              |
    |                           (S increments the observer counter |
    |                            for the group observation of /r .)|
    |                                                              |
   C_1 <-------------------- [ Unicast ] ---------------------     S
    |   5.03                                                       |
    |   Token: 0x4a                                                |
    |   Content-Format: application/informative-response+cbor      |
    |   <Other options>                                            |
    |   Payload: {                                                 |
    |     tp_info    : [1, bstr(SRV_ADDR), SRV_PORT,               |
    |                   0x7b, bstr(GRP_ADDR), GRP_PORT],           |
    |     ph_req     : bstr(0x01 | OPT),                           |
    |     last_notif : bstr(0x45 | OPT | 0xff | PAYLOAD)           |
    |   }                                                          |
    |                                                              |
   C_2     ----------------- [ Unicast ] ------------------------> S  /r
    |   GET                                                        |
    |   Token: 0x01                                                |
    |   Observe: 0 (Register)                                      |
    |   <Other options>                                            |
    |                                                              |
    |                           (S increments the observer counter |
    |                            for the group observation of /r .)|
    |                                                              |
```

```
   |                                                               |
  C_2 <-------------------- [ Unicast ] ---------------------      S
   |  5.03                                                         |
   |  Token: 0x01                                                  |
   |  Content-Format: application/informative-response+cbor       |
   |  <Other options>                                              |
   |  Payload: {                                                   |
   |    tp_info    : [1, bstr(SRV_ADDR), SRV_PORT,                 |
   |                  0x7b, bstr(GRP_ADDR), GRP_PORT],             |
   |    ph_req     : bstr(0x01 | OPT),                             |
   |    last_notif : bstr(0x45 | OPT | 0xff | PAYLOAD)             |
   |  }                                                            |
   |                                                               |
   |        (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"                                            |
   |                                                               |
```

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:

- L is computed as L = ceil(log2(N / M)).

- N is the current value of the observer counter, possibly rounded up to 1, i.e. N = max(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.

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.

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.

- '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.

- 'sec_gp', with value the name of the OSCORE group, encoded as a CBOR text string.

- 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.

- 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].

- 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].

- 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].

- 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].

- 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:

- As 'kid', the Sender ID of the server in the OSCORE group.
- 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-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.

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-oscore-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-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.

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.

- The client MUST NOT perform any replay check. That is, the client skips step 3 in Section 8.2 of [RFC8613].

- 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.

- 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.

- 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.

- 'request_kid' takes the value of the 'kid' field from the OSCORE option of the phantom registration request (see Section 7.3.1).

- 'request_piv' takes the value of the 'piv' field from the OSCORE option of the phantom registration request (see Section 7.3.1).

- '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:

- 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 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 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', 'contexId', '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.

- 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-oscore]. 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-oscore], 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-oscore] 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.

```
C1     C2     P        S
|      |      |        |
|      |      |        |  (The value of the resource /r is "1234")
|      |      |        |
+------------->        |   Token: 0x4a
| GET  |      |        |   Observe: 0 (Register)
|      |      |        |   Proxy-Uri: coap://sensor.example/r
|      |      |        |
|      |      +-------->   Token: 0x5e
|      |      | GET    |   Observe: 0 (Register)
|      |      |        |   Uri-Host: sensor.example
|      |      |        |   Uri-Path: r
|      |      |        |
|      |      |        |   (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)
|      |      |        |
|      |      |  ------+
|      |      | /      |
|      |      | \----->|   Token: 0x7b
|      |      |   GET  |   Observe: 0 (Register)
|      |      |        |   Uri-Host: sensor.example
|      |      |        |   Uri-Path: r
|      |      |        |
```

```
 |      |      |        |
 |      |      |        |  (S creates a group observation of /r)
 |      |      |        |
 |      |      |        |  (S increments the observer counter
 |      |      |        |  for the group observation of /r)
 |      |      |        |
 |      |      |        |
 |      |      |<-------+  Token: 0x5e
 |      |      | 5.03   |  Content-Format: application/
 |      |      |        |     informative-response+cbor
 |      |      |        |  <Other options>
 |      |      |        |  Payload: {
 |      |      |        |    tp_info    : [1, bstr(SRV_ADDR), SRV_PORT,
 |      |      |        |                 0x7b, bstr(GRP_ADDR),
 |      |      |        |                 GRP_PORT],
 |      |      |        |    ph_req     : bstr(0x01 | OPT),
 |      |      |        |    last_notif : bstr(0x45 | OPT |
 |      |      |        |                     0xff | PAYLOAD)
 |      |      |        |  }
 |      |      |        |
 |      |      |        |  (PAYLOAD in 'last_notif' : "1234")
 |      |      |        |
 |      |      |        |
 |      |      |        |  (The proxy starts listening to the
 |      |      |        |   GRP_ADDR address and the GRP_PORT port.)
 |      |      |        |
 |      |      |        |  (The proxy adds C1 to its list of observers.)
 |      |      |        |
 |      |      |        |
 |<------------+        |  Token: 0x4a
 | 2.05 |      |        |  Content-Format: application/cbor
 |      |      |        |  <Other options>
 |      |      |        |  Payload: "1234"
 :      :      :        :
 :      :      :        :
 :      :      :        :
 |      +----->|        |  Token: 0x01
 |      | GET  |        |  Observe: 0 (Register)
 |      |      |        |  Proxy-Uri: coap://sensor.example/r
 |      |      |        |
 |      |      |        |  (The proxy has a fresh cache representation)
 |      |      |        |
 |      |<-----+        |  Token: 0x01
 |      | 2.05 |        |  Content-Format: application/cbor
 |      |      |        |  <Other options>
 |      |      |        |  Payload: "1234"
 |      |      |        |
```

```
 |      |      |        |
 :      :      :        :
 :      :      :        :  (The value of the resource
 :      :      :        :  /r changes to "5678".)
 :      :      :  (*)   :
 |      |      |<-------+  Token: 0x7b
 |      |      | 2.05   |  Observe: 11
 |      |      |        |  Content-Format: application/cbor
 |      |      |        |  <Other options>
 |      |      |        |  Payload: "5678"
 |      |      |        |
 |<------------+        |  Token: 0x4a
 | 2.05 |      |        |  Observe: 54123
 |      |      |        |  Content-Format: application/cbor
 |      |      |        |  <Other options>
 |      |      |        |  Payload: "5678"
 |      |      |        |
 |      |<-----+        |  Token: 0x01
 |      | 2.05 |        |  Observe: 54123
 |      |      |        |  Content-Format: application/cbor
 |      |      |        |  <Other options>
 |      |      |        |  Payload: "5678"

 (*) 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.

```
 C1      C2      P          S
 |       |       |          | (The value of the resource /r is "1234")
 |       |       |          |
 +-------------->|          |  Token: 0x4a
 | FETCH |       |          |  Observe: 0 (Register)
 |       |       |          |  OSCORE: {kid: 1 ; piv: 101 ; ...}
 |       |       |          |  Uri-Host: sensor.example
 |       |       |          |  Proxy-Scheme: coap
 |       |       |          |  <Other class U/I options>
 |       |       |          |  0xff
 |       |       |          |  Encrypted_payload {
 |       |       |          |    0x01 (GET),
 |       |       |          |    Observe: 0 (Register),
 |       |       |          |    Uri-Path: r,
 |       |       |          |    <Other class E options>
 |       |       |          |  }
 |       |       |          |
 |       |       +--------->|  Token: 0x5e
 |       |       | FETCH    |  Observe: 0 (Register)
 |       |       |          |  OSCORE: {kid: 1 ; piv: 101 ; ...}
 |       |       |          |  Uri-Host: sensor.example
 |       |       |          |  <Other class U/I options>
 |       |       |          |  0xff
 |       |       |          |  Encrypted_payload {
 |       |       |          |    0x01 (GET),
 |       |       |          |    Observe: 0 (Register),
 |       |       |          |    Uri-Path: r,
 |       |       |          |    <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)
 |       |       |          |
 |       |       |  -------+
 |       |       | /        |
 |       |       | \------->|  Token: 0x7b
 |       |       |   FETCH  |  Observe: 0 (Register)
 |       |       |          |  OSCORE: {kid: 5 ; piv: 501 ;
 |       |       |          |           kid context: 57ab2e; ...}
```

```
   |         |         |           |  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)
   |         |         |           |
   |         |         |           |
   |         |         |<----------+  Token: 0x5e
   |         |         | 2.05      |  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"
   |         |         |           |    }
   |         |         |           |  }
   |         |         |           |
```

```
 |       |       |         |
 |<--------------+         |  Token: 0x4a
 | 2.05  |       |         |  OSCORE: {piv: 301 ; ...}
 |       |       |         |  <Other class U/I options>
 |       |       |         |  0xff
 |       |       |         |  (Same Encrypted_payload)
 |       |       |         |
 |       |       |         |
 +-------------->|         |  Token: 0x4b
 | FETCH |       |         |  Observe: 0 (Register)
 |       |       |         |  OSCORE: {kid: 5 ; piv: 501 ; ...}
 |       |       |         |  Uri-Host: sensor.example
 |       |       |         |  Proxy-Scheme: coap
 |       |       |         |  Listen-To-
 |       |       |         |  Multicast-Responses: {[1, bstr(SRV_ADDR),
 |       |       |         |                         SRV_PORT, 0x7b,
 |       |       |         |                         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 starts listening to the
 |       |       |         |   GRP_ADDR address and the GRP_PORT port.)
 |       |       |         |
 |       |       |         |  (The proxy adds C1 to
 |       |       |         |   its list of observers.)
 :       :       :         :
 :       :       :         :
 :       :       :         :
 |       +------>|         |  Token: 0x01
 |       | FETCH |         |  Observe: 0 (Register)
 |       |       |         |  OSCORE: {kid: 2 ; piv: 201 ; ...}
 |       |       |         |  Uri-Host: sensor.example
 |       |       |         |  Proxy-Scheme: coap
 |       |       |         |  <Other class U/I options>
 |       |       |         |  0xff
 |       |       |         |  Encrypted_payload {
 |       |       |         |    0x01 (GET),
 |       |       |         |    Observe: 0 (Register),
```

```
 |       |       |         |    Uri-Path: r,
 |       |       |         |    <Other class E options>
 |       |       |         |  }
 |       |       |         |
 |       |       +-------->|  Token: 0x5e
 |       |       | FETCH   |  Observe: 0 (Register)
 |       |       |         |  OSCORE: {kid: 2 ; piv: 201 ; ...}
 |       |       |         |  Uri-Host: sensor.example
 |       |       |         |  <Other class U/I options>
 |       |       |         |  0xff
 |       |       |         |  Encrypted_payload {
 |       |       |         |    0x01 (GET),
 |       |       |         |    Observe: 0 (Register),
 |       |       |         |    Uri-Path: r,
 |       |       |         |    <Other class E options>
 |       |       |         |  }
 |       |       |         |
 |       |       |<--------+  Token: 0x5e
 |       |       | 2.05    |  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"
 |       |       |         |    }
 |       |       |         |  }
 |       |       |         |
 |       |<------+         |  Token: 0x01
 |       | 2.05  |         |  OSCORE: {piv: 401 ; ...}
 |       |       |         |  <Other class U/I options>
 |       |       |         |  0xff
 |       |       |         |  (Same Encrypted_payload)
 |       |       |         |
```

```
 |        |        |           |
 |        +------->|           |  Token: 0x02
 |        | FETCH  |           |  Observe: 0 (Register)
 |        |        |           |  OSCORE: {kid: 5 ; piv: 501 ; ...}
 |        |        |           |  Uri-Host: sensor.example
 |        |        |           |  Proxy-Scheme: coap
 |        |        |           |  Listen-To-
 |        |        |           |  Multicast-Responses: {[1, bstr(SRV_ADDR),
 |        |        |           |                         SRV_PORT, 0x7b,
 |        |        |           |                         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, and serves the cached
 |        |        |           |   response)
 |        |        |           |
 |        |<-------+           |  Token: 0x02
 |        | 2.05   |           |  OSCORE: {piv: 301 ; ...}
 |        |        |           |  <Other class U/I options>
 |        |        |           |  0xff
 |        |        |           |  (Same as earlier to C1)
 :        :        :           :
 :        :        :           :
 :        :        :           :
 |        |        |           |  (The value of the resource
 |        |        |           |  /r changes to "5678".)
 |        |        |           |
 |        |        |           |
 |        |        |    (*)    |
 |        |        |<----------+  Token: 0x7b
 |        |        | 2.05      |  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>
 |       |       |         |
 |       |       |         |
 |<--------------+         |  Token: 0x4b
 | 2.05  |       |         |  Observe: 54123
 |       |       |         |  OSCORE: {kid: 5; piv: 502 ;
 |       |       |         |           kid context: 57ab2e; ...}
 |       |       |         |  <Other class U/I options>
 |       |       |         |  0xff
 |       |       |         |  (Same Encrypted_payload)
 |       |       |         |
 |       |<------+         |  Token: 0x02
 |       | 2.05  |         |  Observe: 54123
 |       |       |         |  OSCORE: {kid: 5; piv: 502 ;
 |       |       |         |           kid context: 57ab2e; ...}
 |       |       |         |  <Other class U/I options>
 |       |       |         |  0xff
 |       |       |         |  (Same Encrypted_payload)
```

(*) 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 M. Tiloca
Internet-Draft RISE AB
Intended status: Standards Track C. Amsuess
Expires: August 26, 2021

P. van der Stok
Consultant
February 22, 2021

Discovery of OSCORE Groups with the CoRE Resource Directory
draft-tiloca-core-oscore-discovery-08

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

Copyright Notice

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 10
2.3.1. Example in Link Format 10
2.3.2. Example in CoRAL 11
3. Addition and Update of Security Groups 11
3.1. Addition Example 12
3.1.1. Example in Link Format 12
3.1.2. Example in CoRAL 13
4. Discovery of Security Groups 15
4.1. Discovery Example #1 16
4.1.1. Example in Link Format 16
4.1.2. Example in CoRAL 18
4.2. Discovery Example #2 19
4.2.1. Example in Link Format 19
4.2.2. Example in CoRAL 20
5. Use Case Example With Full Discovery 21
6. Security Considerations 25
7. IANA Considerations 25
8. References 25
8.1. Normative References 25
8.2. Informative References 27
Appendix A. Use Case Example With Full Discovery (CoRAL) 28
Acknowledgments 33
Authors' Addresses 33

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 show the CoRAL textual serialization format, its binary serialization format is used 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.

- '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 21.11 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 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 Administator 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.

- 'alg', specifying the AEAD 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].

- '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 defined in [COSE.Algorithms].

- 'cs_alg', specifying the algorithm used to countersign 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].

- '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, which is taken from the 'Value' column of the "COSE Elliptic Curves" Registry [COSE.Elliptic.Curves].

- '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, which is taken from the 'Value' column of the "COSE Key Types" Registry [COSE.Key.Types].

- '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, which is taken from the 'Value' column of the "COSE Elliptic Curves" Registry defined in [COSE.Elliptic.Curves].

- 'cs_kenc', specifying the encoding of the public keys used in the security group. If present, this parameter MUST specify a single value. 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.

- '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 2.3 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 defined in [COSE.Elliptic.Curves].

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 'cs_alg', 'cs_alg_crv', 'cs_key_kty', 'cs_key_crv', 'cs_kenc', '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 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.

- The values of 'alg', 'hkdf', 'cs_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 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.

- '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 (-8), with elliptic curve Ed25519 (6). Public keys used in the security group are encoded as COSE Keys (1) [I-D.ietf-cose-rfc8152bis-struct]. 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 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=gm1
Content-Format: 40
Payload:
</ace-group/feedca570000>;ct=65000;rt="core.osc.gm";if="ace.group";
                             sec-gp="feedca570000";app-gp="group1";
                             cs_alg="-8";cs_alg_crv="6";
                             cs_kenc="1";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=gm1
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"
   cs_alg -8
   cs_alg_crv 6
   cs_kenc 1
   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.

- o 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 to the security group with name "feedca570000".

- An additional group-membership resource associated to 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=gm1
Content-Format: 40
Payload:
</ace-group/feedca570000>;ct=65000;rt="core.osc.gm";if="ace.group";
                             sec-gp="feedca570000";app-gp="group1";
                             cs_alg="-8";cs_alg_crv="6";
                             cs_kenc="1";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";
                             cs_alg="-8";cs_alg_crv="6";
                             cs_kenc="1";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";cs_alg="-8";
                             cs_alg_crv="6";cs_kenc="1";
                             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=gm1
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"
      cs_alg -8
      cs_alg_crv 6
      cs_kenc 1
      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"
      cs_alg -8
      cs_alg_crv 6
      cs_kenc 1
      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"
      cs_alg -8
      cs_alg_crv 6
      cs_kenc 1
      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 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.

- '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 public keys 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";cs_alg="-8";cs_alg_crv="6";
    cs_kenc="1";ecdh_alg="-27";ecdh_alg_crv="4";
    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]";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"
   cs_alg -8
   cs_alg_crv 6
   cs_kenc 1
   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";cs_alg="-8";cs_alg_crv="6";
    cs_kenc="1";ecdh_alg="-27";ecdh_alg_crv="4";
    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>;ct=65000;
    rt="core.osc.gm";if="ace.group";sec-gp="feedca570000";
    app-gp="group1";cs_alg="-8";cs_alg_crv="6";
    cs_kenc="1";ecdh_alg="-27";ecdh_alg_crv="4";
    anchor="coap://[2001:db8::ab]",
<coap://[2001:db8::ab]/ace-group/ech0ech00000>;ct=65000;
    rt="core.osc.gm";if="ace.group";sec-gp="ech0ech00000";
    app-gp="group2";cs_alg="-8";cs_alg_crv="6";
    cs_kenc="1";ecdh_alg="-27";ecdh_alg_crv="4";
    anchor="coap://[2001:db8::ab]",
<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";cs_alg="-8";cs_alg_crv="6";
    cs_kenc="1";ecdh_alg="-27";ecdh_alg_crv="4";
    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:ct 65000
   reef:rt "core.osc.gm"
   reef:if "ace.group"
   sec-gp "feedca570000"
   app-gp "group1"
   cs_alg -8
   cs_alg_crv 6
   cs_kenc 1
   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 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 -> 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=gm1&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";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>;ct=65000;
    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-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-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-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>.

[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-11 (work in progress), February 2021.

[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-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-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-02 (work in progress), February 2021.

[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-45 (work in progress), November 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>.

[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=gm1
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 -> 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
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