CoRE Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: 4 September 2025

M. Tiloca
RISE AB
E. Dijk
IoTconsultancy.nl

3 March 2025

Proxy Operations for CoAP Group Communication
draft-ietf-core-groupcomm-proxy-04

Abstract

This document specifies the operations performed by a proxy, when using the Constrained Application Protocol (CoAP) in group communication scenarios. Such a proxy processes a single request sent by a client typically over unicast, and distributes the request to a group of servers, e.g., over UDP/IP multicast as the defined default transport protocol. Then, the proxy collects the individual responses from those servers and relays those responses back to the client, in a way that allows the client to distinguish the responses and their origin servers through embedded addressing information. This document updates RFC7252 with respect to caching of response messages at proxies.

Discussion Venues

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

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

Source for this draft and an issue tracker can be found at https://github.com/core-wg/groupcomm-proxy.

Status of This Memo

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

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

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

This Internet-Draft will expire on 4 September 2025.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] allows the presence of proxies, as intermediary entities supporting clients by performing requests on their behalf and relaying back responses.

CoAP supports also group communication over IP multicast [I-D.ietf-core-groupcomm-bis], where a group request can be addressed to multiple recipient servers, each of which may reply with an individual unicast response. As discussed in Sections E and F 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 typically sends to the proxy a single unicast request, which the proxy forwards to a group of CoAP servers, e.g., using UDP/IP multicast as the defined default transport protocol for CoAP group requests (see Section 1.1 of [I-D.ietf-core-groupcomm-bis]). Later on, the proxy replies to the client's original unicast request, by relaying back the responses from the servers.

As per [RFC7252], a CoAP-to-CoAP proxy relays those responses to the client as separate CoAP messages, all matching (by Token) with the client's original unicast request. A possible alternative approach for aggregating those responses into a single CoAP response sent to the client would require a specific aggregation Content-Format, which is not available yet. Both these approaches have open issues.

This document takes the former approach. That is, after forwarding a CoAP group request from the client to the group of CoAP servers, the proxy relays the individual responses back to the client as separate CoAP messages. The described method addresses all the related issues raised in Sections E and F of [I-D.ietf-core-groupcomm-bis]. To this end, this document defines a dedicated signaling protocol based on two new CoAP options and used by the client and the proxy.

By using this protocol, the client explicitly confirms its intent to perform a proxied group request and its support for receiving multiple responses as a result, i.e., one or more from each origin server. Also, the client signals for how long it is willing to wait for responses. When relaying to the client a response to the group

request, the proxy indicates addressing information pertaining to the origin server. This enables the client to distinguish multiple different responses by origin and to possibly contact one or more of the respective servers by sending individual unicast request(s) to the indicated address(es). In doing these follow-up unicast requests, the client can optionally bypass the proxy.

Like [I-D.ietf-core-groupcomm-bis], this document refers to UDP/IP multicast as the transport protocol that a proxy uses to forward a CoAP group request to a group of servers. While other transport protocols such as broadcast, non-IP multicast, and geocast can also be possible to employ, their use is not considered in this document.

This document also defines how the proposed protocol is used between an HTTP client and an HTTP-to-CoAP cross-proxy, in order to forward an HTTP group request from the client to a group of CoAP servers, and relay back the individual CoAP responses as HTTP responses.

Finally, this document defines a caching model for proxies and specifies how they can serve a group request by using cached responses. Therefore, this document updates [RFC7252].

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts from the following specifications:

* CoAP [RFC7252] and Group Communication for CoAP [I-D.ietf-core-groupcomm-bis].

* Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

* Concise Data Definition Language (CDDL) [RFC8610], Concise Binary Object Representation (CBOR) [RFC8949], and CBOR sequences [RFC8742]

* Constrained Resource Identifiers (CRIs) [I-D.ietf-core-href].

Unless specified otherwise, the term "proxy" refers to a CoAP-to-CoAP forward-proxy, as defined in Section 5.7.2 of [RFC7252].

This document also uses the following terminology.

* Individual request: a request that an origin client sends to a single origin server within a group, either directly or instead indirectly via a proxy.

2. The Multicast-Timeout Option

The Multicast-Timeout Option defined in this section has the properties summarized in Table 1, which extends Table 4 of [RFC7252].

Since the option is not Safe-to-Forward, the column "N" indicates a dash for "not applicable". The value of the Multicast-Timeout Option specifies a timeout value in seconds, encoded as an unsigned integer (see Section 3.2 of [RFC7252]).

No.	C	U	N	R	Name	Format	Length	Default
TBD1	x		-		Multicast-Timeout	uint	0-4	(none)

Table 1: The Multicast-Timeout Option. C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

This document specifically defines how this option is used by a client in a CoAP request, to indicate to a proxy its support for and interest in receiving multiple responses to a proxied CoAP group request (i.e., one or more responses from each origin server) and for how long it is willing to wait for receiving responses via that proxy (see Section 5.1.1 and Section 5.2.1).

When sending a CoAP group request to a proxy via IP unicast, to be forwarded by the proxy to a targeted group of servers, the client includes the Multicast-Timeout Option in the request. The option value indicates after how much time in seconds the client will stop accepting responses matching its original unicast request, with the exception of notifications if the CoAP Observe Option [RFC7641] is used in the same request. This allows the proxy to stop relaying responses back to the client, if those are received from servers after the indicated amount of time has elapsed.

The Multicast-Timeout Option is of class U in terms of OSCORE processing (see Section 4.1 of [RFC8613]).

3. The Reply-From Option

The Reply-From Option defined in this section has the properties summarized in Table 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			-		Reply-From	(*)	5-1034	(none)

Table 2: The Reply-From Option. C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable, (*) See below

This document specifically defines how this option is used by a proxy that can perform proxied CoAP group requests.

Upon receiving a response to such a request from an origin server, the proxy includes the Reply-From Option in the response sent to the origin client (see Section 5). The proxy uses the option to indicate addressing information pertaining to that origin server, which the client can use in order to send an individual request intended to that server.

In particular, the client can use the addressing information specified in the option in order to identify the response originator and to possibly send it individual unicast requests later on, either directly or instead indirectly via the proxy.

When used as defined in this document, the option value is set to the byte serialization of a CBOR sequence [RFC8742], which is composed of at most two CBOR arrays.

* The first CBOR array is REQUIRED and specifies a CRI (see [I-D.ietf-core-href]). In particular, both ’scheme’ and ’authority’ are given, while ’path’, ’query’, and ’fragment’ are not given.

* The second CBOR array is OPTIONAL and specifies a CRI reference (see [I-D.ietf-core-href]). In particular, ’scheme’ is set to null (0xf6), at least one of ’authority’ and ’path’ is given, and both ’query’ and ’fragment’ are not given.

If ’authority’ is given in this CRI reference, then ’host’ MUST NOT be of the form host-name within ’authority’ of the CRI specified by the first CBOR array.

This CRI reference is relevant in some scenarios where the proxy is a reverse-proxy.

The detailed use of this option is specified in Section 5.4.1 and Section 5.5.1 when the proxy is a forward-proxy, and in Section 6.1 and Section 6.2 when the proxy is a reverse-proxy.

The Reply-From Option is of class U in terms of OSCORE processing (see Section 4.1 of [RFC8613]).

4. Requirements and Objectives

In this section, the word "proxy" is not limited to forward-proxies. Instead, it comprises also reverse-proxies and HTTP-to-CoAP proxies.

This document assumes that the following requirements are fulfilled.

* REQ1. The proxy is explicitly configured with an allow-list for performing proxied group requests on behalf of specific allowed clients.

* REQ2. The proxy MUST identify a client sending a unicast group request to be proxied, in order to verify whether the client is allowed-listed to do so. For example, this can rely on one of the following security associations.

  - A TLS [RFC8446] or DTLS [RFC9147] channel between the client and the proxy, where the client has been authenticated during the secure channel establishment.

  - A pairwise OSCORE [RFC8613] Security Context between the client and the proxy, as defined in [I-D.ietf-core-oscore-capable-proxies].

* REQ3. If end-to-end secure 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 Sections E and F of [I-D.ietf-core-groupcomm-bis]. This requires the client and the servers to have previously joined the correct OSCORE group, for instance by using the approach described in [I-D.ietf-ace-key-groupcomm-oscore]. The correct OSCORE group to join can be pre-configured or alternatively discovered, for instance by using the approach described in [I-D.tiloca-core-oscore-discovery].

This document defines how to achieve the following objectives.

* OBJ1. The proxy gets an indication from the client that the client is in fact interested in multiple responses to a proxied group request and is capable to handle those. With particular reference to a unicast CoAP group request sent to the proxy, this means that the client is capable to receive those responses as separate CoAP responses, each matching with the original unicast request.

* OBJ2. The proxy learns for how long it should wait for responses to a proxied group request, before starting to ignore following responses to it (except for notifications, if a CoAP Observe Option is used [RFC7641]).

* OBJ3. The proxy relays to the client any multiple responses to the proxied group request. With particular reference to a client’s original CoAP unicast request sent to the proxy, those responses are sent to the client as separate CoAP responses, each matching with the original unicast request.

* OBJ4. The client is able to distinguish the different responses to the proxied group request, as well as their corresponding origin servers.

* OBJ5. The client is enabled to optionally contact one or more of the responding origin servers in the future, either directly or via the proxy.

5. Protocol Description

This section specifies the steps of the signaling protocol.

5.1. Request Sending at the Client

This section defines the operations performed by the client, for sending a request targeting a group of servers via the proxy.

5.1.1. Request Sending

The client proceeds according to the following steps.

1. The client prepares a unicast CoAP group request addressed to the proxy, and specifies the group URI where the request has to be forwarded to.

   The client can specify the group URI as a string in the Proxy-Uri Option, or by using the Proxy-Scheme Option together with the Uri-* options (see Section 3.5.1 of [I-D.ietf-core-groupcomm-bis]).

   Alternatively, the client can rely on the analogous options defined in [I-D.ietf-core-href], i.e., on the Proxy-Cri Option conveying a CRI equivalent to the group URI, or on the Proxy-Scheme-Number Option together with the Uri-* options.

2. The client MUST retain the Token value used for this original unicast request beyond the reception of a first CoAP response matching with it. To this end, the client follows the same rules for Token retention defined for multicast CoAP requests in Section 3.1.5 of [I-D.ietf-core-groupcomm-bis].

   In particular, the client picks an amount of time T that it is fine to wait for before freeing up the Token value. Specifically, the value of T MUST be such that:

   * $T < T_r$, where T_r is the amount of time that the client is fine to wait for before potentially reusing the Token value. Note that T_r MUST NOT be less than MIN_TOKEN_REUSE_TIME defined in Section 3.1.5 of [I-D.ietf-core-groupcomm-bis].

   * T should be at least the expected worst-case time taken by the request and response processing on the proxy and on the servers in the addressed CoAP group.

   * T should be at least the expected worst-case round-trip delay between the client and the proxy plus the worst-case round-trip delay between the proxy and any of the origin servers.

3. The client MUST include the Multicast-Timeout Option defined in Section 2 in the unicast request to send to the proxy. The option value specifies an amount of time T' < T. The difference (T - T') should be at least the expected worst-case round-trip time between the client and the proxy.

   The client can specify T' = 0 as option value, thus indicating to be not interested in receiving responses from the origin servers through the proxy. In such a case, the client SHOULD also include a No-Response Option [RFC7967] with value 26 (suppress all response codes), if it supports the option.

   Consistently, if the unicast request to send to the proxy already included a No-Response Option with value 26, the client SHOULD specify T' = 0 as value of the Multicast-Timeout Option.

4. The client processes the request as defined in [I-D.ietf-core-groupcomm-bis], and also as in [I-D.ietf-core-oscore-groupcomm] when secure group communication is used between the client and the servers.

5. The client sends the request to the proxy as a unicast CoAP message. When doing so, the client protects the request according to the security association that it has with the proxy.

The exact method that the client uses to estimate the worst-case processing times and round-trip delays mentioned above is out of the scope of this document. However, such a method is expected to be already used by the client when generally determining an appropriate Token lifetime and reuse interval.

5.1.2. Supporting Observe

When using CoAP Observe [RFC7641], the client follows what is specified in Section 3.7 of [I-D.ietf-core-groupcomm-bis], with the difference that it sends a unicast request to the proxy, to be forwarded to the group of servers as defined in Section 5.1.1 of this document.

Furthermore, the client especially follows what is specified in Section 5 of [RFC7641], i.e., it registers its interest to be an observer with the proxy, as if it was communicating with the servers.

5.1.3. Cancellation of Ongoing Response Forwarding

After having sent the unicast request to the proxy but before timeout expiration, the client might become not interested in receiving further corresponding responses, i.e., from that point in time until T seconds have elapsed since the request was sent to the proxy.

In such a case the client MAY ask the proxy for an early stop of the ongoing response forwarding, i.e., to not forward to the client any further response received to the forwarded group request from the servers.

To this end, the client can rely on one of the following two approaches.

* The client acts like it would at timeout expiration (see Section 5.5.1), i.e., it simply frees up its local Token value associated with the original unicast request sent to the proxy.

  Consequently, further responses forwarded by the proxy would result in the client sending a CoAP Reset message (RST), which the proxy would interpret as a loss of interest from the client.

  While this approach would work when using CoAP over UDP between the client and the proxy, it might not be suitable in case a different transport is used instead.

* The client sends to the proxy a new CoAP unicast request, namely Early Stop Request, such that:

  - It MUST use the method GET.

  - It MUST use the same Token value of the original unicast request sent to the proxy. This explicitly relates the present Early Stop Request to the original unicast request.

  - It MUST include the Multicast-Timeout Option, specifying 0 as option value. This explicitly indicates the client’s wish to stop receiving further responses to the original unicast request.

  - It MUST NOT include any of the following: the Proxy-Uri Option or the Proxy-Cri Option; the Proxy-Scheme Option or the Proxy-Scheme-Number Option, together with the Uri-* options. This explicitly indicates the proxy to not forward the Early Stop Request.

After sending the Early Stop Request, the client frees up its local Token value associated with the original unicast request sent to the proxy.

Note that, irrespective of the approach used by the client, freeing up the Token value does not make it eligible for possible reuse yet (see Section 5.1.1).

5.2. Request Processing at the Proxy

This section defines the operations performed by the proxy, when receiving a request to forward to a group of servers.

5.2.1. Request Processing

Upon receiving the request from the client, the proxy proceeds according to the following steps.

1. The proxy decrypts and verifies the request, according to the security association that it has with the client.

2. The proxy identifies the client, and verifies that the client is in fact allowed-listed to have its requests proxied to CoAP group URIs.

   If the verification fails, the proxy MUST stop processing the request and MUST reply to the client with a 4.01 (Unauthorized) response. The proxy protects the response according to the security association that it has with the client.

3. The proxy verifies the presence of the Multicast-Timeout Option, as a confirmation that the client is fine to receive multiple CoAP responses matching with the same original request.

   If the Multicast-Timeout Option is not present, the proxy MUST stop processing the request and MUST reply to the client with a 4.00 (Bad Request) response. The proxy protects the response according to the security association that it has with the client.

   The response MUST include a Multicast-Timeout Option, whose value MUST be set to 0. As per Section 3.2 of [RFC7252], this is represented with an empty option value (a zero-length sequence of bytes). By doing so, the proxy indicates that the Multicast-Timeout Option was missing and has to be included in the request. As per Section 5.9.2 of [RFC7252] The response SHOULD include a diagnostic payload.

4. The proxy retrieves the value T’ from the Multicast-Timeout Option, and then removes the option from the client’s request.

5. The proxy forwards the client’s request to the group of servers. In particular, the proxy sends it as a CoAP group request over IP multicast, addressed to the group URI specified by the client.

6. The proxy sets a timeout with the value T’ retrieved from the Multicast-Timeout Option of the original unicast request.

   In case T’ > 0, the proxy will ignore responses to the forwarded group request coming from servers, if received after the timeout expiration, with the exception of Observe notifications (see Section 5.4).

   In case T’ = 0, the proxy will ignore all responses to the forwarded group request coming from servers.

If the proxy supports caching of responses, it can serve the original unicast request also by using cached responses, as per Section 7.

5.2.2. Supporting Observe

When using CoAP Observe [RFC7641], the proxy takes the role of the client and registers its own interest to observe the target resource with the servers as per Section 5 of [RFC7641].

When doing so, the proxy especially follows what is specified for the client in Section 3.7 of [I-D.ietf-core-groupcomm-bis], by forwarding the group request to the servers over IP multicast as defined in Section 5.2.1 of this document.

5.2.3. Cancellation of Ongoing Response Forwarding

As defined in Section 5.1.3, the client might ask the proxy for an early stop of the ongoing response forwarding, i.e., to stop forwarding to the client any further responses received to the forwarded group request from the servers.

Consistently, the proxy stops forwarding such responses to the client, after receiving a CoAP Reset message (RST) in reply to one of such responses, or after receiving an Early Stop Request related to the ongoing response forwarding (i.e., using the same Token value of the original unicast request from the client).

After that, in case the proxy receives further responses to the forwarded group request from the servers, the proxy MUST NOT forward those responses to the client. In fact, the proxy can safely free up

its local Token value associated with that group request, which results in discarding any further responses to the same group request received from then on from the servers.

5.3. Request and Response Processing at the Server

This section defines the operations performed by the server, when receiving a group request from the proxy.

5.3.1. Request and Response Processing

Upon receiving the request from the proxy, the server proceeds according to the following steps.

1. The server processes the group request as defined in [I-D.ietf-core-groupcomm-bis], and also as in [I-D.ietf-core-oscore-groupcomm] when secure group communication is used between the client and the server.

2. The server processes the response to be relayed to the client as defined in [I-D.ietf-core-groupcomm-bis], and also as in [I-D.ietf-core-oscore-groupcomm] when secure group communication is used between the client and the server.

5.3.2. Supporting Observe

When using CoAP Observe [RFC7641], the server especially follows what is specified in Section 3.7 of [I-D.ietf-core-groupcomm-bis] and Section 5 of [RFC7641].

5.4. Response Processing at the Proxy

This section defines the operations performed by the proxy, when receiving a response matching with a forwarded group request.

5.4.1. Response Processing

Upon receiving a response matching with the group request before the amount of time T’ has elapsed (see Step 6 in Section 5.2.1), the proxy proceeds according to the following steps.

1. The proxy MUST include the Reply-From Option defined in Section 3 in the response. The proxy sets the option value as follows.

   The CRI present as first element of the CBOR sequence specifies the addressing information of the server generating the response. The second element of the CBOR sequence MUST NOT be present.

If the proxy supports caching of responses (see Section 7), the proxy MUST include the Reply-From Option in the response before caching the response. This ensures that a response to a group request conveys the addressing information of the origin server that generated the response, also when the response is forwarded to a client as retrieved from the proxy's cache.

2. The proxy forwards the response back to the client. When doing so, the proxy protects the response according to the security association that it has with the client.

As discussed in Section 3.1.6 of [I-D.ietf-core-groupcomm-bis], it is possible that a same server replies with multiple responses to the same group request, i.e., with the same Token. As long as the proxy forwards responses to a group request back to the origin client, the proxy MUST follow the steps defined above and forward also such multiple responses "as they come".

Upon timeout expiration, i.e., T' seconds after having sent the group request over IP multicast, the proxy frees up its local Token value associated with that request. Thus, following late responses to the same group request will be discarded and not forwarded back to the client.

5.4.2. Supporting Observe

When using CoAP Observe [RFC7641], the proxy acts as a client registered with the servers, as described earlier in Section 5.2.2.

Furthermore, the proxy takes the role of a server when forwarding notifications from origin servers back to the client. To this end, the proxy follows what is specified in Section 3.7 of [I-D.ietf-core-groupcomm-bis] and Section 5 of [RFC7641], with the following additions.

* At Step 1 in Section 5.4, the proxy includes the Reply-From Option in every notification, including non-2.xx notifications resulting in removing the proxy from the list of observers of the origin server.

* The proxy frees up its Token value used for a group observation only if, after the timeout expiration, no 2.xx (Success) responses matching with the group request and also including an Observe Option have been received from any origin server.

  Otherwise, after the timeout expiration and as long as observations are active with servers in the group for the target resource of the group request, notifications from those servers

are forwarded back to the client, as defined in Section 5.4, and the Token value used for the group observation is not freed during this time.

Finally, the proxy SHOULD regularly verify that the client is still interested in receiving observe notifications for a group observation. To this end, the proxy can rely on the same approach discussed for servers in Section 3.7 of [I-D.ietf-core-groupcomm-bis], with more details available in Section 4.5 of [RFC7641].

5.5. Response Processing at the Client

This section defines the operations performed by the client, when receiving a response matching with a request that targeted a group of servers via the proxy.

5.5.1. Response Processing

Upon receiving from the proxy a response matching with the original unicast request before the amount of time T has elapsed (see Step 2 in Section 5.1.1), the client proceeds according to the following steps.

1. The client processes the response as defined in [I-D.ietf-core-groupcomm-bis]. When doing so, the client decrypts and verifies the response according to the security association that it has with the proxy.

2. If secure group communication is used end-to-end between the client and the servers, the client processes the response resulting at the end of Step 1, as defined in [I-D.ietf-core-oscore-groupcomm].

3. The client retrieves the CRI from the value of the Reply-From Option, and identifies the origin server whose addressing information is specified by the CRI. This allows the client to distinguish different responses as generated by different origin servers.

   Optionally, the client may contact one or more of those servers individually, i.e., directly (bypassing the proxy) or instead indirectly (via a proxied unicast request). To this end, the client composes the correct URI for the individual request to the origin server, by using the information specified in the CRI retrieved from the Reply-From Option.

In order to individually reach the origin server again through the proxy, the client is not required to support the transport protocol indicated by 'scheme' in the CRI and used between the proxy and the origin server, in case the protocol is not CoAP over UDP (CRI scheme number: 0).

That is, the client simply specifies the URI for the individual request in the unicast request to the proxy. To this end, the client can specify the URI as a string in the Proxy-Uri Option, or by using the Proxy-Scheme Option together with the Uri-* options. Alternatively, the client can rely on the analogous options defined in [I-D.ietf-core-href], i.e., on the Proxy-Cri Option conveying a CRI equivalent to the URI, or on the Proxy-Scheme-Number Option together with the Uri-* options. In either case, the client uses the transport protocol that it supports, and has used before, to send the unicast request to the proxy.

As discussed in Section 3.1.6 of [I-D.ietf-core-groupcomm-bis], it is possible that the client receives multiple responses to the same group request, i.e., with the same Token, from the same origin server. The specific client implementation determines at which layer deduplication of responses is performed, or whether it is necessary in an application at all. If the processing of a response succeeds, then the client delivers the response to the application as usual. Depending on its available context information, the application itself can be in a good position to decide how to handle such responses.

Upon the timeout expiration, i.e., T seconds after having sent the original unicast request to the proxy, the client frees up its local Token value associated with that request. Note that, upon this timeout expiration, the Token value is not eligible for possible reuse yet (see Section 5.1.1). Thus, until the actual amount of time before enabling Token reusage has elapsed, following late responses to the same request forwarded by the proxy will be discarded, as these are not matching (by Token) with any active request from the client.

5.5.2. Supporting Observe

When using CoAP Observe [RFC7641], the client frees up its Token value only if, after the timeout T expiration, no 2.xx (Success) responses matching with the original unicast request and also including an Observe Option have been received.

Instead, if at least one such response has been received, the client continues receiving those notifications as they are 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.

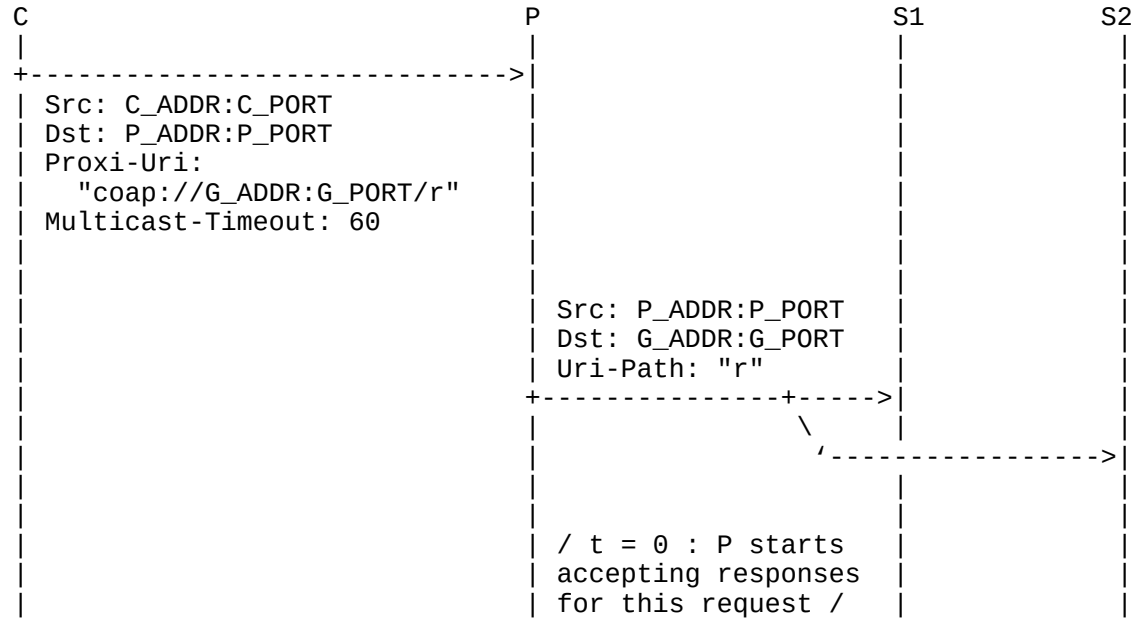
* One proxy P, with address P_ADDR and port number P_PORT.

* 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, cri'X' denotes a CRI corresponding to the URI X.



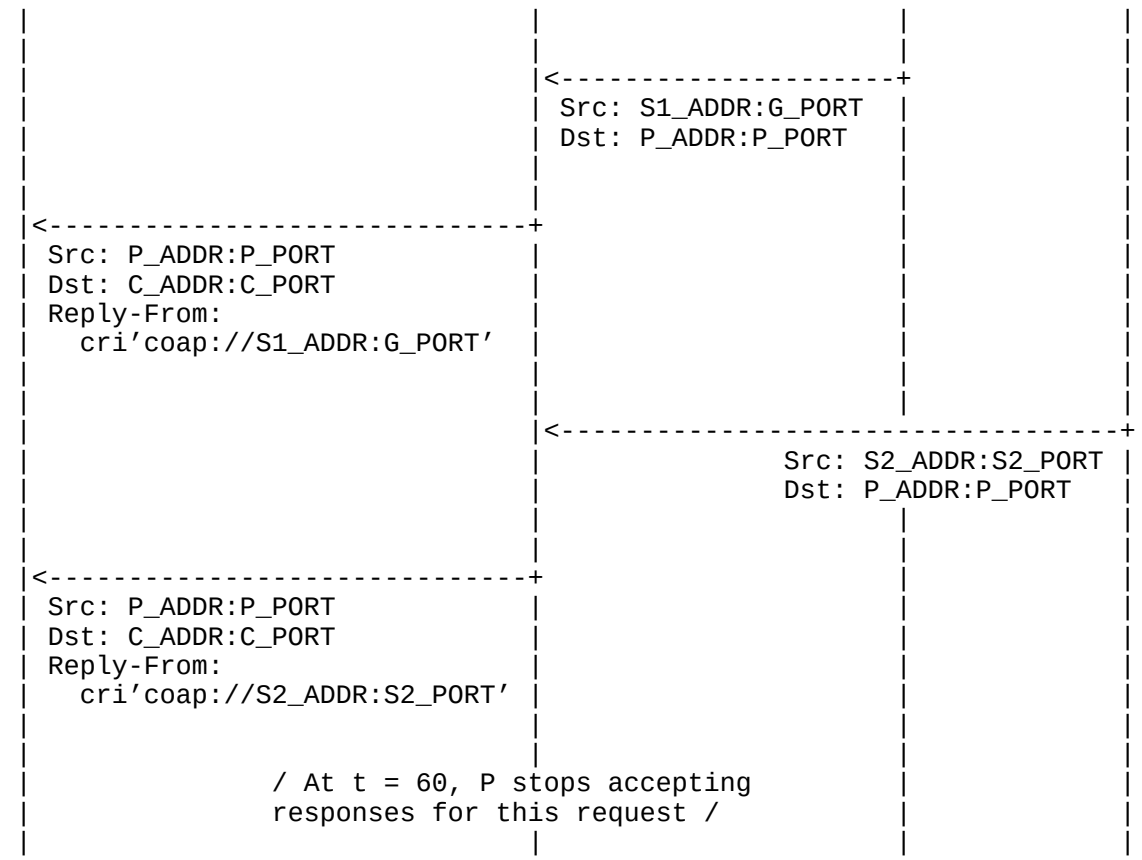


Figure 1: Workflow Example with a Forward-Proxy

6. Reverse-Proxies

The use of reverse-proxies in group communication scenarios is defined in Section 3.5.2 of [I-D.ietf-core-groupcomm-bis].

This section clarifies how the Multicast-Timeout Option is effective also in such a context, in order for:

* The proxy to effectively reveal itself as a reverse-proxy to the client.

* The client to indicate to the proxy of being aware that it is communicating with a reverse-proxy, and for how long it is willing to receive responses to a proxied group request.

This practically addresses the additional issues compared to the case with a forward-proxy, as compiled in Appendix F of [I-D.ietf-core-groupcomm-bis].

Appendix A provides examples with a reverse-proxy.

6.1. Processing on the Proxy Side

If the proxy receives a CoAP request and determines that it should be forwarded to a group of servers over IP multicast, then the proxy performs the steps defined in Section 5.2.

In particular, when such a request does not include a Multicast-Timeout Option, the proxy effectively reveals itself as a reverse-proxy, by replying with a 4.00 (Bad Request) response including a Multicast-Timeout Option with value 0 (which is ultimately represented with an empty option value).

The proxy processes the CoAP responses forwarded back to the client as defined in Section 5.4, with the following additions.

* As a first possible case, the proxy stands in both for the whole group of servers and for the individual origin servers in the group. That is, the origin client cannot reach the individual servers directly, but only through the proxy.

  In such a case, within a response forwarded back to the client, the value of the Reply-From Option specifies an addressing information TARGET that is directly associated with the proxy. The addressing information is such that, when receiving a unicast request that has been sent according to what is specified in TARGET, the proxy forwards the request to the origin server that generated the response. In particular, the proxy sets the option value as follows.

  - The CRI present as first element of the CBOR sequence specifies an addressing information TARGET_1, such that a unicast request reaches the proxy if it is sent according to TARGET_1.

  - A CRI reference MUST be present as second element of the CBOR sequence in case, upon receiving a unicast request that has been sent according to TARGET_1, the proxy forwards the request based on what is specified by the Uri-Host, Uri-Port, and Uri-Path Options included in the request. The CRI reference specifies the same information that the proxy expects to be specified in the Uri-Host, Uri-Port, and Uri-Path Options of such a unicast request.

Otherwise, the second element of the CBOR sequence MUST NOT be present, in which case the proxy forwards the unicast request solely based on the addressing information TARGET_1 according to which the request has been sent to.

The client will be able to communicate individually with the origin server that generated the response, by sending a follow-up unicast request to the proxy at the specified addressing information TARGET, according to which the proxy forwards the request to that server. This is further specified in Section 6.2. Examples are provided in Appendix A.1 and Appendix A.2.

* As a second possible case, the proxy stands in only for the whole group of servers, but not for the individual servers in the group. That is, the origin client can reach the individual servers directly, without recourse to the proxy.

  In such a case, within a response forwarded back to the client, the value of the Reply-From Option specifies an addressing information TARGET that is directly associated with the origin server that generated the response. In particular, the proxy sets the option value as follows.

  - The CRI present as first element of the CBOR sequence specifies the addressing information TARGET, such that a unicast request reaches the origin server if sent according to TARGET.

  - The second element of the CBOR sequence MUST NOT be present.

  The client will be able to use that information for sending a follow-up unicast request directly to that server, i.e., bypassing the proxy. This is further specified in Section 6.2. An example is provided in Appendix A.3.

6.2. Processing on the Client Side

If a client sends a CoAP request intended to a group of servers and is aware of actually communicating with a reverse-proxy, then the client MUST perform the steps defined in Section 5.1.1. In particular, this results in a request sent to the proxy including a Multicast-Timeout Option.

The client processes the CoAP responses forwarded back by the proxy as defined in Section 5.5, with the following differences at Step 3.

* If the client wishes to send a follow-up unicast request intended only to the origin server that generated the response, then the client sends such a request according to the addressing information specified by the CRI retrieved from the value of the Reply-From Option.

  Effectively, the client sends the unicast request either directly to the origin server (in case the proxy stands in only for the whole group of servers, but not for the individual servers in the group), or to the proxy (in case the proxy stands in for both the whole group of servers and the individual servers in the group).

  In case the value of the Reply-From Option specifies also a CRI reference as second element of the CBOR sequence, then the client includes the Uri-Host, Uri-Port, and Uri-Path Options in the unicast request, according to what is specified by the corresponding elements of the CRI reference. If the client wants to specify additional path segments that identify a specific resource at the origin server, then the corresponding Uri-Path Options are included in the request after the Uri-Path Options corresponding to the path component of the CRI reference.

7. Caching

A proxy MAY cache responses to a group request, as defined in Section 5.7.1 of [RFC7252]. In particular, the same rules apply to determine the set of request options used as "Cache-Key" and to determine the max-age values offered for responses served from the cache.

A cache entry is associated with one server and stores one response from that server, regardless of whether it is a response to a unicast request or to a group request. The following two types of requests can produce a hit to a cache entry.

* A matching request intended to that server, i.e., to the corresponding unicast URI.

  When the stored response is a response to a unicast request to the server, the unicast URI of the matching request is the same target URI used for the original unicast request.

  When the stored response is a response to a group request to the CoAP group, the unicast URI of the matching request is the target URI obtained by replacing the authority component of the group URI in the original group request with the transport-layer source address and port number of the response.

* A matching group request intended to the CoAP group, i.e., to the corresponding group URI.

  That is, a matching group request produces a hit to multiple cache entries, each of which associated with one of the CoAP servers currently member of the CoAP group.

  Note that, as per the freshness model defined in Section 7.1, the proxy might serve a group request exclusively from its cached responses only when it knows all the CoAP servers that are current members of the CoAP group and it has a valid cache entry for each of them.

When forwarding a GET or FETCH group request to the servers in the CoAP group, the proxy behaves like a CoAP client as defined in Section 3.2 of [I-D.ietf-core-groupcomm-bis], with the following additions.

* As discussed in Section 5.4.1, the proxy can receive multiple responses to the same group request from a same origin server, and forwards them back to the origin client "as they come". When this happens, each of such multiple responses is stored in the cache entry associated with the server "as it comes", possibly replacing an already stored response from that server.

* As discussed in Section 7.4, when communications in the group are secured with Group OSCORE [I-D.ietf-core-oscore-groupcomm], additional means are required to enable cacheability of responses at the proxy.

The following subsections define the freshness model and validation model that the proxy uses for cached responses.

7.1. Freshness Model

The proxy relies on the same freshness model defined in Section 3.2.1 of [I-D.ietf-core-groupcomm-bis], by taking the role of a CoAP client with respect to the servers in the CoAP group.

In particular, when receiving a unicast group request from the client, the proxy MAY serve it by using exclusively cached responses without forwarding the group request to the servers in the CoAP group, but only if both the following conditions hold.

* The proxy knows all the CoAP servers that are currently members of the CoAP group for which the group request is intended to.

* The proxy's cache currently stores a fresh response for each of those CoAP servers.

The specific way that the proxy uses to determine the CoAP servers currently members of the target CoAP group is out of scope for this document. As possible examples, the proxy can synchronize with a group manager server; rely on well-known time patterns used by the application or in the network for the addition of new CoAP group members; observe group join requests or IGMP/MLD multicast group join messages, e.g., if embedded in a multicast router.

When forwarding the group request to the servers, the proxy may have fresh responses stored in its cache for (some of) those servers. In such a case, the proxy uses (also) those cached responses to serve the original unicast group request, as defined below.

* The request processing in Section 5.2.1 is extended as follows.

  After setting the timeout with value T' > 0 in Step 6, the proxy checks whether its cache currently stores fresh responses to the group request. For each of such responses, the proxy compares the residual lifetime L of the corresponding cache entry against the value T'.

  If a cached response X is such that L < T', then the proxy forwards X back to the client at its earliest convenience. Otherwise, the proxy does not forward X back to the client right away, and rather waits for approaching the timeout expiration, as discussed in the next point.

* The response processing in Section 5.4.1 is extended as follows.

  Before the timeout with original value T' > 0 expires and the proxy stops accepting responses to the group request, the proxy checks whether it stores in its cache any fresh response X to the group request such that both the following conditions hold.

  - The cache entry E storing X was already existing when the proxy forwarded the group request.

  - The proxy has received no response to the forwarded group request from the server associated with E.

  Then, the proxy sends back to the client each response X stored in its cache and selected as above, before the timeout expires.

Note that, from the forwarding of the group request until the timeout expiration, the proxy still forwards responses to the group request back to the client "as they come" (see Section 5.4.1). Also, such responses possibly refresh older responses from the same servers that the proxy has stored in its cache, as defined earlier in Section 7.

7.2. Validation Model

This section defines the revalidation of responses, separately between the proxy and the origin servers, as well as between the origin client and the proxy.

7.2.1. Proxy-Servers Revalidation with Unicast Requests

The proxy MAY revalidate a cached response by making a GET or FETCH request on the related unicast request URI, i.e., by taking the role of a CoAP client with respect to a server in the CoAP group.

As discussed in Section 7.4, this is however not possible for the proxy if communications in the group are secured end-to-end between origin client and origin servers by using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

[TODO

It can be actually possible to enable revalidation of responses between proxy and server, also in this case where Group OSCORE is used end-to-end between client and origin servers.

Fundamentally, this requires to define the possible use of the ETag Option also as an outer option for OSCORE. Thus, in addition to the normal inner ETag, a server can add also an outer ETag option intended to the proxy.

Since validation of responses assumes that cacheability of responses is possible in the first place, it would be convenient to define the use of ETag as outer option in [I-D.amsuess-core-cachable-oscore].

In case OSCORE is also used between the proxy and an individual origin server as per [I-D.ietf-core-oscore-capable-proxies], then the outer ETag Option would be seamlessly protected with the OSCORE Security Context shared between the proxy and the origin server.

The following text can be used to replace the last paragraph above.

As discussed in Section 7.4, the following applies when Group OSCORE [I-D.ietf-core-oscore-groupcomm] is used to secure communications end-to-end between the origin client and the origin servers in the group.

* Additional means are required to enable cacheability of responses at the proxy (see Section 7.4.1).

* If a cached response included an outer ETag Option intended to the proxy, then the proxy can perform revalidation of the cached response, by making a request to the unicast URI targeting the server, and including outer ETag Option(s).

  This is possible also in case the proxy and the origin server use OSCORE to further protect the exchanged request and response, as defined in [I-D.ietf-core-oscore-capable-proxies]. In such a case, the originally outer ETag Option is protected with the OSCORE Security Context shared between the proxy and the origin server, before transferring the message over the communication leg between the proxy and origin server.

]

7.2.2. Proxy-Servers Revalidation with Group Requests

When forwarding a group request to the servers in the CoAP group, the proxy MAY revalidate one or more stored responses that it has cached.

To this end, the proxy relies on the same validation model defined in Section 3.2.2 of [I-D.ietf-core-groupcomm-bis] and using the ETag Option, by taking the role of a CoAP client with respect to the servers in the CoAP group.

As discussed in Section 7.4, this is however not possible for the proxy if communications in the group are secured end-to-end between origin client and origin servers by using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

[TODO

See the notes in Section 7.2.1.

The following text can be used to replace the last paragraph above.

As discussed in Section 7.4, the following applies when Group OSCORE [I-D.ietf-core-oscore-groupcomm] is used to secure communications end-to-end between the origin client and the origin servers in the group.

* Additional means are required to enable cacheability of responses at the proxy (see Section 7.4.1).

* If a cached response included an outer ETag Option intended to the proxy, then the proxy can perform revalidation of the cached response, by making a request to the group URI targeting the CoAP group, and including outer ETag Option(s).

  This is possible also in case the proxy and the origin servers use Group OSCORE to further protect the exchanged request and response, as defined in [I-D.ietf-core-oscore-capable-proxies]. In such a case, the originally outer ETag Option is protected with the Group OSCORE Security Context shared between the proxy and the origin server, before transferring the message over the communication leg between the proxy and origin server.

]

7.3. Client-Proxy Revalidation with Group Requests

A client MAY revalidate the full set of responses to a group request by leveraging the corresponding cache entries at the proxy. To this end, this document defines the new Group-ETag Option.

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

The option is intended for group requests sent to a proxy to be forwarded to the servers in a CoAP group, as well as for the associated responses.

No.	C	U	N	R	Name	Format	Length	Default
TBD3				x	Group-ETag	opaque	1-8	(none)

Table 3: The Group-ETag Option. C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

The Group-ETag Option has the same properties of the ETag Option defined in Section 5.10.6 of [RFC7252].

The Group-ETag Option is of class U in terms of OSCORE processing (see Section 4.1 of [RFC8613]).

A proxy MUST NOT provide this form of validation if it is not in a position to serve a group request by using exclusively cached responses, i.e., without sending the group request to the servers in the CoAP group (see Section 7.1).

If the proxy supports this form of response revalidation, the following applies.

* The proxy defines J as a joint set including all the cache entries currently storing fresh responses that satisfy a group request. A set J is "complete" if it includes a valid cache entry for each of the CoAP servers currently members of the CoAP group.

* When the set J becomes "complete", the proxy assigns it an entity-tag value. The proxy MUST update the current entity-tag value, when J is "complete" and one of its cache entry is updated.

* When forwarding to the client a 2.05 (Content) response to a GET or FETCH group request, the proxy MAY include one Group-ETag Option, in case the set J is "complete". Such a response MUST NOT include more than one Group-ETag Option. The option value specifies the entity-tag value currently associated with the set J.

When sending to the proxy a GET or FETCH request to be forwarded to the servers in the CoAP group, the client MAY include one or more Group-ETag Options. Each option specifies one entity-tag value, applicable to the set J of cache entries that can be hit by the group request.

The proxy MAY perform the following actions, in case the group request produces a hit to the cache entry of each CoAP server currently member of the CoAP group, i.e., in case the set J associated with the group request is "complete".

* The proxy checks whether the current entity-tag value of the set J matches with one of the entity-tag values specified in the Group-ETag Options of the unicast group request from the client.

* In case of positive match, the proxy replies with a single 2.03 (Valid) response. This response has no payload and MUST include one Group-ETag Option, specifying the current entity-tag value of the set J.

That is, the 2.03 (Valid) response from the proxy indicates to the client that the stored responses identified by the entity-tag given in the response's Group-ETag Option can be reused, after updating each of them as described in Section 5.9.1.3 of [RFC7252]. In effect, the client can determine if any of the stored representations from the respective cache entries at the proxy is current, without needing to transfer any of them again.

7.4. Caching of End-To-End Protected Responses at Proxies

When using Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect communications end-to-end between a client and multiple servers in the group, it is normally not possible for an intermediary proxy to effectively cache protected responses.

In fact, when starting from the same plain CoAP message, different clients generate different protected requests to send on the wire. This prevents different clients to generate potential cache hits, and thus makes response caching at the proxy pointless.

7.4.1. Deterministic Requests to Achieve Cacheability

For application scenarios that use secure group communication, it is still possible to achieve cacheability of responses at proxies by using the approach defined in [I-D.amsuess-core-cachable-oscore], which is based on Deterministic Requests protected with the pairwise mode of Group OSCORE. This approach is limited to group requests that are safe (in the RESTful sense) to process and do not yield side effects at the servers. As for any protected group request, it requires the clients and all the servers in the CoAP group to have already joined the correct OSCORE group.

Starting from the same plain CoAP request, this allows different clients in the OSCORE group to deterministically generate a same request protected with Group OSCORE, which is sent to the proxy for being forwarded to the CoAP group. The proxy can now effectively cache the resulting responses from the servers in the CoAP group, since the same plain CoAP request will result again in the same Deterministic Request and thus will produce a cache hit.

When caching of Group OSCORE secured responses is enabled at the proxy, the same as defined in Section 7 applies, with respect to cache entries and their lifetimes.

Note that different Deterministic Requests result in different cache entries at the proxy. This includes the case where different plain group requests differ only in their set of ETag Options, as defined in Section 3.2.2 of [I-D.ietf-core-groupcomm-bis].

That is, even though the servers would produce the same plain CoAP responses when replying to two different Deterministic Requests, those will result in different protected responses to each respective Deterministic Request, hence in different cache entries at the proxy.

Thus, given a plain group request, a client needs to reuse the same set of ETag Options, in order to send that group request as a Deterministic Request that can actually produce a cache hit at the proxy. However, while this would prevent the caching at the proxy from being inefficient and unnecessarily redundant, it would also limit the flexibility of end-to-end response revalidation for a client.

7.4.2. Validation of Responses

Response revalidation remains possible end-to-end between the client and the servers in the group, by including inner ETag Options as defined in Sections 3.2 and 3.2.2 of [I-D.ietf-core-groupcomm-bis].

Furthermore, it remains possible for a client to attempt revalidating responses to a group request from a "complete" set of cache entries at the proxy, by using the Group-ETag Option as defined in Section 7.3.

When directly interacting with the servers in the CoAP group to refresh its cache entries, the proxy cannot rely on response revalidation anymore. This applies to both the case where the request is addressed to a single server and sent to the related unicast URI (see Section 7.2.1) or instead is a group request addressed to the CoAP group and sent to the related group URI (see Section 7.2.2).

[TODO

See the notes in Section 7.2.1.

The following text can be used to replace the last paragraph above.

When directly interacting with the servers in the CoAP group to refresh its cache entries, the proxy also remains able to perform response revalidation. That is, if a cached response included an outer ETag Option intended to the proxy, then the proxy can perform revalidation of the cached response, by making a request to the unicast URI addressed to a single server and sent to the related unicast URI (see Section 7.2.1) or a group request addressed to the CoAP group and sent to the related group URI (see Section 7.2.2).

]

8. Chain of Proxies

A client may be interested to access a resource at a group of origin servers that is reached through a chain of two or more proxies.

That is, these proxies are configured into a chain, where each non-last proxy is configured to forward (group) requests to the next hop towards the origin servers. Also, each non-first proxy is configured to forward back responses to the previous hop proxy towards the origin client.

This section specifies how the signaling protocol defined in Section 5 is used in that setting. Except for the last proxy before the origin servers, every other proxy in the chain takes the role of client with respect to the next hop towards the origin servers. Also, every proxy in the chain except the first takes the role of server towards the previous proxy closer to the origin client.

Accordingly, possible caching of responses at each proxy works as defined in Section 7 and Section 7.4. Also, possible revalidation of responses cached at each proxy and based on the Group-ETag Option works as defined in Section 7.3 and Section 7.4.2.

The requirements REQ1 and REQ2 defined in Section 4 MUST be fulfilled for each proxy in the chain. That is, every proxy in the chain has to be explicitly configured with an allow-list that allows proxied group requests from specific senders, and MUST identify those senders upon receiving their group request. For the first proxy in the chain, that sender is the origin client. For each other proxy in the chain, that sender is the previous hop proxy closer to the origin client. In either case, a proxy can identify the sender of a group request by the same means mentioned in Section 4.

8.1. Request Processing at the Proxy

Upon receiving a group request to be forwarded to a CoAP group URI, a 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 5.2, 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 when the proxy is the first one in the chain, or to the previous hop proxy closer to the origin client otherwise.

* At Step 4, the proxy rather performs the following actions.

  1. The proxy retrieves the value T' from the Multicast-Timeout Option, and does not remove the option.

  2. In case T' > 0, the proxy picks an amount of time T that it is fine to wait for before freeing up its local Token value to use with the next hop towards the origin servers. To this end, the proxy MUST follow what is defined at Step 2 of Section 5.1.1 for the origin client, with the following differences.

     - T MUST be greater than the retrieved value T', i.e., T' < T.

     - The worst-case message processing time takes into account all the next hops towards the origin servers, as well as the origin servers themselves.

     - The worst-case round-trip delay takes into account all the legs between the proxy and the origin servers.

  3. In case T' > 0, the proxy replaces the value of the Multicast-Timeout Option with a new value T'', such that:

     - T'' < T. The difference (T - T'') should be at least the expected worst-case round-trip time between the proxy and the next hop towards the origin servers.

     - T'' < T'. The difference (T' - T'') should be at least the expected worst-case round-trip time between the proxy and the (previous hop proxy closer to the) origin client.

     If the proxy is not able to determine a value T'' that fulfills both the requirements above, the proxy MUST stop processing the request and MUST respond with a 5.05 (Proxying Not Supported) error response to the (previous hop proxy closer to the) origin client. The proxy SHOULD include a Multicast-Timeout Option, set to the minimum value T' that would be acceptable in the Multicast-Timeout Option of a group request to forward.

If the proxy is the first one in the chain, then the error response is sent to the origin client. Upon receiving the error response, the origin client MAY send an updated group request to the same first proxy in the chain. In the updated request, the Multicast-Timeout Option SHOULD specify a value T' such that: it is greater than the one specified in the original group request; and it is greater than or equal to the one specified in the error response (if present therein).

Otherwise, upon receiving the error response, any other proxy in the chain MAY send an updated group request to the next hop towards the origin servers. In the updated group request, the Multicast-Timeout Option MUST specify a value T' such that: it is greater than the one specified in the previous forwarded request; and it is greater than or equal to the one specified in the error response (if present therein). If the proxy does not send an updated group request, the proxy MUST also send a 5.05 (Proxying Not Supported) error response to the previous hop proxy closer to the origin client. Like the received one, also this error response SHOULD include a Multicast-Timeout Option, set to the minimum value T' acceptable by the proxy sending the error response.

* At Step 5, the proxy forwards the request to the next hop towards the origin servers.

* At Step 6, the proxy sets a timeout with the value T' retrieved from the Multicast-Timeout Option of the request received from the (previous hop proxy closer to the) origin client.

  In case T' > 0, the proxy will ignore responses to the forwarded group request coming from the next hop towards the origin servers, if received after the timeout expiration, with the exception of Observe notifications (see Section 5.4).

  In case T' = 0, the proxy will ignore all responses to the forwarded group request coming from the next hop towards the origin servers.

8.1.1. Supporting Observe

When using CoAP Observe [RFC7641], what is defined in Section 5.2.2 applies for the last proxy in the chain, i.e., the last hop before the origin servers.

Any other proxy in the chain acts as a client and registers its own interest to observe the target resource with the next hop towards the origin servers, as per Section 5 of [RFC7641].

8.1.2. Cancellation of Ongoing Response Forwarding

Consistently with what is described in Section 5.2.3, a proxy might be asked by the (previous hop proxy closer to the) origin client for an early stop of the ongoing response forwarding.

That is, the proxy is asked to stop forwarding to the (previous hop proxy closer to the) origin client any further responses received to the forwarded group request from the (next hop proxy towards the) origin servers.

When this happens, the proxy proceeds as described in Section 5.2.3. Furthermore, if the proxy is not the last one in the chain, the proxy MAY send to the next hop proxy towards the origin servers an Early Stop Request (see Section 5.2.1), with the same Token value of the group request that the proxy forwarded to the next hop proxy towards the origin servers.

8.2. Response Processing at the Proxy

Upon receiving a response matching with the group request before the amount of time T' has elapsed, the proxy proceeds as follows.

If the proxy is the last one in the chain, i.e., it is the last hop before the origin servers, the proxy performs the steps defined in Section 5.4 if it is a forward-proxy or in Section 6.1 if it is a reverse-proxy, with no modifications.

Otherwise, the proxy performs the steps defined in Section 5.4, with the following differences.

* In any of the two following cases, the proxy skips Step 1, hence the proxy MUST NOT remove, alter, or replace the Reply-From Option.

  - The chain is composed of forward-proxies.

  - The chain is composed of reverse-proxies, and the last reverse-proxy (in fact, the whole chain) stands in only for the whole group of servers, but not for the individual servers in the group (see Section 6.1).

  This ensures that, when receiving a response to a group request and consuming the Reply-From Option, the origin client can retrieve addressing information that is directly associated with the origin server that generated the response.

* At Step 1, the following applies in case the chain is composed of reverse-proxies, and the last reverse-proxy (in fact, the whole chain) stands in both for the whole group of servers and for the individual origin servers in the group (see Section 6.1).

  In the Reply-From Option, the proxy MUST replace the old value TARGET_OLD. The new value TARGET_NEW specifies addressing information directly associated with the proxy. The new value is such that, when receiving a unicast request that has been sent according to what is specified in TARGET_NEW, the proxy forwards the request according to what was specified in TARGET_OLD, i.e., to the next hop towards the origin server that generated the response.

  This ensures that, when receiving a response to a group request and consuming the Reply-From Option, the origin client can retrieve addressing information that is directly associated with the first reverse-proxy in the chain, i.e., with the next hop towards the origin server that generated the response.

* At Step 2, "client" refers to the origin client when the proxy is the first one in the chain, or to the previous hop proxy closer to the origin client otherwise.

As to the possible reception of multiple responses to the same group request from the same (next hop proxy towards the) origin server, the same as defined in Section 5.4.1 applies. That is, as long as the proxy forwards responses to a group request back to the (previous hop proxy closer to the) origin client, the proxy MUST follow the steps above and forward also such multiple responses "as they come".

Upon timeout expiration, i.e., T’ seconds after having forwarded the group request to the next hop towards the origin servers, the proxy frees up its local Token value associated with that request. Thus, following late responses to the same group request will be discarded and not forwarded back to the (previous hop proxy closer to the) origin client.

8.2.1. Supporting Observe

When using CoAP Observe [RFC7641], what is defined in Section 5.4.2 applies for the last proxy in the chain, i.e., the last hop before the origin servers.

As to any other proxy in the chain, the following applies.

* The proxy acts as a client registered with the next hop towards the origin servers, as described earlier in Section 8.1.1.

* The proxy takes the role of a server when forwarding notifications from the next hop towards the origin servers back to the (previous hop proxy closer to the) origin client, as per Section 5 of [RFC7641].

* The proxy frees up its Token value used for a group observation only if, after the timeout expiration, no 2.xx (Success) responses matching with the group request and also including an Observe Option have been received from the next hop towards the origin servers.

  Otherwise, after the timeout expiration and as long as the observation for the target resource of the group request is active with the next hop towards the origin servers in the group, notifications from that hop are forwarded back to the (previous hop proxy closer to the) origin client, as defined in Section 8.2.

* The proxy SHOULD regularly verify that the (previous hop proxy closer to the) origin client is still interested in receiving observe notifications for a group observation. To this end, the proxy can rely on the same approach defined in Section 4.5 of [RFC7641].

9. HTTP-to-CoAP Proxies

This section defines the components needed to use the signaling protocol specified in this document, when an HTTP client wishes to send a group request to the servers of a CoAP group via an HTTP-to-CoAP cross-proxy.

The following builds on the mapping of the CoAP request/response model to HTTP and vice versa as defined in Section 10 of [RFC7252], as well as on the additional details about the HTTP-to-CoAP mapping defined in [RFC8075].

Furthermore, the components defined in Section 11 of [RFC8613] are also used to map and transport OSCORE-protected messages over HTTP. This allows an HTTP client to use Group OSCORE end-to-end with the servers in the CoAP group.

9.1. The HTTP Multicast-Timeout Header Field

The HTTP Multicast-Timeout header field (see Section 11.2) is used for carrying the content otherwise specified in the CoAP Multicast-Timeout Option defined in Section 2.

Using the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] and including the core ABNF syntax rule DIGIT (decimal digits) defined by that specification, the HTTP Multicast-Timeout header field value is as follows.

```
Multicast-Timeout = *DIGIT
```

The empty header field is equivalent to the header field conveying the value 0.

When translating a CoAP message into an HTTP message, the HTTP Multicast-Timeout header field is set with the content of the CoAP Multicast-Timeout Option, or is left empty in case the option is empty.

When translating an HTTP message into a CoAP message, the CoAP Multicast-Timeout Option is set with the content of the HTTP Multicast-Timeout header field, or is left empty in case the header field is empty.

9.2. The HTTP Reply-From Header Field

The HTTP Reply-From header field (see Section 11.2) is used for carrying the content otherwise specified in the CoAP Reply-From Option defined in Section 3. Its use is intended only for HTTP responses.

Reply-From is a List Structured Header Field [RFC9651]. The List MUST be composed of exactly one or two members. Each member of the List MUST be a Byte Sequence Item. Any deviation from such format MUST cause the entire header field to be ignored.

The value of the header field specifies addressing information pertaining to the origin server that generated the CoAP response corresponding to the HTTP response. The client can use this information in order to send an individual request intended to that server.

When translating a CoAP message into an HTTP message, the value of the HTTP Reply-From header field is built as follows.

* The first Byte Sequence Item encodes the byte serialization of the first CBOR array of the CBOR sequence that is specified as value of the CoAP Reply-From Option.

* The second Byte Sequence Item encodes the byte serialization of the second CBOR array (if present) of the CBOR sequence that is specified as value of the CoAP Reply-From Option.

If the CBOR sequence in the CoAP Reply-From Option does not include the second CBOR array, then this Byte Sequence Item MUST NOT be included in the List of the HTTP Reply-From header field.

When translating an HTTP message into a CoAP message, the value of the CoAP Reply-From Option is built as follows.

* The first CBOR array of the CBOR sequence is obtained by decoding the first Byte Sequence Item in the List that is specified as value of the HTTP Reply-From header field.

* The second CBOR array of the CBOR sequence is obtained by decoding the second Byte Sequence Item (if present) in the List that is specified as value of the HTTP Reply-From header field.

  If the List of the HTTP Reply-From header field does not include the second Byte Sequence Item, then this second CBOR array MUST NOT be included in the CBOR sequence of the CoAP Reply-From Option.

9.3. The HTTP Group-ETag Header Field

The HTTP Group-ETag header field (see Section 11.2) is used for carrying the content otherwise specified in the CoAP Group-ETag Option defined in Section 7.3.

Group-ETag is a List Structured Header Field [RFC9651]. The List MUST be composed of one or more members in HTTP requests and by exactly one member in HTTP responses. Each member of the List MUST be a Byte Sequence Item. Any deviation from such format MUST cause the entire header field to be ignored.

The value of the header field specifies a set of entity-tag values, each of which is associated with a set of cache entries at the proxy that can be hit by a group request (see Section 7.3).

When translating a CoAP message into an HTTP message, the value of the HTTP Group-ETag header field is built as follows.

* When translating a CoAP request to an HTTP request, the List of the HTTP Group-ETag header field MUST include N members, where N is the number of CoAP Group-ETag Options in the CoAP request. The i-th member of the List encodes the value specified in the i-th CoAP Group-ETag Option in the CoAP request.

* When translating a CoAP response to an HTTP response, the List of the HTTP Group-ETag header field MUST include one member, which encodes the value specified in the CoAP Group-ETag Option in the CoAP response.

When translating an HTTP message into a CoAP message, the value of the CoAP Group-ETag Options is built as follows.

* When translating an HTTP request to a CoAP request, N CoAP Group-ETag Options are included in the CoAP request, where N is the number of members of the List of the HTTP Group-ETag header field. The value of the i-th CoAP Group-ETag Option is obtained by decoding the i-th member of the List of the HTTP Group-ETag header field.

* When translating an HTTP response to a CoAP response, one CoAP Group-ETag Option is included in the CoAP response. The value of the CoAP Group-ETag Option is obtained by decoding the only member of the List of the HTTP Group-ETag header field.

When sending to the HTTP-to-CoAP proxy an HTTP GET request to be translated into a CoAP GET request intended to the CoAP group, the client MAY include one HTTP Group-ETag header field in the request. The field value is a list of one or more members, each of which encodes one entity-tag value that is applicable to the set J of cache entries that can be hit by the request (see Section 7.3).

An HTTP-to-CoAP proxy that performs the form of validation defined in Section 7.3 proceeds like defined in Section 7.3 for a CoAP-to-CoAP proxy, with the following differences.

* When sending to the client an HTTP 200 (OK) response to an HTTP GET request that was translated into a CoAP GET request sent to the CoAP group, the proxy MAY include one HTTP Group-ETag header field in the response, in case the set J is "complete". The field value is a List composed of one member, which encodes the entity-tag value currently associated with the set J.

* When the HTTP-to-CoAP proxy receives an HTTP GET request to be translated into a CoAP GET request intended to the CoAP group and that includes an HTTP Group-ETag header field, the following applies.

  - As to the entity-tag values used to check for possible cache hits, the HTTP-to-CoAP proxy obtains those values by decoding the members of the List of the HTTP Group-ETag header field in the HTTP request.

- If the same conditions for which a CoAP-to-CoAP proxy would reply with a single CoAP 2.03 (Valid) response hold, then the HTTP-to-CoAP proxy replies with a single HTTP 304 (Not Modified) response. The response MUST include one HTTP Group-ETag header field whose value is a List composed of one member, which encodes the current entity-tag value of the set J.

An HTTP 304 (Not Modified) response from the HTTP-to-CoAP proxy indicates to the client that it is possible to reuse the stored responses identified by the entity-tag encoded by the only member of the List of the HTTP Group-ETag header field.

9.4. Request Sending at the Client

The client proceeds according to the following steps.

1. The client prepares an HTTP request to send to the proxy via IP unicast, and to be forwarded by the proxy to the targeted group of CoAP servers over IP multicast. With reference to Section 5 of [RFC8075], the request is addressed to a Hosting HTTP URI, such that the proxy can extract the Target CoAP URI as the group URI where to forward the request.

2. The client determines the amount of time T that it is fine to wait for a response to the request from the proxy. Then, the client determines the amount of time T' < T, where the difference (T - T') should be at least the expected worst-case round-trip time between the client and the proxy.

3. If Group OSCORE is used end-to-end between the client and the servers, the client translates the HTTP request into a CoAP request, as per [RFC8075]. Then, the client protects the resulting CoAP request by using Group OSCORE, as defined in [I-D.ietf-core-oscore-groupcomm]. Finally, the protected CoAP request is mapped to HTTP as defined in Section 11.2 of [RFC8613]. Later on, the resulting HTTP request MUST be sent in compliance with the rules in Section 11.1 of [RFC8613].

4. The client includes the HTTP Multicast-Timeout header field in the request, specifying T' as its value. The client can specify T' = 0, thus indicating to be not interested in receiving responses from the origin servers through the proxy.

5. If the client wishes to revalidate responses to a previous group request from the corresponding cache entries at the proxy (see Section 7.3), the client includes one or multiple HTTP Group-ETag header fields in the request (see Section 9.3), each specifying an entity-tag value like they would in a corresponding CoAP Group E-Tag Option.

6. The client sends the request to the proxy, as a unicast HTTP message. In particular, the client protects the request according to the security association that it has with the proxy.

9.5. Request Processing at the Proxy

The proxy translates the HTTP request to a CoAP request, as per [RFC8075]. The additional rules for HTTP messages with the HTTP Multicast-Timeout header field and HTTP Group-ETag header field are defined in Section 9.1 and Section 9.3, respectively.

Once translated the HTTP request into a CoAP request, the proxy MUST perform the steps defined in Section 5.2. If the proxy supports caching of responses, it can serve the unicast request also by using cached responses as per Section 7, considering the CoAP request above as the potentially matching request.

In addition, in case the HTTP Multicast-Timeout header field had value 0, the proxy replies to the client with an HTTP response with status code 204 (No Content), right after forwarding the group request to the group of servers.

9.6. Response Processing at the Proxy

Upon receiving a CoAP response matching with the group request before the amount of time T’ > 0 has elapsed, the proxy includes the Reply-From Option in the response, as per Step 1 of Section 5.4.1. Then, the proxy translates the CoAP response to an HTTP response, as per Section 10.1 of [RFC7252] and [RFC8075], as well as Section 11.2 of [RFC8613] if Group OSCORE is used end-to-end between the client and servers. The additional rules for CoAP messages specifying the Reply-From Option are defined in Section 9.2.

After that, the proxy stores the resulting HTTP response until the timeout with original value T’ > 0 expires. If, before then, the proxy receives another response to the same group request from the same CoAP server, the proxy performs the steps above, and stores the resulting HTTP response by superseding the currently stored one from that server.

When the timeout expires, if no responses have been received from the servers, the proxy replies to the client's original unicast group request with an HTTP response with status code 204 (No Content).

Otherwise, the proxy relays to the client all the collected and stored HTTP responses to the group request, according to the following steps.

1. The proxy prepares a single HTTP batch response, which MUST have 200 (OK) status code and MUST have its HTTP Content-Type header field with value multipart/mixed [RFC2046].

2. For each stored individual HTTP response RESP, the proxy prepares a corresponding batch part to include in the HTTP batch response, such that:

   * The batch part has its own HTTP Content-Type header field with value application/http [RFC9112].

   * The body of the batch part is the individual HTTP response RESP, including its status code, headers, and body.

3. The proxy includes each batch part prepared at Step 2 in the HTTP batch response.

4. The proxy replies to the client's original unicast group request, by sending the HTTP batch response. When doing so, the proxy protects the response according to the security association that it has with the client.

9.7. Response Processing at the Client

When it receives an HTTP response as a reply to the original unicast group request, the client proceeds as follows.

1. The client decrypts and verifies the response, according to the security association that it has with the proxy.

2. From the resulting HTTP batch response, the client extracts the different batch parts.

3. From each of the extracted batch parts, the client extracts the body as one of the individual HTTP response RESP.

4. For each individual HTTP response RESP, the client performs the following steps.

* If Group OSCORE is used end-to-end between the client and servers, the client translates the HTTP response RESP into a CoAP response, as per Section 11.3 of [RFC8613]. Then, the client decrypts and verifies the resulting CoAP response by using Group OSCORE, as defined in [I-D.ietf-core-oscore-groupcomm]. Finally, the decrypted CoAP response is mapped to HTTP as per Section 10.2 of [RFC7252] as well as [RFC8075]. The additional rules for HTTP messages with the HTTP Reply-From header field are defined in Section 9.2.

* The client delivers to the application the individual HTTP response.

Similarly to Step 3 in Section 5.5.1, the client identifies the origin server that originated the CoAP response corresponding to the HTTP response RESP, by means of the addressing information specified as value of the HTTP Reply-From header field. This allows the client to distinguish different individual HTTP responses as corresponding to different CoAP responses from the servers in the CoAP group.

9.8. Example

The examples in this section build on Section 5.6, with the difference that the origin client C is an HTTP client and the proxy P is an HTTP-to-CoAP cross-proxy. The examples are simply illustrative and are not to be intended as a test vector.

The following is an example of unicast group request sent by C to P. The URI mapping and notation are based on the "Simple Form" defined in Section 5.4.1 of [RFC8075].

```
POST https://proxy.url/hc/?target_uri=coap://G_ADDR:G_PORT/ HTTP/1.1
Content-Length: <REQUEST_TOTAL_CONTENT_LENGTH>
Content-Type: text/plain
Multicast-Timeout: 60

Body: Do that!
```

The following is an example of HTTP batch response sent by P to C, as a reply to the client’s original unicast group request

For readability, base64url(cri’X’) denotes the base64url encoding of cri’X’ without padding (see Section 5 of [RFC4648]), and cri’X’ denotes the byte serialization of a CRI corresponding to the URI X.

```
HTTP/1.1 200 OK
Content-Length: <BATCH_RESPONSE_TOTAL_CONTENT_LENGTH>
Content-Type: multipart/mixed; boundary=batch_foo_bar

--batch_foo_bar
Content-Type: application/http

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: <INDIVIDUAL_RESPONSE_1_CONTENT_LENGTH>
Reply-From: base64url(cri'coap://S1_ADDR:G_PORT')

Body: Done!
--batch_foo_bar
Content-Type: application/http

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: <INDIVIDUAL_RESPONSE_2_CONTENT_LENGTH>
Reply-From: base64url(cri'coap://S2_ADDR:S2_PORT')

Body: More than done!
--batch_foo_bar--
```

9.9. Streamed Delivery of Responses to the Client

[TODO

The proxy might still be able to forward back individual responses to the client in a streamed fashion.

Individual responses can be forwarded back one by one as they come (like a CoAP-to-CoAP proxy does), or as soon as a certain amount of them has been received from the servers.

This can be achieved by combining the Content-Type multipart/mixed used in the previous sections with the Transfer-Coding "chunked" specified in RFC 9112.

The above applies to HTTP 1.1, while HTTP/2 has its own mechanisms for data streaming.

]

9.10. Reverse-Proxies

In case an HTTP-to-CoAP proxy acts specifically as a reverse-proxy, the same principles defined in Section 6 apply, as specified below.

9.10.1. Processing on the Client Side

If an HTTP client sends a request intended to a group of servers and is aware of actually communicating with a reverse-proxy, then the client MUST perform the steps defined in Section 9.4. In particular, this results in a request sent to the proxy including a Multicast-Timeout header field.

The client processes the HTTP response forwarded back by the proxy as defined in Section 9.7. If the client wishes to send a follow-up unicast request intended only to one of the CoAP servers that generated the response, the same concepts defined in Section 6.2 apply to the composition of HTTP requests.

9.10.2. Processing on the Proxy Side

If the proxy receives a request and determines that the request should be forwarded to a group of servers over IP multicast, then the same as defined in Section 9.5 applies, with the following difference.

* Once translated the HTTP request into a CoAP request, the proxy performs what is defined in Section 6.1.

The proxy processes the HTTP response sent to the client as defined in Section 9.6.

10. Security Considerations

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

When a chain of proxies is used (see Section 8), the secure communication between any two adjacent hops is independent of that between any other two adjacent hops.

When Group OSCORE is used for end-to-end secure group communication between the origin client and the origin servers, this security association is unaffected by the possible presence of a proxy or a chain of proxies.

Furthermore, the following additional considerations hold.

10.1. Client Authentication

As per the requirement REQ2 (see Section 4), the client has to authenticate to the proxy when sending a group request to forward. This leverages an established security association between the client and the proxy, which the client uses to protect the group request before sending it to the proxy.

If the group request is also protected end-to-end between the client and the origin servers using the group mode of Group OSCORE, the proxy can act as external signature checker (see Section 7.5 of [I-D.ietf-core-oscore-groupcomm]) and authenticate the client by successfully verifying the signature embedded in the group request. However, this requires the proxy to store, for each client to authenticate, the authentication credential that the client uses in the OSCORE group and the public key included therein, and to also store the authentication credential of the Group Manager responsible for the OSCORE group. This in turn would require a form of active synchronization between the proxy and the Group Manager for that group [I-D.ietf-core-oscore-groupcomm].

Nevertheless, the client and the proxy SHOULD still rely on a full-fledged pairwise secure association. In addition to ensuring the integrity of group requests sent to the proxy (see Section 10.2, Section 10.3, and Section 10.4), this prevents the proxy from forwarding replayed group requests with a valid signature, as possibly injected by an active, on-path adversary.

The same considerations apply when a chain of proxies is used (see Section 8), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

10.2. Multicast-Timeout Option

The Multicast-Timeout Option is of class U for OSCORE [RFC8613]. Hence, also when Group OSCORE is used between the client and the servers [I-D.ietf-core-oscore-groupcomm], a proxy is able to access the option value and retrieve the timeout value T', as well as to remove the option altogether before forwarding the group request to the servers. When a chain of proxies is used (see Section 8), this also allows each proxy but the last one in the chain to update the option value, as an indication for the next hop towards the origin servers (see Section 8.1).

The security association between the client and the proxy MUST provide message integrity, so that further intermediaries between the two as well as on-path active adversaries are not able to undetectably remove the option or alter its content, before the group request reaches the proxy.

Removing the option would result in not forwarding the group request to the servers. Altering the option content would result in the proxy accepting and forwarding back responses for an amount of time different from the one actually indicated by the client.

The security association between the client and the proxy SHOULD also provide message confidentiality. Otherwise, any further intermediaries between the two as well as any on-path passive adversaries would be able to 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 by an on-path active adversary to perform a more efficient, selective suppression of responses from the servers.

When the client protects the unicast request sent to the proxy using OSCORE (see [I-D.ietf-core-oscore-capable-proxies]) and/or (D)TLS, both message integrity and message confidentiality are achieved in the leg between the client and the proxy.

The same considerations above about security associations apply when a chain of proxies is used (see Section 8), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

10.3. Reply-From Option

The Reply-From 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.

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 undetectably remove the option from a forwarded server response or alter its content. This ensures that the client can correctly distinguish the different responses and identify the corresponding origin servers.

The security association between the client and the proxy SHOULD also provide message confidentiality. Otherwise, any further intermediaries between the two as well as any on-path passive adversaries would be able to access the option content, and thus learn the addressing information of servers in the group. This may in turn be used by an on-path active adversary to perform a more efficient, selective suppression of follow-up requests that the client sends to a specific server, either directly or instead indirectly via the proxy.

When the proxy protects the response forwarded back to the client using OSCORE (see [I-D.ietf-core-oscore-capable-proxies]) and/or (D)TLS, both message integrity and message confidentiality are achieved in the leg between the client and the proxy.

The same considerations above about security associations apply when a chain of proxies is used (see Section 8), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

10.4. Group-ETag Option

The Group-ETag Option is of class U for OSCORE [RFC8613]. Hence, also when Group OSCORE is used between the client and the servers [I-D.ietf-core-oscore-groupcomm], a proxy is able to access the option value and use it to possibly perform response revalidation at its cache entries associated with the servers in the CoAP group, as well as to remove the option altogether before forwarding the group request to the servers. When a chain of proxies is used (see Section 8), this also allows each proxy but the last one in the chain to update the option value, to possibly ask the next hop towards the origin servers to perform response revalidation at its cache entries.

The security association between the client and the proxy MUST provide message integrity, so that further intermediaries between the two as well as on-path active adversaries are not able to undetectably remove the option or alter its content, before the group request reaches the proxy.

Removing the option would result in the proxy not performing response revalidation at its cache entries associated with the servers in the CoAP group, even though that was what the client asked for.

Altering the option content in a group request would result in the proxy performing response revalidation based on different entity-tag values from those actually specified by the client. Consequently, the proxy would erroneously reply with multiple 2.05 (Content) responses conveying the full resource representations from its cache

entries instead of with a single 2.03 (Valid) response, or vice versa. Instead, altering the option content in a 2.03 (Valid) or 2.05 (Content) response would result in the client wrongly believing that the already stored or the just received representation, respectively, is also the current one, as per the entity value of the tampered Group-ETag Option.

The security association between the client and the proxy SHOULD also provide message confidentiality. Otherwise, any further intermediaries between the two as well as any on-path passive adversaries would be able to access the option content, and thus learn the rate and pattern according to which the group resource in question changes over time, as inferable from the entity values read over time.

When the client protects the unicast request sent to the proxy using OSCORE (see [I-D.ietf-core-oscore-capable-proxies]) and/or (D)TLS, both message integrity and message confidentiality are achieved in the leg between the client and the proxy.

The same considerations above about security associations apply when a chain of proxies is used (see Section 8), with each proxy but the last one in the chain acting as client with the next hop towards the origin servers.

When caching of Group OSCORE secured responses is enabled at the proxy, the same as defined in Section 7 applies, with respect to cache entries and the way they are maintained.

10.5. HTTP-to-CoAP Proxies

Consistently with what is discussed in Section 10.1, an HTTP client has to authenticate to the HTTP-to-CoAP proxy, and they SHOULD rely on a full-fledged pairwise secure association. This can rely on a TLS [RFC8446] channel as also recommended in Section 12.1 of [RFC8613] for when OSCORE is used with HTTP, or on a pairwise OSCORE Security Context shared between the client and the proxy as defined in [I-D.ietf-core-oscore-capable-proxies].

[TODO

Revisit security considerations from [RFC8075]

]

11. IANA Considerations

This document has the following actions for IANA.

Note to RFC Editor: Please replace all occurrences of "[RFC-XXXX]" with the RFC number of this specification and delete this paragraph.

11.1. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Number	Name	Reference
TBD1	Multicast-Timeout	[RFC-XXXX]
TBD2	Reply-From	[RFC-XXXX]
TBD3	Group-ETag	[RFC-XXXX]

Table 4: Registrations in the CoAP Option Numbers Registry

Note to RFC Editor: Please replace "TBD1", "TBD2", and "TBD3" in Table 4 with the assigned option numbers. Then please delete this paragraph and the following text within the present Section 11.1.

For all the three requested options, it is preferred to assign an option number from the value range 0-255.

11.2. Hypertext Transfer Protocol (HTTP) Field Name Registry

IANA is asked to enter the following HTTP header fields to the "Hypertext Transfer Protocol (HTTP) Field Name" registry.

Field Name	Status	Structured Type	Reference
Multicast-Timeout	permanent		[RFC-XXXX]
Reply-From	permanent	List	[RFC-XXXX]
Group-ETag	permanent	List	[RFC-XXXX]

Table 5: Registrations in the Hypertext Transfer Protocol (HTTP) Field Name Registry

12. References

12.1. Normative References

[I-D.ietf-core-groupcomm-bis]
Dijk, E. and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-13, 24 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-bis-13>.

[I-D.ietf-core-href]
Bormann, C. and H. Birkholz, "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-18, 3 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-href-18>.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and R. H旦glund, "Group Object Security for Constrained RESTful Environments (Group OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-24, 8 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-24>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/rfc/rfc2046>.

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

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

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/rfc/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/rfc/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/rfc/rfc7967>.

[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/rfc/rfc8075>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/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/rfc/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/rfc/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/rfc/rfc8613>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/rfc/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/rfc/rfc8949>.

[RFC9112] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/rfc/rfc9112>.

[RFC9651] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 9651, DOI 10.17487/RFC9651, September 2024, <https://www.rfc-editor.org/rfc/rfc9651>.

12.2. Informative References

[I-D.amsuess-core-cachable-oscore]
Ams端ss, C. and M. Tiloca, "Cacheable OSCORE", Work in Progress, Internet-Draft, draft-amsuess-core-cachable-oscore-10, 8 January 2025, <https://datatracker.ietf.org/doc/html/draft-amsuess-core-cachable-oscore-10>.

[I-D.bormann-coap-misc]
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", Work in Progress, Internet-Draft, draft-bormann-coap-misc-27, 14 November 2014, <https://datatracker.ietf.org/doc/html/draft-bormann-coap-misc-27>.

[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-16, 6 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-oscore-16>.

[I-D.ietf-core-oscore-capable-proxies]
Tiloca, M. and R. H旦glund, "OSCORE-capable Proxies", Work in Progress, Internet-Draft, draft-ietf-core-oscore-capable-proxies-03, 21 October 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-capable-proxies-03>.

[I-D.tiloca-core-oscore-discovery]
Tiloca, M., Ams端ss, C., and P. Van der Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-16, 4 September 2024, <https://datatracker.ietf.org/doc/html/draft-tiloca-core-oscore-discovery-16>.

[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/rfc/rfc8446>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

Appendix A. Examples with Reverse-Proxy

The examples in this section refer to the following actors.

* One origin client C, with address C_ADDR and port number C_PORT.

* One proxy P, with address P_ADDR and server port number P_PORT.

* Two origin servers S1 and S2, where the server Sx has address Sx_ADDR and port number Sx_PORT.

The origin servers are members of a CoAP group with IP multicast address G_ADDR and port number G_PORT. Also, the origin servers are members of a same application group, and share the same resource /r.

The communication between C and P is based on CoAP over TCP, as per [RFC8323]. The group communication between P and the origin servers is based on CoAP over UDP and IP multicast, as per [I-D.ietf-core-groupcomm-bis].

Finally, cri'X' denotes a CRI or CRI reference corresponding to the URI or URI reference X.

A.1. Example 1

The example shown in Figure 2 considers a reverse-proxy P that provides access to both the whole group of servers {S1,S2} and also to each of those servers individually. The client C may not have a way to reach the servers directly (e.g., P is acting as a firewall).

After the client C has received two responses to its group request sent via the proxy, it selects one server (S1) and requests another resource from it in unicast, again via the proxy.

In particular:

* In its group request to P, the client C includes the Uri-Host Option with value "group1.com" and the Uri-Path Option with value "r".

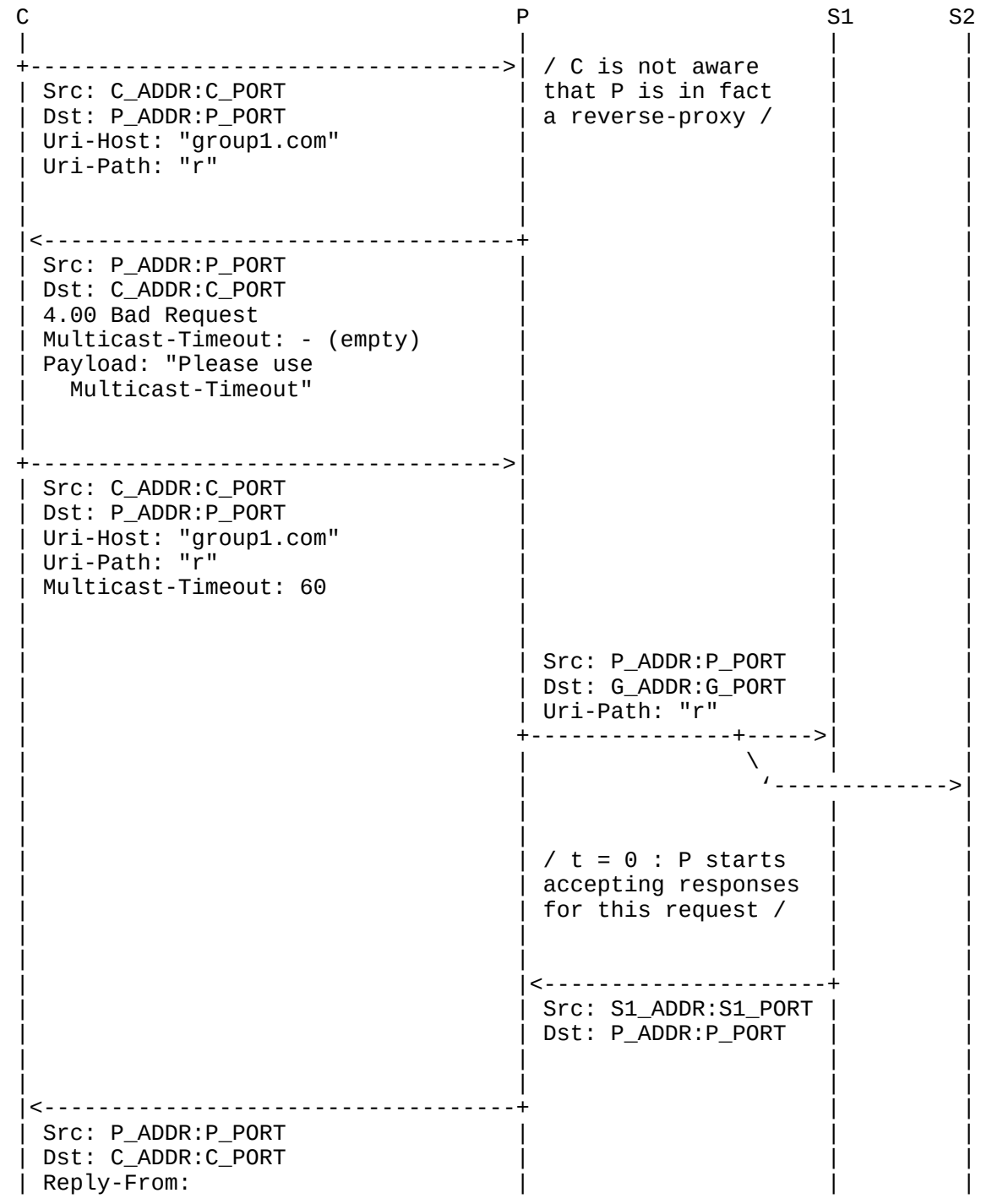
* The hostname 'group1.com' resolves to the IPv6 multicast address G_ADDR. The proxy P performs this resolution upon receiving the group request from C.

  Since such a request does not include the Uri-Port Option, P infers G_PORT to be the default port number 5683 for the "coap" URI scheme.

  Based on this information, P composes the group request and sends it to the CoAP group at G_ADDR:G_PORT.

* Typically, S1_PORT and S2_PORT will be equal to G_PORT, but a server Sx is allowed to reply to the multicast request from another port number not equal to G_PORT. For this reason, the notation Sx_PORT is used.

Note that this type of reverse-proxy only requires one unicast IP address (P_ADDR) for the proxy, so it scales well with a large number of servers Sx. Instead, the type of reverse-proxy in the example in Appendix A.2 requires one IP address for each server Sx and one for each CoAP group that the proxy supports.


C
P
S1
S2
/ C is not aware that P is in fact a reverse-proxy /
Src: C_ADDR:C_PORT
Dst: P_ADDR:P_PORT
Uri-Host: "group1.com"
Uri-Path: "r"
Src: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT
4.00 Bad Request
Multicast-Timeout: - (empty)
Payload: "Please use Multicast-Timeout"
Src: C_ADDR:C_PORT
Dst: P_ADDR:P_PORT
Uri-Host: "group1.com"
Uri-Path: "r"
Multicast-Timeout: 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:S1_PORT
Dst: P_ADDR:P_PORT
Src: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT
Reply-From:

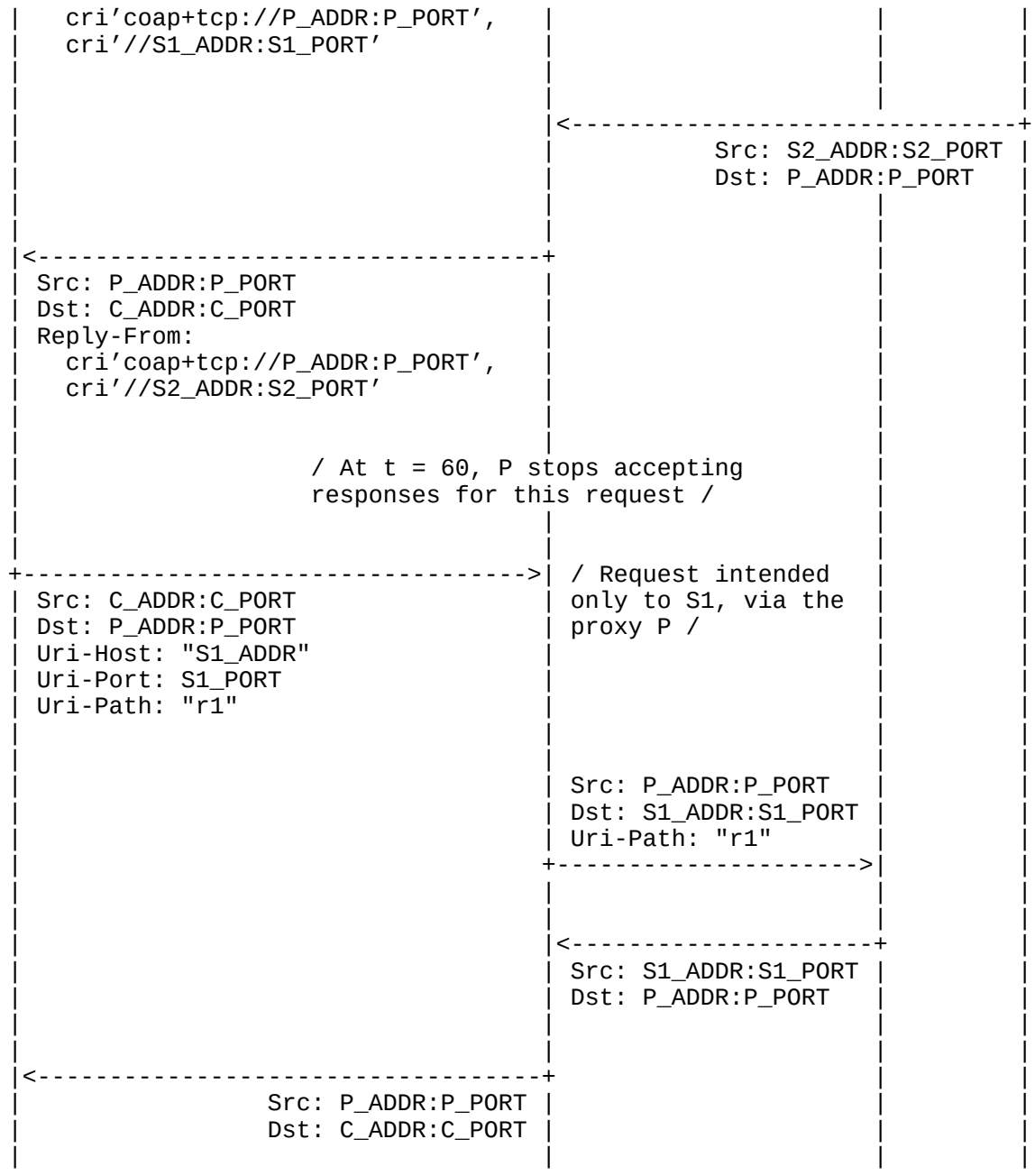

cri'coap+tcp://P_ADDR:P_PORT',
cri'//S1_ADDR:S1_PORT'
Src: S2_ADDR:S2_PORT
Dst: P_ADDR:P_PORT
Src: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT
Reply-From:
cri'coap+tcp://P_ADDR:P_PORT',
cri'//S2_ADDR:S2_PORT'
/ At t = 60, P stops accepting responses for this request /
Src: C_ADDR:C_PORT
Dst: P_ADDR:P_PORT
Uri-Host: "S1_ADDR"
Uri-Port: S1_PORT
Uri-Path: "r1"
/ Request intended only to S1, via the proxy P /
Src: P_ADDR:P_PORT
Dst: S1_ADDR:S1_PORT
Uri-Path: "r1"
Src: S1_ADDR:S1_PORT
Dst: P_ADDR:P_PORT
Src: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT

Figure 2: Workflow Example with a Reverse-Proxy Standing in for Both the Whole Group of Servers and Each Individual Server. This Requires the Proxy to Have Only One Pair (IP Address, Port Number).

A.2. Example 2

The example shown in Figure 3 considers a reverse-proxy that stands in for both the whole group of servers {S1,S2} and for each of those servers Sx. The client C may not have a way to reach the servers directly (e.g., P is acting as a firewall).

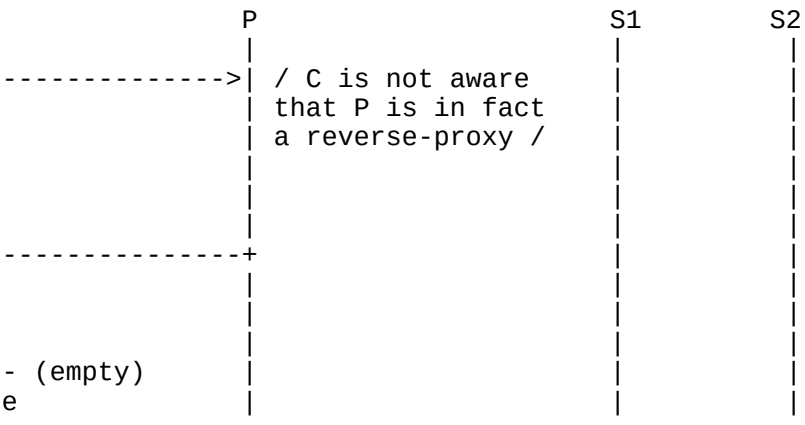
After the client C has received two responses to its group request sent via the proxy, it selects one server (S1) and requests at a later time the same resource from it in unicast, again via the proxy.

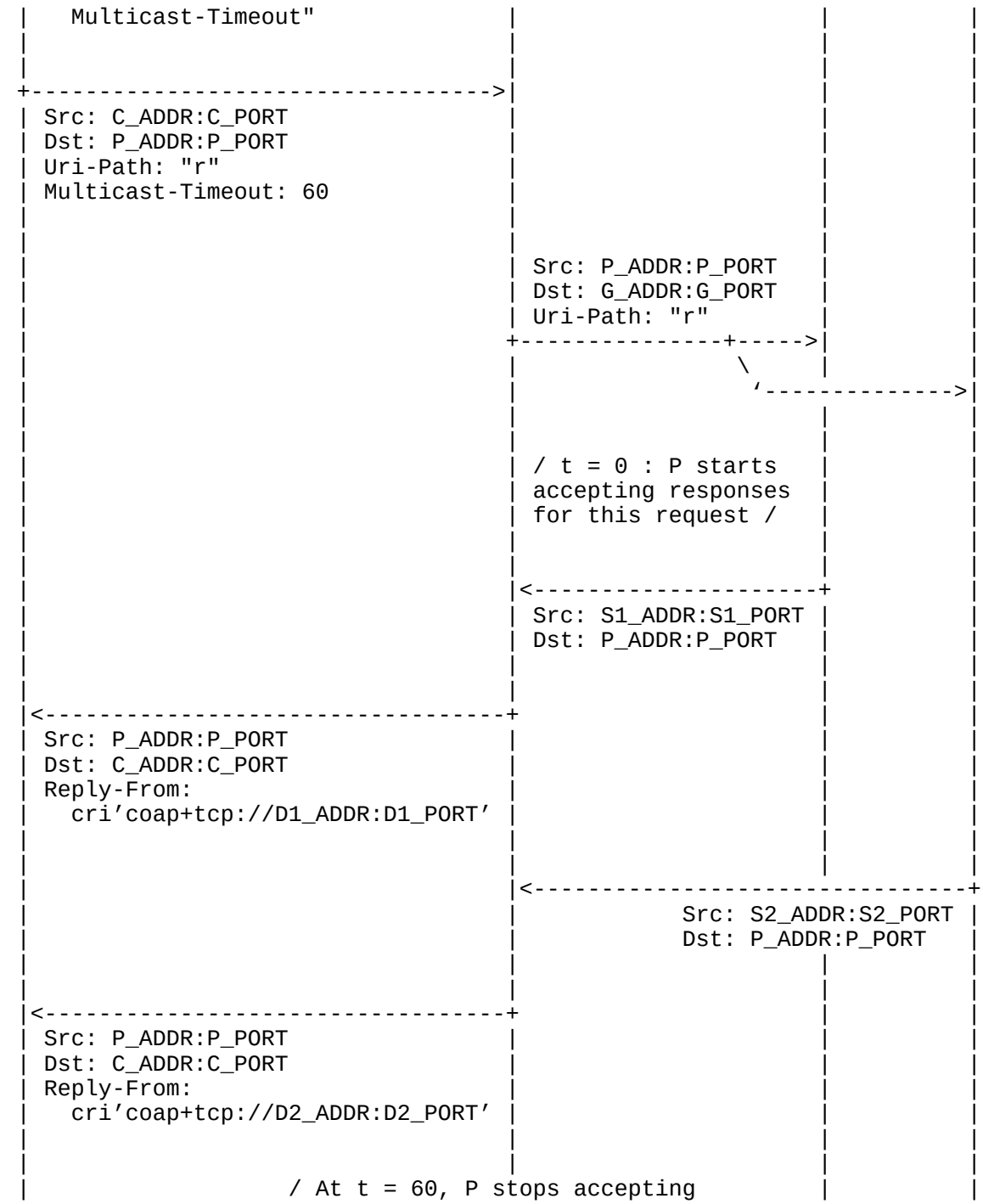
In particular:

* When receiving a request addressed to the unicast address P_ADDR and port number P_PORT, the proxy forwards the request towards the CoAP group at G_ADDR:G_PORT leaving the URI path unchanged.

* The address Dx_ADDR and port number Dx_PORT are also used by the proxy, which forwards an incoming request to that address towards the server at Sx_ADDR:Sx_PORT. The different Dx_ADDR are effectively "proxy IP addresses" used to provide access to the servers.

Note that this type of reverse-proxy implementation requires the proxy to use (potentially) a large number of distinct IP addresses, hence it is not very scalable. Instead, the type of reverse-proxy shown in the example in Appendix A.1 uses only one IPv6 unicast address to provide access to all servers and all CoAP groups.




Multicast-Timeout"
Src: C_ADDR:C_PORT
Dst: P_ADDR:P_PORT
Uri-Path: "r"
Multicast-Timeout: 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:S1_PORT
Dst: P_ADDR:P_PORT
Src: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT
Reply-From:
cri'coap+tcp://D1_ADDR:D1_PORT'
Src: S2_ADDR:S2_PORT
Dst: P_ADDR:P_PORT
Src: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT
Reply-From:
cri'coap+tcp://D2_ADDR:D2_PORT'
/ At t = 60, P stops accepting

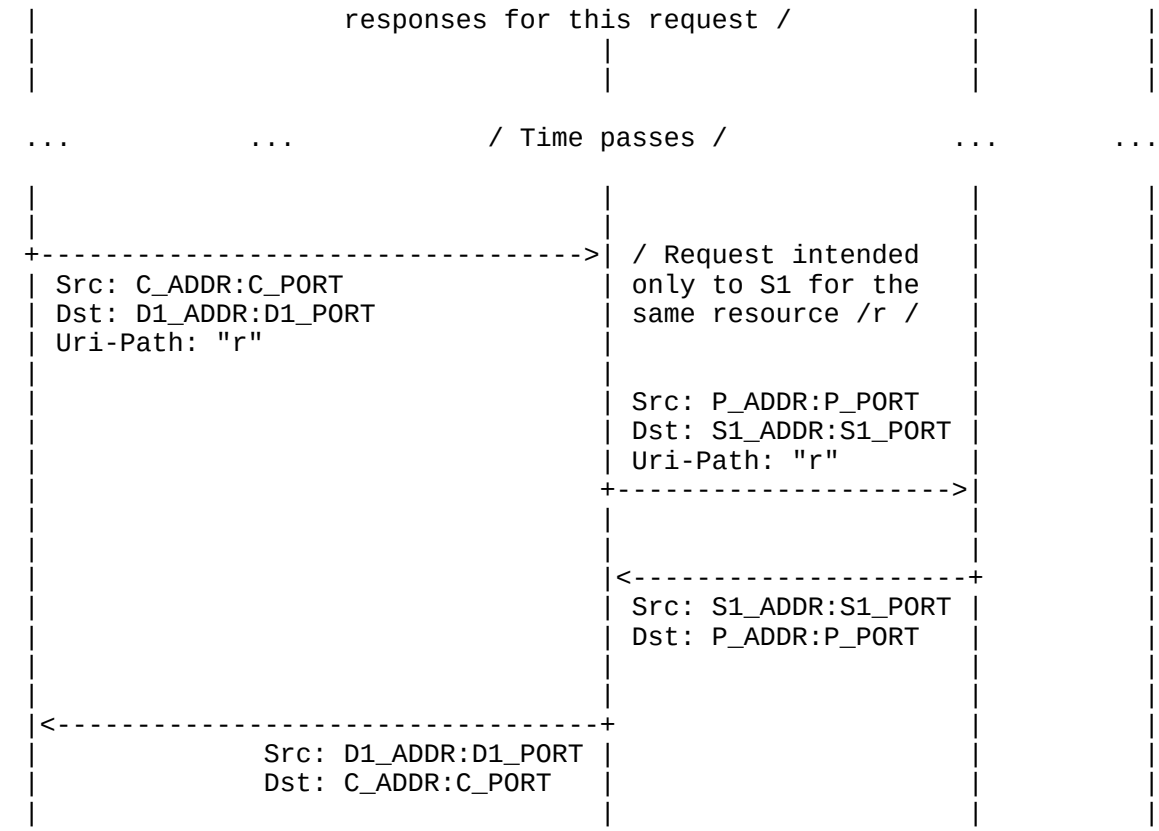


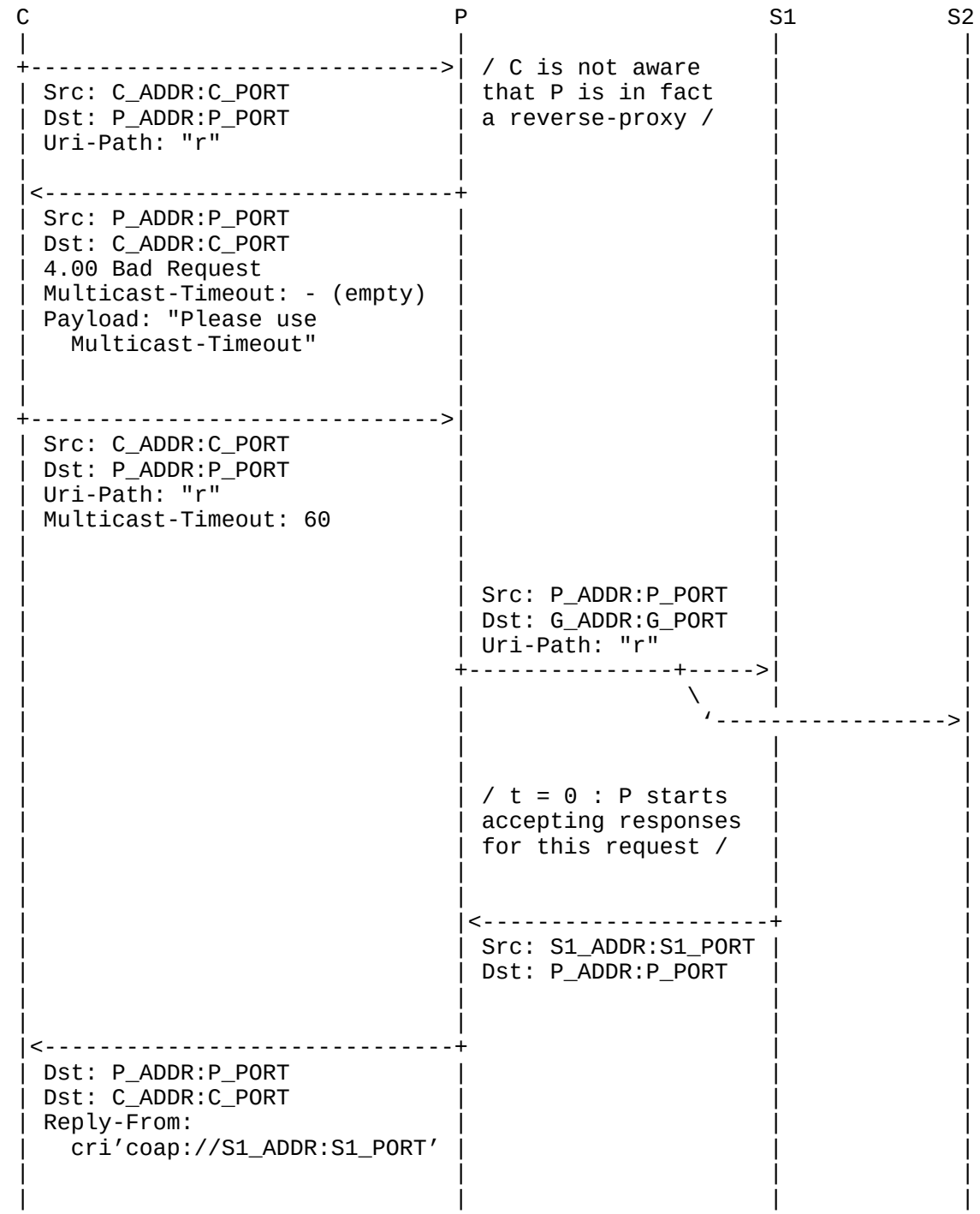
Figure 3: Workflow Example With a Reverse-Proxy Standing in for Both the Whole Group of Servers and Each Individual Server. This Requires the Proxy to Have One Pair (IP Address, Port Number) for Each Group and One for Each Origin Server.

A.3. Example 3

The example shown in Figure 4 builds on the example in Appendix A.2.

However, it considers a reverse-proxy that stands in for only the whole group of servers, but not for each individual server Sx. Therefore, it is possible for the client C to reach the servers directly.

The final exchange between C and S1 occurs with CoAP over UDP.


C
P
S1
S2
/ C is not aware that P is in fact a reverse-proxy /
Src: C_ADDR:C_PORT
Dst: P_ADDR:P_PORT
Uri-Path: "r"
Src: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT
4.00 Bad Request
Multicast-Timeout: - (empty)
Payload: "Please use Multicast-Timeout"
Src: C_ADDR:C_PORT
Dst: P_ADDR:P_PORT
Uri-Path: "r"
Multicast-Timeout: 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:S1_PORT
Dst: P_ADDR:P_PORT
Dst: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT
Reply-From: cri'coap://S1_ADDR:S1_PORT'

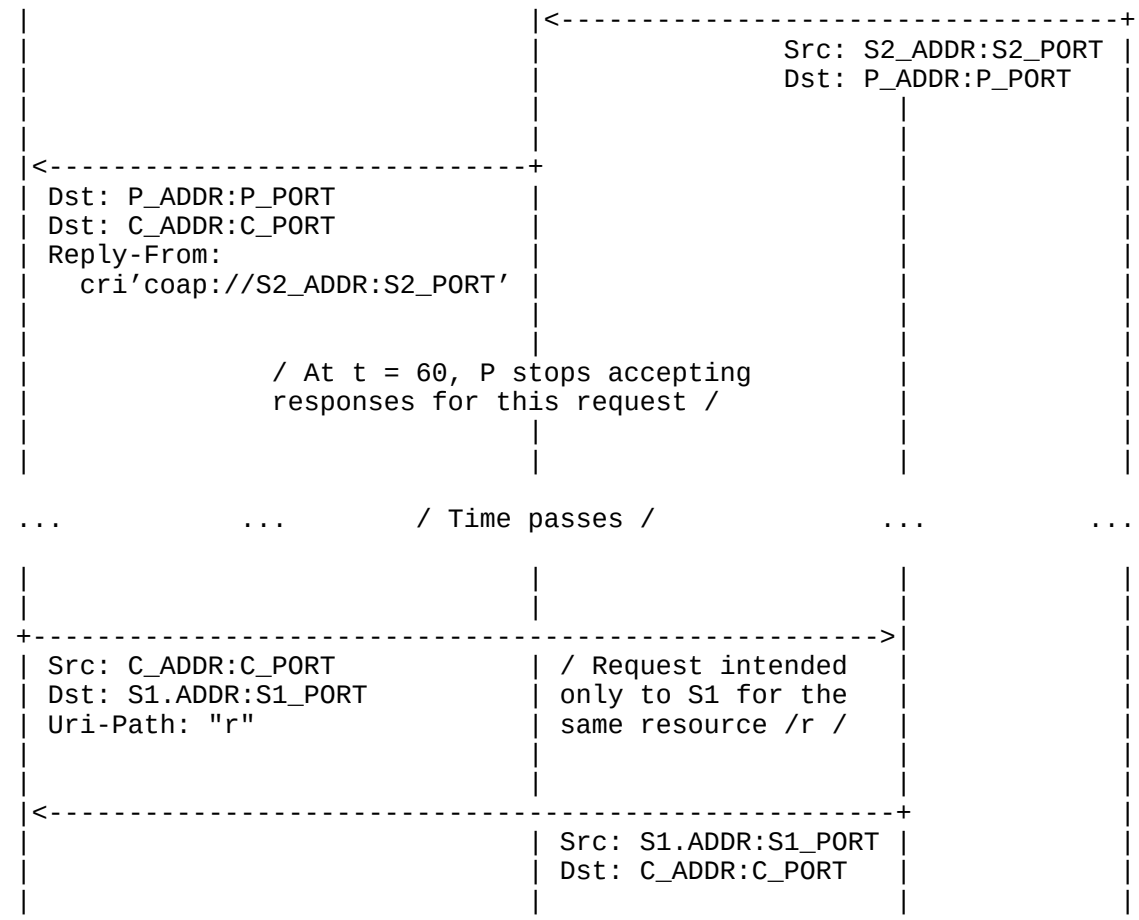


Figure 4: Workflow Example with a Reverse-Proxy Standing in for Only the Whole Group of Servers, but Not for Each Individual Server. This Requires the Proxy to Have One Pair (IP Address, Port Number) for Each Group.

Appendix B. Document Updates

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

B.1. Version -03 to -04

* More appropriate pointers to sections of draft-ietf-core-groupcomm-bis.

* More precise semantics for the Reply-From Option.

* Defined early cancellation of ongoing response forwarding.

* Suggested value ranges for codepoints to register.

* Clarifications and editorial improvements.

B.2. Version -02 to -03

* Made RFC 7967 a normative reference.

* Improved error handling for request reception at the proxy.

* Improved security considerations for the new CoAP options.

* Aligned handling of multiple responses with draft-ietf-core-groupcomm-bis.

* Revised HTTP Reply-From header field to be a Structured Header Field.

* Revised HTTP Group-ETag header field to be a Structured Header Field.

* Clarifications and editorial improvements.

B.3. Version -01 to -02

* Reply-To Option renamed as Reply-From.

* Multicast-Timeout Option set to 0 ultimately yields an empty value.

* Removed moot text on reverse-proxies that might use default timeouts.

* Improved description on using Proxy-Cri and Proxy-Scheme-Number.

* Improved error handling for inadequate timeouts with proxy chains.

* Revised the examples of message exchange with a reverse-proxy.

* Fixes in the IANA considerations.

* Editorial fixes and improvements.

B.4. Version -00 to -01

* Definition of "individual request" in the terminology.
* UDP/IP multicast treated as the default transport protocol.
* Always use the Multicast-Timeout Option, also with reverse-proxies.
* Response-Forwarding Option:
  - Renamed as "Reply-To".
  - Revised encoding to use CRIs.
  - Revised semantics to better address setups with reverse-proxies.
  - Added before possible response caching.
* Clarified response processing at reverse-proxies.
* Updated IANA considerations.
* Editorial fixes and improvements.

Acknowledgments

The authors sincerely thank Christian Ams端ss, Carsten Bormann, Rikard H旦glund, Jim Schaad, and G旦ran Selander for their comments and feedback.

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

Authors’ Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

Esko Dijk
IoTconsultancy.nl
Utrecht
Email: esko.dijk@iotconsultancy.nl