Workgroup: CoRE Working Group
Internet-Draft:
draft-tiloca-core-groupcomm-proxy-07
Updates: 7252 (if approved)
Published: 5 September 2022
Intended Status: Standards Track
Expires: 9 March 2023
Authors: M. Tiloca E. Dijk
RISE AB IoTconsultancy.nl

Proxy Operations for CoAP Group Communication

Abstract

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

Discussion Venues

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

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

Source for this draft and an issue tracker can be found at https://gitlab.com/crimson84/draft-tiloca-core-groupcomm-proxy.

Status of This Memo

This Internet-Draft will expire on 9 March 2023.

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 Section 3.5 of [I-D.ietf-core-groupcomm-bis], this group communication scenario poses a number of issues and limitations to proxy operations.

In particular, the client sends to the proxy a single unicast request, which the proxy forwards to a group of servers over IP multicast. Later on, the proxy replies to the client's original unicast request, by relaying back the responses from the servers.

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

This document considers the former approach. That is, after forwarding a CoAP group request from the client to the group of CoAP servers, the proxy relays the individual responses back to the client as separate CoAP messages. The described method addresses all the related issues raised in Section 3.5 of [I-D.ietf-core-groupcomm-bis]. To this end, a dedicated signaling protocol is defined, using two new CoAP options.

Using this protocol, the client explicitly confirms its intent to perform a proxied group request and its support for receiving multiple responses as a result, i.e., one or more from each origin server. Also, the client signals for how long it is willing to wait for responses. When relaying to the client a response to the group request, the proxy indicates the addressing information of the origin server. This enables the client to distinguish, multiple diffent responses by origin and to possibly contact one or more of the respective servers by sending individual unicast request(s) to the indicated address(es). In doing these follow-up unicast requests, the client may optionally bypass the proxy.

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

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

1.1. Terminology

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

Readers are expected to be familiar with terms and concepts defined in CoAP [RFC7252], Group Communication for CoAP [I-D.ietf-core-groupcomm-bis], CBOR [RFC8949], OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

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

2. The Multicast-Timeout Option

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

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

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

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

Figure 1: The Multicast-Timeout Option.

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

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

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

3. The Response-Forwarding Option

The Response-Forwarding Option defined in this section has the properties summarized in Figure 2, which extends Table 4 of [RFC7252]. The option is intended only for inclusion in CoAP responses, and builds on the Base-Uri option from Section 3 of [I-D.bormann-coap-misc].

Since the option is intended only for responses, the column "N" indicates a dash for "not applicable".

No.	C	U	N	R	Name	Format	Length	Default
TBD2			-		Response-Forwarding	(*)	10-25	(none)

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

(*) See below.

Figure 2: The Response-Forwarding Option.

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

Upon receiving a response to such request from a server, the proxy includes the Response-Forwarding Option into the response sent to the origin client (see Section 5). The proxy uses the option to indicate the addressing information where the client can send an individual request intended to that origin server.

In particular, the client can use the addressing information specified in the option to identify the response originator and

possibly send it individual requests later on, either directly, or indirectly via the proxy, as unicast requests.

The option value is set to the byte serialization of the CBOR array 'tp_info' defined in Section 4.2.1 of [I-D.ietf-core-observe-multicast-notifications], including only the set of elements 'srv_addr'. In turn, the set includes the integer 'tp_id' identifying the used transport protocol, and further elements whose number, format and encoding depend on the value of 'tp_id'.

The value of 'tp_id' MUST be taken from the "Value" column of the "CoAP Transport Information" registry defined in Section 16.5 of [I-D.ietf-core-observe-multicast-notifications]. The elements of 'srv_addr' following 'tp_id' are specified in the corresponding entry of the Registry, under the "Server Addr" column.

If the server is reachable through CoAP transported over UDP, the 'tp_info' array includes the following elements, encoded as defined in Section 4.2.1.1 of [I-D.ietf-core-observe-multicast-notifications].

* 'tp_id': the CBOR integer with value 1. This element MUST be present.

* 'srv_host': a CBOR byte string, encoding the unicast IP address of the server. This element is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)". This element MUST be present.

* 'srv_port': a CBOR unsigned integer or the CBOR simple value "null" (0xf6). This element MAY be present.

  If present as a CBOR unsigned integer, it has as value the destination UDP port number to use for individual requests to the server.

  If present as the CBOR simple value "null" (0xf6), the client MUST assume that the same port number specified in the group URI of the original unicast CoAP group request sent to the proxy (see Section 5.1.1) can be used for individual requests to the server.

  If not present, the client MUST assume that the default port number 5683 defined in [RFC7252] can be used as the destination UDP port number for individual requests to the server.

The CDDL notation [RFC8610] provided below describes the 'tp_info' CBOR array using the format defined above.

```
tp_info = [
       tp_id : 1,              ; UDP as transport protocol
    srv_host : #6.260(bstr),   ; IP address where to reach the server
  ? srv_port : uint / null     ; Port number where to reach the server
]
```

At present, 'tp_id' is expected to take only value 1 (UDP) when using forward proxies, UDP being the only currently available transport for CoAP to work over IP multicast. While additional multicast-friendly transports may be defined in the future, other current tranport protocols can still be useful in applications relying on a reverse-proxy (see Section 6).

The rest of this section considers the new values of 'tp_id' registered by this document (see Section 11.2), and specifies:

- The encoding for the elements of 'tp_info' following 'tp_id' (see Section 3.1).

- The port number assumed by the client if the element 'srv_port' of 'tp_info' is not present (see Section 3.2).

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

3.1. Encoding of Server Address

This document defines some values used as transport protocol identifiers, whose respective new entries are included in the "CoAP Transport Information" registry defined in Section 16.5 of [I-D.ietf-core-observe-multicast-notifications].

For each of these values, the following table summarizes the elements specified under the "Srv Addr" and "Req Info" columns of the registry, together with their CBOR encoding and short description.

While not listed here for brevity, the element 'tp_id' is always present as a CBOR integer in the element set "Srv Addr".

'tp_id' Values	Element Set	Element	CBOR Type	Description
2, 3, 4, 5, 6	Srv Addr	srv_host	#6.260(bstr) (*)	Address of the server
		srv_port	uint / null	Port number of the server
	Req Info	cli_host	#6.260(bstr) (*)	Address of the client
		cli_port	uint	Port number of the client

* The CBOR byte string is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)".

3.2. Default Values of the Server Port Number

If the 'srv_port' element of the 'tp_info' array is not present, the client MUST assume the following value as port number where to send individual requests intended to the server, based on the value of 'tp_id'.

- If 'tp_id' is equal to 1, i.e., CoAP over UDP, the default port number 5683 as defined in [RFC7252].
- If 'tp_id' is equal to 2, i.e., CoAP over UDP secured with DTLS, the default port number 5684 as defined in [RFC7252].
- If 'tp_id' is equal to 3, i.e., CoAP over TCP, the default port number 5683 as defined in [RFC8323].
- If 'tp_id' is equal to 4, i.e., CoAP over TCP secured with TLS, the default port number 5684 as defined in [RFC8323].
- If 'tp_id' is equal to 5, i.e., CoAP over WebSockets, the default port number 80 as defined in [RFC8323].
- If 'tp_id' is equal to 6, i.e., CoAP over WebSockets secured with TLS, the default port number 443 as defined in [RFC8323].

4. Requirements and Objectives

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

This document assumes that the following requirements are fulfilled.

- REQ1. The proxy is explicitly configured (allow-list) to perform proxied group requests on behalf of specific allowed client(s).

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

  - A TLS [RFC8446] or DTLS [RFC6347][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.tiloca-core-oscore-capable-proxies].

- REQ3. If secure, end-to-end communication is required between the client and the servers in the CoAP group, exchanged messages MUST be protected by using Group OSCORE [I-D.ietf-core-oscore-groupcomm], as discussed in Section 5 of [I-D.ietf-core-groupcomm-bis]. This requires the client and the servers to have previously joined the correct OSCORE group, for instance by using the approach described in [I-D.ietf-ace-key-groupcomm-oscore]. The correct OSCORE group to join can be pre-configured or alternatively discovered, for instance by using the approach described in [I-D.tiloca-core-oscore-discovery].

This document defines how to achieve the following objectives.

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

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

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

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

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

5. Protocol Description

This section specifies the steps of the signaling protocol.

5.1. Request Sending at the Client

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

5.1.1. Request Sending

The client proceeds according to the following steps.

1. The client prepares a unicast CoAP group request addressed to the proxy. The request specifies the group URI where the request has to be forwarded to, as a string in the Proxi-URI option or by using the Proxy-Scheme option with the group URI constructed from the URI-* options (see Section 3.5.1 of [I-D.ietf-core-groupcomm-bis]).

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

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

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

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

   *T should be at least the expected worst-case round-trip delay between the client and the proxy plus the worst-case

round-trip delay between the proxy and any one of the origin servers.

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

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

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

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

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

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

5.1.2. Supporting Observe

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

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

5.2. Request Processing at the Proxy

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

5.2.1. Request Processing

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

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

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

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

   If the Multicast-Timeout Option is not present, the proxy MUST stop processing the request and MUST reply to the client with a 4.00 (Bad Request) response. The response MUST include a Multicast-Timeout Option with an empty (zero-length) value, indicating that the Multicast-Timeout Option was missing and has to be included in the request. As per Section 5.9.2 of [RFC7252] The response SHOULD include a diagnostic payload.

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

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

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

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

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

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

5.2.2. Supporting Observe

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

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

5.3. Request and Response Processing at the Server

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

5.3.1. Request and Response Processing

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

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

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

5.3.2. Supporting Observe

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

5.4. Response Processing at the Proxy

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

5.4.1. Response Processing

Upon receiving a response matching with the group request before the amount of time T' has elapsed, the proxy proceeds according to the following steps.

1. The proxy MUST include the Response-Forwarding Option defined in Section 3 into the response. The proxy specifies as option value the addressing information of the server generating the response, encoded as defined in Section 3. In particular:

   * The 'srv_addr' element of the 'srv_info' array MUST specify the server IPv6 address if the multicast request was destined for an IPv6 multicast address, and MUST specify the server IPv4 address if the multicast request was destined for an IPv4 multicast address.

   * If present, the 'srv_port' element of the 'srv_info' array MUST specify the port number of the server as the source port number of the response. This element MUST be present if the source port number of the response differs from the default port number for the transport protocol specified in the 'tp_id' element.

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

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

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

5.4.2. Supporting Observe

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

Furthermore, the proxy takes the role of a server when forwarding notifications from origin servers back to the client. To this end, the proxy follows what is specified in Section 3.7 of [I-D.ietf-

core-groupcomm-bis] and Section 5 of [RFC7641], with the following additions.

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

* The proxy frees up its Token value used for a group observation only if, after the timeout expiration, no 2.xx (Success) responses matching with the group request and also including an Observe option have been received from any origin server. After that, as long as observations are active with servers in the group for the target resource of the group request, notifications from those servers are forwarded back to the client, as defined in Section 5.4, and the Token value used for the group observation is not freed during this time.

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

5.5. Response Processing at the Client

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

5.5.1. Response Processing

Upon receiving from the proxy a response matching with the original unicast request before the amount of time T has elapsed, the client proceeds according to the following steps.

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

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

3. The client identifies the origin server, whose addressing information is specified as value of the Response-Forwarding Option. If the 'srv_port' element of the 'tp_info' array in the Response-Forwarding Option is not present or specifies the CBOR simple value "null" (0xf6), then the client determines the port

number where to send unicast requests to the server -- in case this is needed -- as defined in Section 3. In the former case, the assumed default port number depends on the transport protocol specified by the 'tp_id' element of the 'tp_info' array (see Section 3.2).

In particular, the client is able to distinguish different responses as originated by different servers. Optionally, the client may contact one or more of those servers individually, i.e., directly (bypassing the proxy) or indirectly (via a proxied unicast request).

In order to individually reach an origin server again through the proxy, the client is not required to understand or support the transport protocol indicated in the Response-Forwarding Option, as used between the proxy and the origin server, in case it differs from "UDP" (1). That is, using the IPv4/IPv6 address value and optional port value from the Response-Forwarding Option, the client simply creates the correct URI for the individual request, by means of the Proxy-Uri or Uri-Scheme Option in the unicast request to the proxy. The client uses the transport protocol it knows, and has used before, to send the request to the proxy.

As discussed in Section 3.1.6 of [I-D.ietf-core-groupcomm-bis], it is possible that the client receives multiple responses to the same group request, i.e., with the same Token, from the same origin server. The client normally processes at the CoAP layer each of those responses from the same origin server, and decides how to exactly handle them depending on its available context information (see Section 3.1.6 of [I-D.ietf-core-groupcomm-bis]).

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

5.5.2. Supporting Observe

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

Instead, if at least one such response has been received, the client continues receiving those notifications as forwarded by the proxy, as long as the observation for the target resource of the original unicast request is active.

5.6. Example

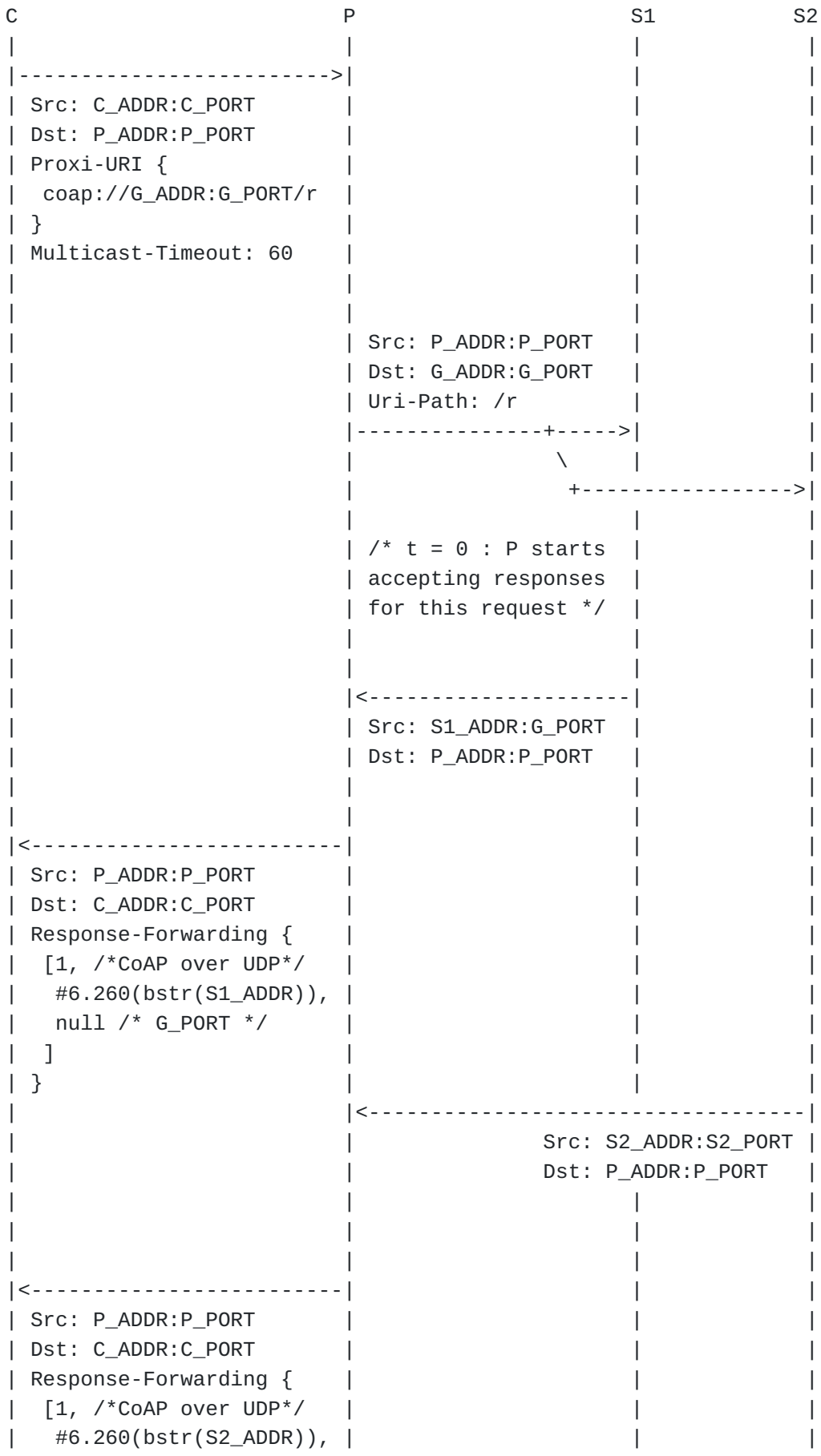
The example in this section refers to the following actors.

- One origin client C, with address C_ADDR and port number C_PORT.
- One proxy P, with address P_ADDR and port number P_PORT.
- Two origin servers S1 and S2, where the server Sx has address Sx_ADDR and port number Sx_PORT.

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

The communication between C and P is based on CoAP over UDP, as per [RFC7252]. The communication between P and the origin servers is based on CoAP over UDP and IP multicast, as per [I-D.ietf-core-groupcomm-bis].

Finally, 'bstr(X)' denotes a CBOR byte string where its value is the byte serialization of X.


C
P
S1
S2
Src: C_ADDR:C_PORT
Dst: P_ADDR:P_PORT
Proxi-URI {
coap://G_ADDR:G_PORT/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:G_PORT
Dst: P_ADDR:P_PORT
Src: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT
Response-Forwarding {
[1, /*CoAP over UDP*/
#6.260(bstr(S1_ADDR)),
null /* G_PORT */
]
}
Src: S2_ADDR:S2_PORT
Dst: P_ADDR:P_PORT
Src: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT
Response-Forwarding {
[1, /*CoAP over UDP*/
#6.260(bstr(S2_ADDR)),

```
|   S2_PORT                |                      |             |
|  ]                       |                      |             |
| }                        |                      |             |
|            /* At t = 60, P stops accepting     |             |
|            responses for this request */       |             |
|                          |                      |             |
```

Figure 3: Workflow example with a forward-proxy

6. Reverse-Proxies

The use of reverse-proxies in group communication scenarios is defined in Section 3.5.2 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 explictly 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 addional issues compared to the case with a forward-proxy, as compiled in Section 3.5.2 of [I-D.ietf-core-groupcomm-bis]. A reverse-proxy may also operate without support of the Multicast-Timeout Option, as defined in that section.

Appendix A provides examples with a reverse-proxy.

6.1. Processing on the Client Side

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

An exception is the case where the reverse-proxy has a pre-configured timeout value T_PROXY, as the default timeout value to use for when to stop accepting responses from the servers, after the reception of the original unicast request from the client. In this case, a client aware of such a configuration MAY omit the Multicast-Timeout Option in the request sent to the proxy.

The client processes the CoAP responses forwarded back by the proxy as defined in Section 5.5.

6.2. Processing on the Proxy Side

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

In particular, when such a request does not include a Multicast-Timeout Option, the proxy SHOULD explicitly reveal itself as a

reverse-proxy, by sending a 4.00 (Bad Request) response including a Multicast-Timeout Option with empty (zero-length) value.

An exception is the case where the reverse-proxy has a pre-configured timeout value T_PROXY, as default timeout value to use for when to stop accepting responses from the servers, after the reception of the original unicast request from the client. In this case, the proxy MAY replace the steps 3 and 4 in Section 5.2.1 with the following step.

A. The proxy verifies the presence of the Multicast-Timeout Option, as a confirmation that the client is willing to receive multiple CoAP responses matching with the same original request. Then, the proxy performs the following actions.

* If the Multicast-Timeout Option is present, the proxy retrieves the value T' from the Multicast-Timeout Option, and then removes the option from the client's request. That is, the timeout value indicated in the option overrides the pre-configured timeout value T_PROXY.

* If the Multicast-Timeout option is not present, the proxy checks that, according to its local configuration, both the following conditions hold for the client (which, at this point, has been successfully authenticated).

  - COND_1 : The client is aware of the default timeout value T_PROXY pre-configured at the proxy.

  - COND_2 : The client is able to process multiple responses to the same request.

  These conditions are expected to hold for clients that are locally registered at the proxy, successfully authenticated and allowed-listed to have their requests proxied to CoAP group URIs.

  If the proxy is able to successfully assert that both the two conditions hold, then the proxy considers the value T' as equal to T_PROXY and proceeds to step 5.

  If the proxy is not able to successfully assert that both the two conditions hold, the proxy MUST stop processing the request and MUST reply to the client with a 4.00 (Bad Request) response. The response MUST include a Multicast-Timeout Option with an empty (zero-length) value, indicating that the Multicast-Timeout Option was missing and has to be included in the request. As per Section 5.9.2 of [RFC7252] The response SHOULD include a diagnostic payload.

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

7. Caching

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

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

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

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

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

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

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

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

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

* As discussed in Section 5.4.1, the proxy can receive multiple responses to the same group request from a same origin server, and forwards them back to the origin client "as they come". When

this happens, each of such multiple responses is stored in the cache entry associated with the server "as it comes", possibly replacing an already stored response from that server.

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

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

7.1. Freshness Model

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7.2. Validation Model

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

7.2.1. Proxy-Servers Revalidation with Unicast Requests

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

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

[TODO

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

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

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

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

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

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

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

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

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

]

7.2.2. Proxy-Servers Revalidation with Group Requests

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

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

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

[TODO

See the notes in Section 7.2.1.

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

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

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

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

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

]

7.3. Client-Proxy Revalidation with Group Requests

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

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

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

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

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

Figure 4: The Group-ETag Option.

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

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

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

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

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

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

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

When sending to the proxy a GET or FETCH request to be forwarded to the servers in the CoAP group, the client MAY include one or more Group-ETag Options. Each option specifies one entity-tag value,

applicable to the set J of cache entries that can be hit by the group request.

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

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

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

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

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

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

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

7.4.1. Deterministic Requests to Achieve Cacheability

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

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

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

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

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

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

7.4.2. Validation of Responses

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

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

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

[TODO

See the notes in Section 7.2.1.

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

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

]

8. Chain of Proxies

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

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

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

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

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

8.1. Request Processing at the Proxy

Upon receiving a group request to be forwarded to a CoAP group URIs, a proxy proceed as follows.

If the proxy is the last one in the chain, i.e., it is the last hop before the origin servers, the proxy performs the steps defined in Section 5.2, with no modifications.

Otherwise, the proxy performs the steps defined in Section 5.2, with the following differences.

* At steps 1-3, "client" refers to the origin client for the first proxy in the chain; or to the previous hop proxy closer to the origin client, otherwise.

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

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

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

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

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

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

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

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

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

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

Upon receiving such an error response, any proxy in the chain MAY send an updated group request to the next hop towards the origin servers, specifying in the Multicast-Timeout Option a value T' greater than in the previous request. If this does not happen, the proxy receiving the error response MUST also send a 5.05 (Proxying Not Supported) error response to the previous hop proxy closer to the origin client. Like the received one, also this error response SHOULD include a Multicast-Timeout Option, set to the minimum value T' acceptable by the proxy sending the error response.

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

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

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

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

8.1.1. Supporting Observe

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

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

8.2. Response Processing at the Proxy

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

If the proxy is the last one in the chain, i.e., it is the last hop before the origin servers, the proxy performs the steps defined in Section 5.4, with no modifications.

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

- The proxy skips step 1. In particular, the proxy MUST NOT remove, alter or replace the Response-Forwarding Option.

- At step 2, "client" refers to the origin client for the first proxy in the chain; or to the previous hop proxy closer to the origin client, otherwise.

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

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

8.2.1. Supporting Observe

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

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

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

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

* The proxy frees up its Token value used for a group observation only if, after the timeout expiration, no 2.xx (Success) responses matching with the group request and also including an Observe option have been received from the next hop towards the origin servers. After that, as long as the observation for the target resource of the group request is active with the next hop towards the origin servers in the group, notifications from that hop are forwarded back to the (previous hop proxy closer to the) origin client, as defined in Section 8.2.

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

9. HTTP-CoAP Proxies

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

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

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

9.1. The HTTP Multicast-Timeout Header Field

The HTTP Multicast-Timeout header field (see Section 11.3) is used for carrying the content otherwise specified in the CoAP Multicast-Timeout Option defined in Section 2.

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

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

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

The same applies in the opposite direction, when translating an HTTP message into a CoAP message.

9.2. The HTTP Response-Forwarding Header Field

The HTTP Response-Forwarding header field (see Section 11.3) is used for carrying the content otherwise specified in the CoAP Response-Forwarding Option defined in Section 3.

Using the Uniform Resource Identifier (URI) syntax components defined in [RFC3986], the HTTP Response-Forwarding header field value is as follows.

scheme = <scheme, see Section 3.1 of [RFC3986]>

authority = <authority, see Section 3.2 of [RFC3986]>

Response-Forwarding = scheme "://" authority

In particular:

- The scheme component indicates the URI scheme otherwise specified in the CoAP Response-Forwarding Option, as per the 'tp_id' element of the 'tp_info' array (see Section 3). That is, the 'tp_id' element with integer value 1 results in the scheme "coap".

- The authority component indicates the URI authority otherwise specified in the CoAP Response-Forwarding Option, as per the 'srv_host' and 'srv_port' elements of the 'tp_info' array (see Section 3).

When translating a CoAP message into an HTTP message, the HTTP Response-Forwarding header field is set to the URI specified in the CoAP Response-Forwarding Option, as per the rules defined above. In particular, consistently with what is defined in Section 3:

- If the 'srv_port' element of the 'tp_info' array is present and specifies the CBOR simple value "null" (0xf6), the URI authority of the header field includes the same port number that was specified in the group URI where the group request was forwarded.

- If the 'srv_port' element of the 'tp_info' array is not present, the URI authority of the header field includes the default port number for the transport protocol specified by the 'tp_id' element of the 'tp_info' array, as per Section 3.2.

When translating an HTTP message into a CoAP message, the CoAP Response-Forwarding Option is set to the URI specified by the HTTP

Response-Forwarding header field. In particular, the URI is encoded according to the format specified in Section 3.

9.3. The HTTP Group-ETag Header Field

The HTTP Group-ETag header field (see Section 11.3) is used for carrying the content otherwise specified in the CoAP Group-ETag Option defined in Section 7.3.

Using the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] and including the following core ABNF syntax rules defined by that specification: ALPHA (letters) and DIGIT (decimal digits), the HTTP Group-ETag header field value is as follows.

```
group-etag-char = ALPHA / DIGIT / "-" / "_"

Group-ETag = 2*group-etag-char
```

When translating a CoAP message into an HTTP message, the HTTP Group-ETag header field is set to the value of the CoAP Group-ETag Option in base64url (see Section 5 of [RFC4648]) encoding without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].

When translating an HTTP message into a CoAP message, the CoAP Group-ETag Option is set to the value of the HTTP Group-ETag header field decoded from base64url (see Section 5 of [RFC4648]) without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].

9.4. Request Sending at the Client

The client proceeds according to the following steps.

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

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

3. If Group OSCORE is used end-to-end between the client and the servers, the client translates the HTTP request into a CoAP request, as per [RFC8075]. Then, the client protects the

resulting CoAP request by using Group OSCORE, as defined in [I-D.ietf-core-oscore-groupcomm]. Finally, the protected CoAP request is mapped to HTTP as defined in Section 11.2 of [RFC8613]. Later on, the resulting HTTP request MUST be sent in compliance with the rules in Section 11.1 of [RFC8613].

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

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

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

9.5. Request Processing at the Proxy

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

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

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

9.6. Response Processing at the Proxy

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

messages specifying the Response-Forwarding Option are defined in Section 9.2.

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

When the timout 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 it has with the client.

9.7. Response Processing at the Client

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

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

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

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

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

   * If Group OSCORE is used end-to-end between the client and servers, the client translates the HTTP response RESP into a CoAP response, as per Section 11.3 of [RFC8613]. Then, the client decrypts the resulting CoAP response by using Group OSCORE, as defined in [I-D.ietf-core-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 Response-Forwarding header field are defined in Section 9.2.

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

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

9.8. Example

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

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

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

Body: Do that!
```

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

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

--batch_foo_bar
Content-Type: application/http

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: <INDIVIDUAL_RESPONSE_1_CONTENT_LENGTH>
Response-Forwarding: coap://S1_ADDR:G_PORT

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

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: <INDIVIDUAL_RESPONSE_2_CONTENT_LENGTH>
Response-Forwarding: coap://S2_ADDR:S2_PORT

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

9.9. Streamed Delivery of Responses to the Client

[TODO

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

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

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

9.10.1. Processing on the Client Side

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

An exception is the case where the reverse-proxy has a pre-configured timeout value T_PROXY, as the default timeout value to use for when to stop accepting responses from the servers, after the reception of the original unicast request from the client. In this case, a client aware of such a configuration MAY omit the Multicast-Timeout header field in the request sent to the proxy.

The client processes the HTTP response forwarded back by the proxy as defined in Section 9.7.

9.10.2. Processing on the Proxy Side

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

Once translated the HTTP request into a CoAP request, the proxy performs what is defined in Section 6.2. Note that, in this case, the condition COND_2 always holds, since the proxy is going to send to the client at most one response, i.e., the HTTP batch response (see Section 9.6).

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

10. Security Considerations

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

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

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

Furthermore, the following additional considerations hold.

10.1. Client Authentication

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

If the group request is (also) protected end-to-end between the client and the servers using the group mode of Group OSCORE, the proxy can act as external signature checker (see Section 8.5 of [I-D.ietf-core-oscore-groupcomm]) and authenticate the client by successfully verifying the signature embedded in the group request. However, this requires that, for each client to authenticate, the proxy stores the authentication credential and public key included therin used by that client in the OSCORE group. This in turn would require a form of active synchronization between the proxy and the Group Manager for that group [I-D.ietf-core-oscore-groupcomm].

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

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

10.2. Multicast-Timeout Option

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

The security association between the client and the proxy MUST provide message integrity, so that further intermediaries between the two as well as on-path active adversaries are not able to remove the option or alter its content, before the group request reaches the proxy. Removing the option would otherwise result in not forwarding the group request to the servers. Instead, altering the option content would result in the proxy accepting and forwarding

back responses for an amount of time different than the one actually indicated by the client.

The security association between the client and the proxy SHOULD also provide message confidentiality. Otherwise, any further intermediaries between the two as well as any on-path passive adversaries would be able to simply access the option content, and thus learn for how long the client is willing to receive responses from the servers in the group via the proxy. This may in turn be used to perform a more efficient, selective suppression of responses from the servers.

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

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

10.3. Response-Forwarding Option

The Response-Forwarding Option is of class U for OSCORE [RFC8613]. Hence, also when Group OSCORE is used between the client and the servers [I-D.ietf-core-oscore-groupcomm], the proxy that has forwarded the group request to the servers is able to include the option into a server response, before forwarding this response back to the (previous hop proxy closer to the) origin client.

Since the security association between the client and the proxy provides message integrity, any further intermediaries between the two as well as any on-path active adversaries are not able to undetectably remove the Response-Forwarding Option from a forwarded server response. This ensures that the client can correctly distinguish the different responses and identify their corresponding origin server.

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

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

10.4. Group-ETag Option

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

The security association between the client and the proxy MUST provide message integrity, so that further intermediaries between the two as well as on-path active adversaries are not able to remove the option or alter its content, before the group request reaches the proxy. Removing the option would otherwise result in the proxy not performing response revalidation at its cache entries associated with the servers in the CoAP group, even though that was what the client asked for.

Altering the option content in a group request would result in the proxy replying with 2.05 (Content) responses conveying the full resource representations from its cache entries, rather than with a single 2.03 (Valid) response. Instead, altering the option content in a 2.03 (Valid) or 2.05 (Content) response would result in the client wrongly believing that the already stored or the just received representation, respectively, is also the current one, as per the entity value of the tampered Group-ETag Option.

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

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

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

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

10.5. HTTP-to-CoAP Proxies

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

[TODO

Revisit security considerations from [RFC8075]

]

11. IANA Considerations

This document has the following actions for IANA.

11.1. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry within the "CoRE Parameters" registry group.

Number	Name	Reference
TBD1	Multicast-Timeout	[[this document]]
TBD2	Response-Forwarding	[[this document]]
TBD3	Group-ETag	[[this document]]

11.2. CoAP Transport Information Registry

IANA is asked to add the following entries to the "CoAP Transport Information" registry defined in Section 16.5 of [I-D.ietf-core-observe-multicast-notifications].

Transport Protocol	Description	Value	Srv Addr	Req Info	Reference
UDP secured with DTLS	UDP with DTLS is used as per RFC8323	2	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]
TCP	TCP is used as per RFC8323	3	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]
TCP secured with TLS	TCP with TLS is used as per RFC8323	4	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]
WebSockets	WebSockets are used as per RFC8323	5	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]
WebSockets secured with TLS	WebSockets with TLS are used as per RFC8323	6	tp_id srv_host srv_port	token cli_host ?cli_port	[This document]

11.3. Header Field Registrations

IANA is asked to enter the following HTTP header fields to the "Message Headers" registry.

Header Field Name	Protocol	Status	Reference
Multicast-Timeout	http	standard	[This document]
Response-Forwarding	http	standard	[This document]
Group-ETag	http	standard	[This document]

12. References

12.1. Normative References

[I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-07, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-07.txt>.

[I-D.ietf-core-observe-multicast-notifications]
Tiloca, M., Höglund, R., Amsüss, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-ietf-core-observe-multicast-notifications-04, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-core-observe-multicast-notifications-04.txt>.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-14, 7 March 2022, <https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-14.txt>.

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

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

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

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

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/

RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

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

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

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

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

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

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

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[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/info/rfc9112>.

12.2. Informative References

[I-D.amsuess-core-cachable-oscore] Amsüss, C. and M. Tiloca, "Cacheable OSCORE", Work in Progress, Internet-Draft, draft-amsuess-core-cachable-oscore-05, 11 July 2022,

<https://www.ietf.org/archive/id/draft-amsuess-core-cachable-oscore-05.txt>.

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

[I-D.ietf-ace-key-groupcomm-oscore] Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-14, 28 April 2022, <https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-14.txt>.

[I-D.tiloca-core-oscore-capable-proxies] Tiloca, M. and R. Höglund, "OSCORE-capable Proxies", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-capable-proxies-03, 11 July 2022, <https://www.ietf.org/archive/id/draft-tiloca-core-oscore-capable-proxies-03.txt>.

[I-D.tiloca-core-oscore-discovery] Tiloca, M., Amsuess, C., and P. V. D. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-11, 7 March 2022, <https://www.ietf.org/archive/id/draft-tiloca-core-oscore-discovery-11.txt>.

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

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <https://www.rfc-editor.org/info/rfc7967>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[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/info/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, 'bstr(X)' denotes a CBOR byte string where its value is the byte serialization of X.

A.1. Example 1

The example shown in Figure 5 considers a reverse-proxy P that provides access to both the whole group of servers {S1,S2} and also to each of those servers individually. The client C may not have a way to reach the servers directly (e.g., P is acting as a firewall). After the client C has received two responses to its group request sent via the proxy, it selects one server (S1) and requests another resource from it in unicast, again via the proxy.

In particular:

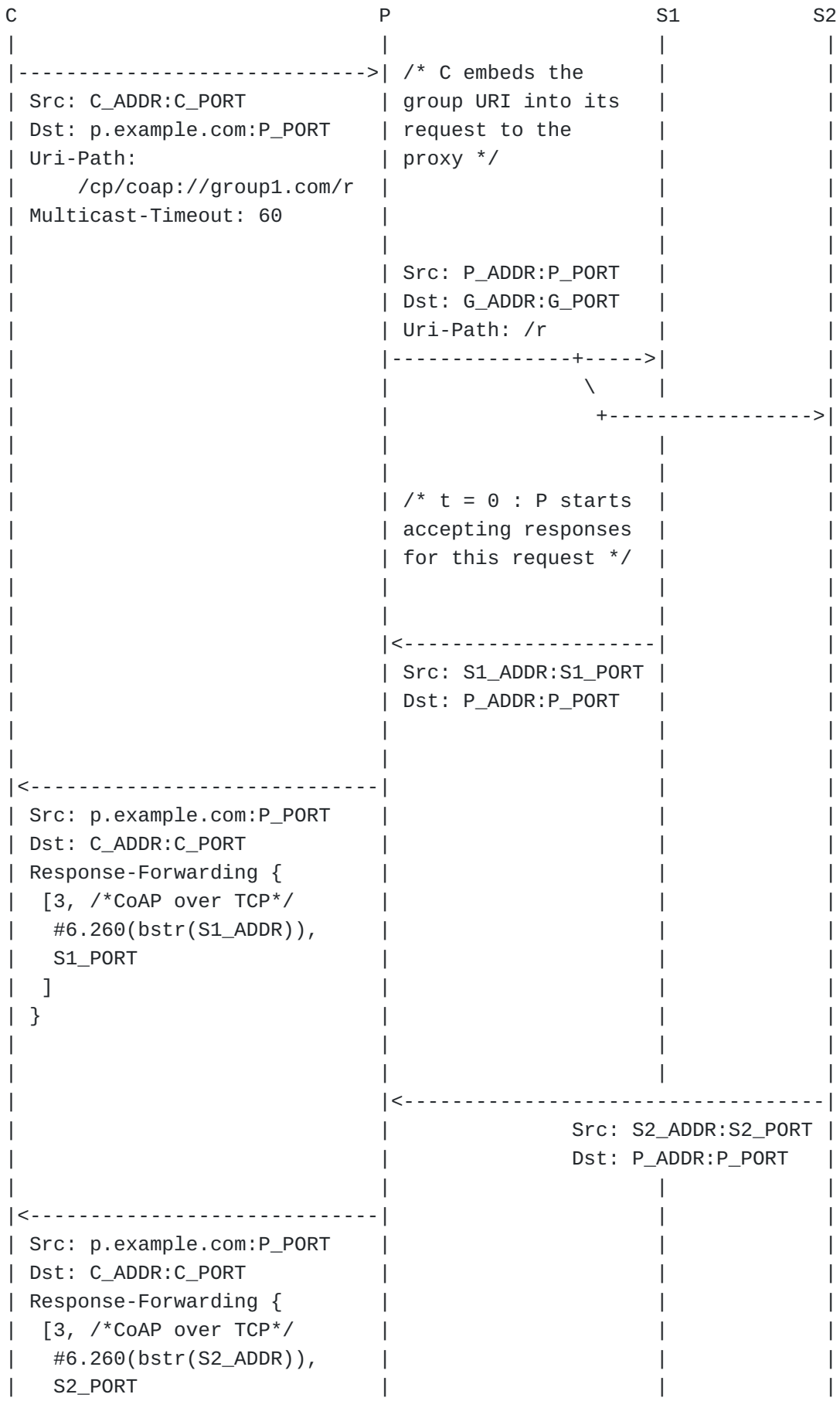
- The client C encodes the group URI 'coap://group1.com/r' within the URI path of its request to P. This encoding follows the "default mapping" defined in Section 5.3 of [RFC8075] for HTTP-to-CoAP proxies, but now applied to a CoAP-to-CoAP proxy. The proxy P decodes the embedded group URI from the request.
- The client's request URI path starts with '/cp', which is the resource on P that provides the CoAP proxy function. Since C in this example constructs the URI in its request including this resource '/cp', it is aware that is requesting to a proxy.
- Because the embedded group URI omits the CoAP port, P infers G_PORT to be the default port 5683 for the 'coap' scheme.

* The hostname 'p.example.com' resolves to the proxy's unicast IPv6 address P_ADDR.

* The hostname 'group1.com' resolves to the IPv6 multicast address G_ADDR. The proxy P performs this resolution upon receiving the request from C. P constructs the group request and sends it to the CoAP group at G_ADDR:G_PORT.

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

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


C
P
S1
S2
/* C embeds the group URI into its request to the proxy */
Src: C_ADDR:C_PORT
Dst: p.example.com:P_PORT
Uri-Path: /cp/coap://group1.com/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.example.com:P_PORT
Dst: C_ADDR:C_PORT
Response-Forwarding {
[3, /*CoAP over TCP*/
#6.260(bstr(S1_ADDR)),
S1_PORT
]
}
Src: S2_ADDR:S2_PORT
Dst: P_ADDR:P_PORT
Src: p.example.com:P_PORT
Dst: C_ADDR:C_PORT
Response-Forwarding {
[3, /*CoAP over TCP*/
#6.260(bstr(S2_ADDR)),
S2_PORT

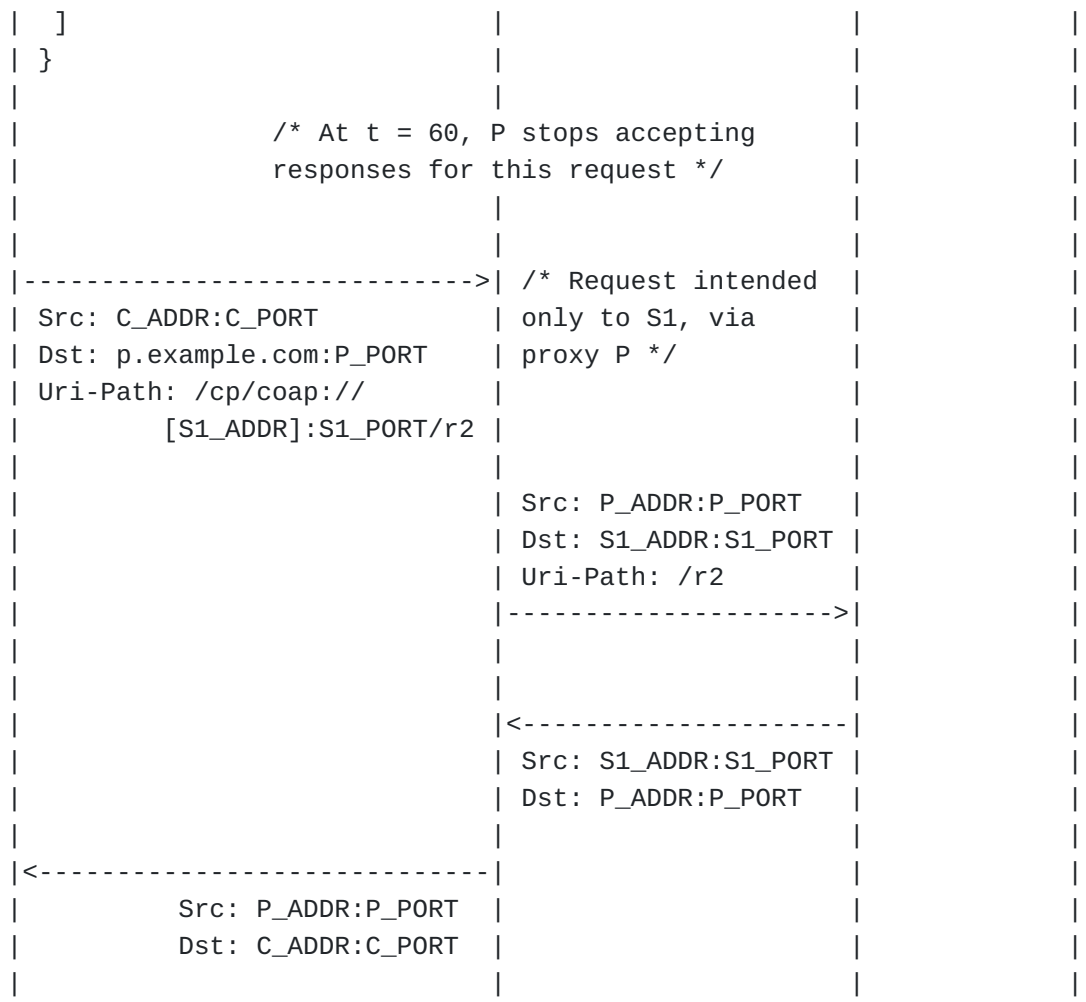

]
}
/* At t = 60, P stops accepting
responses for this request */
/* Request intended
only to S1, via
proxy P */
Src: C_ADDR:C_PORT
Dst: p.example.com:P_PORT
Uri-Path: /cp/coap://
[S1_ADDR]:S1_PORT/r2
Src: P_ADDR:P_PORT
Dst: S1_ADDR:S1_PORT
Uri-Path: /r2
Src: S1_ADDR:S1_PORT
Dst: P_ADDR:P_PORT
Src: P_ADDR:P_PORT
Dst: C_ADDR:C_PORT

Figure 5: Workflow example with reverse-proxy that processes an embedded group URI in a client's request

A.2. Example 2

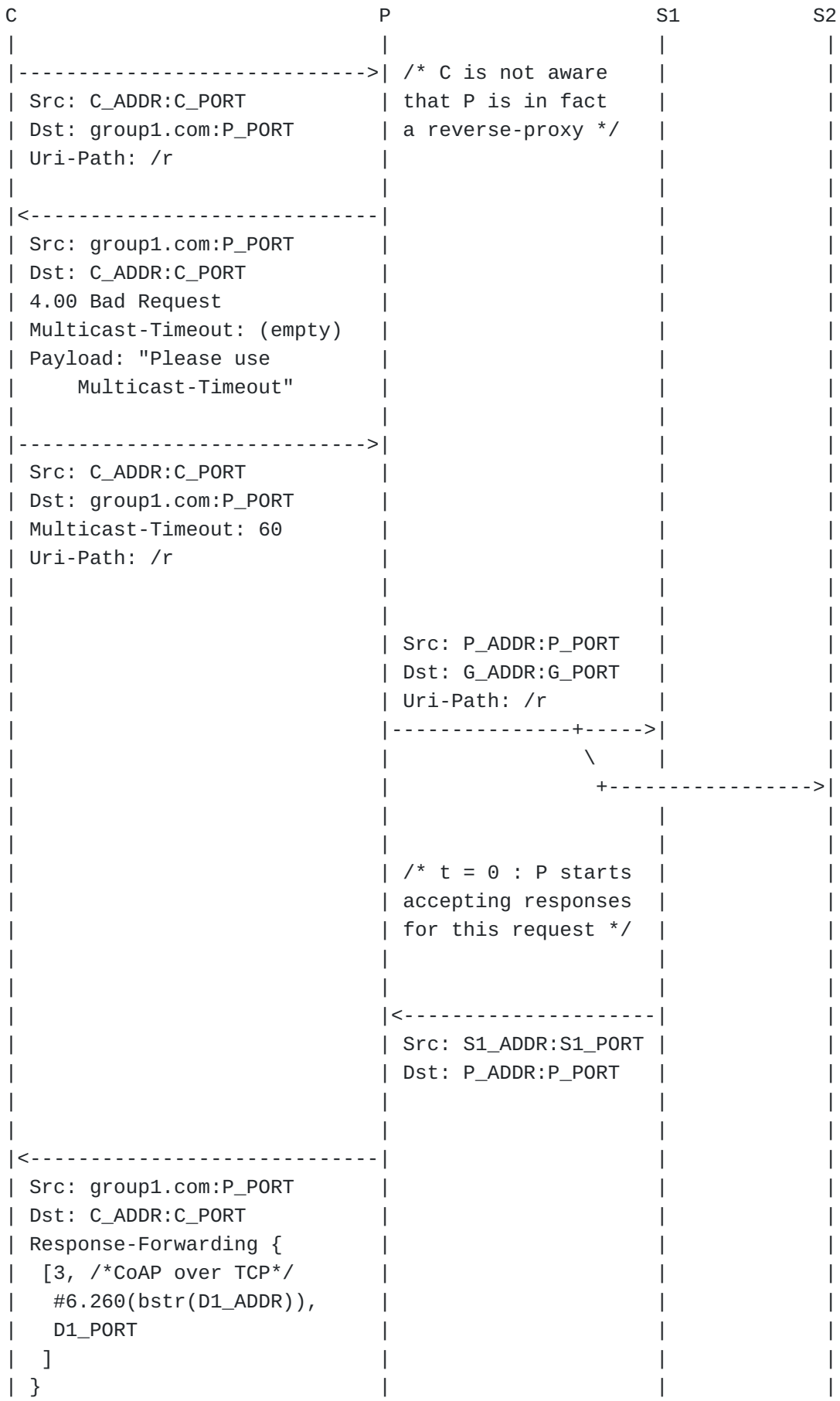
The example shown in Figure 6 considers a reverse-proxy that stands in for both the whole group of servers {S1,S2} and for each of those servers Sx. The client C may not have a way to reach the servers directly (e.g., P is acting as a firewall). After the client C has received two responses to its group request sent via the proxy, it selects one server (S1) and requests at a later time the same resource from it in unicast, again via the proxy.

In particular:

- The hostname 'group1.com' resolves to the unicast address P_ADDR. The proxy forwards an incoming request to that address, for any resource i.e., URI path, towards the CoAP group at G_ADDR:G_PORT leaving the URI path unchanged.

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

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


C
P
S1
S2
/* C is not aware that P is in fact a reverse-proxy */
Src: C_ADDR:C_PORT
Dst: group1.com:P_PORT
Uri-Path: /r
Src: group1.com: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: group1.com:P_PORT
Multicast-Timeout: 60
Uri-Path: /r
Src: P_ADDR:P_PORT
Dst: G_ADDR:G_PORT
Uri-Path: /r
/* t = 0 : P starts accepting responses for this request */
Src: S1_ADDR:S1_PORT
Dst: P_ADDR:P_PORT
Src: group1.com:P_PORT
Dst: C_ADDR:C_PORT
Response-Forwarding {
[3, /*CoAP over TCP*/
#6.260(bstr(D1_ADDR)),
D1_PORT
]
}

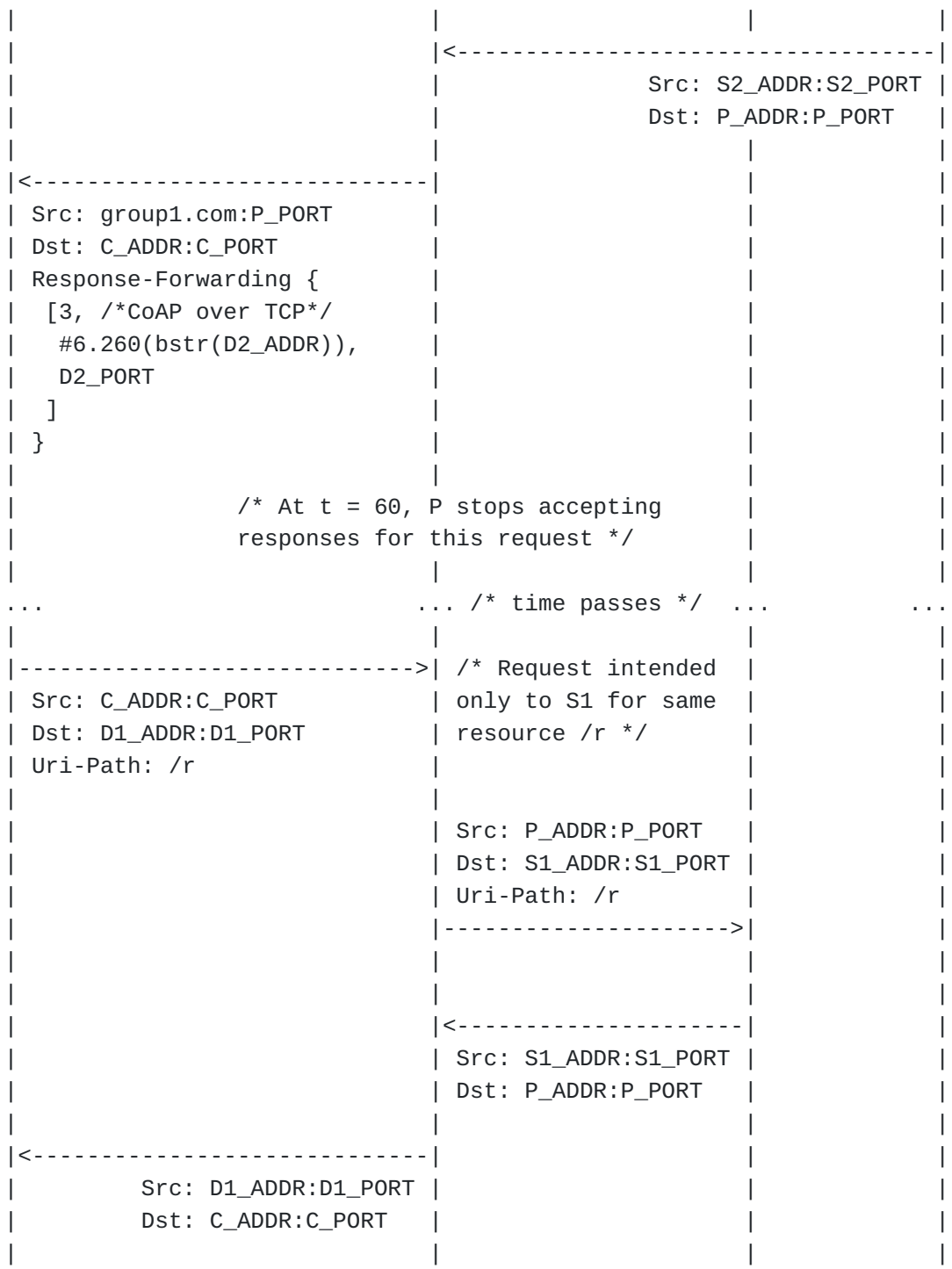

Src: S2_ADDR:S2_PORT
Dst: P_ADDR:P_PORT
Src: group1.com:P_PORT
Dst: C_ADDR:C_PORT
Response-Forwarding {
[3, /*CoAP over TCP*/
#6.260(bstr(D2_ADDR)),
D2_PORT
]
}
/* At t = 60, P stops accepting
responses for this request */
... /* time passes */ ...
Src: C_ADDR:C_PORT
Dst: D1_ADDR:D1_PORT
Uri-Path: /r
/* Request intended
only to S1 for same
resource /r */
Src: P_ADDR:P_PORT
Dst: S1_ADDR:S1_PORT
Uri-Path: /r
Src: S1_ADDR:S1_PORT
Dst: P_ADDR:P_PORT
Src: D1_ADDR:D1_PORT
Dst: C_ADDR:C_PORT

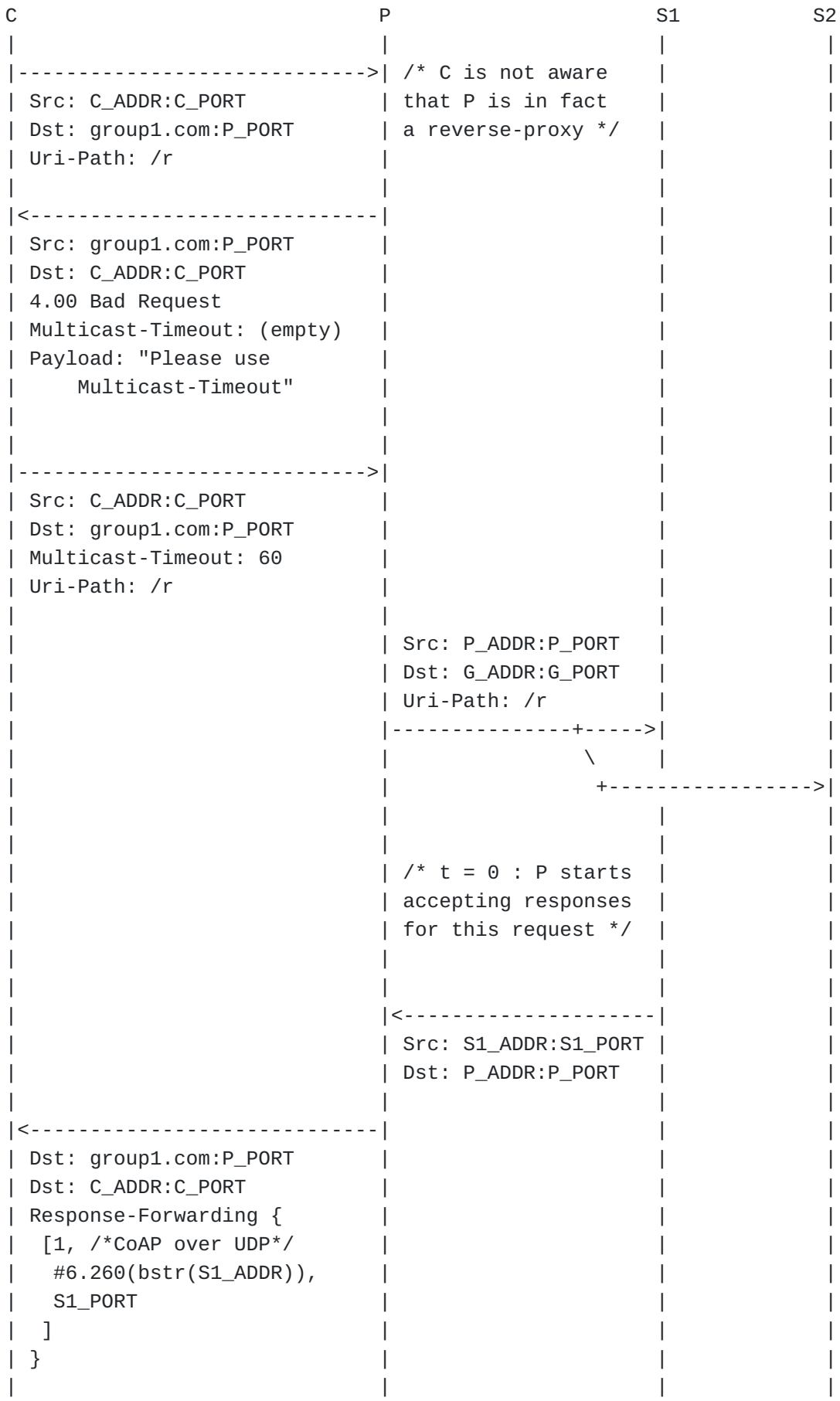
Figure 6: Workflow example with reverse-proxy standing in for both the whole group of servers and each individual server

A.3. Example 3

The example shown in Figure 7 builds on the example in Appendix A.2.

However, it considers a reverse-proxy that stands in for only the whole group of servers, but not for each individual server Sx.

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: group1.com:P_PORT
Uri-Path: /r
Src: group1.com: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: group1.com:P_PORT
Multicast-Timeout: 60
Uri-Path: /r
Src: P_ADDR:P_PORT
Dst: G_ADDR:G_PORT
Uri-Path: /r
/* t = 0 : P starts accepting responses for this request */
Src: S1_ADDR:S1_PORT
Dst: P_ADDR:P_PORT
Dst: group1.com:P_PORT
Dst: C_ADDR:C_PORT
Response-Forwarding {
[1, /*CoAP over UDP*/
#6.260(bstr(S1_ADDR)),
S1_PORT
]
}

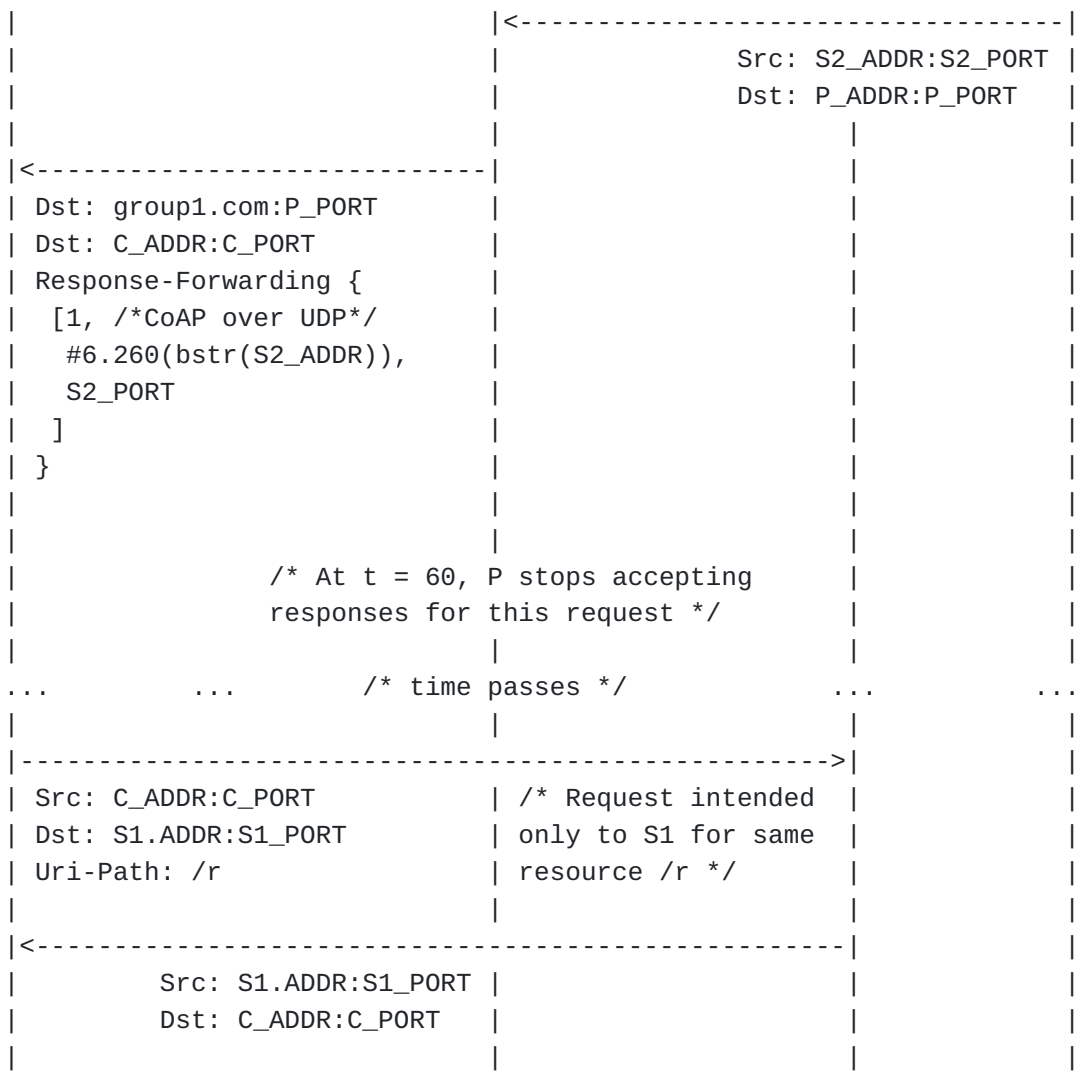

Src: S2_ADDR:S2_PORT
Dst: P_ADDR:P_PORT
Dst: group1.com:P_PORT
Dst: C_ADDR:C_PORT
Response-Forwarding {
[1, /*CoAP over UDP*/
#6.260(bstr(S2_ADDR)),
S2_PORT
]
}
/* At t = 60, P stops accepting
responses for this request */
/* time passes */
Src: C_ADDR:C_PORT
Dst: S1.ADDR:S1_PORT
Uri-Path: /r
/* Request intended
only to S1 for same
resource /r */
Src: S1.ADDR:S1_PORT
Dst: C_ADDR:C_PORT

Figure 7: Workflow example with reverse-proxy standing in for only the whole group of servers, but not for each individual server

Acknowledgments

The authors sincerely thank Christian Amsuess, Jim Schaad and Goeran Selander for their comments and feedback.

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

Authors' Addresses

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

Email: marco.tiloca@ri.se

Esko Dijk
IoTconsultancy.nl
_____\
Utrecht

Email: esko.dijk@iotconsultancy.nl