Workgroup: CoRE Working Group
Internet-Draft:
draft-ietf-core-observe-multicast-
notifications-08
Updates: 7252, 7641 (if approved)
Published: 4 March 2024
Intended Status: Standards Track
Expires: 5 September 2024
Authors: M. Tiloca RISE AB
R. Höglund RISE AB
C. Amsüss
F. Palombini Ericsson AB

# Observe Notifications as CoAP Multicast Responses

## Abstract

The Constrained Application Protocol (CoAP) allows clients to "observe" resources at a server, and receive notifications as unicast responses upon changes of the resource state. In some use cases, such as based on publish-subscribe, it would be convenient for the server to send a single notification addressed to all the clients observing a same target resource. This document updates RFC7252 and RFC7641, and defines how a server sends observe notifications as response messages over multicast, synchronizing all the observers of a same resource on a same shared Token value. Besides, this document defines how Group OSCORE can be used to protect multicast notifications end-to-end between the server and the observer clients.

## Discussion Venues

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

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

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

## Status of This Memo

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

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

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

This Internet-Draft will expire on 5 September 2024.

## Copyright Notice



## Table of Contents

## 1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] has been extended with a number of mechanisms, including resource Observation [RFC7641]. This enables CoAP clients to register at a CoAP server as "observers" of a resource, and hence being automatically notified with an unsolicited response upon changes of the resource state.

CoAP supports group communication [I-D.ietf-core-groupcomm-bis], e.g., over IP multicast. This includes support for Observe registration requests over multicast, in order for clients to efficiently register as observers of a resource hosted at multiple servers.

However, in a number of use cases, using multicast messages for responses would also be desirable. That is, it would be useful that a server sends observe notifications for a same target resource to multiple observers as responses over IP multicast.

For instance, in CoAP publish-subscribe [I-D.ietf-core-coap-pubsub], multiple clients can subscribe to a topic, by observing the related resource hosted at the responsible broker. When a new value is published on that topic, it would be convenient for the broker to send a single multicast notification at once, to all the subscriber clients observing that topic.

A different use case concerns clients observing a same registration resource at the CoRE Resource Directory [RFC9176]. For example, multiple clients can benefit of observation for discovering (to-be-created) OSCORE groups [I-D.ietf-core-oscore-groupcomm], by retrieving from the Resource Directory updated links and descriptions to join them through the respective Group Manager [I-D.tiloca-core-oscore-discovery].

More in general, multicast notifications would be beneficial whenever several CoAP clients observe a same target resource at a CoAP server, and can be all notified at once by means of a single response message. However, CoAP does not currently define response messages addressed to multiple clients, e.g., over IP multicast.

This document fills this gap and provides the following twofold contribution.

First, it updates [RFC7252] and [RFC7641], by defining a method to deliver Observe notifications as CoAP responses addressed to multiple clients, e.g., over IP multicast. In the proposed method, the group of potential observers entrusts the server to manage the Token space for multicast notifications. By doing so, the server provides all the observers of a target resource with the same Token value to bind to their own observation. That Token value is then used in every multicast notification for the target resource. This is achieved by means of a unicast informative response sent by the server to each observer client.

Second, this document defines how to use Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect multicast notifications end-to-end between the server and the observer clients. This is also achieved by means of the unicast informative response mentioned above, which additionally includes parameter values used by the server to protect every multicast notification for the target resource by using Group OSCORE. This provides a secure binding between each of such notifications and the observation of each of the clients.

### 1.1. Terminology

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

Readers are expected to be familiar with terms and concepts described in CoAP [RFC7252], group communication for CoAP [I-D.ietf-core-groupcomm-bis], Observe [RFC7641], CDDL [RFC8610], CBOR [RFC8949], OSCORE [RFC8613], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This document additionally defines the following terminology.

- Traditional observation. A resource observation associated with a single observer client, as defined in [RFC7641].

- Group observation. A resource observation associated with a group of clients. The server sends notifications for the group-observed resource over IP multicast to all the observer clients.

- Phantom request. The CoAP request message that the server would have received to start a group observation on one of its

resources. A phantom request is generated inside the server and does not hit the wire.

* Informative response. A CoAP response message that the server sends to a given client via unicast, providing the client with information on a group observation.

## 2. Prerequisites

In order to use multicast notifications as defined in this document, the following prerequisites have to be fulfilled.

* The server and the clients need to be on a network on which multicast notifications can reach a sufficiently large portion of the clients. These may leverage intermediaries such as proxies, if some clients are not able to directly listen to multicast traffic.

* The server needs to be provisioned with multicast addresses whose token space is placed under its control. On general purpose networks, unmanaged multicast addresses such as "All CoAP Nodes" (see Section 12.8 of [RFC7252]) are not suitable for this purpose.

* The server and the clients need to agree out of band that multicast notifications may be used.

  This document does not describe a way for a client to influence the server's decision to start group observations and thus to use multicast notifications. This is done on purpose.

  That is, the mechanism specified in this document is expected to be used in situations where sending individual notifications is not feasible, or is not preferred beyond a certain number of clients observing a target resource.

  If applications arise where a negotiation between the clients and the server does make sense, those applications are welcome to specify additional means for clients to opt in for receiving multicast notifications.

## 3. High-Level Overview of Available Variants

The method defined in this document fundamentally enables a server to set up a group observation. This is associated with a phantom observation request generated by the server, and to which the multicast notifications of the group observation are bound.

While the server can provide the phantom request in question to the interested clients as they reach out for registering to the group

observation, the server may alternatively distribute the phantom request in advance by alternative means (e.g., see Appendix A). Clients that have already retrieved the phantom request can immediately start listening to multicast notifications if able to directly do so, or rather instruct an assisting intermediary such as a proxy to do that on their behalf.

The following provides an overview of the available variants to enforce a group observation, depending on whether a proxy is deployed or not, and on whether exchanged messages are protected end-to-end between the observer clients and the server.

* Without proxy - This is the simplest network configuration, where the clients participating in the group observation are capable to listen to multicast traffic. In such a setup, the clients directly receive multicast notifications from the server.

  - Without end-to-end security - Messages pertaining to the group observation are not protected. This basic case is defined in Section 4 and Section 5 from the server and the client side, respectively. An example is provided in Section 7.

  - With end-to-end security - Messages pertaining to the group observation are protected end-to-end between the clients and the server, by using the Group OSCORE security protocol [I-D.ietf-core-oscore-groupcomm]. This case is defined in Section 9. An example is provided in Section 10.

    If the participating endpoints using Group OSCORE also support the concept of Deterministic Client [I-D.amsuess-core-cachable-oscore], then the possible early distribution of the phantom request can specifically make available its smaller, plain version. Then, all the clients are able to compute the same protected phantom request to use (see Appendix D).

* With proxy - This network configuration is expected in case (some of) the clients participating in the group observation are not capable to listen to multicast traffic. In such a setup, the proxy directly receives multicast notifications from the server, and relays them back to the clients.

  - Without end-to-end security - Messages pertaining to the group observation are not protected end-to-end between the clients and the server. This basic case is defined in Section 11. An example is provided in Appendix E.

  - With end-to-end security - Messages pertaining to the group observation are protected end-to-end between the clients and the server, by using the Group OSCORE security protocol

[I-D.ietf-core-oscore-groupcomm]. In particular, the clients are required to separately provide the proxy with the obtained phantom request, thus enabling the proxy to receive the multicast notifications from the server. This case is defined in Section 12. An example is provided in Appendix F.

If the participating endpoints using Group OSCORE also support the concept of Deterministic Client [I-D.amsuess-core-cachable-oscore], the same advantages mentioned above for the case without a proxy applies (see Appendix D). In addition, this allows for a more efficient setup and enforcement of the group observation, by reducing the amount of message exchanges and allowing the proxy to effectively serve protected multicast notifications from its cache. An example is provided in Appendix G.2.

## 4. Server-Side

The server can, at any time, start a group observation on one of its resources. Practically, the server may want to do that under the following circumstances.

* In the absence of observations for the target resource, the server receives a registration request from a first client wishing to start a traditional observation on that resource.

* When a certain amount of traditional observations has been established on the target resource, the server decides to make the corresponding observer clients part of a group observation on that resource.

The server maintains an observer counter for each group observation to a target resource, as a rough estimation of the observers actively taking part in the group observation.

The server initializes the counter to 0 when starting the group observation, and increments it after a new client starts taking part in that group observation. Also, the server should keep the counter up-to-date over time, for instance by using the method described in Section 8. This allows the server to possibly terminate a group observation in case, at some point in time, not enough clients are estimated to be still active and interested.

### 4.1. Request

Assuming that the server is reachable at the address SRV_ADDR and port number SRV_PORT, the server starts a group observation on one of its resources as defined below. The server intends to send

multicast notifications for the target resource to the multicast IP address GRP_ADDR and port number GRP_PORT.

1. The server builds a phantom observation request, i.e., a GET request with an Observe Option set to 0 (register).

2. The server selects an available value T, from the Token space of a CoAP endpoint used for messages having:

   * As source address and port number, the IP multicast address GRP_ADDR and port number GRP_PORT.

   * As destination address and port number, the server address SRV_ADDR and port number SRV_PORT, intended for accessing the target resource.

   This Token space is under exclusive control of the server.

3. The server processes the phantom observation request above, without transmitting it on the wire. The request is addressed to the resource for which the server wants to start the group observation, as if sent by the group of observers, i.e., with GRP_ADDR as source address and GRP_PORT as source port.

4. Upon processing the self-generated phantom registration request, the server interprets it as an observe registration received from the group of potential observer clients. In particular, from then on, the server MUST use T as its own local Token value associated with that observation, with respect to the (previous hop towards the) clients.

5. The server does not immediately respond to the phantom observation request with a multicast notification sent on the wire. The server stores the phantom observation request as is, throughout the lifetime of the group observation.

6. The server builds a CoAP response message INIT_NOTIF as initial multicast notification for the target resource, in response to the phantom observation request. This message is formatted as other multicast notifications (see Section 4.3) and MUST include the current representation of the target resource as payload. The server stores the message INIT_NOTIF and does not transmit it. The server considers this message as the latest multicast notification for the target resource, until it transmits a new multicast notification for that resource as a CoAP message on the wire. After that, the server deletes the message INIT_NOTIF.

### 4.2. Informative Response

After having started a group observation on a target resource, the server proceeds as follows.

For each traditional observation ongoing on the target resource, the server MAY cancel that observation. Then, the server considers the corresponding clients as now taking part in the group observation, for which it increases the corresponding observer counter accordingly.

The server sends to each of such clients an informative response message, encoded as a unicast response with response code 5.03 (Service Unavailable). As per [RFC7641], such a response does not include an Observe Option. The response MUST be Confirmable and MUST NOT encode link-local addresses.

The Content-Format of the informative response is set to application/informative-response+cbor, defined in Section 16.2. The payload of the informative response is a CBOR map including the following parameters, whose CBOR labels are defined in Section 13.

* 'tp_info', with value a CBOR array. This includes the transport-specific information required to correctly receive multicast notifications bound to the phantom observation request. Typically, this comprises the Token value associated with the group observation, as well as the source and destination addressing information of the related multicast notifications. The CBOR array is formatted as defined in Section 4.2.1. This parameter MUST be included.

* 'ph_req', with value the byte serialization of the transport-independent information of the phantom observation request (see Section 4.1), encoded as a CBOR byte string. The value of the CBOR byte string is formatted as defined in Section 4.2.2.

  This parameter MAY be omitted, in case the phantom request is, in terms of transport-independent information, identical to the registration request from the client. Otherwise, this parameter MUST be included.

  Note that the registration request from the client may indeed differ from the phantom observation request in terms of transport-independent information, but still be acceptable for the server to register the client as taking part in the group observation.

* 'last_notif', with value the byte serialization of the transport-independent information of the latest multicast notification for the target resource, encoded as a CBOR byte string. The value of

the CBOR byte string is formatted as defined in Section 4.2.2. This parameter MAY be included.

* 'next_not_before', with value the amount of seconds that will minimally elapse before the server sends the next multicast notification for the group observation of the target resource, encoded as a CBOR unsigned integer. This parameter MAY be included.

  This information can help a new client to align itself with the server's timeline, especially in scenarios where multicast notifications are regularly sent. Also, it can help synchronizing different clients when orchestrating a content distribution through multicast notifications.

The CDDL notation [RFC8610] provided below describes the payload of the informative response.

```
informative_response_payload = {
   0 => array, ; 'tp_info' (transport-specific information)
 ? 1 => bstr,  ; 'ph_req' (transport-independent information)
 ? 2 => bstr   ; 'last_notif' (transport-independent information)
 ? 3 => uint   ; 'next_not_before'
}
```

Figure 1: Format of the informative response payload

Upon receiving a registration request to observe the target resource, the server does not create a corresponding individual observation for the requesting client. Instead, the server considers that client as now taking part in the group observation of the target resource, of which it increments the observer counter by 1. Then, the server replies to the client with the same informative response message defined above, which MUST be Confirmable.

Note that this also applies when, with no ongoing traditional observations on the target resource, the server receives a registration request from a first client and decides to start a group observation on the target resource.

#### 4.2.1. Transport-Specific Message Information

[ This encoding might be replaced by CRIs [I-D.ietf-core-href] in a later version of this document. ]

The CBOR array specified in the 'tp_info' parameter is formatted according to the following CDDL notation.

```
tp_info = [
    srv_addr  ; Addressing information of the server
  ? req_info  ; Request data extension
]

srv_addr = (
    tp_id : int,  ; Identifier of the used transport protocol
  * elements      ; Number, format, and encoding
                  ; based on the value of 'tp_id'
)

req_info = (
  + elements  ; Number, format, and encoding based on
              ; the value of 'tp_id' in 'srv_addr'
)
```

Figure 2: General format of 'tp_info'

The 'srv_addr' element of 'tp_info' specifies the addressing information of the server, i.e., the source addressing information of the multicast notifications that are sent for the group observation. Such addressing information MUST be equal to the destination addressing information of the registration requests sent by the clients to observe the target resource at the server.

The 'srv_addr' element includes at least one inner element 'tp_id', which is formatted as follows.

* 'tp_id' : this element is a CBOR integer, which specifies the transport protocol used to transport the CoAP response from the server, i.e., a multicast notification in this document.

  This element takes value from the "Value" column of the "CoAP Transport Information" registry defined in Section 16.5 of this document. This element MUST be present. The value of this element determines:

  - How many elements are required to follow in 'srv_addr', as well as what information they convey, their encoding, and their semantics.

  - How many elements are required in the 'req_info' element of the 'tp_info' array, as well as what information they convey, their encoding, and their semantics.

  This document registers the integer value 1 ("UDP") to be used as value for the 'tp_id' element, when CoAP responses are transported over UDP. In such a case, the full encoding of the 'tp_info' CBOR array is as defined in Section 4.2.1.1.

Future specifications that consider CoAP multicast notifications transported over different transport protocols MUST:

- Register an entry with an integer value to be used for 'tp_id', in the "CoAP Transport Information" registry defined in Section 16.5 of this document.

- Accordingly, define the elements of the 'tp_info' CBOR array, i.e., the elements following 'tp_id' in 'srv_addr' as well as the elements in 'req_info', as to what information they convey, their encoding, and their semantics.

The 'req_info' element of 'tp_info' specifies transport-specific information related to a pertinent request message, i.e., the phantom observation request in this document. The exact format of 'req_info' depends on the value of 'tp_id'.

Given a specific value of 'tp_id', the complete set of elements composing 'srv_addr' and 'req_info' in the 'tp_info' CBOR array is indicated by the two columns "Srv Addr" and "Req Info" of the "CoAP Transport Information" registry defined in Section 16.5, respectively.

#### 4.2.1.1. UDP Transport-Specific Information

When CoAP multicast notifications are transported over UDP as per [RFC7252] and [I-D.ietf-core-groupcomm-bis], the server specifies the integer value 1 ("UDP") as value of 'tp_id' in the 'srv_addr' element of the 'tp_info' CBOR array in the informative response. Then, the rest of the 'tp_info' CBOR array is defined as follows.

* 'srv_addr' includes two more elements following 'tp_id':

  - 'srv_host': this element is a CBOR byte string, with value the destination IP address of the phantom observation request. This parameter is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)". That is, the value of the CBOR byte string is the IP address SRV_ADDR of the server hosting the target resource, from where the server will send multicast notifications for the target resource. This element MUST be present.

  - 'srv_port': this element is a CBOR unsigned integer, with value the destination port number of the phantom observation request. That is, the specified value is the port number SRV_PORT, from where the server will send multicast notifications for the target resource. This element MUST be present.

*'req_info' includes the following elements:

- 'token': this element is a CBOR byte string, with value the Token value of the phantom observation request generated by the server (see Section 4.1). Note that the same Token value is used for the multicast notifications bound to that phantom observation request (see Section 4.3). This element MUST be present.

- 'cli_host': this element is a CBOR byte string, with value the source IP address of the phantom observation request. This parameter is tagged and identified by the CBOR tag 260 "Network Address (IPv4 or IPv6 or MAC Address)". That is, the value of the CBOR byte string is the IP multicast address GRP_ADDR, where the server will send multicast notifications for the target resource. This element MUST be present.

- 'cli_port': this element is a CBOR unsigned integer, with value the source port number of the phantom observation request. That is, the specified value is the port number GRP_PORT, where the server will send multicast notifications for the target resource. This element is OPTIONAL. If not included, the default port number 5683 is assumed.

The CDDL notation provided below describes the full 'tp_info' CBOR array using the format above.

```
tp_info = [
    tp_id    : 1,             ; UDP as transport protocol
    srv_host : #6.260(bstr),  ; Src. address of multicast notifications
    srv_port : uint,          ; Src. port of multicast notifications
    token    : bstr,          ; Token of the phantom request and
                              ; associated multicast notifications
    cli_host : #6.260(bstr),  ; Dst. address of multicast notifications
  ? cli_port : uint           ; Dst. port of multicast notifications
]
```

Figure 3: Format of 'tp_info' with UDP as transport protocol

#### 4.2.2. Transport-Independent Message Information

For both the parameters 'ph_req' and 'last_notif' in the informative response, the value of the byte string is the concatenation of the following components, in the order specified below.

When defining the value of each component, "CoAP message" refers to the phantom observation request for the 'ph_req' parameter, and to

the corresponding latest multicast notification for the 'last_notif' parameter.

- A single byte, with value the content of the Code field in the CoAP message.

- The byte serialization of the complete sequence of CoAP options in the CoAP message.

- If the CoAP message includes a non-zero length payload, the one-byte Payload Marker (0xff) followed by the payload.

### 4.3. Notifications

Upon a change in the status of the target resource under group observation, the server sends a multicast notification, intended to all the clients taking part in the group observation of that resource. In particular, each of such multicast notifications is formatted as follows.

- It MUST be Non-confirmable.

- It MUST include an Observe Option, as per [RFC7641].

- It MUST have the same Token value T of the phantom registration request that started the group observation. This Token value is specified in the 'token' element of 'req_info' under the 'tp_info' parameter, in the informative response sent to all the observer clients.

  That is, every multicast notification for a target resource is not bound to the observation requests from the different clients, but rather to the phantom registration request associated with the whole set of clients taking part in the group observation of that resource.

- It MUST be sent from the same IP address SRV_ADDR and port number SRV_PORT where: i) the original Observe registration requests are sent to by the clients; and ii) the corresponding informative responses are sent from by the server (see Section 4.2). These are indicated to the observer clients as value of the 'srv_host' and 'srv_port' elements of 'srv_addr' under the 'tp_info' parameter, in the informative response (see Section 4.2.1.1). That is, redirection MUST NOT be used.

- It MUST be sent to the IP multicast address GRP_ADDR and port number GRP_PORT. These are indicated to the observer clients as value of the 'cli_host' and 'cli_port' elements of 'req_info' under the 'tp_info' parameter, in the informative response (see Section 4.2.1.1).

For each target resource with an active group observation, the server MUST store the latest multicast notification.

### 4.4. Congestion Control

In order to not cause congestion, the server should conservatively control the sending of multicast notifications. In particular:

- The multicast notifications MUST be Non-confirmable.

- In constrained environments such as low-power, lossy networks (LLNs), the server should only support multicast notifications for resources that are small. Following related guidelines from Section 3.6 of [I-D.ietf-core-groupcomm-bis], this can consist, for example, in having the payload of multicast notifications as limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size, so that it fits into a single link-layer frame in case IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) (see Section 4 of [RFC4944]) is used.

- The server SHOULD provide multicast notifications with the smallest possible IP multicast scope that fulfills the application needs. For example, following related guidelines from Section 3.6 of [I-D.ietf-core-groupcomm-bis], site-local scope is always preferred over global scope IP multicast, if this fulfills the application needs. Similarly, realm-local scope is always preferred over site-local scope, if this fulfills the application needs. Ultimately, it is up to the server administrator to explicitly configure the most appropriate IP multicast scope.

- Following related guidelines from Section 4.5.1 of [RFC7641], the server SHOULD NOT send more than one multicast notification every 3 seconds, and SHOULD use an even less aggressive rate when possible (see also Section 3.1.2 of [RFC8085]). The transmission rate of multicast notifications should also take into account the avoidance of a possible "broadcast storm" problem [MOBICOM99]. This prevents a following, considerable increase of the channel load, whose origin would be likely attributed to a router rather than the server.

### 4.5. Cancellation

At any point in time, the server may want to cancel a group observation of a target resource. For instance, the server may realize that no clients or not enough clients are interested in taking part in the group observation anymore. A possible approach that the server can use to assess this is defined in Section 8.

In order to cancel the group observation, the server sends a multicast response with response code 5.03 (Service Unavailable),

signaling that the group observation has been terminated. The response has the same Token value T of the phantom registration request, it has no payload, and it does not include an Observe Option.

The server sends the response to the same multicast IP address GRP_ADDR and port number GRP_PORT used to send the multicast notifications related to the target resource. Finally, the server releases the resources allocated for the group observation, and especially frees up the Token value T used at its CoAP endpoint.

## 5. Client-Side

### 5.1. Request

A client sends an observation request to the server as described in [RFC7641], i.e., a GET request with an Observe Option set to 0 (register). The request MUST NOT encode link-local addresses. If the server is not configured to accept registrations on that target resource specifically for a group observation, this would still result in a positive notification response to the client as described in [RFC7641], in case the server is able and willing to add the client to the list of observers.

In a particular setup, the information typically specified in the 'tp_info' parameter of the informative response (see Section 4.2) can be pre-configured on the server and the clients. For example, the destination multicast address and port number where to send multicast notifications for a group observation, as well as the associated Token value to use, can be set aside for particular tasks (e.g., enforcing observations of a specific resource). Alternative mechanisms can rely on using some bytes from the hash of the observation request as the last bytes of the multicast address or as part of the Token value.

In such a particular setup, the client may also have an early knowledge of the phantom request, i.e., it will be possible for the server to safely omit the parameter 'ph_req' from the informative response to the observation request (see Section 4.2). In this case, the client can include a No-Response Option [RFC7967] with value 16 in its Observe registration request, which results in the server suppressing the informative response. As a consequence, the observation request only informs the server that there is one additional client interested to take part in the group observation.

While the considered client is able to simply set up its multicast address and start receiving multicast notifications for the group observation, sending an observation request as above allows the server to increment the observer counter. This helps the server to

assess the current number of clients interested in the group observation over time (e.g., by using the method in Section 8), which in turn can play a role in deciding to cancel the group observation.

### 5.2. Informative Response

Upon receiving the informative response defined in Section 4.2, the client proceeds as follows.

1. The client configures an observation of the target resource. To this end, it relies on a CoAP endpoint used for messages having:

   * As source address and port number, the server address SRV_ADDR and port number SRV_PORT intended for accessing the target resource. These are specified as value of the 'srv_host' and 'srv_port' elements of 'srv_addr' under the 'tp_info' parameter, in the informative response (see Section 4.2.1.1).

   * As destination address and port number, the IP multicast address GRP_ADDR and port number GRP_PORT. These are specified as value of the 'cli_host' and 'cli_port' elements of 'req_info' under the 'tp_info' parameter, in the informative response (see Section 4.2.1.1). If the 'cli_port' element is omitted in 'req_info', the client MUST assume the default port number 5683 as GRP_PORT.

2. The client rebuilds the phantom registration request as follows.

   * The client uses the Token value T, specified in the 'token' element of 'req_info' under the 'tp_info' parameter of the informative response.

   * If the 'ph_req' parameter is not present in the informative response, the client uses the transport-independent information from its original Observe registration request.

   * If the 'ph_req' parameter is present in the informative response, the client uses the transport-independent information specified in the parameter.

3. If the informative response includes the parameter 'ph_req', and the transport-independent information specified therein differs from the one in the original Observe registration request, then the client checks whether a response to the rebuilt phantom request can, if available in a cache entry, be used to satisfy the original observation request. If this is

not the case, the client SHOULD explicitly withdraw from the group observation.

4. The client stores the phantom registration request, as associated with the observation of the target resource. In particular, the client MUST use the Token value T of this phantom registration request as its own local Token value associated with that group observation, with respect to the server. The particular way to achieve this is implementation specific.

5. If the informative response includes the parameter 'last_notif', the client rebuilds the latest multicast notification, by using:

   * The transport-independent information, specified in the 'last_notif' parameter of the informative response.

   * The Token value T, specified in the 'token' element of 'req_info' under the 'tp_info' parameter of the informative response.

6. If the informative response includes the parameter 'last_notif', the client processes the multicast notification rebuilt at step 5 as defined in Section 3.2 of [RFC7641]. In particular, the value of the Observe Option is used as initial baseline for notification reordering in this group observation.

7. If a traditional observation to the target resource is ongoing, the client MAY silently cancel it without notifying the server.

If any of the expected fields in the informative response are not present or malformed, the client MAY try sending a new registration request to the server (see Section 5.1). Otherwise, the client SHOULD explicitly withdraw from the group observation.

Appendix A describes possible alternative ways for clients to retrieve the phantom registration request and other information related to a group observation.

### 5.3. Notifications

After having successfully processed the informative response as defined in Section 5.2, the client will receive, accept, and process multicast notifications about the state of the target resource from the server, as responses to the phantom registration request and with Token value T.

The client relies on the value of the Observe Option for notification reordering, as defined in Section 3.4 of [RFC7641].

### 5.4. Cancellation

At a certain point in time, a client may become not interested in receiving further multicast notifications about a target resource. When this happens, the client can simply "forget" about being part of the group observation for that target resource, as per Section 3.6 of [RFC7641].

When, later on, the server sends the next multicast notification, the client will not recognize the Token value T in the message. Since the multicast notification is Non-confirmable, it is OPTIONAL for the client to reject the multicast notification with a Reset message, as defined in Section 3.5 of [RFC7641].

In case the server has canceled a group observation as defined in Section 4.5, the client simply forgets about the group observation and frees up the used Token value T for that endpoint, upon receiving the multicast error response defined in Section 4.5.

## 6. Web Linking

The possible use of multicast notifications in a group observation may be indicated by a target attribute "gp-obs" in a web link [RFC8288] to a resource, e.g., using a link-format document [RFC6690].

The "gp-obs" attribute is a hint, indicating that the server might send multicast notifications for observations of the resource targeted by the link. Note that this is simply a hint, i.e., it does not include any information required to participate in a group observation, and to receive and process multicast notifications.

A value MUST NOT be given for the "gp-obs" attribute and any present value MUST be ignored by the recipient. The "gp-obs" attribute MUST NOT appear more than once in a given link-value; occurrences after the first MUST be ignored by the recipient.

The example in Figure 4 shows a use of the "gp-obs" attribute: the client does resource discovery on a server and gets back a list of resources, one of which includes the "gp-obs" attribute indicating that the server might send multicast notifications for observations of that resource. The link-format notation (see Section 5 of [RFC6690]) is used.

```
REQ: GET /.well-known/core

RES: 2.05 Content
    </sensors/temp>;gp-obs,
    </sensors/light>;if="sensor"
```

Figure 4: The Web Link

## 7. Example

The following example refers to two clients C_1 and C_2 that register to observe a resource /r at a Server S, which has address SRV_ADDR and listens to the port number SRV_PORT. Before the following exchanges occur, no clients are observing the resource /r , which has value "1234".

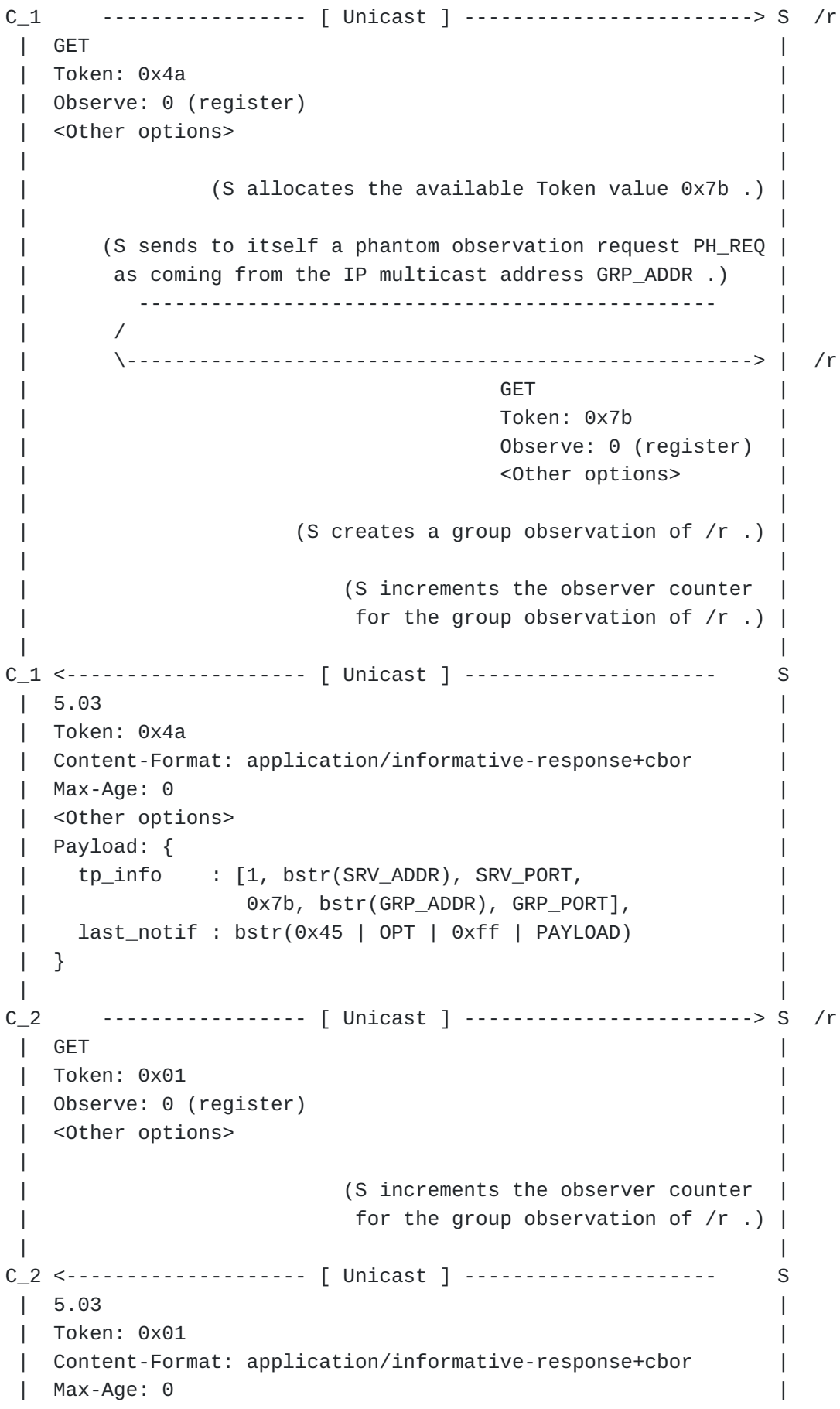
The server S sends multicast notifications to the IP multicast address GRP_ADDR and port number GRP_PORT, and starts the group observation upon receiving a registration request from a first client that wishes to start a traditional observation on the resource /r.

The following notation is used for the payload of the informative responses:

* 'bstr(X)' denotes a CBOR byte string with value the byte serialization of X, with '|' denoting byte concatenation.

* 'OPT' denotes a sequence of CoAP options. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.

* 'PAYLOAD' denotes a CoAP payload. This refers to the latest multicast notification encoded by the 'last_notif' parameter.


C_1
[ Unicast ]
S
/r
GET
Token: 0x4a
Observe: 0 (register)
<Other options>
(S allocates the available Token value 0x7b .)
(S sends to itself a phantom observation request PH_REQ as coming from the IP multicast address GRP_ADDR .)
/r
GET
Token: 0x7b
Observe: 0 (register)
<Other options>
(S creates a group observation of /r .)
(S increments the observer counter for the group observation of /r .)
C_1
[ Unicast ]
S
5.03
Token: 0x4a
Content-Format: application/informative-response+cbor
Max-Age: 0
<Other options>
Payload: {
tp_info : [1, bstr(SRV_ADDR), SRV_PORT, 0x7b, bstr(GRP_ADDR), GRP_PORT],
last_notif : bstr(0x45 | OPT | 0xff | PAYLOAD)
}
C_2
[ Unicast ]
S
/r
GET
Token: 0x01
Observe: 0 (register)
<Other options>
(S increments the observer counter for the group observation of /r .)
C_2
[ Unicast ]
S
5.03
Token: 0x01
Content-Format: application/informative-response+cbor
Max-Age: 0

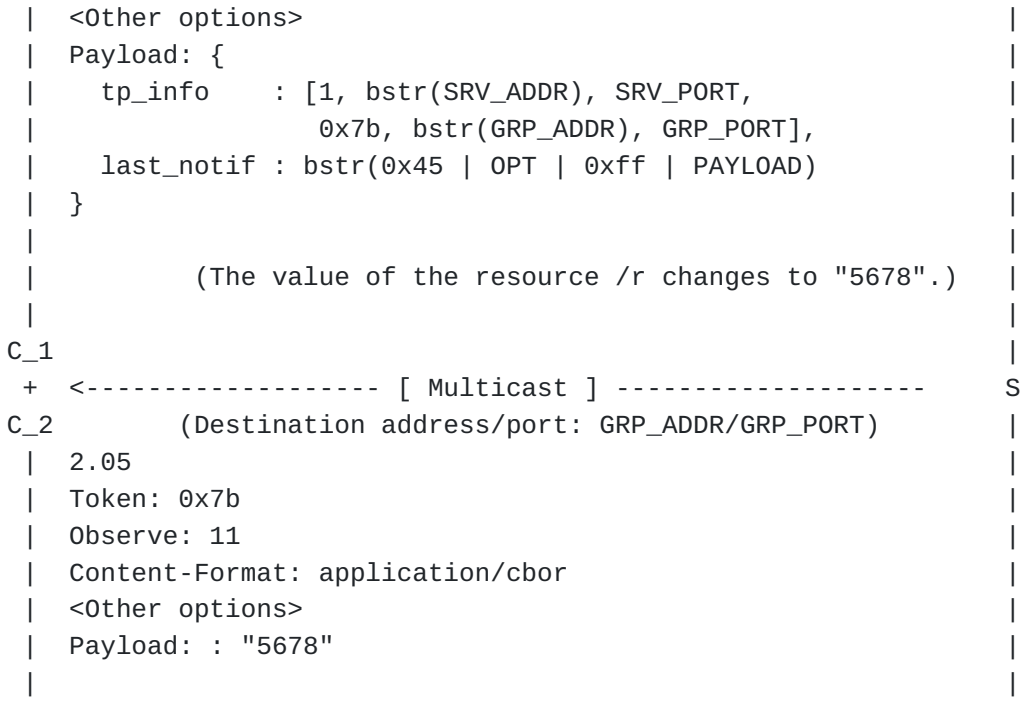



Figure 5: Example of group observation

## 8. Rough Counting of Clients in the Group Observation

This section specifies a method that the server can use to keep an estimate of still active and interested clients, without creating undue traffic on the network.

### 8.1. Multicast-Response-Feedback-Divider Option

In order to enable the rough counting of still active and interested clients, a new CoAP option is introduced, which SHOULD be supported by clients that listen to multicast responses.

The option is called Multicast-Response-Feedback-Divider. As summarized in Figure 6, the option is not Critical, not Safe-to-Forward, and integer valued. Since the option is not Safe-to-Forward, the column "N" indicates a dash for "not applicable".

| No. | C | U | N | R | Name | Format | Len. | Default |
|---|---|---|---|---|---|---|---|---|
| TBD | | x | - | | Multicast-Response-Feedback-Divider | uint | 0-1 | (none) |

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

Figure 6: Multicast-Response-Feedback-Divider

The Multicast-Response-Feedback-Divider Option is of class E for OSCORE [RFC8613][I-D.ietf-core-oscore-groupcomm].

### 8.2. Processing on the Client Side

Upon receiving a response with a Multicast-Response-Feedback-Divider Option, a client SHOULD acknowledge its interest in continuing receiving multicast notifications for the target resource, as described below.

The client picks an integer random number I, from 0 inclusive to the number Z = (2^Q) exclusive, where Q is the value specified in the option and "^" is the exponentiation operator. If I is different than 0, the client takes no further action. Otherwise, the client should wait a random fraction of the Leisure time (see Section 8.2 of [RFC7252]), and then registers a regular unicast observation on the same target resource.

To this end, the client essentially follows the steps that got it originally subscribed to group notifications for the target resource. In particular, the client sends an observation request to the server, i.e., a GET request with an Observe Option set to 0 (register). The request MUST be addressed to the same target resource, and MUST have the same destination IP address and port number used for the original registration request, regardless of the source IP address and port number of the received multicast notification.

Since the Observe registration is only done for its side effect of showing as an attempted observation at the server, the client MUST send the unicast request in a non confirmable way, and with the maximum No-Response setting [RFC7967]. In the request, the client MUST include a Multicast-Response-Feedback-Divider Option, whose value MUST be set to 0. As per Section 3.2 of [RFC7252], this is represented with an empty option value (a zero-length sequence of bytes). The client does not need to wait for responses, and can keep processing further notifications on the same Token.

The client MUST ignore the Multicast-Response-Feedback-Divider Option, if the multicast notification is retrieved from the 'last_notif' parameter of an informative response (see Section 4.2). A client includes the Multicast-Response-Feedback-Divider Option only in a re-registration request triggered by the server as described above, and MUST NOT include it in any other request.

Appendix B.1 and Appendix B.2 provide a description in pseudo-code of the operations above performed by the client.

Section 11 specifies how the approach presented in Section 4 and Section 5 works when a proxy is used between the origin clients and

the origin server. That is, the clients register as observers at the proxy, which in turn registers as a participant to the group observation at the server, receives the multicast notifications from the server, and forwards those to the clients. In such a case, the following applies.

* Since the Multicast-Response-Feedback-Divider Option is unsafe to forward, the proxy needs to recognize and understand the option in order to participate to the rough counting process.

  If the proxy receives a request that includes the Multicast-Response-Feedback-Divider Option but the proxy does not recognize and understand the option, then the proxy has to stop processing the request and send a 4.02 (Bad Option) response to the observer client (see Section 5.7.1 of [RFC7252]). This results in the client terminating its observation at the proxy, after which the client stops receiving notifications for the group observation.

  If the proxy receives a multicast notification that includes the Multicast-Response-Feedback-Divider Option but the proxy does not recognize and understand the option, then the proxy has to stop processing the received multicast notification and send a 5.02 (Bad Gateway) response to each of the observer clients (see Section 5.7.1 of [RFC7252]). This results in all the observer clients terminating their observation at the proxy, after which they stop receiving notifications for the group observation. Consequently, the proxy may decide to forget about its participation to the group observation at the server.

  This is not an issue if communications between the origin endpoints are protected end-to-end, i.e., both for the requests from the origin clients by using OSCORE or Group OSCORE, as well as for the multicast notifications from the origin server by using Group OSCORE (see Section 9 and Section 12). In fact, in such a case, the Multicast-Response-Feedback-Divider Option is protected end-to-end as well, and is thus hidden from the proxy.

  Therefore, if the server uses the rough counting process defined in this section but communications are not protected end-to-end between the origin enpoints, then it is practically required that the proxy recognizes and understands the Multicast-Response-Feedback-Divider Option. If that is not the case, then every execution of the rough counting process will effectively prevent the clients from receiving further notifications for the group observation, until they register again as observers at the proxy.

* The following applies when the proxy receives a multicast notification including the Multicast-Response-Feedback-Divider Option.

  - If the multicast notification is not protected end-to-end by using Group OSCORE (see Section 11), the Multicast-Response-Feedback-Divider Option is visible to the proxy. Then, the proxy proceeds like defined above for an origin client, i.e., by answering to the server on its own in case it picks a random number I equal to 0. When doing so, the proxy will be counted by the server as a single client.

    Instead, if the multicast notification is protected end-to-end by using Group OSCORE (see Section 9 and Section 12), then the Multicast-Response-Feedback-Divider Option is protected end-to-end as well, and is thus hidden from the proxy. As a consequence, the proxy forwards the notification to (the previous hop towards) any of the origin clients, each of which answers to the server if it picks a random number I equal to 0.

  - If the multicast notification is not protected end-to-end by using Group OSCORE, then the Multicast-Response-Feedback-Divider Option is visible to the proxy, which MUST remove the option before forwarding the notification to (the previous hop towards) any of the origin clients.

    The proxy would have to rely on separate means for asserting whether the origin clients are still interested in the observation, e.g., by regularly forwarding notifications to the clients as unicast, Confirmable response messages.

    When no interested origin clients remain, the proxy can simply forget about being part of the group observation for the target resource at the server, like an origin client would do (see Section 5.4).

### 8.3. Processing on the Server Side

In order to avoid needless use of network resources, a server SHOULD keep a rough, updated count of the number of clients taking part in the group observation of a target resource. To this end, the server updates the value COUNT of the associated observer counter (see Section 4), for instance by using the method described below.

#### 8.3.1. Request for Feedback

When it wants to obtain a new estimated count, the server considers a number M of confirmations it would like to receive from the

clients. It is up to applications to define policies about how the server determines and possibly adjusts the value of M.

Then, the server computes the value Q = max(L, 0), where:

- L is computed as L = ceil(log2(N / M)).

- N is the current value of the observer counter, possibly rounded up to 1, i.e., N = max(COUNT, 1).

Finally, the server sets Q as the value of the Multicast-Response-Feedback-Divider Option, which is sent within a successful multicast notification.

If several multicast notifications are sent in a burst fashion, it is RECOMMENDED for the server to include the Multicast-Response-Feedback-Divider Option only in the first notification of such a burst.

#### 8.3.2. Collection of Feedback

The server collects unicast observation requests from the clients, for an amount of time of MAX_CONFIRMATION_WAIT seconds. During this time, the server regularly increments the observer counter when adding a new client to the group observation (see Section 4.2).

It is up to applications to define the value of MAX_CONFIRMATION_WAIT, which has to take into account the transmission time of the multicast notification and of unicast observation requests, as well as the leisure time of the clients, which may be hard to know or estimate for the server.

If this information is not known to the server, it is recommended to define MAX_CONFIRMATION_WAIT as follows.

MAX_CONFIRMATION_WAIT = MAX_RTT + MAX_CLIENT_REQUEST_DELAY

where MAX_RTT is as defined in Section 4.8.2 of [RFC7252] and has default value 202 seconds, while MAX_CLIENT_REQUEST_DELAY is equivalent to MAX_SERVER_RESPONSE_DELAY defined in Section 3.1.5 of [I-D.ietf-core-groupcomm-bis] and has default value 250 seconds. In the absence of more specific information, the server can thus consider a conservative MAX_CONFIRMATION_WAIT of 452 seconds.

If more information is available in deployments, a much shorter MAX_CONFIRMATION_WAIT can be set. This can be based on a realistic round trip time (replacing MAX_RTT) and on the largest leisure time configured on the clients (replacing MAX_CLIENT_REQUEST_DELAY), e.g., DEFAULT_LEISURE = 5 seconds, thus shortening MAX_CONFIRMATION_WAIT to a few seconds.

#### 8.3.3. Processing of Feedback

Once MAX_CONFIRMATION_WAIT seconds have passed, the server counts the R confirmations arrived as unicast observation requests to the target resource, since the multicast notification with the Multicast-Response-Feedback-Divider Option has been sent. In particular, the server considers a unicast observation request as a confirmation from a client only if it includes a Multicast-Response-Feedback-Divider Option with value 0.

Then, the server computes a feedback indicator as E = R * (2^Q), where "^" is the exponentiation operator. According to what is defined by application policies, the server determines the next time when to ask clients for their confirmation, e.g., after a certain number of multicast notifications has been sent. For example, the decision can be influenced by the reception of no confirmations from the clients, i.e., R = 0, or by the value of the ratios (E/N) and (N/E).

Finally, the server computes a new estimated count of the observers. To this end, the server first considers COUNT' as the current value of the observer counter at this point in time. Note that COUNT' may be greater than the value COUNT used at the beginning of this process, if the server has incremented the observer counter upon adding new clients to the group observation (see Section 4.2).

In particular, the server computes the new estimated count value as COUNT' + ((E - N) / D), where D > 0 is an integer value used as dampener. This step has to be performed atomically. That is, until this step is completed, the server MUST hold the processing of an observation request for the same target resource from a new client. Finally, the server considers the result as the current observer counter, and assesses it for possibly canceling the group observation (see Section 4.5).

This estimate is skewed by packet loss, but it gives the server a sufficiently good estimation for further counts and for deciding when to cancel the group observation. It is up to applications to define policies about how the server takes the newly updated estimate into account and determines whether to cancel the group observation.

As an example, if the server currently estimates that N = COUNT = 32 observers are active and considers a constant M = 8, it sends out a notification with Multicast-Response-Feedback-Divider: 2. Then, out of 18 actually active clients, 5 send a re-registration request based on their random draw, of which one request gets lost, thus leaving 4 re-registration requests received by the server. Also, no new clients have been added to the group observation during this

time, i.e., COUNT' is equal to COUNT. As a consequence, assuming that a dampener value D = 1 is used, the server computes the new estimated count value as 32 + (16 - 32) = 16, and keeps the group observation active.

To produce a most accurate updated counter, a server can include a Multicast-Response-Feedback-Divider Option with value Q = 0 in its multicast notifications, as if M is equal to N. This will trigger all the active clients to state their interest in continuing receiving notifications for the target resource. Thus, the amount R of arrived confirmations is affected only by possible packet loss.

Appendix B.3 provides a description in pseudo-code of the operations above performed by the server, including example behaviors for scheduling the next count update and deciding whether to cancel the group observation.

## 9. Protection of Multicast Notifications with Group OSCORE

A server can protect multicast notifications by using Group OSCORE [I-D.ietf-core-oscore-groupcomm], thus ensuring that they are protected end-to-end with the observer clients. This requires that both the server and the clients interested in receiving multicast notifications from that server are members of the same OSCORE group.

In some settings, the OSCORE group to refer to can be pre-configured on the clients and the server. In such a case, a server which is aware of such pre-configuration can simply assume a client to be already member of the correct OSCORE group.

In any other case, the server MAY communicate to clients what OSCORE group they are required to join, by providing additional guidance in the informative response as described in Section 9.1. Note that clients can already be members of the right OSCORE group, in case they have previously joined it to securely communicate with the same server or with other servers to access their resources.

Both the clients and the server MAY join the OSCORE group by using the approach described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [RFC9200]. Further details on how to discover the OSCORE group and join it are out of the scope of this document.

If multicast notifications are protected using Group OSCORE, the original registration requests and related unicast (notification) responses MUST also be secured, including and especially the informative responses from the server.

To this end, alternative security protocols than Group OSCORE, such as OSCORE [RFC8613] and/or DTLS [RFC9147], can be used to protect other exchanges via unicast between the server and each client, including the original client registration (see Section 5).

### 9.1. Signaling the OSCORE Group in the Informative Response

This section describes a mechanism for the server to communicate to the client what OSCORE group to join, in order to decrypt and verify the multicast notifications protected with Group OSCORE. The client MAY use the information provided by the server to start the ACE joining procedure described in [I-D.ietf-ace-key-groupcomm-oscore]. The mechanism defined in this section is OPTIONAL to support for the client and server.

Additionally to what is defined in Section 4, the CBOR map in the informative response payload contains the following fields, whose CBOR labels are defined in Section 13.

* 'join_uri', with value the URI for joining the OSCORE group at the respective Group Manager, encoded as a CBOR text string. If the procedure described in [I-D.ietf-ace-key-groupcomm-oscore] is used for joining, this field specifically indicates the URI of the group-membership resource at the Group Manager.

* 'sec_gp', with value the name of the OSCORE group, encoded as a CBOR text string.

* Optionally, 'as_uri', with value the URI of the Authorization Server associated with the Group Manager for the OSCORE group, encoded as a CBOR text string.

* Optionally, 'hkdf', with value the HKDF Algorithm used in the OSCORE group, encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Algorithms" registry [COSE.Algorithms].

* Optionally, 'cred_fmt', with value the format of the authentication credentials used in the OSCORE group, encoded as a CBOR integer. The value is taken from the 'Label' column of the "COSE Header Parameters" Registry [COSE.Header.Parameters]. Consistently with Section 2.4 of [I-D.ietf-core-oscore-groupcomm], acceptable values denote a format that MUST explicitly provide the comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable).

  At the time of writing this specification, acceptable formats of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claim Sets (CCSs) [RFC8392], X.509 certificates [RFC5280], and

C509 certificates [I-D.ietf-cose-cbor-encoded-cert]. Further formats may be available in the future, and would be acceptable to use as long as they comply with the criteria defined above.

[ As to CWTs and unprotected CWT claim sets, there is a pending registration requested by draft-ietf-lake-edhoc. ]

[ As to C509 certificates, there is a pending registration requested by draft-ietf-cose-cbor-encoded-cert. ]

* Optionally, 'gp_enc_alg', with value the Group Encryption Algorithm used in the OSCORE group to encrypt messages protected with the group mode, encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Algorithms" registry [COSE.Algorithms].

* Optionally, 'sign_alg', with value the Signature Algorithm used to sign messages in the OSCORE group, encoded as a CBOR text string or integer. The value is taken from the 'Value' column of the "COSE Algorithms" registry [COSE.Algorithms].

* Optionally, 'sign_params', encoded as a CBOR array and including the following two elements:

  - 'sign_alg_capab': a CBOR array, with the same format and value of the COSE capabilities array for the algorithm indicated in 'sign_alg', as specified for that algorithm in the 'Capabilities' column of the "COSE Algorithms" Registry [COSE.Algorithms].

  - 'sign_key_type_capab': a CBOR array, with the same format and value of the COSE capabilities array for the COSE key type of the keys used with the algorithm indicated in 'sign_alg', as specified for that key type in the 'Capabilities' column of the "COSE Key Types" Registry [COSE.Key.Types].

The values of 'sign_alg', 'sign_params', and 'cred_fmt' provide an early knowledge of the format of authentication credentials as well as of the type of public keys used in the OSCORE group. Thus, the client does not need to ask the Group Manager for this information as a preliminary step before the (ACE) join process, or to perform a trial-and-error exchange with the Group Manager upon joining the group. Hence, the client is able to provide the Group Manager with its own authentication credential in the correct expected format and including a public key of the correct expected type, at the very first step of the (ACE) join process.

The values of 'hkdf', 'gp_enc_alg', and 'sign_alg' provide an early knowledge of the algorithms used in the OSCORE group. Thus, the

client is able to decide whether to actually proceed with the (ACE) join process, depending on its support for the indicated algorithms.

As mentioned above, since this mechanism is OPTIONAL, all the fields are OPTIONAL in the informative response. However, the 'join_uri' and 'sec_gp' fields MUST be present if the mechanism is implemented and used. If any of the fields are present without the 'join_uri' and 'sec_gp' fields present, the client MUST ignore these fields, since they would not be sufficient to start the (ACE) join procedure. When this happens, the client MAY try sending a new registration request to the server (see Section 5.1). Otherwise, the client SHOULD explicitly withdraw from the group observation.

Appendix C describes a possible alternative approach, where the server self-manages the OSCORE group, and provides the observer clients with the necessary keying material in the informative response. The approach in Appendix C MUST NOT be used together with the mechanism defined in this section for indicating what OSCORE group to join.

### 9.2. Server-Side Requirements

When using Group OSCORE to protect multicast notifications, the server performs the operations described in Section 4, with the following differences.

#### 9.2.1. Registration

The phantom registration request MUST be secured, by using Group OSCORE. In particular, the group mode of Group OSCORE defined in Section 8 of [I-D.ietf-core-oscore-groupcomm] MUST be used.

The server protects the phantom registration request as defined in Section 8.1 of [I-D.ietf-core-oscore-groupcomm], as if it was the actual sender, i.e., by using its Sender Context. As a consequence, the server consumes the current value of its Sender Sequence Number SN in the OSCORE group, and hence updates it to SN* = (SN + 1). Consistently, the OSCORE Option in the phantom registration request includes:

- As 'kid', the Sender ID of the server in the OSCORE group.

- As 'piv', the previously consumed Sender Sequence Number value SN of the server in the OSCORE group, i.e., (SN* - 1).

#### 9.2.2. Informative Response

The value of the CBOR byte string in the 'ph_req' parameter encodes the phantom observation request as a message protected with Group OSCORE (see Section 9.2.1). As a consequence: the specified Code is

always 0.05 (FETCH); the sequence of CoAP options will be limited to the outer, non encrypted options; a payload is always present, as the authenticated ciphertext followed by the signature. Note that, in terms of transport-independent information, the registration request from the client typically differs from the phantom request. Thus, the server has to include the 'ph_req' parameter in the informative response. An exception is the case discussed in Appendix D.

Similarly, the value of the CBOR byte string in the 'last_notif' parameter encodes the latest multicast notification as a message protected with Group OSCORE (see Section 9.2.3). This applies also to the initial multicast notification INIT_NOTIF built at step 6 of Section 4.1.

Optionally, the informative response includes information on the OSCORE group to join, as additional parameters (see Section 9.1).

#### 9.2.3. Notifications

The server MUST protect every multicast notification for the target resource with Group OSCORE. In particular, the group mode of Group OSCORE defined in Section 8 of [I-D.ietf-core-oscore-groupcomm] MUST be used.

The process described in Section 8.3 of [I-D.ietf-core-oscore-groupcomm] applies, with the following additions when building the two OSCORE 'external_aad' to encrypt and sign the multicast notification (see Section 4.4 of [I-D.ietf-core-oscore-groupcomm]).

- The 'request_kid' is the 'kid' value in the OSCORE Option of the phantom registration request, i.e., the Sender ID of the server.

- The 'request_piv' is the 'piv' value in the OSCORE Option of the phantom registration request, i.e., the consumed Sender Sequence Number SN of the server.

- The 'request_kid_context' is the 'kid context' value in the OSCORE Option of the phantom registration request, i.e., the Group Identifier value (Gid) of the OSCORE group used as ID Context.

Note that these same values are used to protect each and every multicast notification sent for the target resource under this group observation.

#### 9.2.4. Cancellation

When canceling a group observation (see Section 4.5), the multicast response with error code 5.03 (Service Unavailable) is also protected with Group OSCORE, as per Section 8.3 of [I-D.ietf-core-oscore-groupcomm]. The server MUST use its own Sender Sequence Number as Partial IV to protect the error response, and include it as Partial IV in the OSCORE Option of the response.

### 9.3. Client-Side Requirements

When using Group OSCORE to protect multicast notifications, the client performs as described in Section 5, with the following differences.

#### 9.3.1. Informative Response

Upon receiving the informative response from the server, the client performs as described in Section 5.2, with the following additions.

When performing step 2, the client expects the 'ph_req' parameter to be included in the informative response, which is otherwise considered malformed. An exception is the case discussed in Appendix D.

Once completed step 2, the client decrypts and verifies the rebuilt phantom registration request as defined in Section 8.2 of [I-D.ietf-core-oscore-groupcomm], with the following differences.

* The client MUST NOT perform any replay check. That is, the client skips step 3 in Section 8.2 of [RFC8613].

* If decryption and verification of the phantom registration request succeed:

  - The client MUST NOT update the Replay Window in the Recipient Context associated with the server. That is, the client skips the second bullet of step 6 in Section 8.2 of [RFC8613].

  - The client MUST NOT take any further process as normally expected according to [RFC7252]. That is, the client skips step 8 in Section 8.2 of [RFC8613]. In particular, the client MUST NOT deliver the phantom registration request to the application, and MUST NOT take any action in the Token space of its unicast endpoint, where the informative response has been received.

  - The client stores the values of the 'kid', 'piv', and 'kid context' fields from the OSCORE Option of the phantom registration request.

* If decryption and verification of the phantom registration request fail, the client MAY try sending a new registration request to the server (see Section 5.1). Otherwise, the client SHOULD explicitly withdraw from the group observation.

After successful decryption and verification, the client performs step 3 in Section 5.2, considering the decrypted phantom registration request.

If the informative response includes the parameter 'last_notif', the client also decrypts and verifies the latest multicast notification rebuilt at step 5 in Section 5.2, just like it would for the multicast notifications transmitted as CoAP messages on the wire (see Section 9.3.2). If decryption and verification succeed, the client proceeds with step 6, considering the decrypted latest multicast notification. Otherwise, the client proceeds to step 7.

### 9.3.2. Notifications

After having successfully processed the informative response as defined in Section 9.3.1, the client will decrypt and verify every multicast notification for the target resource as defined in Section 8.4 of [I-D.ietf-core-oscore-groupcomm], with the following difference.

For both decryption and signature verification, the client MUST set the 'external_aad' defined in Section 4.4 of [I-D.ietf-core-oscore-groupcomm] as follows. The particular way to achieve this is implementation specific.

* 'request_kid' takes the value of the 'kid' field from the OSCORE Option of the phantom registration request (see Section 9.3.1).

* 'request_piv' takes the value of the 'piv' field from the OSCORE Option of the phantom registration request (see Section 9.3.1).

* 'request_kid_context' takes the value of the 'kid context' field from the OSCORE Option of the phantom registration request (see Section 9.3.1).

Note that these same values are used to decrypt and verify each and every multicast notification received for the target resource.

## 10. Example with Group OSCORE

The following example refers to two clients C_1 and C_2 that register to observe a resource /r at a Server S, which has address SRV_ADDR and listens to the port number SRV_PORT. Before the following exchanges occur, no clients are observing the resource /r , which has value "1234".

The server S sends multicast notifications to the IP multicast address GRP_ADDR and port number GRP_PORT, and starts the group observation upon receiving a registration request from a first client that wishes to start a traditional observation on the resource /r.

Pairwise communication over unicast is protected with OSCORE, while S protects multicast notifications with Group OSCORE. Specifically:

- C_1 and S have a pairwise OSCORE Security Context. In particular, C_1 has 'kid' = 0x01 as Sender ID, and SN_1 = 101 as Sender Sequence Number. Also, S has 'kid' = 0x03 as Sender ID, and SN_3 = 301 as Sender Sequence Number.

- C_2 and S have a pairwise OSCORE Security Context. In particular, C_2 has 'kid' = 0x02 as Sender ID, and SN_2 = 201 as Sender Sequence Number. Also, S has 'kid' = 0x04 as Sender ID, and SN_4 = 401 as Sender Sequence Number.

- S is a member of the OSCORE group with name "myGroup", and 'kid context' = 0x57ab2e as Group ID. In the OSCORE group, S has 'kid' = 0x05 as Sender ID, and SN_5 = 501 as Sender Sequence Number.

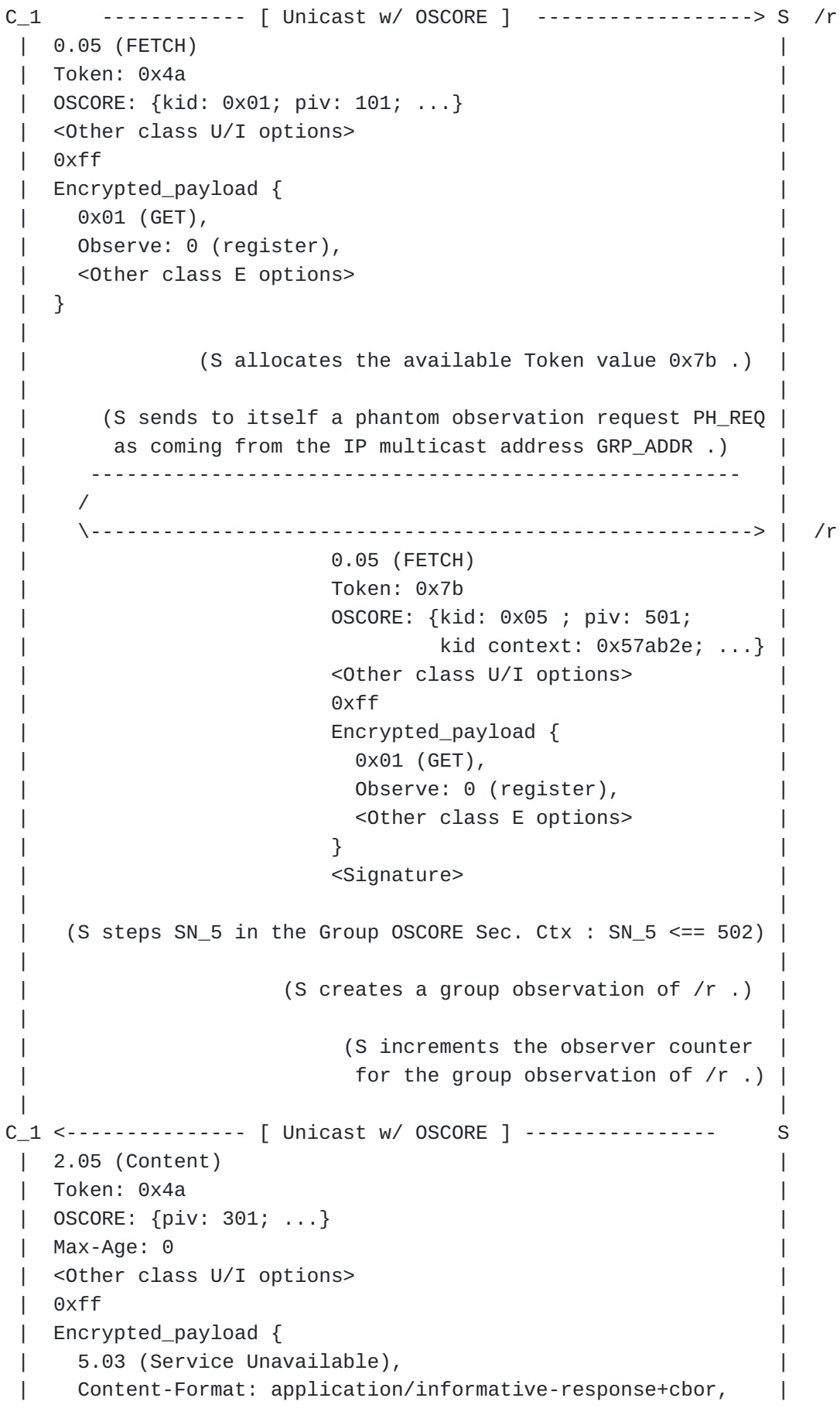
The following notation is used for the payload of the informative responses:

- 'bstr(X)' denotes a CBOR byte string with value the byte serialization of X, with '|' denoting byte concatenation.

- 'OPT' denotes a sequence of CoAP options. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.

- 'PAYLOAD' denotes an encrypted CoAP payload. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.

- 'SIGN' denotes the signature appended to an encrypted CoAP payload. This refers to the phantom registration request encoded by the 'ph_req' parameter, or to the corresponding latest multicast notification encoded by the 'last_notif' parameter.


C_1 ---------- [ Unicast w/ OSCORE ] ----------> S /r
0.05 (FETCH)
Token: 0x4a
OSCORE: {kid: 0x01; piv: 101; ...}
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
<Other class E options>
}
(S allocates the available Token value 0x7b .)
(S sends to itself a phantom observation request PH_REQ
as coming from the IP multicast address GRP_ADDR .)
/
\----------> /r
0.05 (FETCH)
Token: 0x7b
OSCORE: {kid: 0x05 ; piv: 501;
kid context: 0x57ab2e; ...}
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
<Other class E options>
}
<Signature>
(S steps SN_5 in the Group OSCORE Sec. Ctx : SN_5 <== 502)
(S creates a group observation of /r .)
(S increments the observer counter
for the group observation of /r .)
C_1 <---------- [ Unicast w/ OSCORE ] ---------- S
2.05 (Content)
Token: 0x4a
OSCORE: {piv: 301; ...}
Max-Age: 0
<Other class U/I options>
0xff
Encrypted_payload {
5.03 (Service Unavailable),
Content-Format: application/informative-response+cbor,

```
 |    <Other class E options>,                                    |
 |    0xff,                                                       |
 |    CBOR_payload {                                              |
 |      tp_info    : [1, bstr(SRV_ADDR), SRV_PORT,                |
 |                    0x7b, bstr(GRP_ADDR), GRP_PORT],            |
 |      ph_req     : bstr(0x05 | OPT | 0xff | PAYLOAD | SIGN),   |
 |      last_notif : bstr(0x45 | OPT | 0xff | PAYLOAD | SIGN),   |
 |      join_uri   : "coap://myGM/ace-group/myGroup",             |
 |      sec_gp     : "myGroup"                                    |
 |    }                                                           |
 |  }                                                             |
 |                                                                |
C_2    ------------ [ Unicast w/ OSCORE ]  ------------------> S  /r
 |  0.05 (FETCH)                                                  |
 |  Token: 0x01                                                   |
 |  OSCORE: {kid: 0x02; piv: 201; ...}                            |
 |  <Other class U/I options>                                     |
 |  0xff                                                          |
 |  Encrypted_payload {                                           |
 |    0x01 (GET),                                                 |
 |    Observe: 0 (register),                                      |
 |    <Other class E options>                                     |
 |  }                                                             |
 |                                                                |
 |                          (S increments the observer counter    |
 |                           for the group observation of /r .)   |
 |                                                                |
C_2 <--------------- [ Unicast w/ OSCORE ] ----------------     S
 |  2.05 (Content)                                                |
 |  Token: 0x01                                                   |
 |  OSCORE: {piv: 401; ...}                                       |
 |  Max-Age: 0                                                    |
 |  <Other class U/I options>                                     |
 |  0xff,                                                         |
 |  Encrypted_payload {                                           |
 |    5.03 (Service Unavailable),                                 |
 |    Content-Format: application/informative-response+cbor,      |
 |    <Other class E options>,                                    |
 |    0xff,                                                       |
 |    CBOR_payload {                                              |
 |      tp_info    : [1, bstr(SRV_ADDR), SRV_PORT,                |
 |                    0x7b, bstr(GRP_ADDR), GRP_PORT],            |
 |      ph_req     : bstr(0x05 | OPT | 0xff | PAYLOAD | SIGN),   |
 |      last_notif : bstr(0x45 | OPT | 0xff | PAYLOAD | SIGN),   |
 |      join_uri   : "coap://myGM/ace-group/myGroup",             |
 |      sec_gp     : "myGroup"                                    |
 |    }                                                           |
 |  }                                                             |
 |                                                                |
```

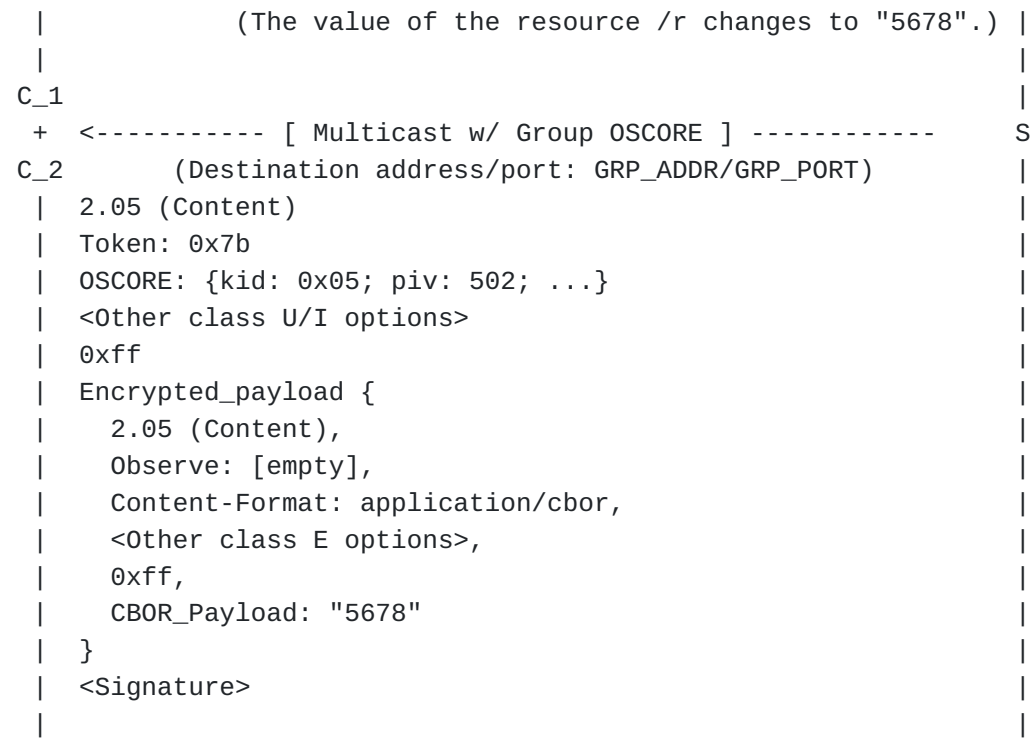



Figure 7: Example of group observation with Group OSCORE

The two external_aad used to encrypt and sign the multicast notification above have 'request_kid' = 5, 'request_piv' = 501, and 'request_kid_context' = 0x57ab2e. These values are specified in the 'kid', 'piv', and 'kid context' field of the OSCORE Option of the phantom observation request, which is encoded in the 'ph_req' parameter of the unicast informative response to the two clients. Thus, the two clients can build the two same external_aad for decrypting and verifying this multicast notification and the following ones in the group observation.

## 11. Intermediaries

This section specifies how the approach presented in Section 4 and Section 5 works when a proxy is used between the clients and the server. In addition to what is specified in Section 5.7 of [RFC7252] and Section 5 of [RFC7641], the following applies.

A client sends its original observation request to the proxy. If the proxy is not already registered at the server for that target resource, the proxy forwards the observation request to the server, hence registering itself as an observer. If the server has an ongoing group observation for the target resource or decides to start one, the server considers the proxy as taking part in the group observation, and replies to the proxy with an informative response.

Upon receiving an informative response, the proxy performs as specified for the client in Section 5, with the peculiarity that "consuming" the last notification (if present) means populating its cache.

In particular, by using the information retrieved from the informative response, the proxy configures an observation of the target resource at the origin server, acting as a client directly taking part in the group observation.

As a consequence, the proxy listens to the IP multicast address and port number indicated by the server in the informative response, as 'cli_host' and 'cli_port' element of 'req_info' under the 'tp_info' parameter, respectively (see Section 4.2.1.1). Furthermore, multicast notifications will match the phantom request stored at the proxy, based on the Token value specified in the 'token' element of 'req_info' under the 'tp_info' parameter in the informative response.

Then, the proxy performs the following actions.

- If the 'last_notif' field is not present, the proxy responds to the client with an Empty Acknowledgement (if indicated by the message type, and if the proxy has not already done so).

- If the 'last_notif' field is present, the proxy rebuilds the latest multicast notification, as defined in Section 5. Then, the proxy responds to the client, by forwarding back the latest multicast notification.

When responding to an observation request from a client, the proxy also adds that client (and its Token) to the list of its registered observers for the target resource, next to the older observations.

Upon receiving a multicast notification from the server, the proxy forwards it back separately to each observer client over unicast. Note that the notification forwarded back to a certain client has the same Token value of the original observation request sent by that client to the proxy.

Note that the proxy configures the observation of the target resource at the server only once, when receiving the informative response associated with a (newly started) group observation for that target resource.

After that, when receiving an observation request from a following new client to be added to the same group observation, the proxy does not take any further action with the server. Instead, the proxy responds to the client either with the latest multicast notification

if available from its cache, or with an Empty Acknowledgement otherwise, as defined above.

As a result, the observer counter at the server (see Section 4) is not incremented when a new origin client behind the proxy registers as an observer at the proxy. Instead, the observer counter takes into account only the proxy, which has registered as observer at the server and has received the informative response from the server.

An example is provided in Appendix E.

In the general case with a chain of two or more proxies, every proxy in the chain takes the role of client with the (next hop towards the) origin server. Note that the proxy adjacent to the origin server is the only one in the chain that receives informative responses, and that listens to an IP multicast address and port number to receive notifications for the group observation. Furthermore, every proxy in the chain takes the role of server with the (previous hop towards the) origin client.

## 12. Intermediaries Together with End-to-End Security

As defined in Section 9, Group OSCORE can be used to protect multicast notifications end-to-end between the origin server and the origin clients. In such a case, additional actions are required when also the informative responses from the origin server are protected specifically end-to-end, by using OSCORE or Group OSCORE.

In fact, the proxy adjacent to the origin server is not able to access the encrypted payload of such informative responses. Hence, the proxy cannot retrieve the 'ph_req' and 'tp_info' parameters necessary to correctly receive multicast notifications and forward them back to the clients.

Then, differently from what is defined in Section 11, each proxy receiving an informative response simply forwards it back to the client that has sent the corresponding observation request. Note that the proxy does not even realize that the message is an actual informative response, since the outer Code field is set to 2.05 (Content).

Upon receiving the informative response, the client does not configure an observation of the target resource. Instead, the client performs a new observe registration request, by transmitting the re-built phantom request as intended to reach the proxy adjacent to the origin server. In particular, the client includes the new Listen-To-Multicast-Responses CoAP option defined in Section 12.1, to provide that proxy with the transport-specific information required for receiving multicast notifications for the group observation.

As a result, the observer counter at the server (see Section 4) is incremented when, after having received the original observation request from a new origin client, the origin server replies with the informative response. In particular, the observer counter at the server reliably takes into account the new, different origin clients behind the proxy, which the server distinguishes through their security identity specified by the pair (OSCORE Sender ID, OSCORE ID Context) in the OSCORE Option of their original observation request. Note that this does not hold anymore if the origin endpoints use phantom observation requests as deterministic requests (see Appendix D).

Details on the additional message exchange and processing are defined in Section 12.2.

### 12.1. Listen-To-Multicast-Responses Option

In order to allow the proxy to listen to the multicast notifications sent by the server, a new CoAP option is introduced. This option MUST be supported by clients interested to take part in group observations through intermediaries, and by proxies that collect multicast notifications and forward them back to the observer clients.

The option is called Listen-To-Multicast-Responses and is intended only for requests. As summarized in Figure 8, the option is critical and not Safe-to-Forward. Since the option is not Safe-to-Forward, the column "N" indicates a dash for "not applicable".

| No. | C | U | N | R | Name | Format | Len. | Default |
|---|---|---|---|---|---|---|---|---|
| TBD | x | x | - | | Listen-To-Multicast-Responses | (*) | 3-1024 | (none) |

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable
(*) See below.

Figure 8: Listen-To-Multicast-Responses

The Listen-To-Multicast-Responses Option includes the serialization of a CBOR array. This specifies transport-specific message information required for listening to the multicast notifications of a group observation, and intended to the proxy adjacent to the origin server sending those notifications. In particular, the serialized CBOR array has the same format specified in Section 4.2.1

for the 'tp_info' parameter of the informative response (see Section 4.2).

The Listen-To-Multicast-Responses Option is of class U for OSCORE [RFC8613][I-D.ietf-core-oscore-groupcomm].

### 12.2. Message Processing

Compared to Section 11, the following additions apply when informative responses are protected end-to-end between the origin server and the origin clients.

After the origin server sends an informative response, each proxy simply forwards it back to the (previous hop towards the) origin client that has sent the observation request.

Once received the informative response, the origin client proceeds in a different way than in Section 9.3.1:

* The client performs all the additional decryption and verification steps of Section 9.3.1 on the phantom request specified in the 'ph_req' parameter and on the last notification specified in the 'last_notif' parameter (if present).

* The client builds a ticket request (see Appendix B of [I-D.amsuess-core-cachable-oscore]), as intended to reach the proxy adjacent to the origin server. The ticket request is formatted as follows.

  - The Token is chosen as the client sees fit. In fact, there is no reason for this Token to be the same as the phantom request's.

  - The outer Code field, the outer CoAP options, and the encrypted payload with AEAD tag (protecting the inner Code, the inner CoAP options, and the possible plain CoAP payload) concatenated with the signature are the same of the phantom request used for the group observation. That is, they are as specified in the 'ph_req' parameter of the received informative response.

  - An outer Observe Option is included and set to 0 (register). This will usually be set in the phantom request already.

  - The client includes: the outer option Proxy-Uri or Proxy-Cri [I-D.ietf-core-href]; or the outer options (Uri-Host, Uri-Port), together with the outer option Proxy-Scheme or Proxy-Scheme-Number [I-D.ietf-core-href]. These options are set in order to specify the same request URI of the original registration request sent by the client.

- The new option Listen-To-Multicast-Responses is included as an outer option. The value is set to the serialization of the CBOR array specified by the 'tp_info' parameter of the informative response.

  Note that, except for transport-specific information such as the Token and Message ID values, every different client participating in the same group observation (hence rebuilding the same phantom request) will build the same ticket request.

  Note also that, identically to the phantom request, the ticket request is still protected with Group OSCORE, i.e., it has the same OSCORE Option, encrypted payload, and signature.

Then, the client sends the ticket request to the next hop towards the origin server. Every proxy in the chain forwards the ticket request to the next hop towards the origin server, until the last proxy in the chain is reached. This last proxy, adjacent to the origin server, proceeds as follows.

* The proxy MUST NOT further forward the ticket request to the origin server.

* The proxy removes the option Proxy-Uri, or Proxy-Scheme, or Proxy-Cri, or Proxy-Scheme-Number from the ticket request.

* The proxy removes the Listen-To-Multicast-Responses Option from the ticket request, and extracts the conveyed transport-specific information.

* The proxy rebuilds the phantom request associated with the group observation, by using the ticket request as directly providing the required transport-independent information. This includes the outer Code field, the outer CoAP options, and the encrypted payload with AEAD tag concatenated with the signature.

* The proxy configures an observation of the target resource at the origin server, acting as a client directly taking part in the group observation. To this end, the proxy uses the rebuilt phantom request and the transport-specific information retrieved from the Listen-To-Multicast-Responses Option. The particular way to achieve this is implementation specific.

After that, the proxy listens to the IP multicast address and port number indicated in the Listen-To-Multicast-Responses Option, as 'cli_host' and 'cli_port' element of the serialized CBOR array, respectively. In particular, multicast notifications will match the phantom request stored at the proxy, based on the Token value specified in the 'token' element of the serialized CBOR array in the Listen-To-Multicast-Responses Option.

An example is provided in Appendix F.

## 13. Informative Response Parameters

This document defines a number of fields used in the informative response defined in Section 4.2.

The table below summarizes them and specifies the CBOR key to use instead of the full descriptive name. Note that the media type application/informative-response+cbor MUST be used when these fields are transported.

| Name | CBOR Key | CBOR Type | Reference |
|---|---|---|---|
| tp_info | 0 | array | Section 4.2 |
| ph_req | 1 | bstr | Section 4.2 |
| last_notif | 2 | bstr | Section 4.2 |
| next_not_before | 3 | uint | Section 4.2 |
| join_uri | 4 | tstr | Section 9.1 |
| sec_gp | 5 | tstr | Section 9.1 |
| as_uri | 6 | tstr | Section 9.1 |
| hkdf | 7 | int / tstr | Section 9.1 |
| cred_fmt | 8 | int | Section 9.1 |
| gp_enc_alg | 9 | int / tstr | Section 9.1 |
| sign_alg | 10 | int / tstr | Section 9.1 |
| sign_params | 11 | array | Section 9.1 |
| gp_material | 12 | map | Appendix C |
| srv_cred | 13 | bstr | Appendix C |
| srv_identifier | 14 | bstr | Appendix C |
| exi | 15 | uint | Appendix C |
| exp | 16 | uint | Appendix C |

Table 1

## 14. Transport Protocol Information

This document defines some values of transport protocol identifiers to use within the 'tp_info' parameter of the informative response defined in Section 4.2.

According to the encoding specified in Section 4.2.1, these values are used for the 'tp_id' element of 'srv_addr', under the 'tp_info' parameter.

The table below summarizes them, specifies the integer value to use instead of the full descriptive name, and provides the corresponding comprehensive set of information elements to include in the 'tp_info' parameter.

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

| Transport Protocol | Description | Value | Srv Addr | Req Info | Reference |
|---|---|---|---|---|---|
| Reserved | This value is reserved | 0 | | | [RFC-XXXX] |
| UDP | UDP is used as per RFC7252 | 1 | tp_id<br>srv_host<br>srv_port | token<br>cli_host<br>?cli_port | [RFC-XXXX] |

Figure 9: Transport protocol information

## 15. Security Considerations

In addition to the security considerations from [RFC7252][RFC7641] [I-D.ietf-core-groupcomm-bis][RFC8613] [I-D.ietf-core-oscore-groupcomm], the following considerations hold for this document.

### 15.1. Unprotected Communications

In case communications are not protected, the server might not be able to effectively authenticate a new client when it registers as an observer. Section 7 of [RFC7641] specifies how, in such a case, the server must strictly limit the number of notifications sent between receiving acknowledgements from the client, as confirming to be still interested in the observation; i.e., any notifications sent in Non-confirmable messages must be interspersed with confirmable messages.

This is not possible to achieve by the same means when using the communication model defined in this document, since multicast notifications are sent as Non-confirmable messages. Nonetheless, the server might obtain such acknowledgements by other means.

For instance, the method defined in Section 8 to perform the rough counting of still interested clients triggers (some of) them to explicitly send a new observation request to acknowledge their interest. Then, the server can decide to terminate the group observation altogether, in case not enough clients are estimated to be still active.

If the method defined in Section 8 is used, the server SHOULD NOT send more than a strict number of multicast notifications for a given group observation, without having first performed a new rough counting of active clients. Note that, when using the method defined in Section 8 with unprotected communications, an adversary can craft and inject multiple new observation requests including the Multicast-Response-Feedback-Divider Option, hence inducing the server to overestimate the number of still interested clients, and thus to inappropriately continue the group observation.

### 15.2. Protected Communications

If multicast notifications are protected using Group OSCORE as per Section 9, the following applies.

* The original registration requests and related unicast (notification) responses MUST also be secured, including and especially the informative responses from the server. This prevents on-path active adversaries from altering the conveyed IP multicast address and serialized phantom registration request. Thus, it ensures secure binding between every multicast notification for a same observed resource and the phantom registration request that started the group observation of that resource.

* A re-registration request, possibly including the Multicast-Response-Feedback-Divider Option to support the rough counting of clients (see Section 8), MUST also be secured.

To this end, clients and servers SHOULD use OSCORE or Group OSCORE, so ensuring that the secure binding above is enforced end-to-end between the server and each observing client.

### 15.3. Listen-To-Multicast-Responses Option

The CoAP option Listen-To-Multicast-Responses defined in Section 12.1 is of class U for OSCORE and Group OSCORE [RFC8613] [I-D.ietf-core-oscore-groupcomm].

This allows the proxy adjacent to the origin server to access the option value conveyed in a ticket request (see Section 12.2), and to retrieve from it the transport-specific information about a phantom request. By doing so, the proxy becomes able to configure an observation of the target resource and to receive multicast notifications matching to the phantom request.

Any proxy in the chain, as well as further possible intermediaries or on-path active adversaries, are thus able to remove the option or alter its content, before the ticket request reaches the proxy adjacent to the origin server.

Removing the option would result in the proxy adjacent to the origin server to not configure the group observation, if that has not happened yet. In such a case, the proxy would not receive the corresponding multicast notifications to be forwarded back to the clients.

Altering the option content would result in the proxy adjacent to the origin server to incorrectly configure a group observation (e.g., by indicating a wrong multicast IP address) hence preventing the correct reception of multicast notifications and their forwarding to the clients; or to configure bogus group observations that are currently not active on the origin server.

In order to prevent what is described above, the ticket requests conveying the Listen-To-Multicast-Responses Option can be additionally protected hop-by-hop. This can be achieved by the client protecting the ticket request sent to the proxy using OSCORE (see [I-D.ietf-core-oscore-capable-proxies]) and/or DTLS [RFC9147].

## 16. IANA Considerations

This document has the following actions for IANA.

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

### 16.1. Media Type Registrations

This document registers the media type "application/informative-response+cbor" for error messages as informative response defined in Section 4.2, when carrying parameters encoded in CBOR. This registration follows the procedures specified in [RFC6838].

* Type name: application

* Subtype name: informative-response+cbor

* Required parameters: N/A

* Optional parameters: N/A

* Encoding considerations: Must be encoded as a CBOR map containing the parameters defined in Section 4.2 of [RFC-XXXX].

* Security considerations: See Section 15 of [RFC-XXXX].

* Interoperability considerations: N/A

* Published specification: [RFC-XXXX]

*Applications that use this media type: The type is used by CoAP servers and clients that support error messages as informative response defined in Section 4.2 of [RFC-XXXX].

*Fragment identifier considerations: N/A

*Additional information: N/A

*Person & email address to contact for further information: CoRE WG mailing list (core@ietf.org) or IETF Applications and Real-Time Area (art@ietf.org)

*Intended usage: COMMON

*Restrictions on usage: None

*Author/Change controller: IETF

*Provisional registration: No

### 16.2. CoAP Content-Formats Registry

IANA is asked to add the following entry to the "CoAP Content-Formats" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Content Type: application/informative-response+cbor

Content Coding: -

ID: TBD

Reference: [RFC-XXXX]

### 16.3. 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 |
|---|---|---|
| TBD | Multicast-Response-Feedback-Divider | [RFC-XXXX] |
| TBD | Listen-To-Multicast-Responses | [RFC-XXXX] |

### 16.4. Informative Response Parameters Registry

This document establishes the "Informative Response Parameters" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 16.7.

The columns of this registry are:

- Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.

- CBOR Key: This is the value used as CBOR key of the item. These values MUST be unique. The value can be a positive integer, a negative integer, or a string.

- CBOR Type: This contains the CBOR type of the item, or a pointer to the registry that defines its type, when that depends on another item.

- Reference: This contains a pointer to the public specification for the item.

This registry has been initially populated by the values in Section 13. The "Reference" column for all of these entries refers to sections of this document.

### 16.5. CoAP Transport Information Registry

This document establishes the "CoAP Transport Information" registry within the "CoRE Parameters" registry group. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 16.7. It should be noted that, in addition to the expert review, some portions of the Registry require a specification, potentially a Standards Track RFC, to be supplied as well.

The columns of this registry are:

- Transport Protocol: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.

- Description: Text giving an overview of the transport protocol referred by this item.

- Value: CBOR abbreviation for the transport protocol referred by this item. Different ranges of values use different registration

policies [RFC8126]. Integer values from -256 to 255 are designated as "Standards Action With Expert Review". Integer values from -65536 to -257 and from 256 to 65535 are designated as "Specification Required". Integer values greater than 65535 are designated as "Expert Review". Integer values less than -65536 are marked as "Private Use".

*Server Addr: List of elements providing addressing information of the server.

*Req Info: List of elements providing transport-specific information related to a pertinent CoAP request. Optional elements are prepended by '?'.

*Reference: This contains a pointer to the public specification for the item.

This registry has been initially populated by the values in Section 14. The "Reference" column for all of these entries refers to sections of this document.

### 16.6. Target Attributes Registry

IANA is asked to register the following entry in the "Target Attributes" registry within the "CoRE Parameters" registry group.

Attribute Name: gp-obs
Brief Description: Observable resource supporting group observation
Change Controller: IETF
Reference: Section 6 of [RFC-XXXX]

### 16.7. Expert Review Instructions

The IANA registries established in this document are defined as "Expert Review". This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason, so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

*Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments. The zones tagged as "Private Use" are intended for testing purposes and closed environments. Code points in other ranges should not be assigned for testing.

* Specifications are required for the "Standards Action With Expert Review" range of point assignment. Specifications should exist for "Specification Required" ranges, but early assignment before a specification is available is considered to be permissible. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.

* Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for Standards Track documents does not mean that a Standards Track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

# 17. References

## 17.1. Normative References


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

**[COSE.Header.Parameters]** IANA, "COSE Header Parameters", <https://www.iana.org/assignments/cose/cose.xhtml#header-parameters>.

**[COSE.Key.Types]** IANA, "COSE Key Types", <https://www.iana.org/assignments/cose/cose.xhtml#key-type>.

**[I-D.ietf-ace-key-groupcomm-oscore]** Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-16, 6 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-oscore-16>.

**[I-D.ietf-core-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-10, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-bis-10>.

**[I-D.ietf-core-href]** Bormann, C. and H. Birkholz, "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-14, 9 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-href-14>.

**[I-D.ietf-core-oscore-groupcomm]** Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group Object Security for Constrained RESTful Environments (Group OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-20, 2 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-20>.

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

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

**[RFC6838]** Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

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

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

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

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

**[RFC8126]** Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

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

**[RFC8288]** Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

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

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

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

**[RFC9203]** Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "The Object Security for Constrained RESTful Environments (OSCORE) Profile of the Authentication and Authorization for Constrained Environments (ACE) Framework", RFC 9203, DOI 10.17487/RFC9203, August 2022, <https://www.rfc-editor.org/rfc/rfc9203>.

### 17.2. 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-08, 10 January 2024, <https://datatracker.ietf.org/doc/html/draft-amsuess-core-cachable-oscore-08>.

**[I-D.ietf-core-coap-pubsub]** Jimenez, J., Koster, M., and A. Keränen, "A publish-subscribe architecture for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-13, 20 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-coap-pubsub-13>.

**[I-D.ietf-core-coral]** Amsüss, C. and T. Fossati, "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-ietf-core-coral-06, 4 March 2024,

<https://datatracker.ietf.org/doc/html/draft-ietf-core-coral-06>.

**[I-D.ietf-core-oscore-capable-proxies]** Tiloca, M. and R. Höglund, "OSCORE-capable Proxies", Work in Progress, Internet-Draft, draft-ietf-core-oscore-capable-proxies-00, 18 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-capable-proxies-00>.

**[I-D.ietf-cose-cbor-encoded-cert]** Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-09, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-cbor-encoded-cert-09>.

**[I-D.tiloca-core-oscore-discovery]** Tiloca, M., Amsüss, C., and P. Van der Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-14, 8 September 2023, <https://datatracker.ietf.org/doc/html/draft-tiloca-core-oscore-discovery-14>.

**[MOBICOM99]** Ni, S., Tseng, Y., Chen, Y., and J. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network", Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking , August 1999, <https://people.eecs.berkeley.edu/~culler/cs294-f03/papers/bcast-storm.pdf>.

**[RFC5280]** Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

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

**[RFC7519]** Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

**[RFC8392]** Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

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

**[RFC9176]** Amsüss, C., Ed., Shelby, Z., Koster, M., Bormann, C., and P. van der Stok, "Constrained RESTful Environments (CoRE) Resource Directory", RFC 9176, DOI 10.17487/RFC9176, April 2022, <https://www.rfc-editor.org/rfc/rfc9176>.

**[RFC9200]** Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/rfc/rfc9200>.

## Appendix A. Different Sources for Group Observation Data

While the clients usually receive the phantom registration request and other information related to the group observation through an informative response (see Section 4.2), the server can make the same data available through different means, such as the following ones.

In such a case, the server has to first start the group observation (see Section 4.1), before making the corresponding data available.

A client that receives such information from different sources may be able to simply set up the right multicast address and start receiving multicast notifications for the group observation. In such a case, the client does not need to perform additional setup traffic, e.g., in order to configure a proxy for listening to multicast notifications on its behalf (see Section 11 and Section 12). Consequently, the server will not receive an observation request due to that client, will not follow-up with a corresponding informative response, and thus its observer counter (see Section 4) is not incremented to reflect the presence of the new client.

### A.1. Topic Discovery in Publish-Subscribe Settings

In a Publish-Subscribe scenario [I-D.ietf-core-coap-pubsub], a group observation can be discovered along with topic metadata.

To this end, together with topic metadata, the server has to publish the same information associated with the group observation that would be conveyed in the informative response returned to observer clients (see Section 4.2).

This information especially includes the phantom observation request associated with the group observation, as well as the addressing information of the server and the addressing information where multicast notifications are sent to.

Figure 10 provides an example where a group observation is discovered. The example assumes a CoRAL namespace [I-D.ietf-core-coral], that contains properties analogous to those in the content-format application/informative-response+cbor.

Note that the information about the transport protocol used for the group observation is not expressed through a dedicated element equivalent to 'tp_id' of the informative response (see Section 4.2.1). Rather, it is expressed through the scheme component of the two URIs specified as 'tp_info_srv' and 'tp_info_cli', where the former specifies the addressing information of the server (like 'srv_host' and 'srv_port' in Section 4.2.1.1), while the latter specifies the addressing information where multicast notifications are sent to (like 'cli_host' and 'cli_port' in Section 4.2.1.1).

Request:

```
GET </ps/topics?rt=oic.r.temperature>
Accept: application/coral+cbor
```

Response:

```
2.05 Content
Content-Format: application/coral+cbor

rdf:type [ = <http://example.org/pubsub/topic-list>,
       topic [ = </ps/topics/1234>,
           tp_info_srv <coap://[2001:db8::1]>,
           tp_info_token "7b"^^xsd::hexBinary,
           tp_info_cli <coap://[ff35:30:2001:db8::123]>,
           ph_req "0160.."^^xsd::hexBinary,
           last_notif "256105.."^^xsd::hexBinary,
       ]
]
```

Figure 10: Group observation discovery in a Pub-Sub scenario

With this information from the topic discovery step, the client can already set up its multicast address and start receiving multicast notifications for the group observation in question. Clients that are not directly able to listen to multicast notifications can instead rely on a proxy to do so on their behalf (see Section 11 and Section 12).

In heavily asymmetric networks like municipal notification services, discovery and notifications do not necessarily need to use the same network link. For example, a departure monitor could use its (costly and usually-off) cellular uplink to discover the topics it needs to update its display to, and then listen on a LoRA-WAN interface for receiving the actual multicast notifications.

### A.2. Introspection at the Multicast Notification Sender

For network debugging purposes, it can be useful to query a server that sends multicast responses as matching a phantom registration request.

Such an interface is left for other documents to specify on demand. As an example, a possible interface can be as follows, and rely on the already known Token value of intercepted multicast notifications, associated with a phantom registration request.

```
Request:

GET </.well-known/core/mc-sender?token=6464>

Response:

2.05 Content
Content-Format: application/informative-response+cbor

{
  / tp_info /    0 : [1, h'7b', h'20010db80100/.../0001', 5683,
                      h'ff35003020010db8/.../1234', 5683],
  / ph_req /     1 : h'0160/.../',
  / last_notif / 2 : h'256105/.../'
}
```

Figure 11: Group observation discovery with server introspection

For example, a network sniffer could offer sending such a request when unknown multicast notifications are seen in a network. Consequently, it can associate those notifications with a URI, or decrypt them, if member of the correct OSCORE group.

## Appendix B. Pseudo-Code for Rough Counting of Clients

This appendix provides a description in pseudo-code of the two algorithms used for the rough counting of active observers, as defined in Section 8.

In particular, Appendix B.1 describes the algorithm for the client side, while Appendix B.2 describes an optimized version for

constrained clients. Finally, Appendix B.3 describes the algorithm for the server side.

### B.1. Client Side

```
input:  int Q, // Value of the MRFD option
        int LEISURE_TIME, // DEFAULT_LEISURE from RFC 7252,
                          // unless overridden

output: None


int RAND_MIN = 0;
int RAND_MAX = (2^Q) - 1;
int I = randomInteger(RAND_MIN, RAND_MAX);

if (I == 0) {
    float fraction = randomFloat(0, 1);

    Timer t = new Timer();
    t.setAndStart(fraction * LEISURE_TIME);
    while(!t.isExpired());

    Request req = new Request();
    // Initialize as NON and with maximum
    // No-Response settings, set options ...

    Option opt = new Option(OBSERVE);
    opt.set(0);
    req.setOption(opt);

    opt = new Option(MRFD);
    opt.set(0);
    req.setOption(opt);

    req.send(SRV_ADDR, SRV_PORT);
}
```

### B.2. Client Side - Optimized Version

```
input:  int Q, // Value of the MRFD option
        int LEISURE_TIME, // DEFAULT_LEISURE from RFC 7252,
                          // unless overridden

output: None


const unsigned int UINT_BIT = CHAR_BIT * sizeof(unsigned int);

if (respond_to(Q) == true) {
    float fraction = randomFloat(0, 1);

    Timer t = new Timer();
    t.setAndStart(fraction * LEISURE_TIME);
    while(!t.isExpired());

    Request req = new Request();
    // Initialize as NON and with maximum
    // No-Response settings, set options ...

    Option opt = new Option(OBSERVE);
    opt.set(0);
    req.setOption(opt);

    opt = new Option(MRFD);
    opt.set(0);
    req.setOption(opt);

    req.send(SRV_ADDR, SRV_PORT);
}

bool respond_to(int Q) {
    while (Q >= UINT_BIT) {
        if (rand() != 0) return false;
        Q -= UINT_BIT;
    }
    unsigned int mask = ~((~0u) << Q);
    unsigned int masked = mask & rand();
    return masked == 0;
}
```

### B.3. Server Side

```
input:  int COUNT, // Current observer counter
        int M, // Desired number of confirmations
        int MAX_CONFIRMATION_WAIT,
        Response notification, // Multicast notification to send

output: int NEW_COUNT // Updated observer counter


int D = 4; // Dampener value
int RETRY_NEXT_THRESHOLD = 4;
float CANCEL_THRESHOLD = 0.2;

int N = max(COUNT, 1);
int Q = max(ceil(log2(N / M)), 0);
Option opt = new Option(MRFD);
opt.set(Q);

notification.setOption(opt);
<Finalize the notification message>
notification.send(GRP_ADDR, GRP_PORT);

Timer t = new Timer();
t.setAndStart(MAX_CONFIRMATION_WAIT); // Time t1
while(!t.isExpired());

// Time t2

int R = <number of requests to the target resource
         received between t1 and t2, and including
         the MRFD option with value 0>;

int E = R * (2^Q);

// Determine after how many multicast notifications
// the next count update will be performed
if ((R == 0) || (max(E/N, N/E) > RETRY_NEXT_THRESHOLD)) {
    <Next count update with the next multicast notification>
}
else {
    <Next count update after 10 multicast notifications>
}

// Compute the new count estimate
int COUNT_PRIME = <current value of the observer counter>;
int NEW_COUNT = COUNT_PRIME + ((E - N) / D);

// Determine whether to cancel the group observation
if (NEW_COUNT < CANCEL_THRESHOLD) {
    <Cancel the group observation>;
    return 0;
```

```
}

return NEW_COUNT;
```

## Appendix C. OSCORE Group Self-Managed by the Server

For simple settings, where no pre-arranged group with suitable memberships is available, the server can be responsible to set up and manage the OSCORE group used to protect the group observation.

In such a case, a client would implicitly request to join the OSCORE group when sending the observe registration request to the server. When replying, the server includes the group keying material and related information in the informative response (see Section 4.2).

Additionally to what is defined in Section 4, the CBOR map in the informative response payload contains the following fields, whose CBOR labels are defined in Section 13.

* 'gp_material': this element is a CBOR map, which includes what the client needs in order to set up the Group OSCORE Security Context.

  This parameter has as value a subset of the Group_OSCORE_Input_Material object, which is defined in Section 6.3 of [I-D.ietf-ace-key-groupcomm-oscore] and extends the OSCORE_Input_Material object encoded in CBOR as defined in Section 3.2.1 of [RFC9203].

  In particular, the following elements of the Group_OSCORE_Input_Material object are included, using the same CBOR labels from the OSCORE Security Context Parameters Registry, as in Section 6.3 of [I-D.ietf-ace-key-groupcomm-oscore].

  - 'ms', 'contexId', 'cred_fmt', 'sign_enc_alg', 'sign_alg' and 'sign_params'. These elements MUST be included.

    Editor's note: as per the text above, the referred version of [I-D.ietf-ace-key-groupcomm-oscore] still uses 'sign_enc_alg' as parameter name. The next version of [I-D.ietf-ace-key-groupcomm-oscore] will be updated in order to use 'gp_enc_alg' instead, consistently with the naming used in the latest version of [I-D.ietf-core-oscore-groupcomm].

  - 'hkdf' and 'salt'. These elements MAY be included.

  The 'group_senderId' element of the Group_OSCORE_Input_Material object MUST NOT be included.

Note that no information is provided as related to the AEAD Algorithm, or to the Pairwise Key Agreement Algorithm and its parameters. In fact, the clients and the server will never use the pairwise mode of Group OSCORE as per Section 9 of [I-D.ietf-core-oscore-groupcomm], and will not need to compute a cofactor Diffie-Hellman shared secret in this OSCORE group. It follows that:

- In the Common Context of the Group OSCORE Security Context, the parameter AEAD Algorithm and the parameter Pairwise Key Agreement Algorithm are not set (see Section 2.1.1 of [I-D.ietf-core-oscore-groupcomm] and Section 2.1.9 of [I-D.ietf-core-oscore-groupcomm]).

- Consistently, when building the two OSCORE 'external_aad' to process messages protected with Group OSCORE in this OSCORE group, (see Section 4.4 of [I-D.ietf-core-oscore-groupcomm]), the elements 'alg_aead' and 'alg_pairwise_key_agreement' within the 'algorithms' arrays are set to the CBOR simple value null (0xf6).

* 'srv_cred': this element is a CBOR byte string, with value the original binary representation of the server's authentication credential used in the OSCORE group. In particular, the original binary representation complies with the format specified by the 'cred_fmt' element of 'gp_material'.

* 'srv_identifier': this element MUST be included and is encoded as a CBOR byte string, with value the Sender ID that the server has in the OSCORE group.

* 'exi': this element has as value the residual lifetime of the keying material of the OSCORE group specified in the 'gp_material' parameter, encoded as a CBOR unsigned integer. The value represents the residual lifetime of the keying material in seconds, i.e., the number of seconds between the current time at the server and the time when the keying material expires. Upon receiving the informative response containing the 'exi' parameter, a client determines the expiration time of the keying material by adding the seconds specified in the 'exi' parameter to its current time.

If the server has a reliable way to synchronize its internal clock with UTC, then the server includes also the following field:

* 'exp': this element has as value the expiration time of the keying material of the OSCORE group specified in the 'gp_material' parameter, encoded as a CBOR unsigned integer. The value represents the number of seconds from 1970-01-01T00:00:00Z

UTC until the specified UTC date/time, ignoring leap seconds, analogous to what is specified for NumericDate in Section 2 of [RFC7519].

If a client has a reliable way to synchronize its internal clock with UTC and the 'exp' parameter is present in the informative response, then the client MUST use the 'exp' parameter value as expiration time for the group keying material.

Note that the informative response does not require to include an explicit proof-of-possession (PoP) of the server's private key. Although the server is also acting as Group Manager and a PoP evidence of the Group Manager's private key is included in a full-fledged Join Response (see Section 6.3 of [I-D.ietf-ace-key-groupcomm-oscore]), such proof-of-possession will be achieved through every multicast notification that the server sends, as protected with the group mode of Group OSCORE and including a signature computed with its private key.

A client receiving an informative response uses the information above to set up the Group OSCORE Security Context, as described in Section 2 of [I-D.ietf-core-oscore-groupcomm]. Note that the client does not obtain a Sender ID of its own, hence it installs a Security Context that a "silent server" would, i.e., without Sender Context. From then on, the client uses the received keying material to process the incoming multicast notifications from the server.

Since the server is also acting as Group Manager, the authentication credential of the server provided in the 'srv_cred' element of the informative response is also used in the 'gm_cred' element of the external_aad for encrypting and signing the phantom request and multicast notifications (see Section 4.4 of [I-D.ietf-core-oscore-groupcomm]).

Furthermore, the server complies with the following points.

* The server MUST NOT self-manage OSCORE groups and provide the related keying material in the informative response for any other purpose than the protection of the phantom requests and the multicast notifications in group observations that it hosts, as defined in this document.

  The server MAY use the same self-managed OSCORE group to protect the phantom request and the multicast notifications of multiple group observations that it hosts.

* The server MUST NOT provide in the informative response the keying material of other OSCORE groups it is or has been a member of.

Upon expiration of the group keying material as indicated in the informative response by the 'exp' parameter (if present) and the 'exi' parameter:

- The server MUST stop using the keying material and MUST cancel the group observations for which that keying material is used (see Section 4.5 and Section 9.2.4). If the server creates a new group observation as a replacement or follow-up using the same OSCORE group:

  - The server MUST update the OSCORE Master Secret.

  - The server MUST update the ID Context used as Group Identifier (Gid), consistently with Section 3.2 of [I-D.ietf-core-oscore-groupcomm].

  - The server MAY update the OSCORE Master Salt.

- The client MUST stop using the keying material and MAY re-register the observation at the server.

Before the keying material has expired, the server can send a multicast response with response code 5.03 (Service Unavailable) to the observing clients, protected with the current keying material. In particular, this is an informative response (see Section 4.2) that: i) additionally contains the abovementioned parameters for the next group keying material to be used; and ii) MAY omit the 'tp_info' and 'ph_req' parameters, since the associated information is immutable throughout the observation lifetime. The response has the same Token value T of the phantom registration request and it does not include an Observe Option. The server MUST use its own Sender Sequence Number as Partial IV to protect the error response, and include it as Partial IV in the OSCORE Option of the response.

When some clients leave the OSCORE group and forget about the group observation, the server does not have to provide the remaining clients with any stale Sender IDs, as normally required for Group OSCORE (see Section 3.2 of [I-D.ietf-core-oscore-groupcomm]). In fact, only two entities in the group have a Sender ID, i.e., the server and possibly the Deterministic Client, if the optimization defined in this appendix is combined with the use of phantom requests as deterministic requests (see Appendix D). In particular, both of them never change their Sender ID during the group lifetime, and they both remain part of the group until the group ceases to exist.

As an alternative to renewing the keying material before it expires, the server can simply cancel the group observation (see Section 4.5 and Section 9.2.4), which results in the eventual re-registration of the clients that are still interested in the group observation.

Applications requiring backward security or forward security are REQUIRED to use an actual group joining process (usually through a dedicated Group Manager), e.g., the ACE joining procedure defined in [I-D.ietf-ace-key-groupcomm-oscore]. The server can facilitate the clients by providing them information about the OSCORE group to join, as described in Section 9.1.

## Appendix D. Phantom Request as Deterministic Request

In some settings, the server can assume that all the approaching clients already have the exact phantom observation request to use, together with the transport-specific information required to listen to corresponding multicast notifications.

For instance, the clients can be pre-configured with the phantom observation request, or they may be expected to retrieve it through dedicated means (see Appendix A). In either case, the server would already have started the group observation, before the associated phantom observation request was disseminated.

Then, the clients either set up the multicast address and group observation for listening to multicast notifications (if able to directly do so), or rely on a proxy to do so on their behalf (see Section 11 and Section 12).

If Group OSCORE is used to protect the group observation (see Section 9), and the OSCORE group supports the concept of Deterministic Client [I-D.amsuess-core-cachable-oscore], then the server and each client in the OSCORE group can also independently compute the protected phantom observation request.

In such a case, the unprotected version of the phantom observation request can be made available to the clients as a smaller, plain CoAP message. As above, this can be pre-configured on the clients, or they can obtain it through dedicated means (see Appendix A). In either case, the clients and the server can independently protect the plain CoAP message by using the approach defined in Section 3 of [I-D.amsuess-core-cachable-oscore], thus all computing the same protected deterministic request. The latter is used as the actual phantom observation request, against which the protected multicast notifications will match for the group observation in question.

If relying on a proxy, each client sends the deterministic request to the proxy as a ticket request (see Section 12). However, differently from what is defined in Section 12 when the ticket request is not a deterministic request, the clients do not include a Listen-to-Multicast-Responses Option. This results in the proxy forwarding the ticket request (i.e., the phantom observation request) to the server and obtaining the information required to

listen to multicast notifications, unless the proxy has already set itself to do so. Also, the proxy will be able to serve multicast notifications from its cache as per [I-D.amsuess-core-cachable-oscore]. An example considering such a setup is shown in Appendix G.

Note that the phantom registration request is, in terms of transport-independent information, identical to the same deterministic request possibly sent by each client (e.g., if a proxy is deployed). Thus, if the server receives such a phantom registration request, the informative response may omit the 'ph_req' parameter (see Section 4.2). If a client receives an informative response that includes the 'ph_req' parameter, and this specifies transport-independent information different from the one of the sent deterministic request, then the client considers the informative response malformed.

When using a deterministic request as a phantom observation request, the observer counter at the server (see Section 4) is not reliably incremented when new clients start participating in the group observation. In fact:

* If a proxy is not deployed, the clients simply set up the right multicast address and port number, and starts listening to multicast notifications bound to the deterministic request. Hence, the observer counter at the server is not incremented as new clients start listening to multicast notifications.

* If a proxy is deployed, the origin server increments its observer counter after having sent the informative response to the proxy, as a reply to the deterministic request forwarded to the origin server on behalf of the first origin client that contacted the proxy. After that, the same deterministic request sent by any origin client will not be forwarded to the origin server, but will instead produce a cache hit at the proxy that will serve the client accordingly. Hence, the observer counter at the server is not further incremented as additional, new origin clients start participating in the group observation through the proxy.

In either case, the security identity associated with the sender of any deterministic request in the OSCORE group is exactly the same one, i.e., the pair (SID, OSCORE ID Context), where SID is the OSCORE Sender ID of the deterministic client in the OSCORE group, which all the clients in the group rely on to produce deterministic requests.

If the optimization defined in Appendix C is also used, the 'gp_material' element in the informative response from the server

MUST also include the following elements from the Group_OSCORE_Input_Material object.

- 'alg', as per Section 6.3 of [I-D.ietf-ace-key-groupcomm-oscore].

- 'det_senderId' and 'det_hash_alg', defined in Section 4 of [I-D.amsuess-core-cachable-oscore]. These specify the Sender ID of the Deterministic Client in the OSCORE group, and the hash algorithm used to compute the deterministic request (see Section 3.4.1 of [I-D.amsuess-core-cachable-oscore]).

Note that, like in Appendix C, no information is provided as related to the Pairwise Key Agreement Algorithm and its parameters. In fact, the clients and the server will not need to compute a cofactor Diffie-Hellman shared secret in this OSCORE group. It follows that:

- In the Common Context of the Group OSCORE Security Context, the parameter Pairwise Key Agreement Algorithm is not set (see Section 2.1.9 of [I-D.ietf-core-oscore-groupcomm]).

- Consistently, when building the two OSCORE 'external_aad' to process messages protected with Group OSCORE in this OSCORE group, (see Section 4.4 of [I-D.ietf-core-oscore-groupcomm]), the element 'alg_pairwise_key_agreement' within the 'algorithms' arrays is set to the CBOR simple value null (0xf6).

If a deterministic request is used as phantom observation request for a group observation, the server does not assist clients that are interested to take part in the group observation but do not support deterministic requests. This is consistent with the fact that the setup in question already relies on a lot of agreed pre-configuration.

Therefore, the following holds when a group observation relies on a deterministic request as a phantom observation request.

- Every client interested to take part in such a group observation: has to support deterministic requests; and has to know the phantom observation request, as a result of pre-configuration or following its retrieval through dedicated means (see Appendix A).

- When running such an observation request, the server does not simultaneously run a parallel group observation for the same target resource, as associated with a different phantom observation request and intended to clients that do not support deterministic requests.

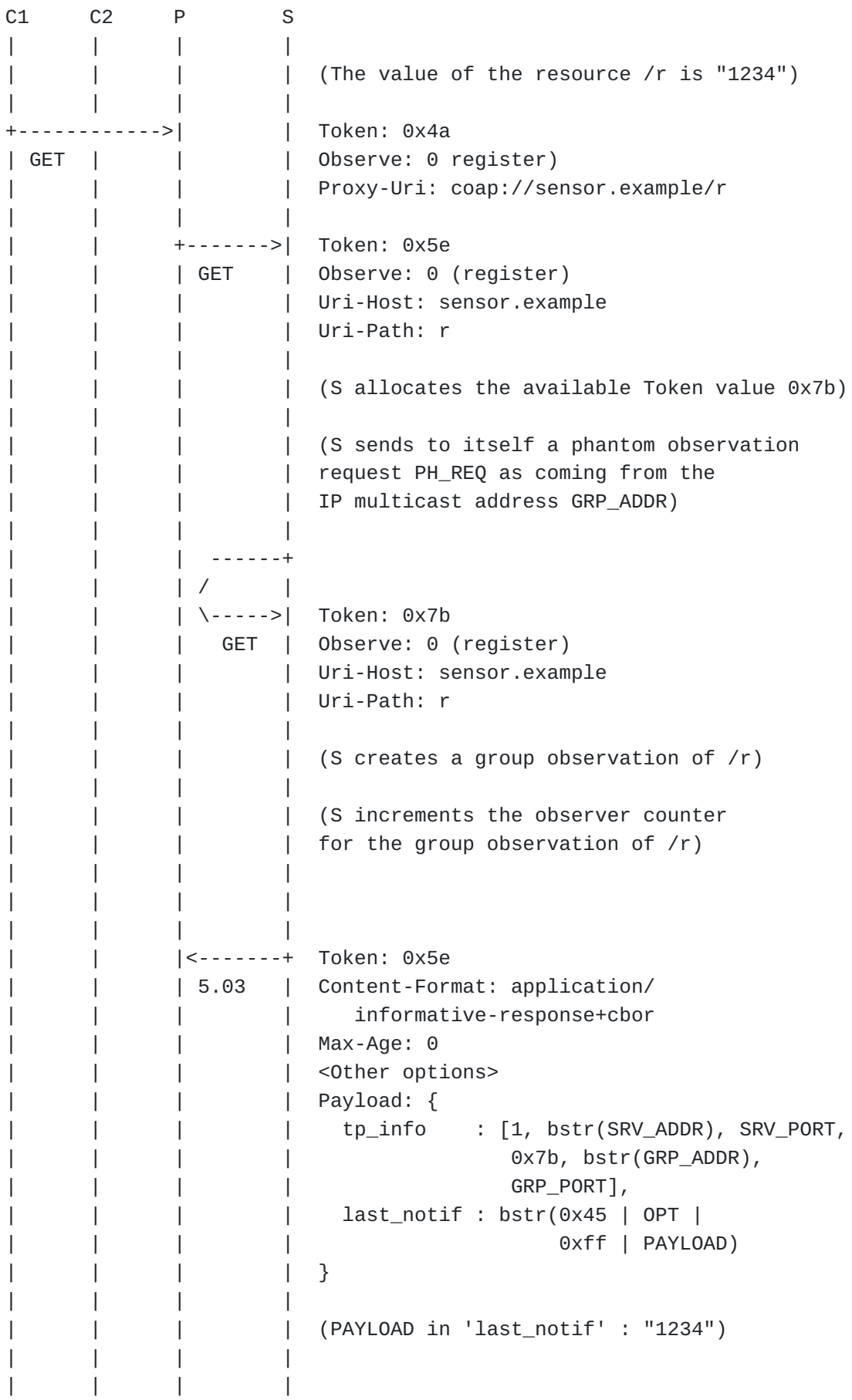
  Upon receiving an individual observation request for the same target resource, the server MUST reply with a generic 5.03 (Service Unavailable) response (i.e., not the informative

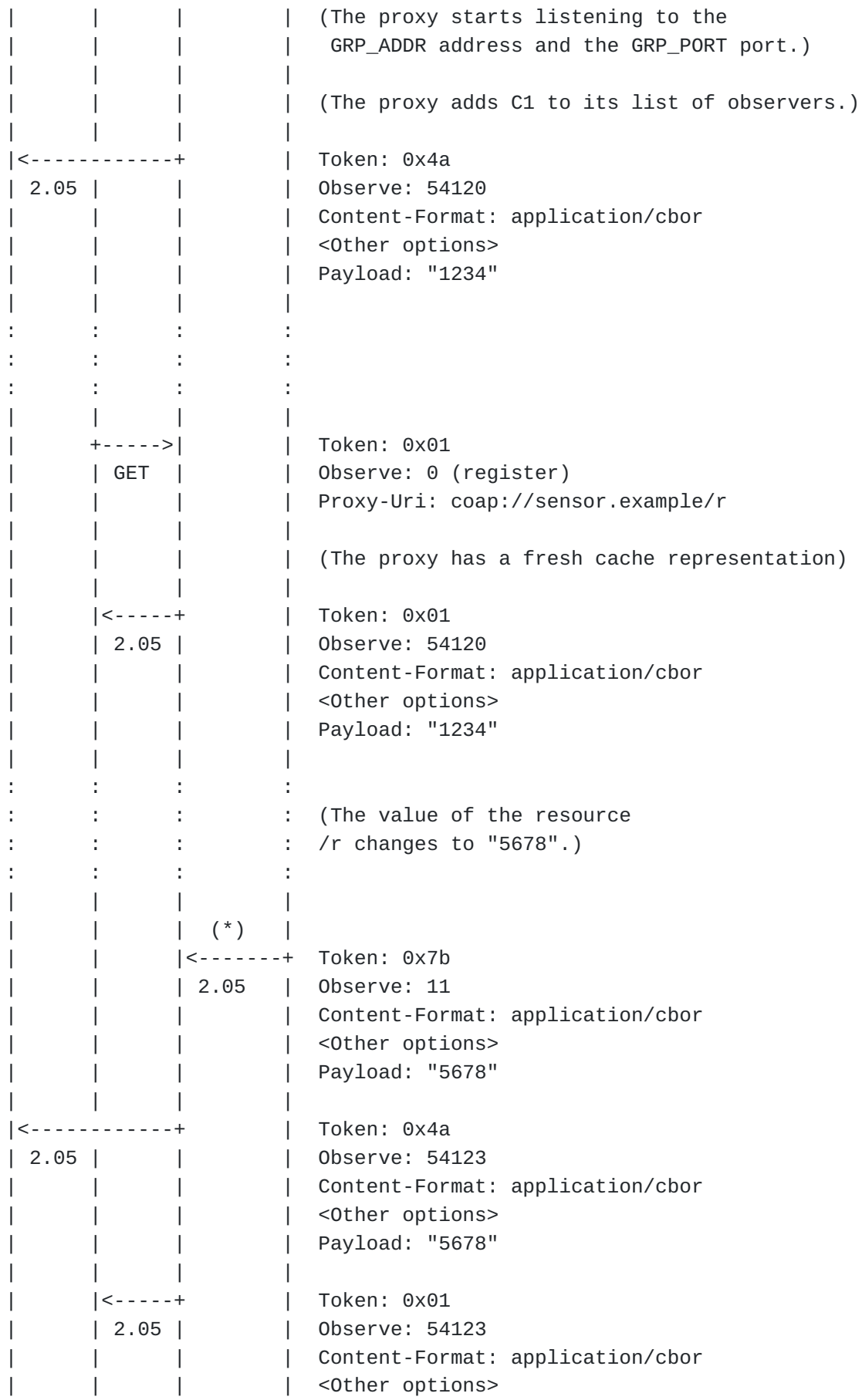
response defined in Section 4.2), if the request differs from the specific deterministic request associated with the group observation.

## Appendix E. Example with a Proxy

This section provides an example when a proxy P is used between the clients and the server. The same assumptions and notation used in Section 7 are used for this example. In addition, the proxy has address PRX_ADDR and listens to the port number PRX_PORT.

Unless explicitly indicated, all messages transmitted on the wire are sent over unicast.


C1
C2
P
S
(The value of the resource /r is "1234")
Token: 0x4a
GET
Observe: 0 register)
Proxy-Uri: coap://sensor.example/r
Token: 0x5e
GET
Observe: 0 (register)
Uri-Host: sensor.example
Uri-Path: r
(S allocates the available Token value 0x7b)
(S sends to itself a phantom observation
request PH_REQ as coming from the
IP multicast address GRP_ADDR)
Token: 0x7b
GET
Observe: 0 (register)
Uri-Host: sensor.example
Uri-Path: r
(S creates a group observation of /r)
(S increments the observer counter
for the group observation of /r)
Token: 0x5e
5.03
Content-Format: application/
informative-response+cbor
Max-Age: 0
<Other options>
Payload: {
tp_info : [1, bstr(SRV_ADDR), SRV_PORT,
0x7b, bstr(GRP_ADDR),
GRP_PORT],
last_notif : bstr(0x45 | OPT |
0xff | PAYLOAD)
}
(PAYLOAD in 'last_notif' : "1234")


(The proxy starts listening to the
GRP_ADDR address and the GRP_PORT port.)
(The proxy adds C1 to its list of observers.)
Token: 0x4a
2.05
Observe: 54120
Content-Format: application/cbor
<Other options>
Payload: "1234"
Token: 0x01
GET
Observe: 0 (register)
Proxy-Uri: coap://sensor.example/r
(The proxy has a fresh cache representation)
Token: 0x01
2.05
Observe: 54120
Content-Format: application/cbor
<Other options>
Payload: "1234"
(The value of the resource
/r changes to "5678".)
(*)
Token: 0x7b
2.05
Observe: 11
Content-Format: application/cbor
<Other options>
Payload: "5678"
Token: 0x4a
2.05
Observe: 54123
Content-Format: application/cbor
<Other options>
Payload: "5678"
Token: 0x01
2.05
Observe: 54123
Content-Format: application/cbor
<Other options>

```
|      |      |        |  Payload: "5678"
|      |      |        |
```

(*) Sent over IP multicast to GROUP_ADDR:GROUP_PORT.

Figure 12: Example of group observation with a proxy

Note that the proxy has all the information to understand the observation request from C2, and can immediately start to serve the still fresh values.

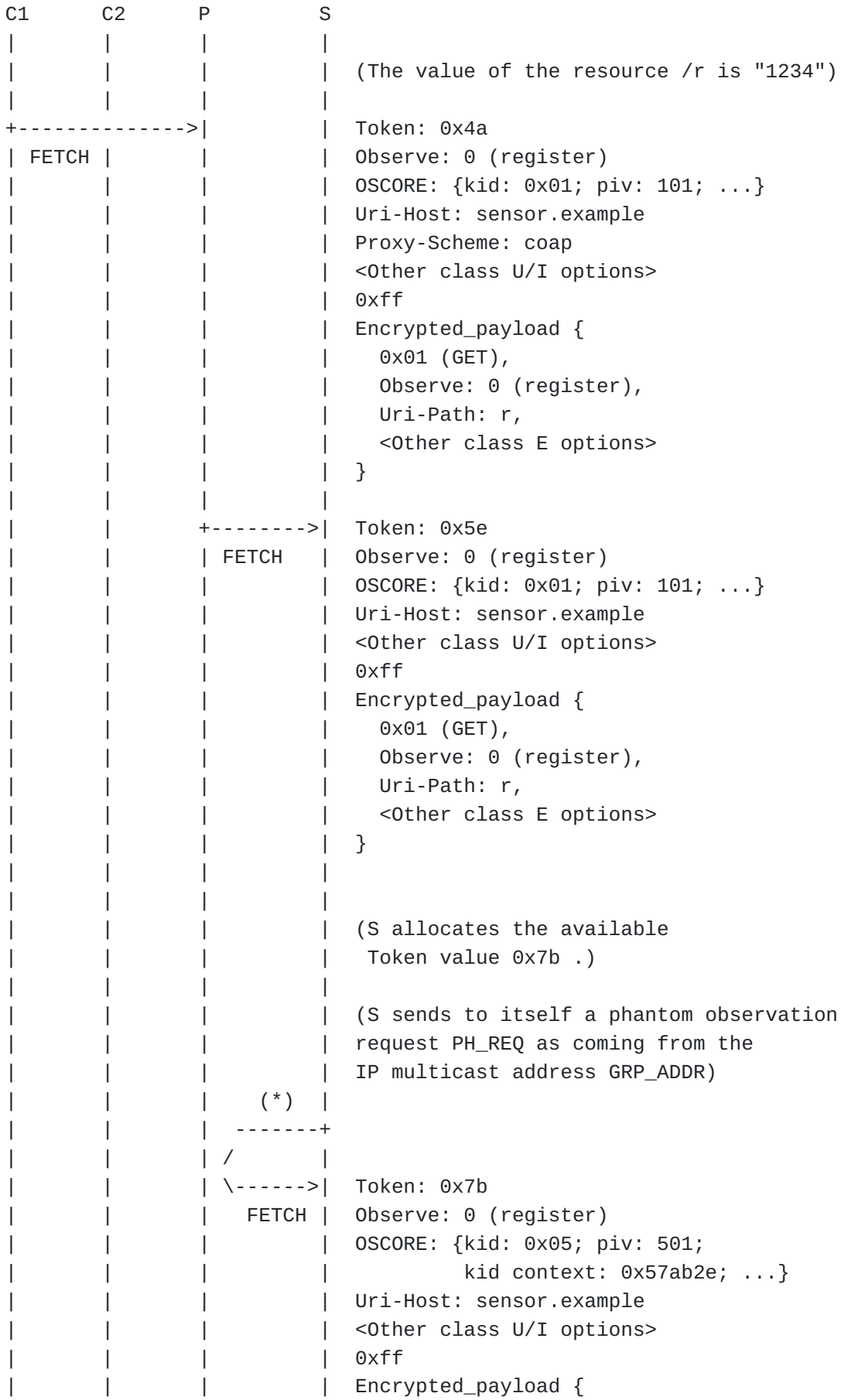
This behavior is mandated by Section 5 of [RFC7641], i.e., the proxy registers itself only once with the next hop and fans out the notifications it receives to all the registered clients.

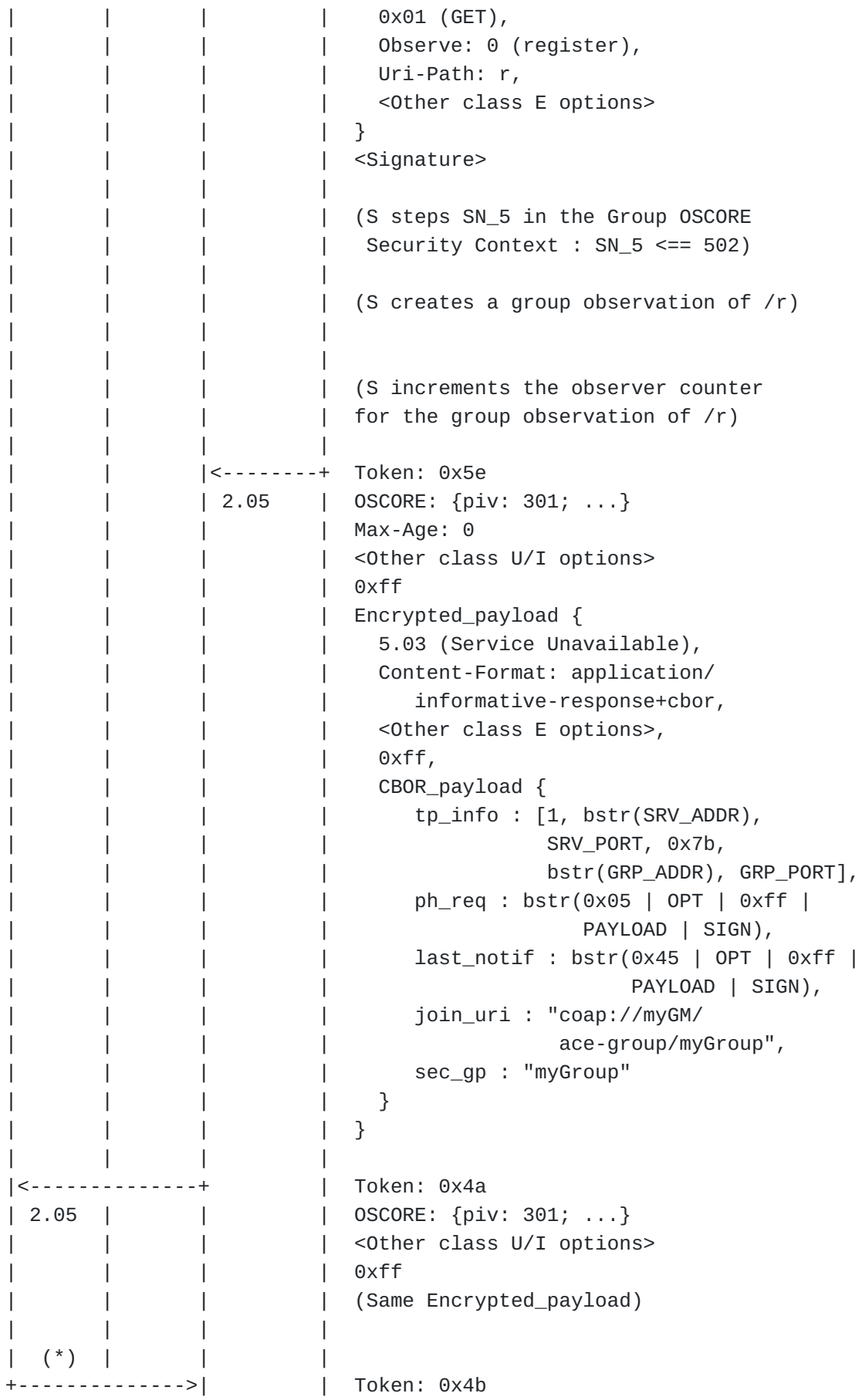
## Appendix F. Example with a Proxy and Group OSCORE

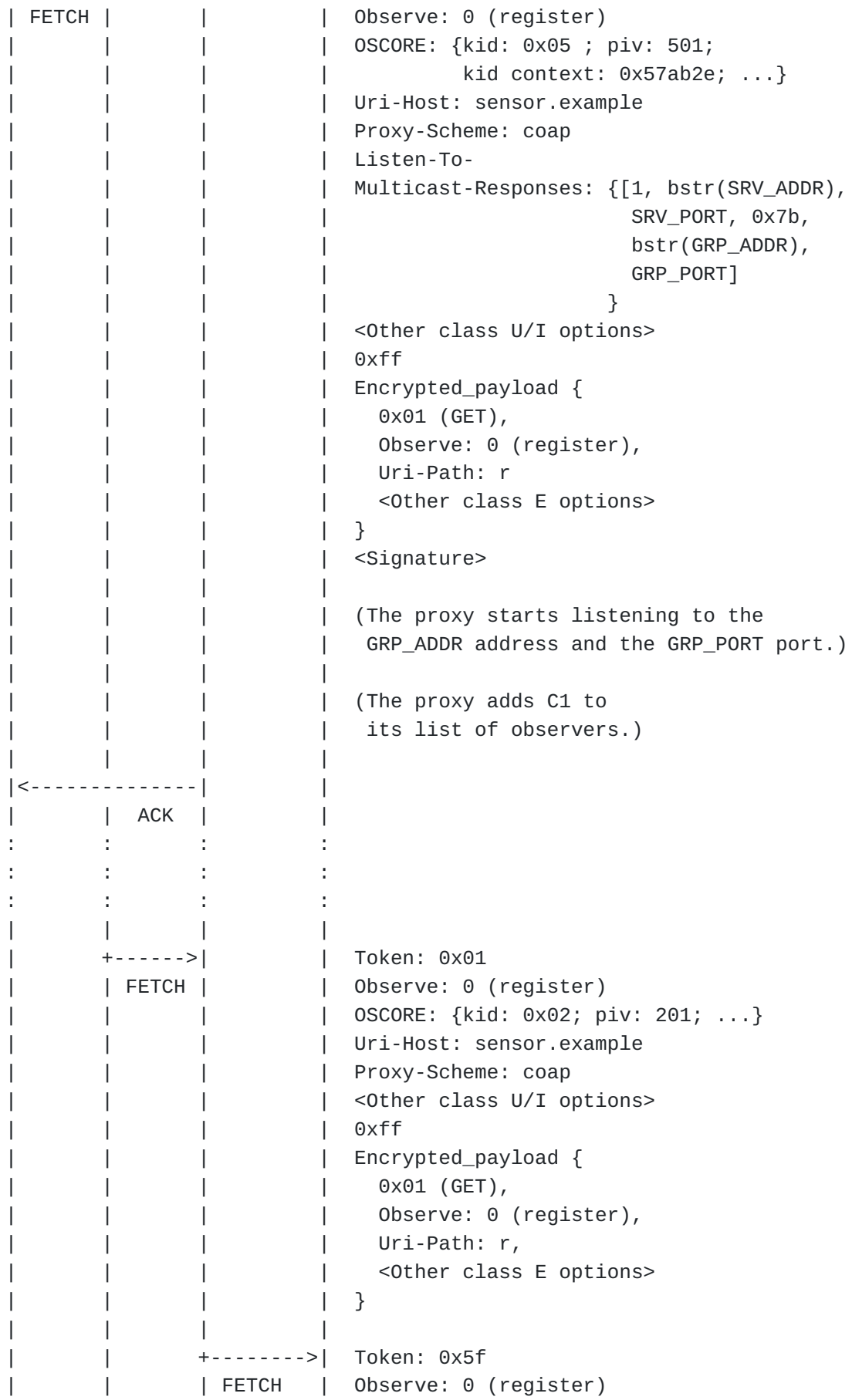
This section provides an example when a proxy P is used between the clients and the server, and Group OSCORE is used to protect multicast notifications end-to-end between the server and the clients.

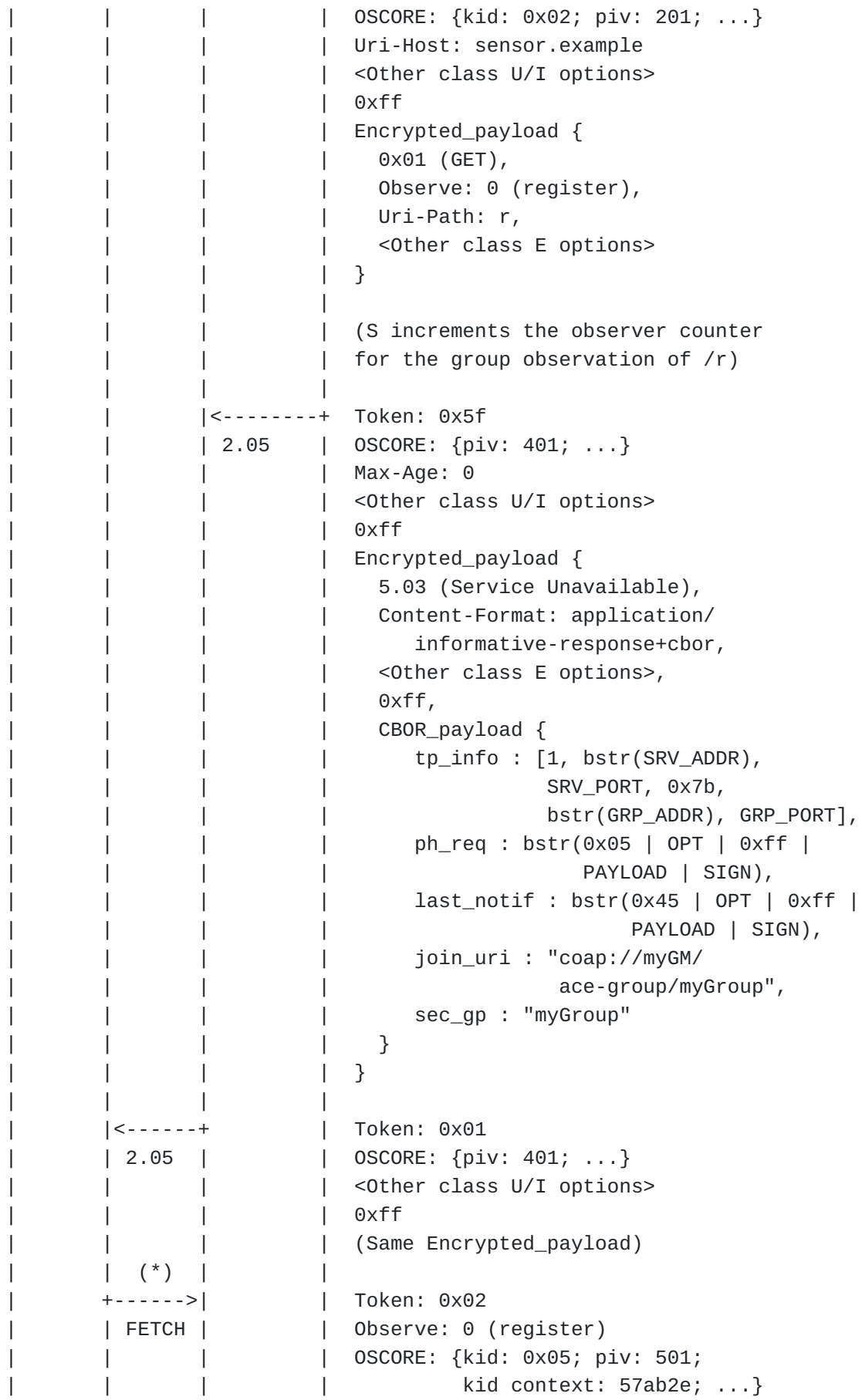
The same assumptions and notation used in Section 10 are used for this example. In addition, the proxy has address PRX_ADDR and listens to the port number PRX_PORT.

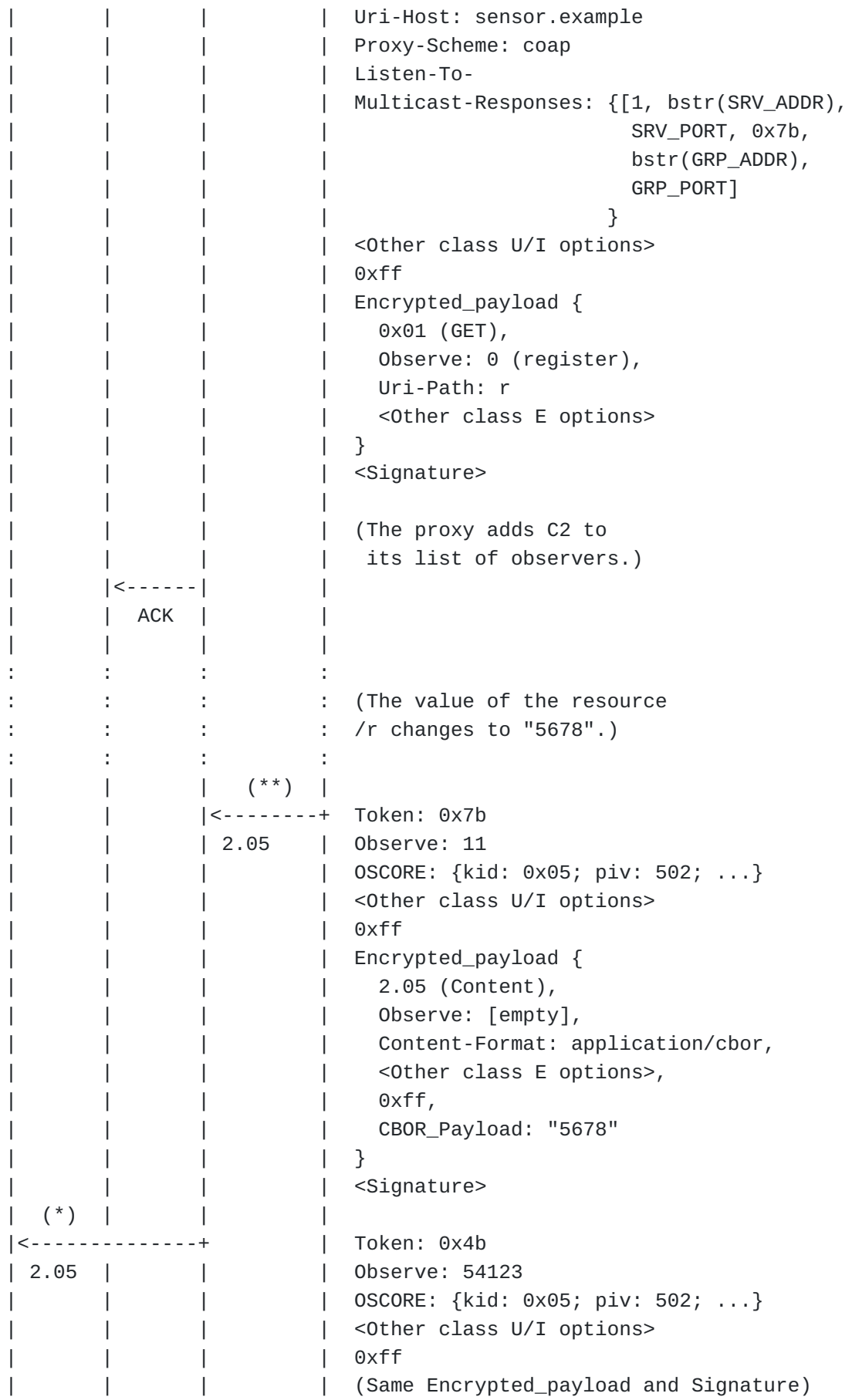
Unless explicitly indicated, all messages transmitted on the wire are sent over unicast and protected with OSCORE end-to-end between a client and the server.

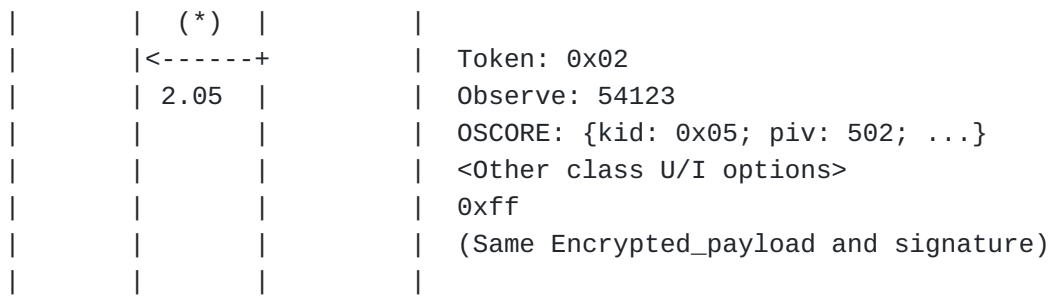

C1
C2
P
S
(The value of the resource /r is "1234")
FETCH
Token: 0x4a
Observe: 0 (register)
OSCORE: {kid: 0x01; piv: 101; ...}
Uri-Host: sensor.example
Proxy-Scheme: coap
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
Uri-Path: r,
<Other class E options>
}
FETCH
Token: 0x5e
Observe: 0 (register)
OSCORE: {kid: 0x01; piv: 101; ...}
Uri-Host: sensor.example
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
Uri-Path: r,
<Other class E options>
}
(S allocates the available
Token value 0x7b .)
(S sends to itself a phantom observation
request PH_REQ as coming from the
IP multicast address GRP_ADDR)
(*)
Token: 0x7b
FETCH
Observe: 0 (register)
OSCORE: {kid: 0x05; piv: 501;
kid context: 0x57ab2e; ...}
Uri-Host: sensor.example
<Other class U/I options>
0xff
Encrypted_payload {


0x01 (GET),
Observe: 0 (register),
Uri-Path: r,
<Other class E options>
}
<Signature>
(S steps SN_5 in the Group OSCORE
Security Context : SN_5 <== 502)
(S creates a group observation of /r)
(S increments the observer counter
for the group observation of /r)
Token: 0x5e
2.05
OSCORE: {piv: 301; ...}
Max-Age: 0
<Other class U/I options>
0xff
Encrypted_payload {
5.03 (Service Unavailable),
Content-Format: application/
informative-response+cbor,
<Other class E options>,
0xff,
CBOR_payload {
tp_info : [1, bstr(SRV_ADDR),
SRV_PORT, 0x7b,
bstr(GRP_ADDR), GRP_PORT],
ph_req : bstr(0x05 | OPT | 0xff |
PAYLOAD | SIGN),
last_notif : bstr(0x45 | OPT | 0xff |
PAYLOAD | SIGN),
join_uri : "coap://myGM/
ace-group/myGroup",
sec_gp : "myGroup"
}
}
Token: 0x4a
2.05
OSCORE: {piv: 301; ...}
<Other class U/I options>
0xff
(Same Encrypted_payload)
(*)
Token: 0x4b


FETCH
Observe: 0 (register)
OSCORE: {kid: 0x05 ; piv: 501;
kid context: 0x57ab2e; ...}
Uri-Host: sensor.example
Proxy-Scheme: coap
Listen-To-
Multicast-Responses: {[1, bstr(SRV_ADDR),
SRV_PORT, 0x7b,
bstr(GRP_ADDR),
GRP_PORT]
}
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
Uri-Path: r
<Other class E options>
}
<Signature>
(The proxy starts listening to the
GRP_ADDR address and the GRP_PORT port.)
(The proxy adds C1 to
its list of observers.)
ACK
FETCH
Token: 0x01
Observe: 0 (register)
OSCORE: {kid: 0x02; piv: 201; ...}
Uri-Host: sensor.example
Proxy-Scheme: coap
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
Uri-Path: r,
<Other class E options>
}
FETCH
Token: 0x5f
Observe: 0 (register)


OSCORE: {kid: 0x02; piv: 201; ...}
Uri-Host: sensor.example
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
Uri-Path: r,
<Other class E options>
}
(S increments the observer counter
for the group observation of /r)
Token: 0x5f
2.05
OSCORE: {piv: 401; ...}
Max-Age: 0
<Other class U/I options>
0xff
Encrypted_payload {
5.03 (Service Unavailable),
Content-Format: application/
informative-response+cbor,
<Other class E options>,
0xff,
CBOR_payload {
tp_info : [1, bstr(SRV_ADDR),
SRV_PORT, 0x7b,
bstr(GRP_ADDR), GRP_PORT],
ph_req : bstr(0x05 | OPT | 0xff |
PAYLOAD | SIGN),
last_notif : bstr(0x45 | OPT | 0xff |
PAYLOAD | SIGN),
join_uri : "coap://myGM/
ace-group/myGroup",
sec_gp : "myGroup"
}
}
Token: 0x01
2.05
OSCORE: {piv: 401; ...}
<Other class U/I options>
0xff
(Same Encrypted_payload)
(*)
Token: 0x02
FETCH
Observe: 0 (register)
OSCORE: {kid: 0x05; piv: 501;
kid context: 57ab2e; ...}


Uri-Host: sensor.example
Proxy-Scheme: coap
Listen-To-
Multicast-Responses: {[1, bstr(SRV_ADDR),
SRV_PORT, 0x7b,
bstr(GRP_ADDR),
GRP_PORT]
}
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
Uri-Path: r
<Other class E options>
}
<Signature>
(The proxy adds C2 to
its list of observers.)
ACK
(The value of the resource
/r changes to "5678".)
(**)
2.05
Token: 0x7b
Observe: 11
OSCORE: {kid: 0x05; piv: 502; ...}
<Other class U/I options>
0xff
Encrypted_payload {
2.05 (Content),
Observe: [empty],
Content-Format: application/cbor,
<Other class E options>,
0xff,
CBOR_Payload: "5678"
}
<Signature>
(*)
2.05
Token: 0x4b
Observe: 54123
OSCORE: {kid: 0x05; piv: 502; ...}
<Other class U/I options>
0xff
(Same Encrypted_payload and Signature)




(*) Sent over unicast, and protected with Group OSCORE end-to-end between the server and the clients.

(**) Sent over IP multicast to GROUP_ADDR:GROUP_PORT, and protected with Group OSCORE end-to-end between the server and the clients.

Figure 13: Example of group observation with a proxy and Group OSCORE

Unlike in the unprotected example in Appendix E, the proxy does *not* have all the information to perform request deduplication, and can only recognize the identical request once the client sends the ticket request.

## Appendix G. Example with a Proxy and Deterministic Requests

This section provides an example when a proxy P is used between the clients and the server, and Group OSCORE is used to protect multicast notifications end-to-end between the server and the clients.

In addition, the phantom request is especially a deterministic request (see Appendix D), which is protected with the pairwise mode of Group OSCORE as defined in [I-D.amsuess-core-cachable-oscore].

### G.1. Assumptions and Walkthrough

The example provided in this appendix as reflected by the message exchange shown in Appendix G.2 assumes the following.

1. The OSCORE group supports deterministic requests. Thus, the server creates the phantom request as a deterministic request [I-D.amsuess-core-cachable-oscore], stores it locally as one of its issued phantom requests, and starts the group observation.

2. The server makes the phantom request available through other means, e.g., a pub-sub broker, together with the transport-specific information for listening to multicast notifications bound to the phantom request (see Appendix A).

3. Since the phantom request is a deterministic request, the server can more efficiently make it available in its smaller, plain version. The clients can obtain it from the particular alternative source and protect it as per Section 3 of [I-D.amsuess-core-cachable-oscore], thus all computing the same deterministic request to be used as phantom observation request.

4. If the client does not rely on a proxy between itself and the server, it simply sets the group observation and starts listening to multicast notifications. Building on point (2) above, the same would happen if the phantom request was not specifically a deterministic request.

5. If the client relies on a proxy between itself and the server, it uses the phantom request as a ticket request (see Section 12). However, unlike the case considered in Section 12 where the ticket request is not a deterministic request, the client does not include a Listen-to-Multicast-Responses Option in the phantom request sent to the proxy.

6. Unlike for the case considered in Section 12, here the proxy does not know that the request is exactly a ticket request for subscribing to multicast notifications. Thus, the proxy simply forwards the ticket request to the server as it normally does for any request.

7. The server receives the ticket request, which is a deviation from the case where the ticket request is not a deterministic request and stops at the proxy (see Section 12). Then, the server can clearly understand what is happening. In fact, as the result of an early check, the server recognizes the phantom request among the stored ones. This happens through a byte-by-byte comparison of the incoming message minus the transport-related fields, i.e., by considering only: i) the outer REST code; ii) the outer options; and iii) the ciphertext from the message payload.

8. Having recognized the incoming request as one of the self-generated deterministic phantom requests made available at external sources, the server does not perform any OSCORE processing on it. This opens for replying to the proxy with an unprotected response, although not signaling any OSCORE-related error.

9. The server starts the group observation and replies with an error response, i.e., the usual 5.03 informative response, including: the transport-specific information, the phantom request, and (optionally) the latest notification.

10. From the received 5.03 (Service Unavailable) response, the proxy retrieves everything needed to set itself as an observer in the group observation, and it starts listening to multicast notifications. If the 5.03 (Service Unavailable) response included a latest notification, the proxy caches it and forwards it back to the client, otherwise it replies with an empty ACK (if it has not done it already and the request from the client was Confirmable).

11. Like in the case with a non-deterministic phantom request considered in Section 12, the proxy fans out the multicast notifications to the origin clients as they come. Also, as new clients following the first one contact the proxy, this does not have to contact the server again as in Section 12, since the deterministic phantom request would produce a cache hit as per [I-D.amsuess-core-cachable-oscore]. Thus, the proxy can serve such clients with the latest fresh multicast notification from its cache.

### G.2. Message Exchange

The same assumptions and notation used in Section 10 are used for this example. As a recap of some specific value:

- Two clients C_1 and C_2 register to observe a resource /r at a Server S, which has address SRV_ADDR and listens to the port number SRV_PORT. Before the following exchanges occur, no clients are observing the resource /r , which has value "1234".

- The server S sends multicast notifications to the IP multicast address GRP_ADDR and port number GRP_PORT, and starts the group observation already after creating the deterministic phantom request to early disseminate.

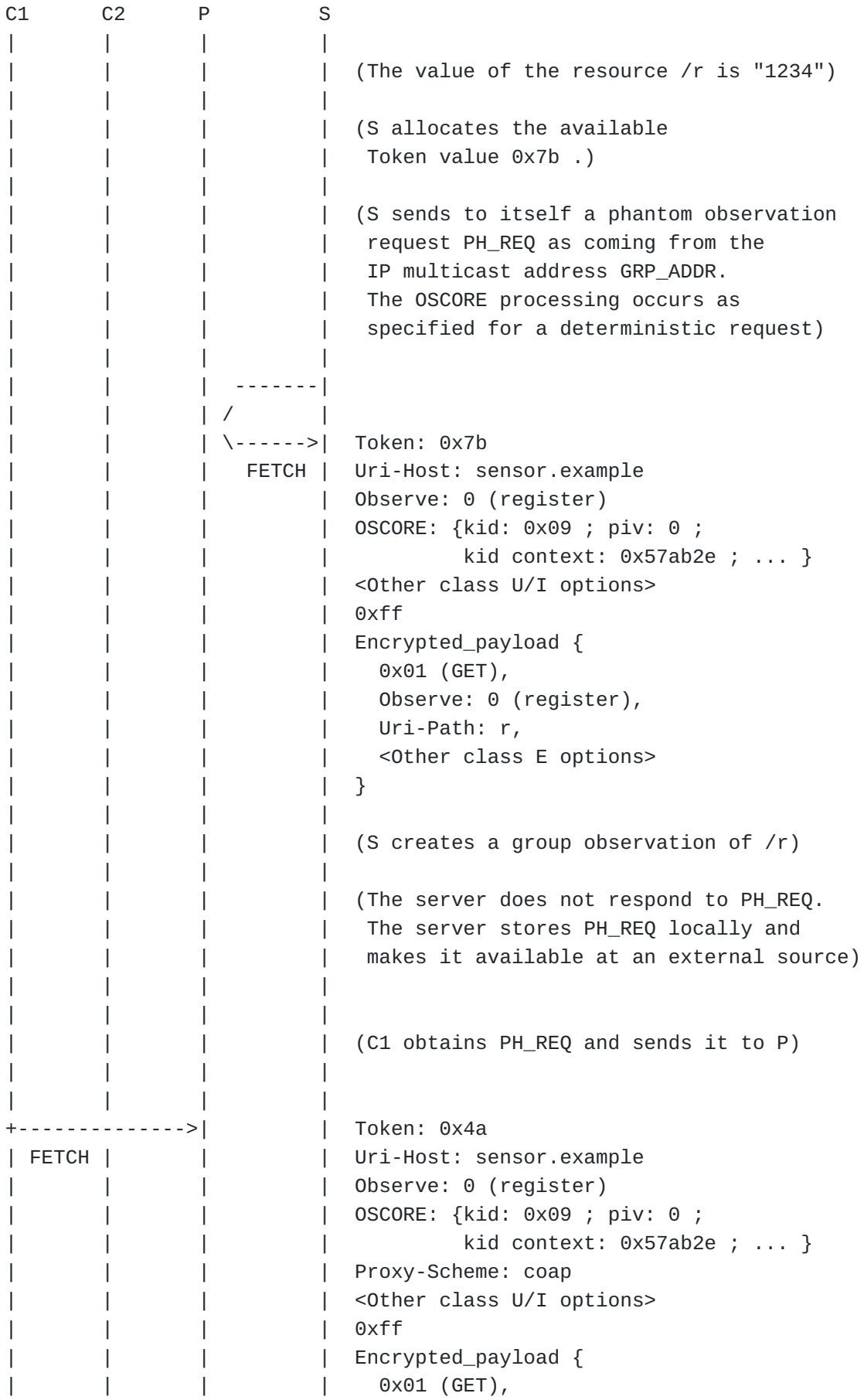
- S is a member of the OSCORE group with 'kid context' = 0x57ab2e as Group ID. In the OSCORE group, S has 'kid' = 0x05 as Sender ID, and SN_5 = 501 as Sender Sequence Number.

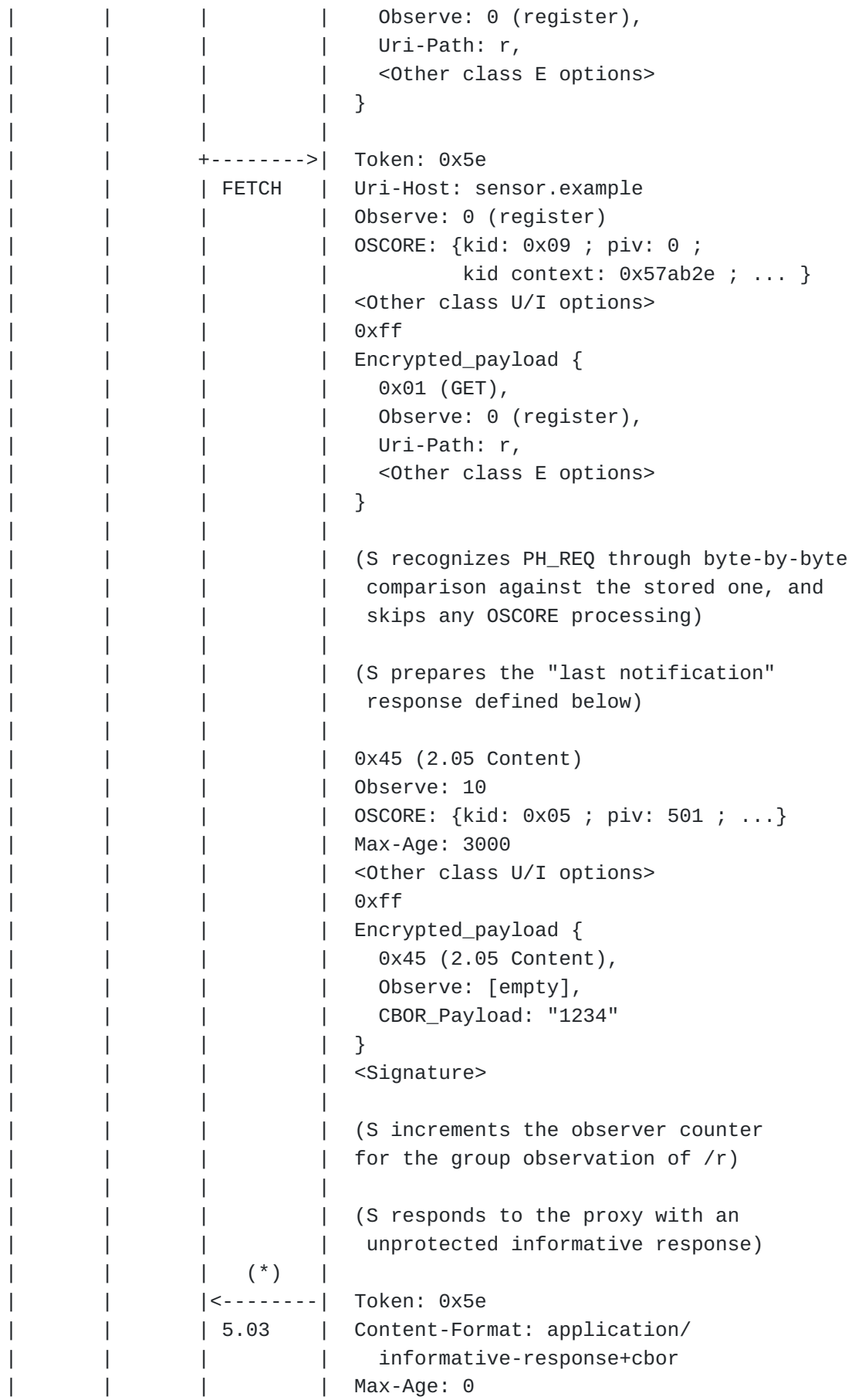
In addition:

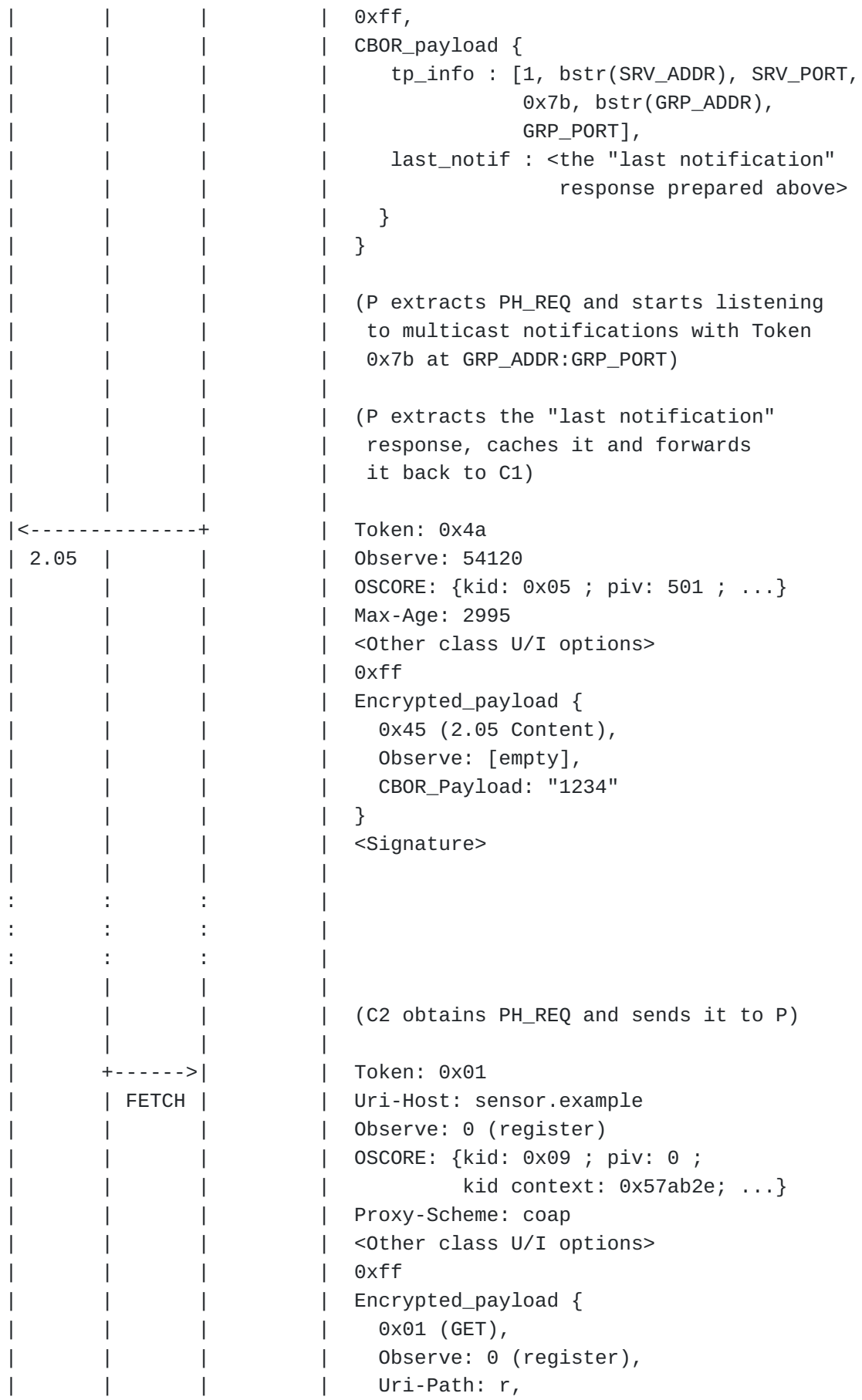
- The proxy has address PRX_ADDR and listens to the port number PRX_PORT.

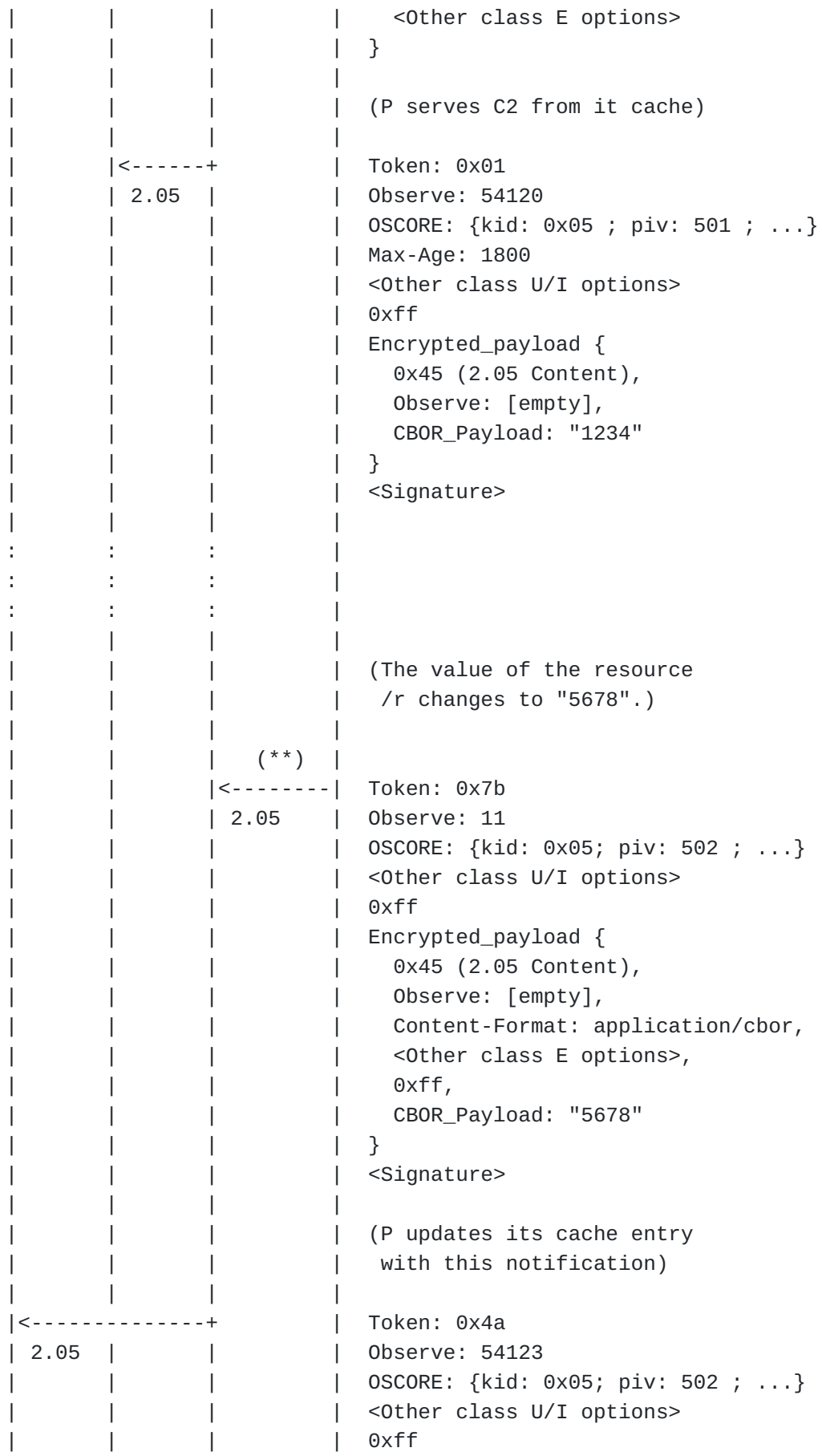
- The deterministic client in the OSCORE group has 'kid' = 0x09.

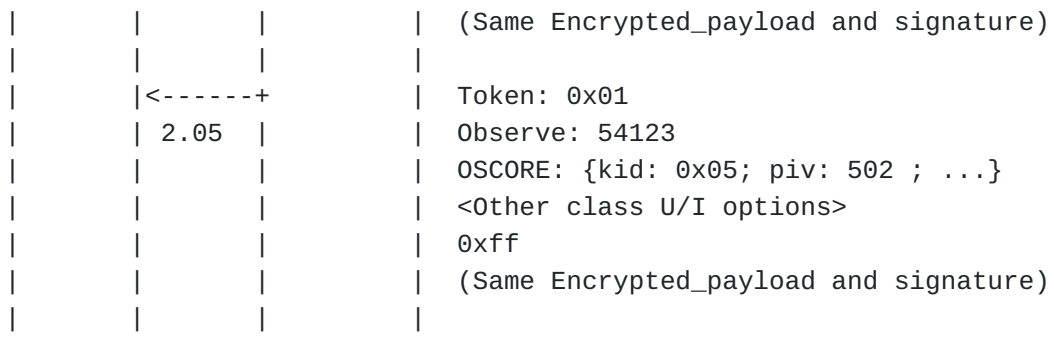
Unless explicitly indicated, all messages transmitted on the wire are sent over unicast and protected with Group OSCORE end-to-end between a client and the server.


C1
C2
P
S
(The value of the resource /r is "1234")
(S allocates the available
Token value 0x7b .)
(S sends to itself a phantom observation
request PH_REQ as coming from the
IP multicast address GRP_ADDR.
The OSCORE processing occurs as
specified for a deterministic request)
FETCH
Token: 0x7b
Uri-Host: sensor.example
Observe: 0 (register)
OSCORE: {kid: 0x09 ; piv: 0 ;
kid context: 0x57ab2e ; ... }
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
Uri-Path: r,
<Other class E options>
}
(S creates a group observation of /r)
(The server does not respond to PH_REQ.
The server stores PH_REQ locally and
makes it available at an external source)
(C1 obtains PH_REQ and sends it to P)
FETCH
Token: 0x4a
Uri-Host: sensor.example
Observe: 0 (register)
OSCORE: {kid: 0x09 ; piv: 0 ;
kid context: 0x57ab2e ; ... }
Proxy-Scheme: coap
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),


Observe: 0 (register),
Uri-Path: r,
<Other class E options>
}
Token: 0x5e
FETCH
Uri-Host: sensor.example
Observe: 0 (register)
OSCORE: {kid: 0x09 ; piv: 0 ;
kid context: 0x57ab2e ; ... }
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
Uri-Path: r,
<Other class E options>
}
(S recognizes PH_REQ through byte-by-byte
comparison against the stored one, and
skips any OSCORE processing)
(S prepares the "last notification"
response defined below)
0x45 (2.05 Content)
Observe: 10
OSCORE: {kid: 0x05 ; piv: 501 ; ...}
Max-Age: 3000
<Other class U/I options>
0xff
Encrypted_payload {
0x45 (2.05 Content),
Observe: [empty],
CBOR_Payload: "1234"
}
<Signature>
(S increments the observer counter
for the group observation of /r)
(S responds to the proxy with an
unprotected informative response)
(*)
Token: 0x5e
5.03
Content-Format: application/
informative-response+cbor
Max-Age: 0


0xff,
CBOR_payload {
tp_info : [1, bstr(SRV_ADDR), SRV_PORT,
0x7b, bstr(GRP_ADDR),
GRP_PORT],
last_notif : <the "last notification"
response prepared above>
}
}
(P extracts PH_REQ and starts listening
to multicast notifications with Token
0x7b at GRP_ADDR:GRP_PORT)
(P extracts the "last notification"
response, caches it and forwards
it back to C1)
2.05
Token: 0x4a
Observe: 54120
OSCORE: {kid: 0x05 ; piv: 501 ; ...}
Max-Age: 2995
<Other class U/I options>
0xff
Encrypted_payload {
0x45 (2.05 Content),
Observe: [empty],
CBOR_Payload: "1234"
}
<Signature>
(C2 obtains PH_REQ and sends it to P)
FETCH
Token: 0x01
Uri-Host: sensor.example
Observe: 0 (register)
OSCORE: {kid: 0x09 ; piv: 0 ;
kid context: 0x57ab2e; ...}
Proxy-Scheme: coap
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
Observe: 0 (register),
Uri-Path: r,


|  |  |  |  <Other class E options>
|  |  |  |  }
|  |  |  |
|  |  |  |  (P serves C2 from it cache)
|  |  |  |
|  |<------+  |  Token: 0x01
|  | 2.05  |  |  Observe: 54120
|  |  |  |  |  OSCORE: {kid: 0x05 ; piv: 501 ; ...}
|  |  |  |  |  Max-Age: 1800
|  |  |  |  |  <Other class U/I options>
|  |  |  |  |  0xff
|  |  |  |  |  Encrypted_payload {
|  |  |  |  |    0x45 (2.05 Content),
|  |  |  |  |    Observe: [empty],
|  |  |  |  |    CBOR_Payload: "1234"
|  |  |  |  |  }
|  |  |  |  |  <Signature>
|  |  |  |  |
:  :  :  :  |
:  :  :  :  |
:  :  :  :  |
|  |  |  |  |
|  |  |  |  |  (The value of the resource
|  |  |  |  |   /r changes to "5678".)
|  |  |  |  |
|  |  |  |   (**)  |
|  |  |  |<--------|  Token: 0x7b
|  |  |  | 2.05    |  Observe: 11
|  |  |  |  |  |  |  OSCORE: {kid: 0x05; piv: 502 ; ...}
|  |  |  |  |  |  |  <Other class U/I options>
|  |  |  |  |  |  |  0xff
|  |  |  |  |  |  |  Encrypted_payload {
|  |  |  |  |  |  |    0x45 (2.05 Content),
|  |  |  |  |  |  |    Observe: [empty],
|  |  |  |  |  |  |    Content-Format: application/cbor,
|  |  |  |  |  |  |    <Other class E options>,
|  |  |  |  |  |  |    0xff,
|  |  |  |  |  |  |    CBOR_Payload: "5678"
|  |  |  |  |  |  |  }
|  |  |  |  |  |  |  <Signature>
|  |  |  |  |  |  |
|  |  |  |  |  |  |  (P updates its cache entry
|  |  |  |  |  |  |   with this notification)
|  |  |  |  |  |  |
|<---------------+  |  Token: 0x4a
| 2.05  |  |  |  |  |  Observe: 54123
|  |  |  |  |  |  |  OSCORE: {kid: 0x05; piv: 502 ; ...}
|  |  |  |  |  |  |  <Other class U/I options>
|  |  |  |  |  |  |  0xff




(*) Sent over unicast and unprotected.

(**) Sent over IP multicast to GROUP_ADDR:GROUP_PORT, and protected with Group OSCORE end-to-end between the server and the clients.

## Appendix H. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

### H.1. Version -07 to -08

*Fixed the CDDL definition 'srv_addr' in 'tp_info'.

*Early mentioning that 'srv_addr' cannot instruct redirection.

*The replay protection of multicast notifications is as per Group OSCORE.

*Consistently use the format uint for the Multicast-Response-Feedback-Divider Option.

*Fixed consumption of proxy-related options in a ticket request sent to the proxy.

*Possible use of the option Proxy-Cri or Proxy-Scheme-Number in a ticket request.

*Explained non-provisioning of some parameters in self-managed OSCORE groups.

*Use of 'exi' for relative expiration time in self-managed OSCORE groups.

*Improved notation in the examples of message exchanges with proxy.

*Clarifications and editorial improvements.

### H.2. Version -06 to -07

* Added more details on proxies that do not support the Multicast-Response-Feedback-Divider Option.

* Added more details on the reliability of the client rough counting.

* Added more details on the unreliability of counting new clients, when the phantom request is obtained from other sources or is an OSCORE deterministic request.

* Revised parameter naming.

* Fixes in IANA considerations.

* Editorial improvements.

### H.3. Version -05 to -06

* Clarified rough counting of clients when a proxy is used

* IANA considerations: registration of target attribute "gp-obs"

* Editorial improvements.

### H.4. Version -04 to -05

* If the phantom request is an OSCORE deterministic request, there is no parallel group observation for clients that lack support.

* Clarification on pre-configured clients.

* Clarified early publication of phantom request.

* Fixes in IANA considerations.

* Fixed example with proxy and Group OSCORE.

* Editorial improvements.

### H.5. Version -03 to -04

* Added a new section on prerequisites.

* Added a new section overviewing alternative variants.

* Consistent renaming of 'cli_addr' to 'cli_host' in 'tp_info'.

* Added pre-requirements for early retrieval of phantom request.

*Revised example with early retrieval of phantom request.

*Clarified use, rationale and example of phantom request as deterministic request.

*Editorial improvements.

### H.6. Version -02 to -03

*Distinction between authentication credential and public key.

*Fixed processing of informative response at the client, when Group OSCORE is used.

*Discussed scenarios with pre-configured address/port and Token value.

### H.7. Version -01 to -02

*Clarifications on client rough counting and IP multicast scope.

*The 'ph_req' parameter is optional in the informative response.

*New parameter 'next_not_before' for the informative response.

*Optimization when rekeying the self-managed OSCORE group.

*Security considerations on unsecured multicast notifications.

*Protection of the ticket request sent to a proxy.

*RFC8126 terminology in IANA considerations.

*Editorial improvements.

### H.8. Version -00 to -01

*Simplified cancellation of the group observation, without using a phantom cancellation request.

*Aligned parameter semantics with core-oscore-groupcomm and ace-key-groupcomm-oscore.

*Clarifications about self-managed OSCORE group and use of deterministic requests for cacheable OSCORE.

*New example with a proxy, Group OSCORE and a deterministic phantom request.

*Fixes in the examples and editorial improvements.

## Acknowledgments

The authors sincerely thank Carsten Bormann, Klaus Hartke, Jaime Jiménez, John Preuß Mattsson, Jim Schaad, Ludwig Seitz, and Göran 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

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

Email: rikard.hoglund@ri.se

Christian Amsüss
Hollandstr. 12/4
1020 Vienna
Austria

Email: christian@amsuess.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden

Email: francesca.palombini@ericsson.com