Workgroup: CoRE Working Group
Internet-Draft:
draft-ietf-core-groupcomm-bis-06
Obsoletes: 7390 (if approved)
Updates: 7252, 7641 (if approved)
Published: 7 March 2022
Intended Status: Standards Track
Expires: 8 September 2022
Authors: E. Dijk — IoTconsultancy.nl; C. Wang — InterDigital; M. Tiloca — RISE AB

Group Communication for the Constrained Application Protocol (CoAP)

Abstract

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

Discussion Venues

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

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

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

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

1.1. Scope

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

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

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

1.2. Terminology

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

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

In addition, the following terms are extensively used.

* Group URI - This is defined as a CoAP URI that has the "coap" scheme and includes in the authority component either an IP multicast address or a group hostname (e.g., a Group Fully Qualified Domain Name (FQDN)) that can be resolved to an IP multicast address. A group URI also contains an optional UDP port number in the authority component. Group URIs follow the regular CoAP URI syntax (see Section 6 of [RFC7252]).

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

1.3. Changes to Other Documents

This document obsoletes and replaces [RFC7390] as follows.

* It defines separate definitions for CoAP groups, application groups and security groups, together with high-level guidelines on their configuration (see Section 2).

* It defines the use of Group OSCORE [I-D.ietf-core-oscore-groupcomm] as the security protocol to protect group communication for CoAP, together with high-level guidelines on secure group maintenance (see Section 5).

* It updates all the guidelines about using group communication for CoAP (see Section 3).

* It strongly discourages unsecured group communication for CoAP based on the "NoSec" mode (see Section 4 and Section 6.1) and highlights the risk of amplification attacks (see Section 6.3).

This document updates [RFC7252] as follows.

* It updates the request/response model for group communication, as to response suppression (see Section 3.1.2) and token reuse time (see Section 3.1.5).

* It updates the freshness model and validation model to use for cached responses (see Section 3.2.1 and Section 3.2.2).

This document updates [RFC7641] as follows.

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

2. Group Definition and Group Configuration

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

2.1. Group Definition

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

2.1.1. CoAP Group

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

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

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

2.1.2. Application Group

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

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

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

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

2.1.3. Security Group

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

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

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

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

2.1.4. Relations Between Group Types

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

```
(application group, CoAP group, security group)
```

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

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

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

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

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

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

Figure 1: Relations Among Different Group Types

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

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

Figure 2: Deployment Example of Different Group Types

2.2. Group Configuration

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

2.2.1. Group Naming

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

2.2.1.1. CoAP Groups

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

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

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

2.2.1.2. Application Groups

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

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

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

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

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

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

```
Application group name: gp1

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

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example.org
   Uri-Path: gp
   Uri-Path: gp1
   Uri-Path: light
   Uri-Query: foo=bar
```

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

```
Application group name: gp1

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

CoAP group request
   Header: POST (T=NON, Code=0.02, MID=0x7d41)
   Uri-Path: g
   Uri-Path: gp1
   Uri-Path: li
```

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

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

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

    A full-fledged example is provided in Figure 5.

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

A full-fledged example is provided in Figure 6.

```
Application group name: gp1

Group URI: coap://grp.example.org:5685/light?gp1

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example.org
   Uri-Path: light
   Uri-Query: gp1
```

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

```
Application group name: gp1

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

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example.org
   Uri-Path: light
   Uri-Query: foo=bar
   Uri-Query: gp=gp1
```

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

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

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

  A full-fledged example is provided in Figure 7.

```
Application group name: grp.example.org:5685

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

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example.org
   Uri-Path: light
   Uri-Query: foo=bar
```

Figure 7: Example of application group name in URI authority

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

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

  A full-fledged example is provided in Figure 8.

```
Application group name: grp.example.org

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

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example.org
   Uri-Path: light
   Uri-Query: foo=bar
```

Figure 8: Example of application group name in URI host

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

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

  A full-fledged example is provided in Figure 9.

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

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

CoAP group request
   Header: GET(T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example.org
   Uri-Path: light
   Uri-Query: foo=bar
```

Figure 9: Example of application group name in URI port

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

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

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

  A full-fledged example is provided in Figure 10.

```
Application group name: grp1

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

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example.org
   Uri-Path: light
   Uri-Query: foo=bar
   App-Group-Name: grp1  // new (e.g., custom) CoAP option
```

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

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

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

2.2.1.3. Security Groups

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

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

2.2.2. Group Creation and Membership

To create a CoAP group, a configuring entity defines an IP multicast address (or hostname) for the group and optionally a UDP port number in case it differs from the default CoAP port number 5683. Then, it configures one or more devices as listeners to that IP multicast address, with a CoAP endpoint listening on the group's associated UDP port. These endpoints/devices are the group members. The configuring entity can be, for example, a local application with pre-configuration, a user, a software developer, a cloud service, or a local commissioning tool. Also, the devices sending CoAP requests to the group in the role of CoAP client need to be configured with the same information, even though they are not necessarily group members. One way to configure a client is to supply it with a group URI. The IETF does not define a mandatory protocol to accomplish CoAP group creation. [RFC7390] defined an experimental protocol for configuration of group membership for unsecured group communication, based on JSON-formatted configuration resources. For IPv6 CoAP groups, common multicast address ranges that are used to configure group addresses from are ff1x::/16 and ff3x::/16.

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

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

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

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

2.2.3. Group Discovery

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

2.2.3.1. Discovery through a Resource Directory

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

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

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

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

2.2.3.2. Discovery from the CoAP Servers

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

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

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

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

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

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

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

A full-fledged example is provided in Figure 11.

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

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

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

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

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

* A CoAP client can discover the CoAP servers that are members of a specific application group, the CoAP group associated with the

application group, and optionally the resources that those servers host for each application group.

This is achieved by sending the GET request above to the "All CoAP Nodes" IP multicast address (see Section 12.8 of [RFC7252]), with a particular chosen scope (e.g., site-local or realm-local) if IPv6 is used. Also, the request specifies the application group name of interest in the URI query component, as defined in Section 2.2.1.2. For example, the request can use a Group URI with path and query components "/.well-known/core?href=/gp/gp1" to specify the application group with name "gp1".

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

A full-fledged example is provided in Figure 12. Note that the servers need to respond now with an absolute URI and not a relative URI like in the previous example, because the group resource "gp1" is hosted at the authority indicated by the CoAP group (grp.example.org:5685) and not by the authority that is serving the Link Format document (which is [ff03::fd]:5683).

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

```
// Request to realm-local members of the application group "gp1"
Req: GET coap://[ff03::fd]/.well-known/core?href=/gp/gp1

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

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

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

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

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

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

A full-fledged example is provided in Figure 13.

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

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

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

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

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

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

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

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

A full-fledged example is provided in Figure 14.

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

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

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

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

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

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

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

For example, some applications may use the "rt" attribute on a parent resource to denote support for a particular REST API to access child resources, as detailed in the example in Figure 15. In this example a custom Link Format attribute "gpt" is used to denote the group type within the scope of the application/system. An alternative, shorter encoding (not shown in the figure) is to use only the value "1" for each "gpt" attribute, to denote that the resource is of type application group. In that case information about the semantics/API of the group resource is disclosed only via the "rt" attribute as shown in the figure.

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

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

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

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

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

2.2.4. Group Maintenance

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

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

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

3. CoAP Usage in Group Communication

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

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

3.1. Request/Response Model

3.1.1. General

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

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

3.1.2. Response Suppression

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

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

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

3.1.3. Repeating a Request

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

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

3.1.4. Request/Response Matching and Distinguishing Responses

A CoAP client can distinguish the origin of multiple server responses by the source IP address of the message containing the

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

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

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

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

3.1.5. Token Reuse

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

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

```
MIN_TOKEN_REUSE_TIME = (NON_LIFETIME + MAX_LATENCY +
                        MAX_SERVER_RESPONSE_DELAY)
```

where NON_LIFETIME and MAX_LATENCY are defined in Section 4.8 of [RFC7252]. This specification defines MAX_SERVER_RESPONSE_DELAY as was done in [RFC7390], that is: the expected maximum response delay over all servers that the client can send a CoAP group request to. This delay includes the maximum Leisure time period as defined in

Section 8.2 of [RFC7252]. However, CoAP does not define a time limit for the server response delay. Using the default CoAP parameters, the Token reuse time MUST be greater than 250 seconds plus MAX_SERVER_RESPONSE_DELAY.

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

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

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

3.1.6. Client Handling of Multiple Responses With Same Token

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

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

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

3.2. Caching

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

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

* A client sending a GET or FETCH group request MAY update a cache with the responses from the servers in the CoAP group. Then, the client uses both cached-still-fresh and new responses as the result of the group request.

* A client sending a GET or FETCH group request MAY use a response received from a server, to satisfy a subsequent sent request intended to that server on the related unicast request URI. In particular, the unicast request URI is obtained by replacing the authority component of the request URI with the transport-layer source address of the cached response message.

* A client MAY revalidate a cached response by making a GET or FETCH request on the related unicast request URI.

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

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

3.2.1. Freshness Model

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

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

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

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

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

3.2.2. Validation Model

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

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

3.2.2.1. ETag Option in a Group Request/Response

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

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

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

3.3. URI Path Selection

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

3.4. Port Selection for UDP Transport

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

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

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

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

3.5. Proxy Operation

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

3.5.1. Forward-Proxies

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

When intending to reach a CoAP group through a proxy, the client sends a unicast CoAP group request to the proxy. This request specifies the group URI where the request has to be forwarded to, either as a string in the Proxy-URI Option, or through the Proxy-Scheme Option with the group URI constructed from the usual Uri-* Options. Then, the forward-proxy resolves the group URI to a destination CoAP group, i.e., it sends (e.g., multicasts) the CoAP group request to the group URI, receives the responses and forwards all the individual (unicast) responses back to the client.

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

* The CoAP client component that has sent the unicast CoAP group request to the proxy may be expecting only one (unicast) response, as usual for a CoAP unicast request. Instead, it receives multiple (unicast) responses, potentially leading to fault conditions in the component or to discarding any received responses following the first one. This issue may occur even if the application calling the CoAP client component is aware that the forward-proxy is going to forward the CoAP group request to the group URI.

* Each individual CoAP response received by the client will appear to originate (based on its IP source address) from the CoAP Proxy, and not from the server that produced the response. This makes it impossible for the client to identify the server that produced each response, unless the server identity is contained

as a part of the response payload or inside a CoAP option in the response.

* The proxy does not necessarily know how many members there are in the CoAP group or how many group members will actually respond. Also, the proxy does not know for how long to collect responses before it stops forwarding them to the client. A CoAP client that is not using a Proxy might face the same problems in collecting responses to a group request. However, the client itself would typically have application-specific rules or knowledge on how to handle this situation, while an application-agnostic CoAP Proxy would typically not have this knowledge. For example, a CoAP client could monitor incoming responses and use this information to decide how long to continue collecting responses - which is something a proxy cannot do.

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

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

* Like for the approach discussed above, the proxy does not know for how long to collect responses before sending back the aggregated response to the client. Analogous considerations apply to this approach too, both on the client and proxy side.

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

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

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

3.5.2. Reverse-Proxies

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

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

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

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

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

* The three issues and limitations defined in Section 3.5.1 for a forward proxy apply to a reverse-proxy as well, and have to be addressed, e.g., using the signaling method defined in [I-D.tiloca-core-groupcomm-proxy] or other means.

* A reverse-proxy MAY have preconfigured time duration(s) that are used for the collecting of server responses and forwarding these back to the client. These duration(s) may be set as global configuration or resource-specific configurations. If there is such preconfiguration, then an explicit signaling of the time period in the client's request as defined in [I-D.tiloca-core-groupcomm-proxy] is not necessarily needed.

* A client that is configured to access a reverse-proxy resource (i.e., one that triggers a CoAP group communication request)

SHOULD be configured also to handle potentially multiple responses with the same Token value caused by a single request.

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

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

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

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

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

3.6. Congestion Control

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

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

* A server may choose not to respond to an IP multicast request if there is nothing useful to respond to, e.g., error or empty response (see Section 8.2 of [RFC7252]).

* A server should limit the support for IP multicast requests to specific resources where multicast operation is required (Section 11.3 of [RFC7252]).

* An IP multicast request MUST be Non-confirmable (Section 8.1 of [RFC7252]).

* A response to an IP multicast request SHOULD be Non-confirmable (Section 5.2.3 of [RFC7252]).

* A server does not respond immediately to an IP multicast request and should first wait for a time that is randomly picked within a predetermined time interval called the Leisure (Section 8.2 of [RFC7252]).

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

* A server in a constrained network SHOULD only support group requests for resources that have a small representation (where the representation may be retrieved via a GET, FETCH or POST method in the request). For example, "small" can be defined as a response payload limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size, so that it fits into a single link-layer frame in case IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN, see Section 3.9.3) is used on the constrained network.

* A server SHOULD minimize the payload size of a response to a group GET or FETCH request on "/.well-known/core" by using hierarchy in arranging link descriptions for the response. An example of this is given in Section 5 of [RFC6690].

* A server MAY minimize the payload size of a response to a group GET or FETCH request (e.g., on "/.well-known/core") by using CoAP block-wise transfers [RFC7959] in case the payload is long, returning only a first block of the CoRE Link Format description. For this reason, a CoAP client sending a CoAP group request to "/.well-known/core" SHOULD support block-wise transfers. See also Section 3.8.

* A client SHOULD be configured to use CoAP groups with the smallest possible IP multicast scope that fulfills the application needs. As an example, site-local scope is always preferred over global scope IP multicast if this fulfills the application needs. Similarly, realm-local scope is always preferred over site-local scope if this fulfills the application needs.

3.7. Observing Resources

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

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

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

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

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

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

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

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

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

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

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

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

3.8. Block-Wise Transfer

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

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

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

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

3.9. Transport Protocols

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

3.9.1. UDP/IPv6 Multicast Transport

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

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

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

3.9.2. UDP/IPv4 Multicast Transport

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

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

3.9.3. 6LoWPAN

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

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

3.9.4. Other Transports

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

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

3.10. Interworking with Other Protocols

3.10.1. MLD/MLDv2/IGMP/IGMPv3

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

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

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

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

3.10.2. RPL

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

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

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

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

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

3.10.3. MPL

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

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

Methods to configure which multicast groups are to be propagated into the MPL Domain could be:

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

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

4. Unsecured Group Communication

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

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

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

5. Secured Group Communication using Group OSCORE

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

5.1. Group OSCORE

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

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

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

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

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

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

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

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

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

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

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

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

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

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

* All further CoAP endpoints configured only as CoAP clients, that may send CoAP group requests to the CoAP group.

5.2. Secure Group Maintenance

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

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

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

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

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

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

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

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

5.3. Proxy Security

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

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

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

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

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

6. Security Considerations

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

6.1. CoAP NoSec Mode

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

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

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

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

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

6.2. Group OSCORE

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

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

6.2.1. Group Key Management

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

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

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

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

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

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

6.2.2. Source Authentication

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

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

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

6.2.3. Countering Attacks

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

* Since Group OSCORE provides end-to-end confidentiality and integrity of request/response messages, proxies capable of group communication cannot break message protection, and thus cannot act as man-in-the-middle beyond their legitimate duties (see Section 11.2 of [RFC7252]). In fact, intermediaries such as

proxies are not assumed to have access to the OSCORE Security Context used by group members. Also, with the notable addition of signatures for the group mode, Group OSCORE protects messages using the same procedure as OSCORE (see Sections 8 and 9 of [I-D.ietf-core-oscore-groupcomm]), and especially processes CoAP options according to the same classification in U/I/E classes.

* Group OSCORE limits the feasibility and impact of amplification attacks, see Section 6.3 of this document and Section 11.3 of [RFC7252].

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

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

  Furthermore, the adversary needs to consider a group request that specifically targets a resource for which the CoAP servers are configured to respond. While this can be often correctly assumed or inferable from the application context, it is not explicit from the group request itself, since Group OSCORE protects the Uri-Path and Uri-Query CoAP Options conveying the respective components of the target URI.

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

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

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

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

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

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

6.3. Risk of Amplification

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

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

Thus, consistently with Section 11.3 of [RFC7252], a server in a CoAP group:

* SHOULD limit the support for CoAP group requests only to the group resources of the application group(s) using that CoAP group;

* SHOULD NOT accept group requests that can not be authenticated in some way;

* SHOULD NOT provide large amplification factors through its responses to a non-authenticated group request, and can possibly rely on CoAP block-wise transfers [RFC7959] to reduce the amount of amplification.

Amplifications attacks using CoAP are further discussed in [I-D.mattsson-t2trg-amplification-attacks], which also highlights how the amplification factor would become even higher when CoAP group communication is combined with resource observation [RFC7641]. That is, a single group request may result in multiple notification responses from each of the responding servers, throughout the observation lifetime.

Thus, consistently with Section 7 of [RFC7641], a server in a CoAP group MUST strictly limit the number of notifications it sends between receiving acknowledgments that confirm the actual interest of the client in continuing the observation.

Moreover, it is especially easy to perform an amplification attack when the NoSec mode is used. Therefore, even in case of non-sensitive and non-critical applications, it is generally NOT RECOMMENDED to use CoAP group communication in NoSec mode, in order to prevent an easy proliferation of high-volume amplification attacks.

Exceptions should be carefully limited to use cases and accesses to a group resource that have a specific, narrow and well understood scope, and where only a few CoAP servers (or, ideally, only one) would possibly respond to a group request.

A relevant exceptional example is a CoAP client performing the discovery of hosts such as a group manager or a Resource Directory [I-D.ietf-core-resource-directory], by probing for them through a group request sent to the CoAP group. This early, unprotected step is relevant for a CoAP client that does not know the address of such hosts in advance, and that does not have yet configured a mutual security relation with them. In this kind of deployments, such a discovery procedure does not result in a considerable and harmful amplification, since only the few CoAP servers object of discovery are going to respond to the group request targeting that specific resource. In particular, those hosts can be the only CoAP servers in that specific CoAP group (hence listening for group requests sent to that group), and/or the only CoAP servers explicitly configured to respond to group requests targeting specific group resources.

With the exception of such particular use cases, group communications MUST be secured using Group OSCORE [I-D.ietf-core-oscore-groupcomm], see Section 5. As discussed in Section 6.2.3, this limits the feasibility and impact of amplification attacks.

6.4. Replay of Non-Confirmable Messages

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

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

This requires a server to have a synchronized, up to date view of the sequence number used by the client. If such synchronization is lost, e.g., due to a reboot, or suspected so, the server should use the challenge-response synchronization method described in Appendix E of [I-D.ietf-core-oscore-groupcomm] and based on the Echo Option for CoAP defined in [RFC9175], in order to (re-)synchronize with the client's sequence number.

6.5. Use of CoAP No-Response Option

When CoAP group communication is used in CoAP NoSec (No Security) mode (see Section 4), the CoAP No-Response Option [RFC7967] could be misused by a malicious client to evoke as much responses from servers to a group request as possible, by using the value '0' - Interested in all responses. This even overrides the default behavior of a CoAP server to suppress the response in case there is nothing of interest to respond with. Therefore, this option can be used to perform an amplification attack (see Section 6.3).

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

6.6. 6LoWPAN and MPL

In a 6LoWPAN network, a multicast IPv6 packet may be fragmented prior to transmission. A 6LoWPAN Router that forwards a fragmented packet may have a relatively high impact on the occupation of the wireless channel and may locally experience high memory load due to packet buffering. For example, the MPL [RFC7731] protocol requires an MPL Forwarder to store the packet for a longer duration, to allow multiple forwarding transmissions to neighboring Forwarders. If one or more of the fragments are not received correctly by an MPL Forwarder during its packet reassembly time window, the Forwarder discards all received fragments and at a future point in time it needs to receive again all the packet fragments (this time, possibly from another neighboring MPL Forwarder).

For these reasons, a fragmented IPv6 multicast packet is a possible attack vector in a Denial of Service (DoS) amplification attack. See Section 6.3 of this document and Section 11.3 of [RFC7252] for more details on amplification. To mitigate the risk, applications sending multicast IPv6 requests to 6LoWPAN hosted CoAP servers SHOULD limit the size of the request to avoid 6LoWPAN fragmentation of the request packet. A 6LoWPAN Router or (MPL) multicast forwarder SHOULD deprioritize forwarding for multi-fragment 6LoWPAN multicast packets. 6LoWPAN Border Routers are typical ingress points where multicast traffic enters into a 6LoWPAN network. Specific MPL Forwarders (whether located on a 6LBR or not) may also be configured as ingress points. Any such ingress point SHOULD implement multicast packet filtering to prevent unwanted multicast traffic from entering a 6LoWPAN network from the outside. For example, it could filter out all multicast packets for which there is no known multicast listener on the 6LoWPAN network. See Section 3.10 for protocols that allow multicast listeners to signal which groups they would like to listen to. As part of multicast packet filtering, the ingress point SHOULD implement a filtering criterium based on the size of the multicast packet. Ingress multicast packets above a defined size may then be dropped or deprioritized.

6.7. Wi-Fi

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

6.8. Monitoring

6.8.1. General Monitoring

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

6.8.2. Pervasive Monitoring

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

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

One mitigation is to restrict the scope of IP multicast to the minimal scope that fulfills the application need. See the congestion control recommendations in the last bullet of Section 3.6 to minimize the scope. Thus, for example, realm-local IP multicast scope is always preferred over site-local scope IP multicast if this fulfills the application needs.

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

7. IANA Considerations

This document has no actions for IANA.

8. References

8.1. Normative References

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

[I-D.ietf-cose-rfc8152bis-algs] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>.

[I-D.ietf-cose-rfc8152bis-struct] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

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

[RFC3376]
Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, DOI 10.17487/RFC3376, October 2002, <https://www.rfc-editor.org/info/rfc3376>.

[RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <https://www.rfc-editor.org/info/rfc3810>.

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

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

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

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

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

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

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

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

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

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

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

[RFC9175] Amsüss, C., Preuß Mattsson, J., and G. Selander, "Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing", RFC 9175, DOI 10.17487/RFC9175, February 2022, <https://www.rfc-editor.org/info/rfc9175>.

8.2. Informative References

[Californium] Eclipse Foundation, "Eclipse Californium", March 2019, <https://github.com/eclipse/californium/tree/2.0.x/californium-core/src/main/java/org/eclipse/californium/core>.

[Go-OCF] Open Connectivity Foundation (OCF), "Implementation of CoAP Server & Client in Go", March 2019, <https://github.com/go-ocf/go-coap>.

[I-D.ietf-ace-key-groupcomm-oscore] Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-13, 7 March 2022, <https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-13.txt>.

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-46, 8 November 2021, <https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>.

[I-D.ietf-ace-oscore-gm-admin] Tiloca, M., Höglund, R., Stok, P. V. D., and F. Palombini, "Admin Interface for the OSCORE

Group Manager", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-gm-admin-05, 7 March 2022, <https://www.ietf.org/archive/id/draft-ietf-ace-oscore-gm-admin-05.txt>.

[I-D.ietf-core-coap-pubsub] Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-09, 30 September 2019, <https://www.ietf.org/archive/id/draft-ietf-core-coap-pubsub-09.txt>.

[I-D.ietf-core-new-block] Boucadair, M. and J. Shallow, "Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission", Work in Progress, Internet-Draft, draft-ietf-core-new-block-14, 26 May 2021, <https://www.ietf.org/archive/id/draft-ietf-core-new-block-14.txt>.

[I-D.ietf-core-resource-directory] Amsüss, C., Shelby, Z., Koster, M., Bormann, C., and P. V. D. Stok, "CoRE Resource Directory", Work in Progress, Internet-Draft, draft-ietf-core-resource-directory-28, 7 March 2021, <https://www.ietf.org/archive/id/draft-ietf-core-resource-directory-28.txt>.

[I-D.mattsson-t2trg-amplification-attacks] Mattsson, J. P., Selander, G., and C. Amsüss, "Amplification Attacks Using the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-mattsson-t2trg-amplification-attacks-00, 11 February 2022, <https://www.ietf.org/archive/id/draft-mattsson-t2trg-amplification-attacks-00.txt>.

[I-D.tiloca-core-groupcomm-proxy] Tiloca, M. and E. Dijk, "Proxy Operations for CoAP Group Communication", Work in Progress, Internet-Draft, draft-tiloca-core-groupcomm-proxy-06, 7 March 2022, <https://www.ietf.org/archive/id/draft-tiloca-core-groupcomm-proxy-06.txt>.

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

[RFC6092] Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for

Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <https://www.rfc-editor.org/info/rfc6092>.

[RFC6550]
Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/info/rfc6550>.

[RFC6636] Asaeda, H., Liu, H., and Q. Wu, "Tuning the Behavior of the Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) for Routers in Mobile and Wireless Networks", RFC 6636, DOI 10.17487/RFC6636, May 2012, <https://www.rfc-editor.org/info/rfc6636>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

[RFC7346] Droms, R., "IPv6 Multicast Address Scopes", RFC 7346, DOI 10.17487/RFC7346, August 2014, <https://www.rfc-editor.org/info/rfc7346>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <https://www.rfc-editor.org/info/rfc7390>.

[RFC7731] Hui, J. and R. Kelsey, "Multicast Protocol for Low-Power and Lossy Networks (MPL)", RFC 7731, DOI 10.17487/RFC7731, February 2016, <https://www.rfc-editor.org/info/rfc7731>.

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

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

[RFC8710] Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for the Constrained Application Protocol

(CoAP)", RFC 8710, DOI 10.17487/RFC8710, February 2020, <https://www.rfc-editor.org/info/rfc8710>.

Appendix A. Use Cases

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

A.1. Discovery

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

A.1.1. Distributed Device Discovery

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

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

A.1.2. Distributed Service Discovery

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

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

A.1.3. Directory Discovery

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

A.2. Operational Phase

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

A.2.1. Actuator Group Control

Group communication can be beneficial to control actuators that need to act in synchrony, as a group, with strict timing (latency) requirements. Examples are office lighting, stage lighting, street lighting, or audio alert/Public Address systems. Sections 3.4 and

3.5 of [RFC7390] showed examples of lighting control of a group of 6LoWPAN-connected lights.

A.2.2. Device Group Status Request

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

A.2.3. Network-wide Query

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

A.2.4. Network-wide / Group Notification

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

A.3. Software Update

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

Appendix B. Examples of Message Exchanges

This section provides examples of different message exchanges when CoAP is used with group communication. The examples consider:

* A client with address ADDR_CLIENT and port number PORT_CLIENT.

* A CoAP group associated with the IP multicast address ADDR_GRP and port number PORT_GRP.

* An application group "gp1" associated with the CoAP group above.

* Three servers A, B and C, all of which are members of the CoAP group above and of the application group "gp1". Each server X (with X equal to A, B or C): listens to its own address ADDR_X and port number PORT_X; and listens to the address ADDR_GRP and port number PORT_GRP. For each server its PORT_X may be different from PORT_GRP or may be equal to it, in general.

In Figure 16, the client sends a Non-confirmable GET request to the CoAP group, targeting the resource "temperature" in the application group "gp1". All servers reply with a 2.05 Content response, although the response from server B is lost. As source port number of their response, servers A and B use the destination port number of the request, i.e, PORT_GRP. Instead, server C uses its own port number PORT_C.

```
Client              A  B  C
   |                |  |  |
   |  GET           |  |  |
   +-------+------->|  |  |  Source: ADDR_CLIENT:PORT_CLIENT
   |        \       |  |  |  Destination: ADDR_GRP:PORT_GRP
   |         `.------->|  |  Header: GET (T=NON, Code=0.01, MID=0x7d41)
   |           `.   |  |  |  Token: 0x86
   |             `------->|  Uri-Path: "gp"
   |                |  |  |  Uri-Path: "gp1"
   |                |  |  |  Uri-Path: "temperature"
   |                |  |  |
   |<---------------+  |  | Source: ADDR_A:PORT_GRP
   |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
   |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x60b1)
   |                |  |  | Token: 0x86
   |                |  |  | Payload: "22.3 C"
   |                |  |  |
   |   X---------------+  | Source: ADDR_B:PORT_GRP
   |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
   |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x01a0)
   |                |  |  | Token: 0x86
   |                |  |  | Payload: "20.9 C"
   |                |  |  |
   |                |  |  |
   |<---------------------+ Source: ADDR_C:PORT_C
   |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
   |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x952a)
   |                |  |  | Token: 0x86
   |                |  |  | Payload: "21.0 C"
   |                |  |  |
```

Figure 16: Example of Non-confirmable group request, followed by Non-confirmable Responses

In Figure 17, the client sends a Non-confirmable GET request to the CoAP group, targeting and requesting to observe the resource "temperature" in the application group "gp1". All servers reply with a 2.05 Content notification response. As source port number of their response, servers A and B use the destination port number of the request, i.e, PORT_GRP. Instead, server C uses its own port number PORT_C. Some time later, all servers send a 2.05 Content notification response, with payload the new representation of the "temperature" resource.

```
   Client              A  B  C
      |                |  |  |
      |  GET           |  |  |
      +-------+------->|  |  |  Source: ADDR_CLIENT:PORT_CLIENT
      |        \       |  |  |  Destination: ADDR_GRP:PORT_GRP
      |         `.------->|  |  Header: GET (T=NON, Code=0.01, MID=0x7d41)
      |           `.   |  |  |  Token: 0x86
      |             `------->|  Observe: 0 (register)
      |                |  |  |  Uri-Path: "gp"
      |                |  |  |  Uri-Path: "gp1"
      |                |  |  |  Uri-Path: "temperature"
      |                |  |  |
      |<---------------+  |  | Source: ADDR_A:PORT_GRP
      |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
      |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x60b1)
      |                |  |  | Token: 0x86
      |                |  |  | Observe: 3
      |                |  |  | Payload: "22.3 C"
      |                |  |  |
      |<------------------+  | Source: ADDR_B:PORT_GRP
      |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
      |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x01a0)
      |                |  |  | Token: 0x86
      |                |  |  | Observe: 13
      |                |  |  | Payload: "20.9 C"
      |                |  |  |
      |<---------------------+ Source: ADDR_C:PORT_C
      |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
      |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x952a)
      |                |  |  | Token: 0x86
      |                |  |  | Observe: 23
      |                |  |  | Payload: "21.0 C"
      |                |  |  |

      // The temperature changes ...

      |                |  |  |
      |<---------------+  |  | Source: ADDR_A:PORT_GRP
      |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
      |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x60b2)
      |                |  |  | Token: 0x86
      |                |  |  | Observe: 7
      |                |  |  | Payload: "32.3 C"
      |                |  |  |
      |<------------------+  | Source: ADDR_B:PORT_GRP
      |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
      |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x01a1)
      |                |  |  | Token: 0x86
      |                |  |  | Observe: 18
```

```
|                 |  |  | Payload: "30.9 C"
|                 |  |  |
|<---------------------+ Source: ADDR_C:PORT_C
|      2.05       |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
|                 |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x952b)
|                 |  |  | Token: 0x86
|                 |  |  | Observe: 29
|                 |  |  | Payload: "31.0 C"
|                 |  |  |
```

Figure 17: Example of Non-confirmable Observe group request, followed by Non-confirmable Responses as Observe notifications

In Figure 18, the client sends a Non-confirmable GET request to the CoAP group, targeting the resource "log" in the application group "gp1", and requesting a blockwise transfer. All servers reply with a 2.05 Content response including the first block. As source port number of its response, each server uses its own port number. After obtaining the first block, the client requests the following blocks separately from each server, by means of unicast exchanges.

```
Client              A  B  C
   |                |  |  |
   |  GET           |  |  |
   +-------+------->|  |  |  Source: ADDR_CLIENT:PORT_CLIENT
   |        \       |  |  |  Destination: ADDR_GRP:PORT_GRP
   |         `.------->|  |  Header: GET (T=NON, Code=0.01, MID=0x7d41)
   |           `.   |  |  |  Token: 0x86
   |             `------->|  Uri-Path: "gp"
   |                |  |  |  Uri-Path: "gp1"
   |                |  |  |  Uri-Path: "log"
   |                |  |  |  Block2: 0/0/64
   |                |  |  |
   |<---------------+  |  | Source: ADDR_A:PORT_A
   |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
   |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x60b1)
   |                |  |  | Token: 0x86
   |                |  |  | Block2: 0/1/64
   |                |  |  | Payload: 0x0a00 ...
   |                |  |  |
   |<------------------+  | Source: ADDR_B:PORT_B
   |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
   |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x01a0)
   |                |  |  | Token: 0x86
   |                |  |  | Block2: 0/1/64
   |                |  |  | Payload: 0x0b00 ...
   |                |  |  |
   |<---------------------+ Source: ADDR_C:PORT_C
   |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
   |                |  |  | Header: 2.05 (T=NON, Code=4.04, MID=0x952a)
   |                |  |  | Token: 0x86
   |                |  |  | Block2: 0/1/64
   |                |  |  | Payload: 0x0c00 ...
   |                |  |  |
   |      GET       |  |  |
   +--------------->|  |  |  Source: ADDR_CLIENT:PORT_CLIENT
   |                |  |  |  Destination: ADDR_A:PORT_A
   |                |  |  |  Header: GET (T=CON, Code=0.01, MID=0x7d42)
   |                |  |  |  Token: 0xa6
   |                |  |  |  Uri-Path: "gp"
   |                |  |  |  Uri-Path: "gp1"
   |                |  |  |  Uri-Path: "log"
   |                |  |  |  Block2: 1/0/64
   |                |  |  |
   |<---------------+  |  | Source: ADDR_A:PORT_A
   |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
   |                |  |  | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d42)
   |                |  |  | Token: 0xa6
   |                |  |  | Block2: 1/1/64
   |                |  |  | Payload: 0x0a01 ...
|                |  |  |
|      GET       |  |  |
+--------------->|  |  |  Source: ADDR_CLIENT:PORT_CLIENT
|                |  |  |  Destination: ADDR_A:PORT_A
|                |  |  |  Header: GET (T=CON, Code=0.01, MID=0x7d43)
|                |  |  |  Token: 0xa7
|                |  |  |  Uri-Path: "gp"
|                |  |  |  Uri-Path: "gp1"
|                |  |  |  Uri-Path: "log"
|                |  |  |  Block2: 2/0/64
|                |  |  |
|<---------------+  |  | Source: ADDR_A:PORT_A
|      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
|                |  |  | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d43)
|                |  |  | Token: 0xa7
|                |  |  | Block2: 2/0/64
|                |  |  | Payload: 0x0a02 ...
|                |  |  |
|      GET       |  |  |
+------------------>|  |  Source: ADDR_CLIENT:PORT_CLIENT
|                |  |  |  Destination: ADDR_B:PORT_B
|                |  |  |  Header: GET (T=CON, Code=0.01, MID=0x7d44)
|                |  |  |  Token: 0xb6
|                |  |  |  Uri-Path: "gp"
|                |  |  |  Uri-Path: "gp1"
|                |  |  |  Uri-Path: "log"
|                |  |  |  Block2: 1/0/64
|                |  |  |
|<------------------+  | Source: ADDR_B:PORT_B
|      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
|                |  |  | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d44)
|                |  |  | Token: 0xb6
|                |  |  | Block2: 1/1/64
|                |  |  | Payload: 0x0b01 ...
|                |  |  |
|      GET       |  |  |
+------------------>|  |  Source: ADDR_CLIENT:PORT_CLIENT
|                |  |  |  Destination: ADDR_C:PORT_B
|                |  |  |  Header: GET (T=CON, Code=0.01, MID=0x7d45)
|                |  |  |  Token: 0xb7
|                |  |  |  Uri-Path: "gp"
|                |  |  |  Uri-Path: "gp1"
|                |  |  |  Uri-Path: "log"
|                |  |  |  Block2: 2/0/64
|                |  |  |
|<------------------+  | Source: ADDR_B:PORT_B
|      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
|                |  |  | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d45)
|                |  |  | Token: 0xb7
|                 |  |  | Block2: 2/0/64
|                 |  |  | Payload: 0x0b02 ...
|                 |  |  |
|      GET        |  |  |
+---------------------->|  Source: ADDR_CLIENT:PORT_CLIENT
|                 |  |  |  Destination: ADDR_C:PORT_C
|                 |  |  |  Header: GET (T=CON, Code=0.01, MID=0x7d46)
|                 |  |  |  Token: 0xc6
|                 |  |  |  Uri-Path: "gp"
|                 |  |  |  Uri-Path: "gp1"
|                 |  |  |  Uri-Path: "log"
|                 |  |  |  Block2: 1/0/64
|                 |  |  |
|<----------------------+ Source: ADDR_C:PORT_C
|      2.05       |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
|                 |  |  | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d46)
|                 |  |  | Token: 0xc6
|                 |  |  | Block2: 1/1/64
|                 |  |  | Payload: 0x0c01 ...
|                 |  |  |
|      GET        |  |  |
+---------------------->|  Source: ADDR_CLIENT:PORT_CLIENT
|                 |  |  |  Destination: ADDR_C:PORT_C
|                 |  |  |  Header: GET (T=CON, Code=0.01, MID=0x7d47)
|                 |  |  |  Token: 0xc7
|                 |  |  |  Uri-Path: "gp"
|                 |  |  |  Uri-Path: "gp1"
|                 |  |  |  Uri-Path: "log"
|                 |  |  |  Block2: 2/0/64
|                 |  |  |
|<----------------------+ Source: ADDR_C:PORT_C
|      2.05       |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
|                 |  |  | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d47)
|                 |  |  | Token: 0xc7
|                 |  |  | Block2: 2/0/64
|                 |  |  | Payload: 0x0c02 ...
|                 |  |  |
```

Figure 18: Example of Non-confirmable group request starting a blockwise transfer, followed by Non-confirmable Responses with the first block. The transfer continues over confirmable unicast exchanges

Appendix C. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

C.1. Version -05 to -06

* Harmonized use of "group URI".

* Clarifications about different group types.

* Revised methods to perform group naming.

* Revised methods to discover application groups and CoAP groups.

* Explicit difference between "authentication credential" and "public key".

* Added examples of application group naming.

* Added examples of application/CoAP group discovery.

* Added examples of message exchanges.

* Reference to draft-mattsson-core-coap-attacks replaced with reference to draft-mattsson-t2trg-amplification-attacks.

* Editorial improvements.

C.2. Version -04 to -05

* Clarified changes to other documents.

* Clarified relation between different group types.

* Clarified discovery of application groups.

* Discussed methods to express application group names in requests.

* Revised and extended text on the NoSec mode and amplification attacks.

* Rephrased backward/forward security as properties.

* Removed appendix on Multi-ETag Option for response revalidation.

* Editorial improvements.

C.3. Version -03 to -04

* Multi-ETag Option for response revalidation moved to appendix.
* ETag Option usage added.
* Q-Block Options added in the block-wise transfer section.
* Caching at proxies moved to draft-tiloca-core-groupcomm-proxy.
* Client-Proxy response revalidation with the Group-ETag Option moved to draft-tiloca-core-groupcomm-proxy.
* Security considerations on amplification attacks.
* Generalized transport protocols to include others than UDP/IP multicast; and security protocols other than Group OSCORE.
* Overview of security cases with proxies.
* Editorial improvements.

C.4. Version -02 to -03

* Multiple responses from same server handled at the application.
* Clarifications about issues with forward-proxies.
* Operations for reverse-proxies.
* Caching of responses at proxies.
* Client-Server response revalidation, with Multi-ETag Option.
* Client-Proxy response revalidation, with the Group-ETag Option.

C.5. Version -01 to -02

* Clarified relation between security groups and application groups.
* Considered also FETCH for requests over IP multicast.
* More details on Observe re-registration.
* More details on Proxy intermediaries.
* More details on servers changing port number in the response.
* Usage of the Uri-Host Option to indicate an application group.

* Response suppression based on classes of error codes.

C.6. Version -00 to -01

* Clarifications on group memberships for the different group types.

* Simplified description of Token reusage, compared to the unicast case.

* More details on the rationale for response suppression.

* Clarifications of creation and management of security groups.

* Clients more knowledgeable than proxies about stopping receiving responses.

* Cancellation of group observations.

* Clarification on multicast scope to use.

* Both the group mode and pairwise mode of Group OSCORE are considered.

* Updated security considerations.

* Editorial improvements.

Acknowledgments

The authors sincerely thank Christian Amsuess, Carsten Bormann, Thomas Fossati, Rikard Hoeglund, Jaime Jimenez, John Mattsson and Jim Schaad for their comments and feedback.

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

Authors' Addresses

Esko Dijk
IoTconsultancy.nl
_________________\
Utrecht
Netherlands

Email: esko.dijk@iotconsultancy.nl

Chonggang Wang
InterDigital

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

Email: Chonggang.Wang@InterDigital.com

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

Email: marco.tiloca@ri.se