CoRE Working Group
Internet-Draft
Obsoletes: 7390 (if approved)
Updates: 7252, 7641 (if approved)
Intended status: Standards Track
Expires: 4 January 2026

E. Dijk
IoTconsultancy.nl
M. Tiloca
RISE AB

3 July 2025

Group Communication for the Constrained Application Protocol (CoAP) draft-ietf-core-groupcomm-bis-14

Abstract

The Constrained Application Protocol (CoAP) is a web transfer protocol for constrained devices and constrained networks. In a number of use cases, constrained devices often naturally operate in groups (e.g., in a building automation scenario, all lights in a given room may need to be switched on/off as a group). This document specifies the use of 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 and obsoletes RFC 7390, while it updates RFC 7252 and RFC 7641.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-core-groupcomm-bis/.

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

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

Status of This Memo

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

This Internet-Draft will expire on 4 January 2026.

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol based on Representational State Transfer (REST) that is used in resource-constrained devices and in resource-constrained networks where packet sizes should be small. This area of use is summarized as Constrained RESTful Environments (CoRE). CoAP has many similarities to HTTP [RFC9110][RFC9112] but also some key differences.

In a number of use cases, constrained devices can be large in number as well as often related to each other in function or by location. For example, in a building automation scenario, all the light switches in a building may belong to one group, and all the thermostats may belong to another group. Groups may be preconfigured before deployment or dynamically formed during operation. If information needs to be sent to or received from a group of devices, group communication mechanisms can improve efficiency and latency of communication and reduce bandwidth requirements for a given application. While CoAP supports group communication via multicast requests (see Section 8 of [RFC7252]), HTTP does not support any equivalent functionality.

This document specifies the use of CoAP for group communication, together with UDP/IP multicast as the default transport for CoAP group communication messages.

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 that support group communication include "Eclipse Californium" [Californium], "Go-CoAP" [Go-CoAP] as well as "libcoap" [libcoap].

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) [RFC9052][RFC9053].

This document 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 not considered. 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 not considered. 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 can contain a 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 provides separate definitions for CoAP groups, application groups, and security groups, together with high-level guidelines on their configuration (see Section 2).

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

* It updates all sections on transport protocols and interworking with other protocols based on new IETF work done for these protocols (see Section 3.9 and Section 3.10).

* It strongly discourages unsecured group communication for CoAP based on the CoAP NoSec (No Security) mode (see Section 4 and Section 6.1), and highlights the risk of amplification attacks (see Section 6.3).

* 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).

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).

* It defines the measures against congestion risk specified in [RFC7252] to be applicable also to alternative transports other than IP multicast, and defines additional guidelines to reduce congestion risks (see Section 3.6).

* It explicitly allows the use of the IPv6 multicast address scopes realm-local (3), admin-local (4), and global (E). In particular, it recommends that an IPv6 CoAP server supports at least link-local (2), admin-local (4), and site-local (5) scopes with the "All CoAP Nodes" multicast CoAP group (see Section 3.9.1). Also, it recommends that the realm-local (3) scope is supported by an IPv6 CoAP server on a 6LoWPAN node (see Section 3.9.1).

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. Types of Groups and Their 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. Types of Groups

Three types of groups and their mutual relationships 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. Membership(s) of an endpoint or node to a CoAP group 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 that CoAP group: it is a member only if it also has a CoAP endpoint listening on the associated IP multicast address and UDP port associated with the CoAP group.

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 relationship between a CoAP group and application group(s). Such relationships 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 relationship between security groups and CoAP groups, but often it is one-to-one. Also, there can be a many-to-many relationship between security groups and application groups, but often it is one-to-one. Such relationships are discussed in more detail in Section 2.1.4.

Further details on identifying a security group are provided in Section 2.2.1.3.

If the NoSec mode is used (see Section 4), group communication does not rely on security at the transport layer nor at the CoAP layer, hence the communicating endpoints do not refer to a security group.

When a security group uses the security protocol Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect group communication (see Section 5 of this document), source authentication is achieved for messages exchanged within the group (see Section 5.1 and Section 6.2.2 of this document). That is, even though the endpoints in the security group do share group security material, a recipient CoAP endpoint is able to verify that a message protected with Group OSCORE has actually been originated and sent by a specific and identified CoAP endpoint as a member of the security group.

2.1.4. Relationships Between Group Types

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

(application group, CoAP group, security group)

A special note is appropriate about the possible relationship 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 server port number used, 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 single 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 the same application group.

In fact, 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. Therefore, access control to use resources in the application group should be separately enforced by leveraging the resource properties or through dedicated access control credentials assessed by separate means.

Figure 1 summarizes the relationships between the different types of groups described above in Unified Modeling Language (UML) class diagram notation. The class attributes in square brackets are optionally defined.

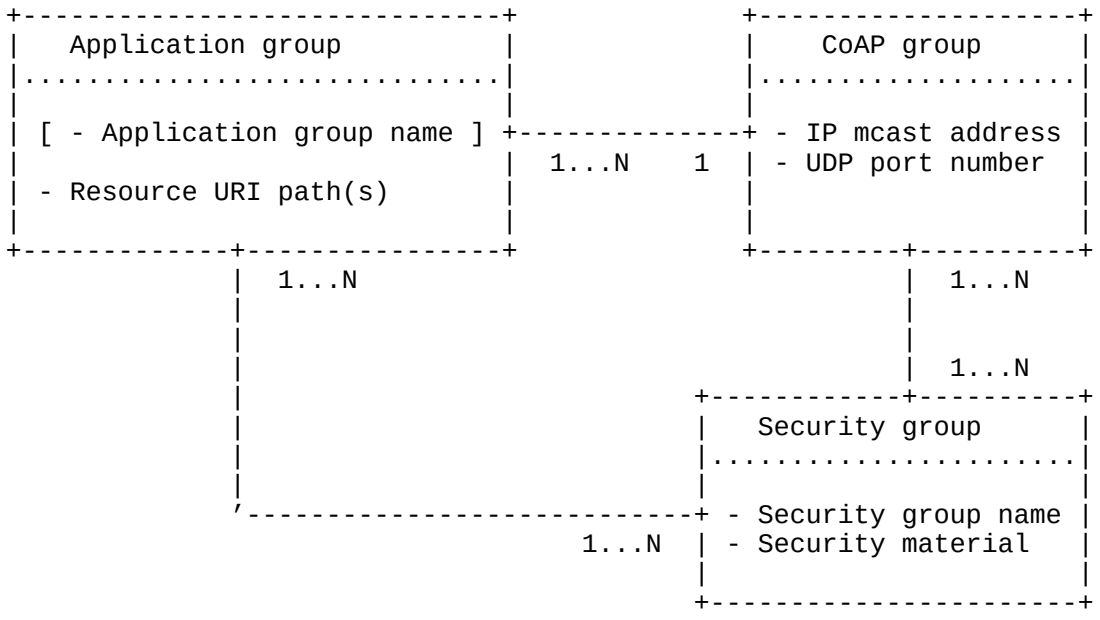


Figure 1: Relationships Among Different Group Types

Figure 2 provides a deployment example of the relationships between the different types of groups. It shows six CoAP servers (Srv1-Srv6) and their respective resources hosted (/resX). Although in real-life deployments using group communication the number of servers and resources would usually be higher, only limited numbers are shown here for ease of representation.

There are three application groups (1, 2, 3) and two security groups (1, 2). The Security Group 1 may, for example, include all lighting devices on a floor of an office building, while Security Group 2 includes all Heating, Ventilation, and Air Conditioning (HVAC) devices of that floor. Security Group 1 is used by both Application Group 1 and 2. The Application Group 1 for example may consist of all lights in a hallway, while Application Group 2 includes all lights in a storage room. Three clients (Cli1, Cli2, Cli3) are configured with security material for Security Group 1. These clients may be motion sensors and a control panel (Cli3), that send multicast messages to /resA to inform the lights of any motion or user activity detected. The control panel Cli3 additionally sends a multicast message to /resB to communicate the latest light preset selected by a user. The latter action only influences the lighting in the storage room (Application Group 2). Two clients (Cli2, Cli4) are configured with security material for Security Group 2. These clients may be temperature/humidity sensors that report measurements periodically to all HVAC devices (Srv5, Srv6) in the Application Group 3, using for example /resC to report temperature and /resD to report humidity.

All the shown application groups may use the same CoAP group (not shown in the figure), for example the CoAP group with site-local, site-specific multicast IP address ff15::3456 and default UDP port number 5683 on which all the shown resources are hosted for each server. Other floors of the same building may replicate the shown structure, but using different security groups and different CoAP groups.

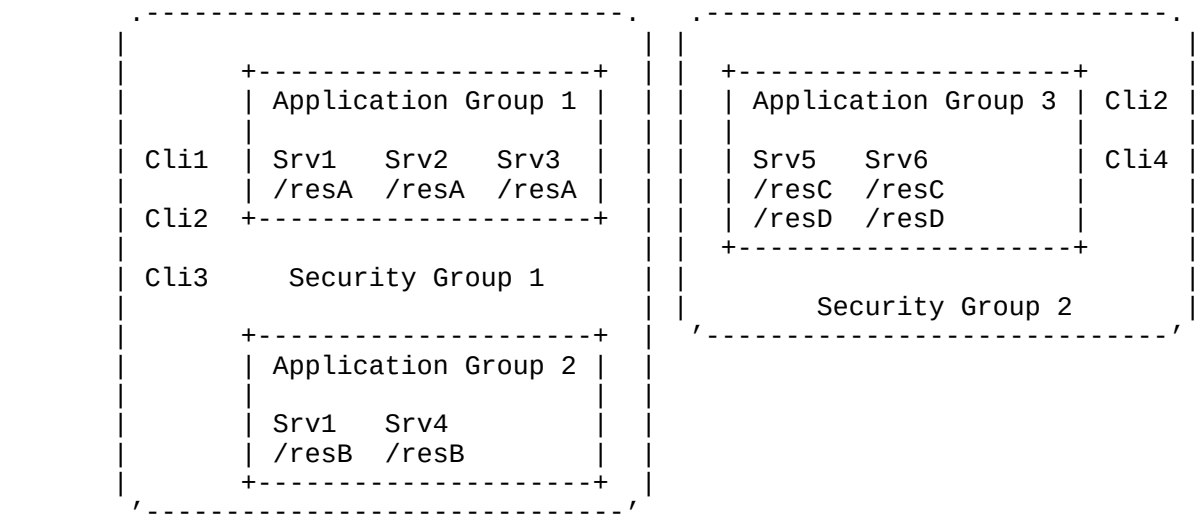


Figure 2: Deployment Example of Different Group Types

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 groups are named as specified below, separately for CoAP groups, application groups, and security groups.

2.2.1.1. CoAP Groups

A CoAP group is always defined by the two properties of IP multicast address and UDP port number (see Section 2.1.1).

However, a CoAP group is for practical purposes identified and named by the authority component in the group URI. This component includes the host subcomponent and an optional UDP port number. The host subcomponent directly defines the IP multicast address of the CoAP group, in case the host consists of an IP literal. The host subcomponent indirectly defines the IP multicast address of the CoAP group, in case the host consists of a hostname: resolving the hostname to an IP address in this case produces the IP multicast address.

It follows that the same CoAP group might have multiple names, which can be simultaneously and interchangeably used. For example, if the two hostnames group1.example and group1.alias.example both resolve to the IP multicast address [ff15::1234], then the following authority components are all names for the same CoAP group.

* group1.example:7700

* group1.alias.example:7700

* [ff15::1234]:7700

Also note that, when using the "coap" scheme, the two authority components <HOST> and <HOST>:5683 both identify the same CoAP group, whose members listen to the CoAP default port number 5683. Therefore, building on the above, the following authority components are all names for the same CoAP group.

* group1.example

* group1.alias.example

* [ff15::1234]

* group1.example:5683

* group1.alias.example:5683

* [ff15::1234]: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.

Some examples of hierarchical CoAP group FQDN naming (and scoping) for a building control application are shown below.

URI authority	Targeted group of nodes
all.bldg6.example	"all nodes in building 6"
all.west.bldg6.example	"all nodes in west wing, building 6"
all.floor1.west.bldg6.example	"all nodes in floor 1, west wing, building 6"
all.bu036.floor1.west.bldg6.example	"all nodes in office bu036, floor 1, west wing, building 6"

Table 1: Examples of Hierarchical Group FQDN Naming

Similarly, if supported, reverse mapping (from IP multicast address to Group FQDN) is possible using the reverse DNS resolution technique [RFC1035]. Reverse mapping is important, for example, in troubleshooting to translate IP multicast addresses back to human-readable hostnames to show in a diagnostics user interface.

2.2.1.2. Application Groups

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

This section summarizes possible methods for encoding an application group name in a CoAP group request. Full examples for these methods are provided in Appendix B.

An application group name can be explicitly encoded in a group URI. Specifically, 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 APPNAME should be kept short.

A best practice 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 containing 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/APPNAME/res1 and /base/gp/APPNAME/res1/res2 conform to this practice. The path segment used as delimiter ('gp' in the examples) should be kept short in constrained networks.

Full examples are provided in Appendix B.1.

* URI query component -- This method can use the following formats. In either case, when using this method in constrained networks, an application group name APPNAME 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 identifier. The query component ?APPNAME conforms to this format.

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

  Full examples are provided in Appendix B.2.

* URI authority component -- If this method is used, the application group is identified by the authority component of the group URI or a subset thereof.

  Because the CoAP group is also defined by the same authority component (see Section 2.2.1.1), even when using this method, a given application group is always associated with exactly one CoAP group. (See Section 2.1.4 for background on group relationships.)

  Note that the host subcomponent within the authority component of the Group URI can be a group hostname, or an IP address literal. For constrained networks, using an IP address literal matching the request's destination IP address has the benefit of reducing the size of the CoAP message. This is because the Uri-Host Option is elided from the CoAP request in this case, since its default value applies (see Sections 5.10.1 and 6.4 of [RFC7252]).

  Full examples are provided in Appendix B.3.

Due to the CoAP client's encoding of the request URI into CoAP options (per Section 6.4 of [RFC7252]) and the possibility of the CoAP server to compose the URI again based on the options received (see Section 6.5 of [RFC7252]), the application group name information can be transported to the server and used to select the intended application group.

Any other method to transport the application group name within a CoAP request, but not using the group URI, would require a new CoAP option to be defined. Such an approach is out of the scope of this document.

Finally, it is also possible to not encode the application group name in the CoAP request, 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 CoAP group, and/or 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. Appendix A of [RFC9176] provides an example of a named 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, while the CoAP group ff35:30:2001:db8:f1:0:8000:1 is specified in the authority component of the URI encoded in the "base" attribute. In subsequent group requests by a client to the "lights" group, the name of the group is not present in the request message. Rather, the URI authority component that selects the CoAP group ff35:30:2001:db8:f1:0:8000:1 will implicitly also select the "lights" application group.

2.2.1.3. Security Groups

A security group can be named in many ways through different types of identifiers, such as name string, (integer) number, URI, or other types of strings. Such a group name is generally not related to other kinds of group identifiers that may be specific to the security solution used.

The name of a security group is not expected to be used in messages exchanged among its members, unless the application requires otherwise. At the same time, it is useful to identify the security group when performing a number of side tasks related to secure group communication, such as the following ones.

* An administrator may have to request an authorization to configure security groups at an available Group Manager (see Section 5). During the authorization process, as well as during the interaction between the administrator and the Group Manager, the group name identifies the specific security group that the administrator wishes to configure and is authorized to.

* A CoAP endpoint may have to request an authorization to join a specific security group through the respective Group Manager, and thus obtain the required group security material (see Section 5). During the authorization process, as well as during the interaction between the CoAP endpoint and the Group Manager, the group name identifies the specific security group that the CoAP endpoint wishes to join and is authorized to.

* A CoAP endpoint may first need to discover the specific security groups to join through the respective Group Manager (see Section 2.2.3.1). Results from the discovery process include the name of the security groups to join, together with additional information such as a pointer to the respective Group Manager.

It is discouraged to use "NoSec" and any of its lowercase/uppercase combinations as name of a security group. Indications that endpoints can use the NoSec mode MUST NOT rely on setting up and advertising a pseudo security group with name "NoSec" or any of its lowercase/uppercase combinations.

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 CoAP 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.

The devices sending CoAP requests to the CoAP group in the role of CoAP client also 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 configuring memberships of CoAP groups for unsecured group

communication, based on JSON-formatted configuration resources. The experiment is concluded as showing that the protocol has not been considered for deployment and use.

For IPv6 CoAP groups, common multicast address ranges from which group addresses can be taken are ff1x::/16 and ff3x::/16.

To create an application group, a configuring entity may configure a resource 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 security 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 configuration specified for the security group.

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 mode (see Section 4), there is no security material to be provided, hence there is no security group for CoAP endpoints to participate in.

The configuration of groups and membership may be performed at different moments in the life-cycle of a device. For example, it can occur 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 [RFC9176].

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 names of 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 Server Members of an Application or CoAP Group

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 discussed 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 (see Section 12.8 of [RFC7252]), 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 discusses a number of methods to discover application groups and CoAP groups, building on the following assumptions.

* Application group names are encoded in the path component of Group URIs (see Section 2.2.1.2). In examples in this document, the path segment "gp" is used as designated delimiter.

* The type of an application group is encoded in the value of the CoRE Link Format attribute "rt" of a group resource.

  In examples presented in the following, this document considers such values for the attribute "rt" to have the semantics "g.<GROUPTYPE>", where GROUPTYPE denotes the type of the application group in question.

  Resource Type values can be registered in the "Resource Type (rt=) Link Target Attribute Values" IANA registry [Resource.Type.Link.Target.Attribute.Values] within the "Constrained RESTful Environments (CoRE) Parameters" registry group. While relying on registered Resource Type values is not strictly necessary, it is encouraged in order to ensure a more effective discovery of application groups and CoAP groups.

Note that the specific way of using the methods discussed below is 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 application groups and CoAP 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.

Full examples for the different methods are provided in Appendix C.

* 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 example is provided in Appendix C.1.

* 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.

  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.

  A full example is provided in Appendix C.2.

* 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.

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.

A full example is provided in Appendix C.3.

* A CoAP client can discover the CoAP servers that are members of any application group configured in the 6LoWPAN 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, 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 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.

  Through the corresponding responses, the query result is a list of group resources hosted by any server in the 6LoWPAN network. Each group resource denotes one application group membership of a server. For each application group, the associated CoAP group is obtained as the URI authority component of the corresponding returned link.

  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.

  Full examples are provided in Appendix C.4.

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., when the NoSec mode is used, 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 as well. 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 (NON), 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 MAY send one or more responses to a CoAP group request; these are always unicast messages. The unicast responses received by the CoAP client may carry a mixture of success (e.g., 2.05 (Content)) and failure (e.g., 4.04 (Not Found)) response 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 that a server supporting this option only takes it into account when processing requests that target resources for which influencing the default suppression has been predetermined to be appropriate, as well as useful, in the application context.

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 (see Section 3 of [RFC7252]) 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

Group requests sent over IP multicast generally have much higher loss rates than messages sent over unicast, particularly in constrained networks. Therefore, it is more urgent to have a strategy in place for handling the loss of group requests than the loss of unicast responses. To this end, CoAP clients can rely on the following two approaches.

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) members of the CoAP group have been reached with the request. This

is useful in case a number of members of the CoAP group 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 a different Message ID (and the same or a different Token value), 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 members of the CoAP group, 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, a server MAY respond from a UDP port number that differs from the destination UDP port number of the request.

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 bounded 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, if the client sends a group request specifying the Observe option set to 0 (see Section 3.1 of [RFC7641]) and this server adds the client to the list of observers for the targeted resource, then the server is set up to send multiple responses as Observe notifications to notify the client of changes to the resource state (see Section 4.2 of [RFC7641]). The use of Observe with group communication is discussed in more details in Section 3.7. As another example, a 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, it is up to the specific client implementation to decide at which layer deduplication of responses is performed, or whether it is necessary in an application at all. If the processing of a response is successful, the client delivers the response to the application as usual.

The application itself can be in a good 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, e.g., when this can trigger a change of state within the application.

As part of a message exchange between the client and any of the servers in the CoAP group, the multiple responses considered above are examples of the more general concept elaborated in Section 2 of [I-D.bormann-core-responses].

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]. The set of request options used as "Cache-Key" is also as defined in Section 5.6 of [RFC7252].

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. With such a cache, the client uses both cached-still-fresh and new responses as the result of further group requests.

* 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 defined in [I-D.ietf-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 members may have joined the CoAP group since the last group request to the same CoAP group or 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 servers that are members of the CoAP group; and, for each group member, the client's cache currently stores a fresh response.

How the client in the case above determines the CoAP servers that are currently members of the CoAP group is out of scope for this document. It may be, for example, via a Group Manager, or by monitoring group joining protocol exchanges.

For caching at proxies, the freshness model defined in [I-D.ietf-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.ietf-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 CoAP group have responded to a past 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 wants a fresh response from server 2, there is no way to achieve this using a single group request. This wish could occur if the client has a cached representation for server 1, but has no cached representation for server 2: for example, because the client needed to remove older items from its cache to make space for newer resource representations.

There are various strategies to avoid problems caused by identical ETag values: one strategy is for a client to repeat a request if a particular server returned 2.03 (Valid) with an ETag value that is not in the client's cache (for that server). The repeated request excludes the "duplicate" ETag, and it may be a group request or a unicast request to the particular server. Another strategy is to mark a cached ETag value as "duplicated - not to be used for revalidation" as soon as another server responds with the same ETag value. Finally, the recommended strategy is for the servers to generate unique ETags as specified below.

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 practically prevents a collision of the ETag values from different servers in the same CoAP group and application group, which in turn allows the client to effectively validate a particular response of an origin server. This can be accomplished, for example, by embedding a compact ID (or hash) of the server within the ETag value, where the ID is unique (or unique with a high probability) in the scope of the CoAP/application groups.

Note: a CoAP server implementation that is unaware of the updates to [RFC7252] made by this document will expect group requests to never contain an ETag Option (see Section 8.2.1 of [RFC7252]). Such a server treats 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 processes 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 members of a CoAP group. However, there are valid use cases where a group request is known to be successful only for a subset of the CoAP group. For instance, the subset may include only members of a specific application group, while the members of the CoAP group for which the request is unsuccessful (for example because they are outside the target application group) either suppress a response as per the default behavior from Section 3.1.2, or reply with an error response, e.g., when the default behavior is overridden by a No-Response Option [RFC7967] included in the group request.

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 and port number. The group's port number is usually the CoAP default UDP port number 5683, or alternatively another non-default UDP port number if configured. Regardless of the method that is used for selecting the group's port number, the same port number is used as the destination port number for requests 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 Internet (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 dedicated 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" CoAP 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. Furthermore, Section 3.5.3 discusses the case where a client sends a group request to multiple proxies at once. 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. The group URI where the request has to be forwarded to is specified in the request, 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.

Issues and limitations of this approach are compiled in Appendix E. A forward-proxying method using this approach and addressing such issues and limitations is defined in [I-D.ietf-core-groupcomm-proxy].

An alternative approach 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. Issues and limitations of this alternative approach are also compiled in Appendix E.

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 an HTTP response -- back to the HTTP client. Resulting issues and limitations are also compiled in Appendix E.

A forward-proxying method for HTTP-to-CoAP proxies addressing such issues and limitations is defined in [I-D.ietf-core-groupcomm-proxy].

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 servers in a CoAP group, over a one-to-many transport such as IP/UDP multicast.

One typical implementation is to allocate specific resources on the reverse-proxy to application groups. A client can then select the application group, and group resource to access, using the URI path in its group request. For example, a request to /proxy/APPNAME/res1 could give access to resource /res1 in the application group APPNAME. In this example, the proxy automatically selects the associated CoAP group.

In general, using the URI path to select application group and/or CoAP group is an efficient way to access a reverse proxy. Other methods are possible, such as using the URI authority component: this requires configuration of more elements on the reverse proxy, like multiple virtual servers and/or multiple IP addresses and/or multiple port numbers.

The reverse-proxy practically stands in for a CoAP group, thus preventing the client from reaching the group as a whole with a single group request directly addressed to that group (e.g., via multicast). In addition to that, the reverse-proxy may also stand in for each of the individual servers in the CoAP group (e.g., if acting as firewall), thus also preventing the client from individually reaching any server in the group with a unicast request directly addressed to that server.

For a reverse-proxy that sends a group request to servers in a CoAP group, the considerations as defined in Section 5.7.3 of [RFC7252] hold. Resulting issues and limitations are compiled in Appendix F. A reverse-proxying method addressing such issues and limitations is defined in [I-D.ietf-core-groupcomm-proxy].

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 servers in the CoAP group, 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.

For the operation of HTTP-to-CoAP reverse proxies, see the last two paragraphs of Section 3.5.1, which apply also to the case of reverse-proxies.

3.5.3. Single Group Request to Multiple Proxies

A client might send a group request to multiple proxies at once (e.g., over IP multicast), so that each of those proxies forwards it to the servers in the CoAP group. Assuming that no message loss occurs and that N proxies receive and forward the group request, this has the following implications.

* Each server receives N copies of the group request, i.e., one copy from each proxy.

* If the NoSec mode is used (see Section 4), each server treats each received copy of the group request as a different request from a different client. As a result:

  - Each server can reply to each of the N received requests with multiple responses over time (see Section 3.1.6). All the responses to the same received request are sent to the same proxy that has forwarded that request, which in turn relays those responses to the client.

  - From each proxy, the client receives all the responses to the group request that each server has sent to that proxy. Even in case the client is able to distinguish the different servers originating the responses (e.g., by using the approach defined in [I-D.ietf-core-groupcomm-proxy]), the client would receive the same response content originated by each server N times, as relayed by the N proxies.

* If secure group communication with Group OSCORE is used (see Section 5), each server is able to determine that each received copy of the group request is in fact originated by the same client. In particular, each server is able to determine that all such received requests are copies of exactly the same group request.

  As a result, each server accepts only the first copy of the group request received from one of the proxies, while discarding as replay any later copies received from any other proxy.

After that, the server can reply to the accepted request with multiple responses over time (see Section 3.1.6). All those responses are sent to the same proxy that forwarded the only accepted request, and that in turn relays those responses to the client.

As a consequence, for each server, the client receives responses originated by that server only from one proxy. That is, the client receives a certain response content only once, like in the case with only one proxy.

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, 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]).

* The same rationale for enforcing congestion control based on the transmission parameter PROBING_RATE defined in Section 4.7 of [RFC7252] holds for CoAP group communication. In particular, group requests are the Non-confirmable requests in question, and an average data rate PROBING_RATE is not to be exceeded by a client that does not receive a response from any server in the targeted CoAP group.

* 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.

Independently of the transport used, 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 3.9.2) 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, which 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], the same rules apply when using 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. If a group member does not have this particular resource, or it does not allow the GET or FETCH method on that resource, then the group member will either suppress a response as per the default behavior from Section 3.1.2, or reply with an error response -- 4.04 (Not Found) or 4.05 (Method Not Allowed), respectively -- e.g., when the default behavior is overridden by a No-Response Option [RFC7967] included in the group request.

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 members of the CoAP group did not respond to the initial request. The client MAY additionally use the same Message ID in the repeated request, to avoid that members of the CoAP group 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 members of the CoAP group 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 member of the CoAP group did not respond to the initial group request, although it was expected to. In this case, the client MUST use a Message ID that differs from that of the initial group request message.

Since the first Observe notification from a server can be lost, a client SHOULD be ready to begin receiving the Observe notifications from a server long after the Non-confirmable group request with the Observe Option was sent.

At the same time, the loss of initial responses with the Observe Option from a server is less problematic than in the case where the group request is a regular request, i.e., when the request does not include the Observe Option. That is, as per Section 4.5 of [RFC7641], servers that have registered a client as an observer have to ensure that the client achieves eventual consistency with respect to the representation of the observed resource. This realistically relies on the sending of new Observe notifications, which are occasionally expected to be sent as Confirmable messages also in order to assess client aliveness (see below).

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 the aliveness 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 CoAP 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 canceled 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 at servers that are members of a CoAP group through a CoAP-to-CoAP proxy, the limitations stated in Section 3.5 apply. The method defined in [I-D.ietf-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.

If a client sends a multicast GET or FETCH request including a Block2 Option with a block number of 0, then the client can rely on two possible approaches in order to retrieve any further blocks of the resource from responding servers.

1. The client uses unicast requests, separately addressing each different server.

2. The client uses follow-up group requests, if all the responses received from different servers specify the same block size in their Block2 Option. In particular, such a block size can be equal to the block size specified in the Block2 Option of the first group request, or instead a smaller one. If the client relies on this approach, then the Block2 Option of follow-up group requests in the same block-wise transfer specifies the same block size used by all the servers in the Block2 Option of their responses.

Furthermore, 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. That is the case when a client sends a multicast GET or FETCH request that does not include a Block2 Option.

After a client receives responses that include a Block2 Option to the first group request that did not include a Block2 Option, the client can rely on either of the two approaches above for any further requests to retrieve more blocks of the resource. Alternatively, the client can compute a block size that is smaller than or equal to the smallest block size among those specified in the Block2 Option of the received responses. If the client relies on this latter approach, then the Block2 Option of follow-up group requests in the same block-wise transfer specifies the block size computed by the client.

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.

[RFC9177] specifies a specialized 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 CoAP group. An IPv6 CoAP server SHOULD support the "All CoAP Nodes" multicast CoAP group with at least link-local (2), admin-local (4), and site-local (5) scopes. An IPv6 CoAP server on a 6LoWPAN node (see Section 3.9.2) 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/IPv6 Multicast Transport over 6LoWPAN

In 6LoWPAN [RFC4944] [RFC6282] networks, an IPv6 packet (up to 1280 bytes) may be fragmented into multiple 6LoWPAN fragments, each fragment small enough to be carried over an IEEE 802.15.4 MAC frame (up to 127 bytes).

These 6LoWPAN fragments are exchanged between 6LoWPAN nodes, potentially involving 6LoWPAN routers operating in a multi-hop network topology. Although 6LoWPAN multicast routing protocols usually define mechanisms to compensate for the loss of transmitted fragments (e.g., using link-layer unicast acknowledgements, or repeated link-layer broadcast transmissions as in MPL -- see Section 3.10.3) a fragment may still be lost in transit. The loss of a single fragment implies the loss of the entire IPv6 packet, because the reassembly back into IPv6 packet will fail in that case. Also, if this fragment loss causes the application-layer retransmission of the entire multi-fragment IPv6 packet, it may happen that much of the same data is transmitted yet again over the constrained network.

For this reason, the performance in terms of packet loss and throughput of using larger, multi-fragment multicast IPv6 packets is on average worse than the performance of smaller, single-fragment IPv6 multicast packets. So it is recommended to design application payloads for group communication sufficiently small: a CoAP request sent over multicast over a 6LoWPAN network interface SHOULD fit in a single IEEE 802.15.4 MAC frame, if possible.

On 6LoWPAN networks, multicast CoAP groups can be defined with realm-local scope [RFC7346]. Such a realm-local CoAP group is restricted to the local 6LoWPAN network/subnet. In other words, a multicast request to that CoAP 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 CoAP group (see Section 3.9.1) in order to discover only CoAP servers on the local 6LoWPAN network.

3.9.3. 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 CoAP 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].

3.9.4. TCP, TLS, and WebSockets

CoAP over TCP, TLS, and WebSockets is defined in [RFC8323]. Although it supports unicast only, it can be employed as a transport for CoAP group communication in situations where unicast is used, such as exchanging messages with a proxy and completing block-wise transfers. In particular:

* A suitable cross-proxy can be set up, such that it receives a unicast CoAP group request over TCP/TLS/WebSockets, and then forwards the request to the servers in the group over UDP/IP multicast (see Section 3.5). The discovery of such a proxy can rely on means defined in [I-D.ietf-core-transport-indication].

* [RFC8323] can be employed to complete block-wise transfers for CoAP group communication, with the limitations discussed in Section 3.8.

  That is, after the first group request including the Block2 Option and sent over UDP, the following unicast CoAP requests targeting individual servers to retrieve further blocks may be sent over TCP or WebSockets, possibly protected with TLS.

  This requires the individually addressed servers to be reachable via a suitable cross-proxy or to also support CoAP over TCP/TLS/WebSockets for the targeted resource. While those transports do not support multicast, it is possible to rely on multicast for discovering that a server has those transports available and that they allow accessing the targeted resource, possibly with block-wise transfer used for random access to blocks within the resource representation. Such discovering can rely on means defined in [I-D.ietf-core-transport-indication].

3.9.5. 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.

3.10. Interworking with Other Protocols

3.10.1. MLDv2 and IGMPv3

A CoAP node that is an IP host (i.e., not an IP router) may be unaware of the specific IP multicast routing/forwarding protocol being used in its network. When such a node needs to join a specific (CoAP) multicast group, the application process would typically subscribe to the particular IP multicast group via an API method of the IP stack on the node. Then the IP stack would execute a particular (e.g., default) method to communicate its subscription to on-link IP (multicast) routers.

The Multicast Listener Discovery Version 2 (MLDv2) protocol [RFC9777] is the standard IPv6 method to communicate multicast subscriptions, when other methods are not defined. The CoAP server nodes then act in the role of MLDv2 Multicast Address Listener. MLDv2 uses link-local communication between Listeners and IP multicast routers. Constrained IPv6 networks such as ones implementing 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 Internet Group Management Protocol Version 3 (IGMPv3) protocol [RFC9776] is the standard IPv4 method to signal subscriptions to multicast group. This SHOULD be used by members of a CoAP group to subscribe to its multicast IPv4 address on IPv4 networks unless another method is defined for the network interface/technology used.

The guidelines from [RFC6636] on the tuning of MLDv2 and IGMPv3 for mobile and wireless networks may be useful when implementing MLDv2 and IGMPv3 in constrained networks.

3.10.2. RPL

IPv6 Routing Protocol for Low-Power and Lossy Networks (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 set to a mode that supports multicast, for example 3 (Storing mode with multicast support) or 5 (Non-Storing Mode of Operation with ingress replication multicast support) defined in [RFC9685].

In mode 3, RPL DAO can be used by an RPL/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 members of the CoAP group.

The same DAO mechanism can be used by an edge router such as a 6LoWPAN Border Router (6LBR, see [RFC6775]), in order to learn CoAP group membership information of the entire RPL network, 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 learns which IP multicast traffic it needs to selectively pass through from the backbone network into the LLN subnet. 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.

See [RFC9685] for more details on RPL Mode 5, and on subscribing to IPv6 multicast groups using 6LoWPAN Neighbor Discovery (ND) and the Extended Address Registration Option (EARO) in RPL networks.

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 members of the CoAP group.

However, a CoAP multicast request to a CoAP 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 [RFC9777], which works only in case all CoAP servers joining a CoAP group are in link-local communication range of an ingress MPL Forwarder.

* Using 6LoWPAN Neighbor Discovery (ND) and Extended Address Registration Option (EARO) as described in [RFC9685], in a network that supports 6LoWPAN-ND, RPL, and MPL.

* 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, other filtering criteria may also be defined at an ingress MPL Forwarder. See Section 6.6 for more details.

4. Unsecured Group Communication (NoSec Mode)

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 NoSec mode does not require and does not make use of a security group. Indications that endpoints can use the NoSec mode MUST NOT rely on setting up and advertising a pseudo security group with name "NoSec" or any of its lowercase/uppercase combinations.

A CoAP server in NoSec mode MUST NOT be accessible through the public Internet. It is NOT RECOMMENDED to use CoAP group communication in NoSec mode.

The possible, exceptional use of the NoSec mode ought to be limited to specific, well-defined "unsecured steps" that unquestionably do not require security or are not able to attain it, e.g., early discovery of devices and resources (see Section 6.1).

Before possibly and exceptionally using the NoSec mode in such circumstances, 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). Consistent with such security implications, the use of the NoSec mode should still be avoided whenever possible.

5. Secured Group Communication using Group OSCORE

This section discusses 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 [RFC9052][RFC9053] 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. Section 4 of [RFC8613] defines in detail if and what protection is applied to the CoAP header fields, payload, and CoAP options specified in the unprotected message, in accordance with their class for OSCORE.

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. That is, the recipient of a CoAP message protected with Group OSCORE is able to securely verify whether the CoAP endpoint that has generated and sent the message is indeed the alleged one. This is achieved 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 security 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 security group members' authentication credentials including the corresponding public keys; manages, renews, and provides security material in the security group; and handles the join process of new members in the security group.

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 configuration of the security group 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 [RFC9200].

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 [RFC9176].

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 that are configured to receive CoAP group messages sent to the particular CoAP group and -- in case of IP multicast transport -- that 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, requires 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 member of that group cannot read the cleartext of 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 participate 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 verify 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 12.2 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 is considered trustworthy in sending a group request on behalf of clients and it is configured as a member of the OSCORE security group(s) that it needs to access. As further clarified below, the first communication leg between the client and the proxy, on one hand, and the second communication leg between the proxy and the servers, on the other hand, are protected individually and independently of one another.

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.ietf-core-groupcomm-proxy].

6. Security Considerations

This section provides security considerations for CoAP group communication, in general and in particular when using IP multicast.

6.1. CoAP NoSec Mode

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

For these reasons and in order to prevent proliferation of high-volume amplification attacks as further discussed in Section 6.3, it is NOT RECOMMENDED to use CoAP group communication in NoSec mode. The requirement in Section 4 on publically accessible CoAP servers also aims to prevent amplification attacks.

Exceptionally, and only after the security implications have been very well considered and understood, some applications may rely on a limited use of the NoSec mode, when performing specific, well-defined "unsecured steps" (see Section 4).

For example, early link-local discovery of devices and resources as part of an onboarding protocol is a typical use case where the NoSec mode or equivalent unsecured mode is used. In such a discovery step, there may be a querying device that needs to discover nearby devices capable of helping it with the network onboarding process. But there are no mutual security relationships configured on the querying device and its neighbor devices at the time it performs the early discovery. These relationships are configured later in the process based on secure device identities. Alternatively, a new device to be onboarded may wait for advertisements of nearby devices able to help it do the network onboarding process. Also in this case, these messages cannot be secured initially because the new device does not yet have any security relationship configured with devices that are already a member of the network. See [I-D.ietf-anima-constrained-voucher] for an example of an onboarding protocol that can use CoAP multicast for early link-local discovery.

As a further example, the NoSec mode may be useful and acceptable in simple read-only applications that do not involve, impact, or disclose sensitive data and personal information. These include, e.g., read-only temperature sensors deployed in a public gathering environment, where unauthenticated clients retrieve temperature values but do not use this data to control actuators or to perform other automated actions.

In the exception cases where NoSec mode is used, high-volume and harmful amplifications need to be prevented through appropriate and conservative device configurations: taking the early discovery query as an example, only a few CoAP servers are expected to be configured for responding to multicast group requests that are sent for discovery. And the time window during which such unsecured requests are accepted, can be limited as well. Furthermore, the scope is also limited: only link-local requests are accepted.

Except for the class of applications discussed above, and all the more so in applications that obviously have hard security requirements (e.g., health monitoring systems and alarm monitoring systems), CoAP group communication MUST NOT be used in NoSec mode.

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 specific dynamics of security group membership (e.g., the problem of distributing security keys across large groups with many members that come and go).

CoAP group communication MUST be protected by using Group OSCORE as specified in [I-D.ietf-core-oscore-groupcomm], with the possible exception of specific, well-defined "unsecured steps" (see Section 4).

The security considerations from Section 14 of [I-D.ietf-core-oscore-groupcomm] hold for this specification.

6.2.1. Group Key Management

Group rekeying, that is, a key management scheme for secure revocation and renewal of group security material, 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 12.2 of [I-D.ietf-core-oscore-groupcomm].

Even though security group policies vary depending on the specific applications and their security requirements, they SHOULD also take into account:

* The expected amount of time that the key management scheme requires to rekey the OSCORE group.

* The expected frequency of group membership changes in the OSCORE group (i.e., nodes joining and leaving).

* The identity of the specific CoAP endpoints as they join and leave the OSCORE group.

In particular, it can be desirable to not rekey the OSCORE group upon every single membership change, in case group members frequently join and leave, and at the same time a single group rekeying instance takes a non-negligible time to complete.

In such a case, the Group Manager can cautiously consider to rekey the OSCORE 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 a slow rekeying scheme that is frequently invoked from paralyzing communications in the OSCORE group.

At the same time, the security implications of delaying the rekeying process have to be carefully considered and understood before employing such security 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 change in the OSCORE group membership. That is, 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 OSCORE group is rekeyed, recently left nodes would retain access to group communications protected with the existing 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 signed 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 and sent by a specific and identified CoAP endpoint as a member of the OSCORE group.

Note that Group OSCORE does not protect the addressing information about the CoAP endpoint that has sent the message, e.g., the source IP address and port number used when sending the message. In particular, Group OSCORE does not provide authentication of such source addressing information.

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 meddler-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 OSCORE 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 7 and 8 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]), thanks to the handling of protected group requests on the server side. That is, upon receiving a group request protected with Group OSCORE, a server verifies whether the request is not a replay, and whether it has been originated by the alleged CoAP endpoint in the OSCORE group.

  In order to perform the latter check of source authentication, the server either: i) verifies the signature included in the request by using the public key of the client, when the request is protected using the group mode (see Section 7.2 of [I-D.ietf-core-oscore-groupcomm]); or ii) decrypts and verifies the request by means of an additionally derived pairwise key associated with the client, when the request is protected using the pairwise mode (see Section 8.4 of [I-D.ietf-core-oscore-groupcomm]).

  As also discussed in Section 7 of [I-D.ietf-core-oscore-groupcomm], it is recommended that, when failing to decrypt and verify an incoming group request protected with the group mode, a server does not send back any error message in case any of the following holds: the server determines that the request was indeed sent to the whole CoAP group (e.g., over IP multicast); or the server is not able to determine it altogether.

Such a message processing on the server 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 often be correctly inferred 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 verify whether 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 of the group request, especially if such an address differs from the one used in previous group requests from the same (authenticated) device. 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 response.

Using the Echo Option does not provide authentication of source addressing information about the sender of a CoAP message. Also, using the Echo Option in itself does not provide source authentication of exchanged messages, which is achieved by means of Group OSCORE (see Section 6.2.2). Using the Echo Option together with Group OSCORE allows a CoAP server in the OSCORE group to verify the freshness of CoAP requests received from other members in the group.

* Group OSCORE limits the impact of attacks based on IP spoofing 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 CoAP endpoints that are legitimate members of the group.

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 7.1 and 7.3 of [I-D.ietf-core-oscore-groupcomm]), and in the pairwise mode by using additionally derived pairwise keys (see Sections 8.3 and 8.5 of [I-D.ietf-core-oscore-groupcomm]). Thus, recipient endpoints can verify a message to have been originated and sent by the alleged, identifiable CoAP endpoint 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 members of an OSCORE group 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 OSCORE group.

* Group OSCORE prevents any single member of an OSCORE group from being a target for subverting security in the whole 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 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, consistent 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 cannot be authenticated in some way, i.e., for which it is not possible to securely verify that they have been originated and sent by the alleged, identifiable CoAP endpoint in the OSCORE group;

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

Amplification attacks using CoAP are further discussed in [I-D.irtf-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, consistent with Section 7 of [RFC7641], a server in a CoAP group MUST strictly limit the number of notifications it sends between receiving acknowledgements 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, also in order to prevent an easy proliferation of high-volume amplification attacks, it is generally NOT RECOMMENDED to use CoAP group communication in NoSec mode (see Section 6.1).

Besides building on very well understood security implications and being limited to specific, well-defined "unsecured steps" (see Section 4), possible exceptions should also be limited to use cases where accesses to a group resource 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 [RFC9176], 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 yet have configured a mutual security relationship 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 that are the 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 Group Requests

After a client has sent a group request over IP multicast, an adversary might capture the group request to be re-injected in the group as a replay to the server nodes. In particular:

* If the adversary re-injects the group request before the client has freed up the corresponding Token value (see Section 3.1.5), the client might receive additional responses from one or more of the servers in the group.

  Due to the group request being Non-confirmable and thus not eliciting Acknowledgement messages, the client might not be able to notice the attack, or to distinguish the responses that a particular server has sent as reply to the original group request (if any) or to the replayed group request.

* If the adversary re-injects the group request after the client has freed up the corresponding Token value, the client would not have anymore a valid, active request matching with responses that the servers sent to the replayed group request.

It follows that, in either case, this replay attack would not accomplish anything on the client, although it does effectively target the servers in the group that do not implement the optional Message ID based deduplication.

If Group OSCORE is used, such a replay attack is prevented on all servers, since a client protects each 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 Replay Window that is in a valid state. If the server's Replay Window is initialized as invalid, e.g., due to a reboot, the server should use the challenge-response synchronization method based on the Echo Option for CoAP defined in [RFC9175] as described in Section 9 of [I-D.ietf-core-oscore-groupcomm], in order to make the Replay Window valid before resuming to accept incoming messages from other group members.

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 many responses from servers to a group request as possible, by using the value '0' -- Interested in all responses. This might even override 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).

As a mitigation, a server that takes into account the No-Response Option can specifically relax the standard suppression rules for a resource only in case the option is sent by an authenticated client. Consequently, if sent by an unauthenticated client, the option can still 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, the MPL [RFC7731] protocol may be used to forward multicast packets throughout the network. A 6LoWPAN Router that forwards a large IPv6 packet may have a relatively high impact on the occupation of the wireless channel because sending a large packet consists of the transmission of multiple link-layer IEEE 802.15.4 frames. Also, a constrained 6LoWPAN Router may experience a high memory load due to buffering of the large packet -- MPL requires an MPL Forwarder to store the packet for a longer duration, to allow multiple forwarding transmissions to neighboring Forwarders. This

could allow an attacker on the 6LoWPAN network or outside the 6LoWPAN network to execute a Denial of Service (DoS) attack by sending large IPv6 multicast packets. This is also an amplification attack in general, because each of potentially multiple MPL Forwarder(s) repeats the transmission of the IPv6 packet potentially multiple times, hence amplifying the original amount of data sent by the attacker considerably.

The amplification factor may be even further increased by the loss of link-layer frames. 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 it will likely at a future point in time trigger a neighboring MPL Forwarder to send the IPv6 packet (fragments) again, because its internal state marks this packet (that it failed to received previously) still as a "new" IPv6 packet. Hence, this leads to an MPL Forwarder signaling to neighbors its "old" state, triggering additional transmission(s) of all packet fragments.

For these reasons, a large IPv6 multicast packet is a possible attack vector in a Denial of Service (DoS) amplification attack on a 6LoWPAN network. 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. If packet forwarding relies on priority-based scheduling, a 6LoWPAN Router or (MPL) multicast forwarder SHOULD deprioritize forwarding for multi-fragment 6LoWPAN multicast packets, which is implementation specific. 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 criterion 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, to simultaneously turn on all the lights in a room. This intrinsically exposes the communicating devices to some unique monitoring risks, which they are not as vulnerable to when not using group communication.

For example, an attacker might be 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 CoAP group, some network topology information may be deduced).

6.8.2. Pervasive Monitoring

CoAP traffic is typically used for the Internet of Things, and CoAP (multicast) group communication may specifically be used for conveniently controlling and monitoring critical infrastructure (e.g., lights, alarms, HVAC, electrical grid, etc.).

This may make it a prime target of pervasive monitoring attacks [RFC7258], which have to be considered as a key additional threat for group communication. 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 instance, the source node (controller) sends out CoAP group messages, which easily identifies it as a controller.

CoAP group communication built on top of IP multicast is inherently more vulnerable compared to communications solely relying on IP unicast, since the same packet may be replicated over many links. In

particular, this yields a higher probability of packet capture by a pervasive monitoring system, which in turn results in more information available to analyze within the same time interval. Moreover, a single CoAP group request potentially results in multiple CoAP responses, thus further contributing to the information available to analyze.

This requires CoAP group communication solutions that are built on top of IP multicast to pay particular attention to these dangers.

In order to limit the ease of interception of group communication messages, 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 it fulfills the application needs.

Even if CoAP group communications are encrypted/protected (see Section 5), 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 R. H旦glund, "Group Object Security for Constrained RESTful Environments (Group OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-25, 16 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-25>.

[Resource.Type.Link.Target.Attribute.Values]
IANA, "Resource Type (rt=) Link Target Attribute Values", <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#rt-link-target-att-value>.

[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/rfc/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/rfc/rfc2119>.

[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/rfc/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/rfc/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/rfc/rfc6282>.

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

[RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <https://www.rfc-editor.org/rfc/rfc6775>.

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

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

[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/rfc/rfc7959>.

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

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

[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/rfc/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/rfc/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/rfc/rfc8613>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

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

[RFC9175] Ams端ss, C., Preu 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/rfc/rfc9175>.

[RFC9776] Haberman, B., Ed., "Internet Group Management Protocol, Version 3", STD 100, RFC 9776, DOI 10.17487/RFC9776, March 2025, <https://www.rfc-editor.org/rfc/rfc9776>.

8.2. Informative References

[Californium]
Eclipse Foundation, "Eclipse Californium", June 2022, <https://github.com/eclipse/californium>.

[Go-CoAP] Open Connectivity Foundation (OCF), "Go-CoAP", June 2022, <https://github.com/go-ocf/go-coap>.

[I-D.bormann-core-responses]
Bormann, C. and C. Amsss, "CoAP: Non-traditional response forms", Work in Progress, Internet-Draft, draft-bormann-core-responses-04, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-bormann-core-responses-04>.

[I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M. and F. Palombini, "Key Management for Group Object Security for Constrained RESTful Environments (Group OSCORE) Using Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-17, 11 April 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-oscore-17>.

[I-D.ietf-ace-oscore-gm-admin]
Tiloca, M., Hglund, R., Van der Stok, P., and F. Palombini, "Admin Interface for the OSCORE Group Manager", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-gm-admin-13, 8 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-oscore-gm-admin-13>.

[I-D.ietf-anima-constrained-voucher]
Richardson, M., Van der Stok, P., Kampanakis, P., and E. Dijk, "Constrained Bootstrapping Remote Secure Key Infrastructure (cBRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-voucher-27, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-constrained-voucher-27>.

[I-D.ietf-core-coap-pubsub]
Jimenez, J., Koster, M., and A. Kernen, "A publish-subscribe architecture for the Constrained Application

Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-18, 28 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-coap-pubsub-18>.

[I-D.ietf-core-groupcomm-proxy]
Tiloca, M. and E. Dijk, "Proxy Operations for CoAP Group Communication", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-proxy-04, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-proxy-04>.

[I-D.ietf-core-transport-indication]
Amsss, C. and M. S. Lenders, "CoAP Transport Indication", Work in Progress, Internet-Draft, draft-ietf-core-transport-indication-08, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-transport-indication-08>.

[I-D.irtf-t2trg-amplification-attacks]
Mattsson, J. P., Selander, G., and C. Amsss, "Amplification Attacks Using the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-irtf-t2trg-amplification-attacks-05, 18 June 2025, <https://datatracker.ietf.org/doc/html/draft-irtf-t2trg-amplification-attacks-05>.

[I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsss, C., and P. Van der Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-17, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-tiloca-core-oscore-discovery-17>.

[libcoap] Bergmann, O., "libcoap", June 2022, <https://github.com/obgm/libcoap>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/rfc/rfc1035>.

[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/rfc/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/rfc/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/rfc/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/rfc/rfc7258>.

[RFC7346] Droms, R., "IPv6 Multicast Address Scopes", RFC 7346, DOI 10.17487/RFC7346, August 2014, <https://www.rfc-editor.org/rfc/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/rfc/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/rfc/rfc7731>.

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

[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/rfc/rfc8710>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/rfc/rfc9019>.

[RFC9124] Moran, B., Tschofenig, H., and H. Birkholz, "A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices", RFC 9124, DOI 10.17487/RFC9124, January 2022, <https://www.rfc-editor.org/rfc/rfc9124>.

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

[RFC9177] Boucadair, M. and J. Shallow, "Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission", RFC 9177, DOI 10.17487/RFC9177, March 2022, <https://www.rfc-editor.org/rfc/rfc9177>.

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

[RFC9685] Thubert, P., Ed., "Listener Subscription for IPv6 Neighbor Discovery Multicast and Anycast Addresses", RFC 9685, DOI 10.17487/RFC9685, November 2024, <https://www.rfc-editor.org/rfc/rfc9685>.

[RFC9777] Haberman, B., Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", STD 101, RFC 9777, DOI 10.17487/RFC9777, March 2025, <https://www.rfc-editor.org/rfc/rfc9777>.

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 resources 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 [RFC9176]; 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 (Section 7 of [RFC7252]), 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 and the CoAP query interface defined in Section 4 of [RFC6690] to find these particular resources.

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 subcase 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.

As defined in [RFC9176], a CoRE Resource Directory (RD) 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 an RD to let it be found by other devices and/or applications.

As an example, the following shows two IPv6 network configurations. Both are based on the topology as shown in Figure 3. The two configurations using this topology are as follows:

1. Subnets are 6LoWPAN networks; the routers Rtr-1 and Rtr-2 are 6LoWPAN Border Routers (6LBRs) [RFC6775].

2. Subnets are Ethernet links; the routers Rtr-1 and Rtr-2 are multicast-capable Ethernet routers.

Both configurations are further specified by the following:

* A large room (Room-A) with three lights (Light-1, Light-2, Light-3) controlled by a light switch (Light Switch). The devices are organized into two subnets. In reality, there could be more lights (up to several hundreds) but, for clarity, only three are shown.

* Light-1 and the light switch are connected to a router (Rtr-1).

* Light-2 and Light-3 are connected to another router (Rtr-2).

* The routers are connected to an IPv6 network backbone (Network Backbone) that is also multicast enabled. In the general case, this means the network backbone and Rtr-1/Rtr-2 support a PIM-based multicast routing protocol and, e.g., Multicast Listener Discovery v2 (MLDv2) for forming groups.

* A CoRE RD using CoAP (CoAP Resource Directory) is connected to the network backbone.

* The DNS server (DNS Server) is optional. If the server is there (connected to the network backbone), then certain DNS-based features are available (e.g., DNS resolution of the hostname to

the IP multicast address). If the DNS server is not there, then different provisioning of the network is required (e.g., IP multicast addresses are hard-coded into devices, or manually configured, or obtained via a service discovery method).

* A controller using CoAP (CoAP Controller Client) is connected to the backbone, which is able to control various building functions including lighting.

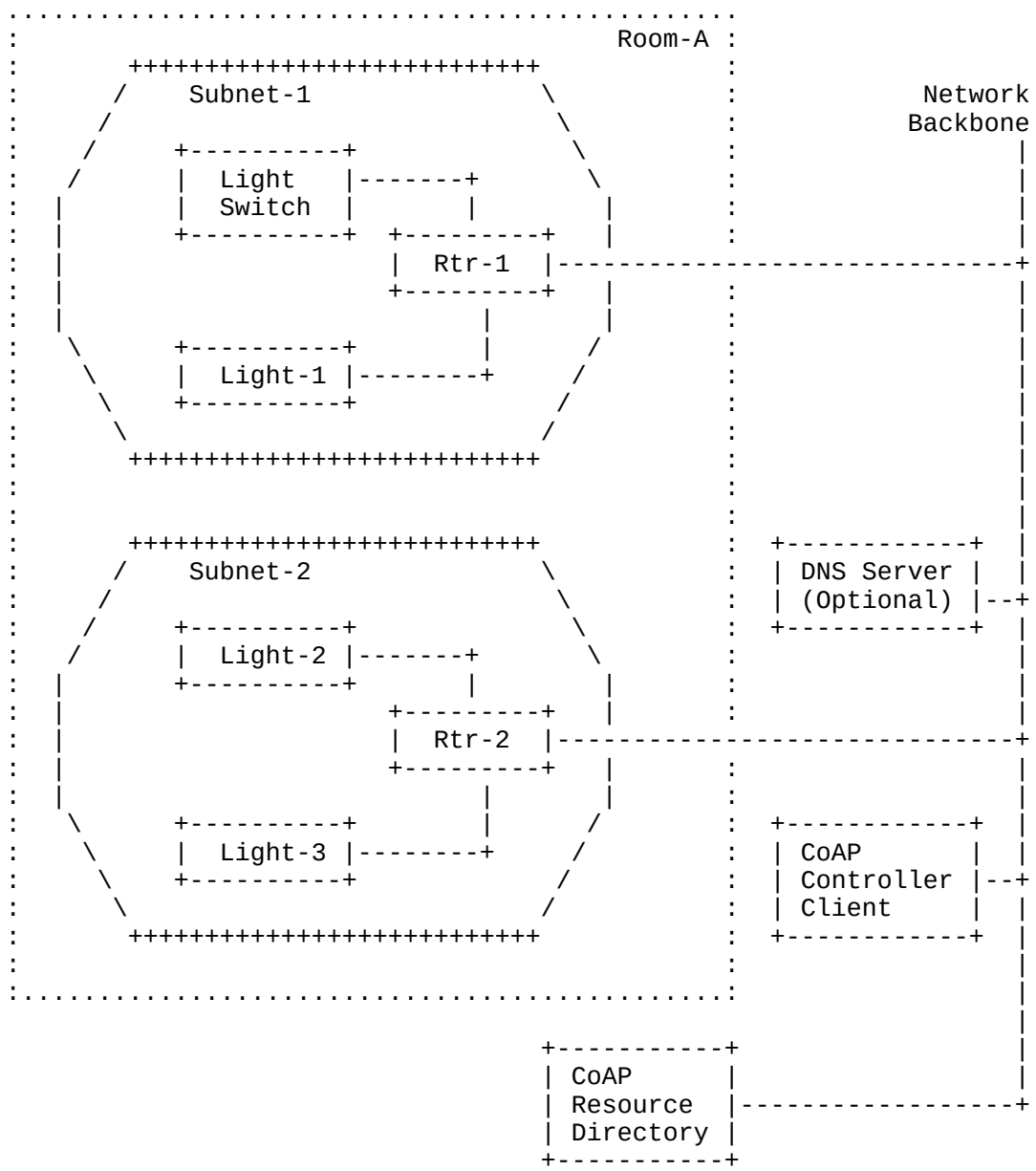


Figure 3: Network Topology of a Large Room (Room-A)

Building on the above, an example of protocol flow for discovery of the RD for the given network (of Figure 3) is shown in Figure 4:

* Light-2 is installed and powered on for the first time.

* Light-2 will then search for the local RD by sending out a group communication GET request (with the "/.well-known/core?rt=core.rd" request URI) to the site-local "All CoAP Nodes" multicast address (ff05::fd).

* This multicast message will then go to each node in Subnet-2. Rtr-2 will then forward it into the network backbone where it will be received by the RD. All other nodes in Subnet-2 will ignore the group communication GET request because it is qualified by the query string "?rt=core.rd" (which indicates it should only be processed by the endpoint if it contains a resource of type "core.rd").

* The RD will then send back a unicast response containing the requested content, which is a CoRE Link Format representation of a resource of type "core.rd".

* Note that the flow is shown only for Light-2 for clarity. Similar flows will happen for Light-1, Light-3, and Light Switch when they are first installed.

The RD may also be discovered by other means such as by assuming a default location (e.g., on a 6LBR), using DHCP, anycast address, etc. However, these approaches do not invoke CoAP group communication so are not further discussed here.

For other discovery use cases such as discovering local CoAP servers, services, or resources, CoAP group communication can be used in a similar fashion as in the above use case. For example, link-local, realm-local, admin-local, or site-local scoped discovery can be done this way.

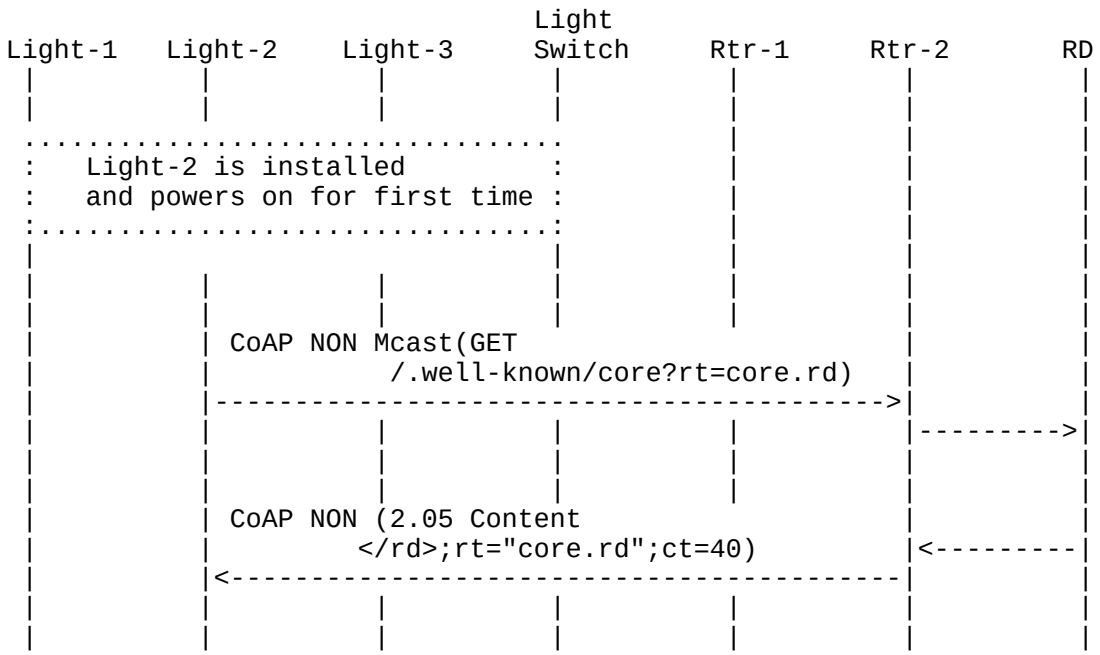


Figure 4: Resource Directory Discovery via Multicast Request

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 cohesive collection, with strict timing (latency) requirements. Examples are office lighting, stage lighting, street lighting, or audio alert/Public Address systems.

The following presents examples of lighting control of a set of 6LoWPAN-connected lights.

Figure 5 shows the protocol flow for a building automation lighting control scenario for the network considered in Figure 3. The network is assumed to be in a 6LoWPAN configuration. Also, it is assumed that the CoAP servers in each light are configured to suppress CoAP responses for any IP multicast CoAP requests related to lighting control. (See Section 3.1.2 for more details on response suppression by a server.)

In addition, Figure 6 shows a protocol flow example for the case where servers do respond to a lighting control IP multicast request with (unicast) CoAP NON responses. There are two success responses and one 5.00 error response. In this particular case, the light switch does not check that all lights in the group received the IP multicast request by examining the responses. This is because the light switch is not configured with an exhaustive list of the IP addresses of all lights belonging to the group. However, based on received error responses, it could take additional action such as logging a fault or alerting the user via its LCD display. In case a CoAP message is delivered multiple times to a light, the subsequent CoAP messages can be filtered out as duplicates, based on the CoAP Message ID.

Reliability of IP multicast is not guaranteed. Therefore, one or more lights in the group may not have received the CoAP control request due to packet loss. In this use case, there is no detection nor correction of such situations: the application layer expects that the IP multicast forwarding/routing will be of sufficient quality to provide on average a very high probability of packet delivery to all CoAP endpoints in an IP multicast group. An example protocol to accomplish this using randomized retransmission is the MPL forwarding protocol for LLNs [RFC7731].

We assume the following steps have already occurred before the illustrated flows:

1) Startup phase: 6LoWPANs are formed. IPv6 addresses are assigned to all devices. The CoAP network is formed.

2) Network configuration (application independent): 6LBRs are configured with IP multicast addresses, or address blocks, to filter out or to pass through to/from the 6LoWPAN.

3a) Commissioning phase (application related): The IP multicast address of the group (Room-A-Lights) has been configured in all the lights and in the light switch.

3b) As an alternative to the previous step, when a DNS server is available, the light switch and/or the lights have been configured with a group hostname that each node resolves to the above IP multicast address of the group.

Note for the Commissioning phase: the switch's 6LoWPAN/CoAP software stack supports sending unicast, multicast, or proxied unicast CoAP requests, including processing of the multiple responses that may be generated by an IP multicast CoAP request.

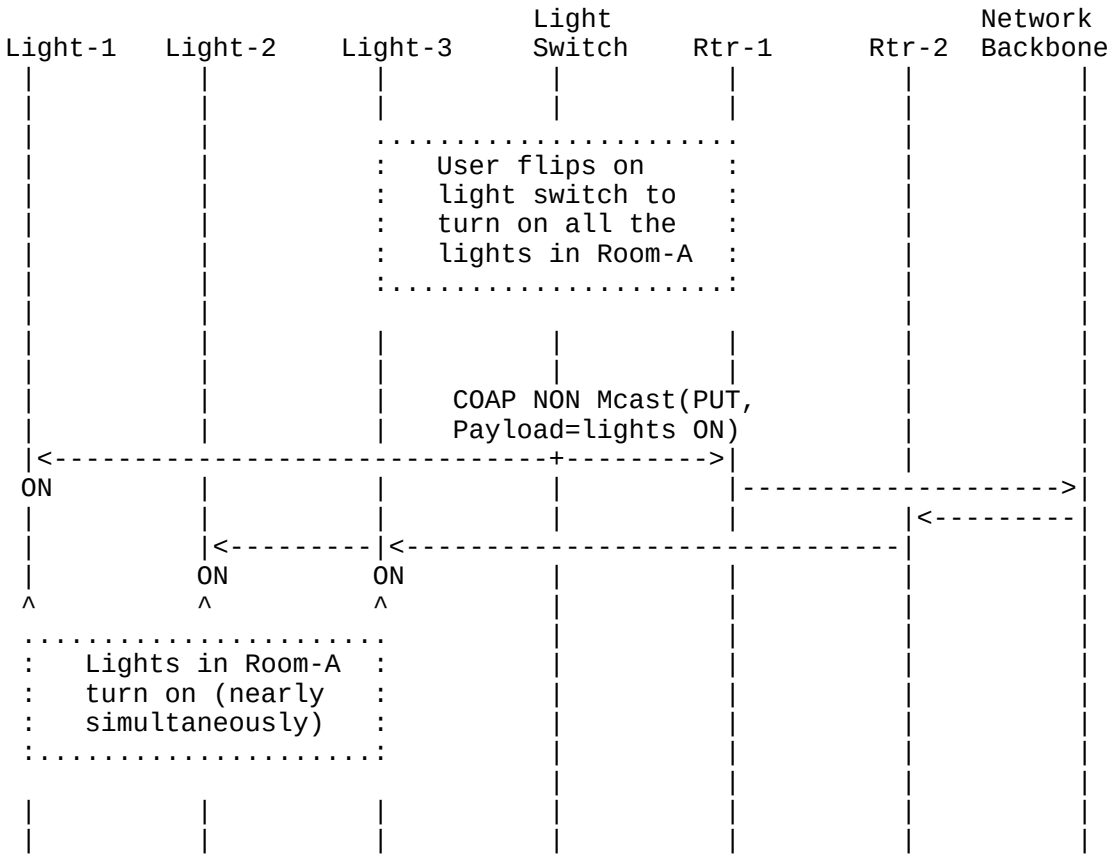


Figure 5: Light Switch Sends Multicast Control Message

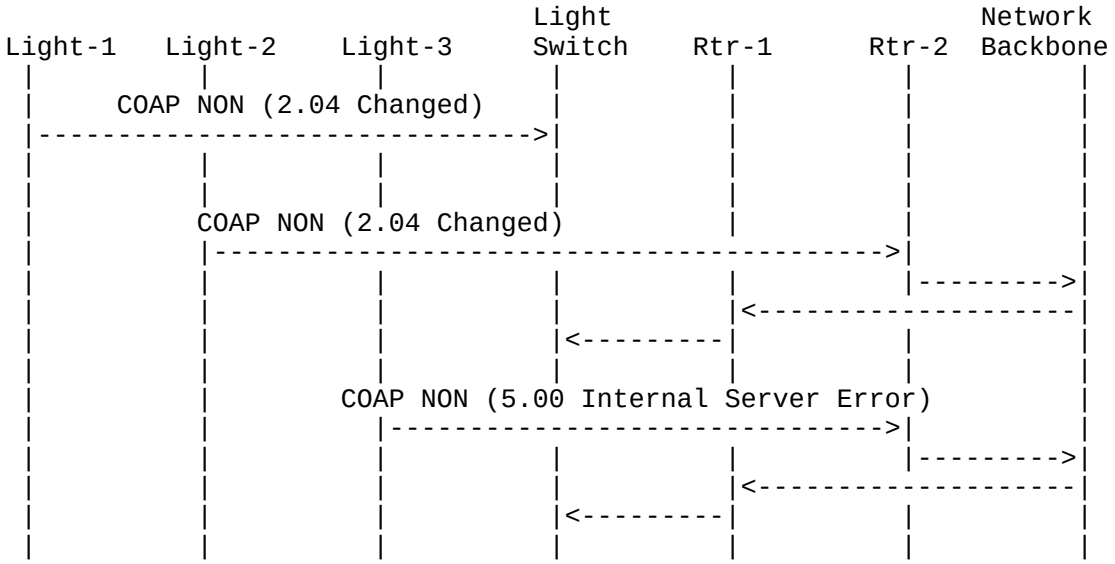


Figure 6: Lights (Optionally) Respond to Multicast CoAP Request

Another, but similar, lighting control use case is shown in Figure 7. In this case, a controller connected to the network backbone sends a CoAP group communication request to turn on all lights in Room-A. Every light sends back a CoAP response to the controller after being turned on.

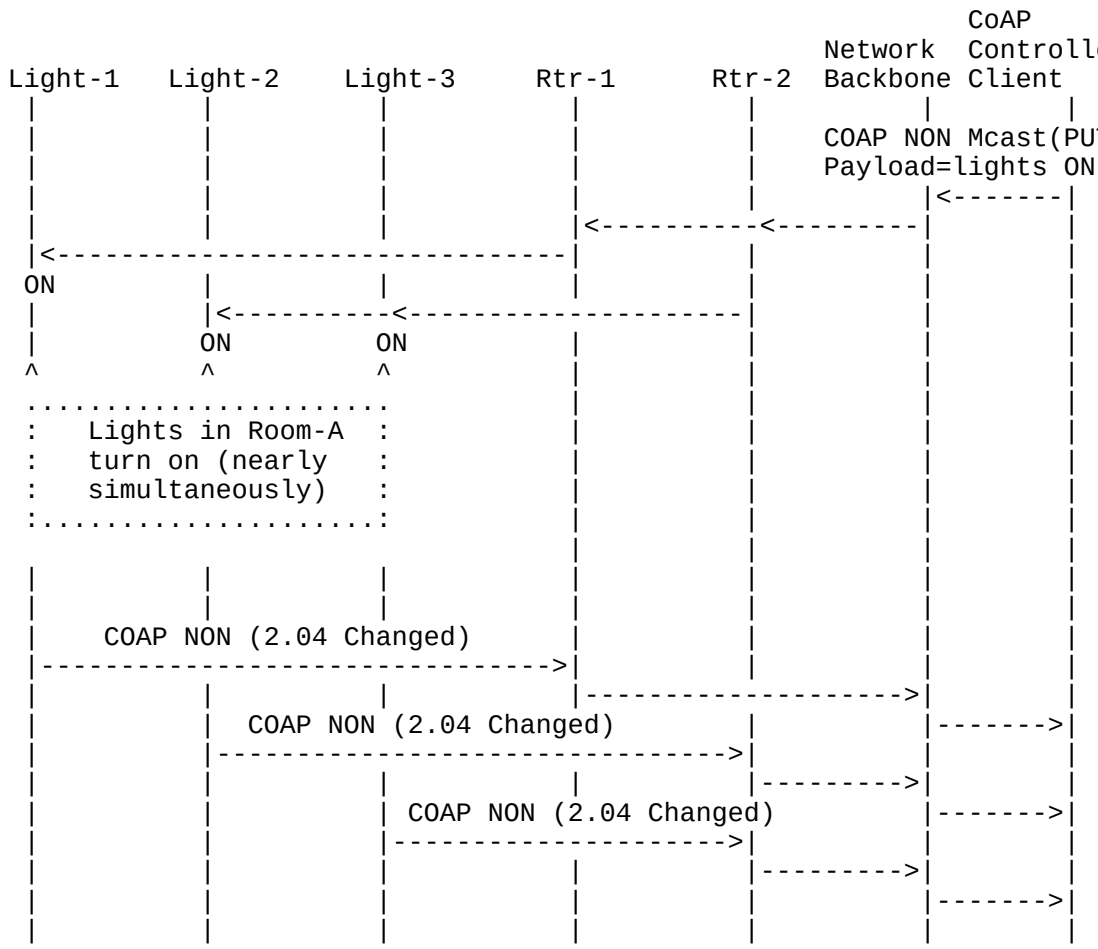


Figure 7: Controller on Backbone Sends Multicast Control Message

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. A set of devices may be defined as a CoAP group, e.g., as intended to cover "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 set of devices 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 set of devices 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 request successfully that provided a notification.

A.3. Software Update

Group communication can be useful to efficiently distribute new software (firmware, image, application, etc.) to a set of multiple devices, e.g., by relying on the SUIT firmware update architecture [RFC9019] and its manifest information model [RFC9124]. In this case, a CoAP group can be defined in terms of the device type of its members: all devices in the target CoAP 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 CoAP 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 CoAP 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 Group Naming for Application Groups

This section provides examples for the different methods that can be used to name application groups, as defined in Section 2.2.1.2.

The content of this section is purely illustrative and has no ambition to be comprehensive. That is, while the methods defined in Section 2.2.1.2 are presented as viable to use, further viable methods might exist and can be defined in the future. Furthermore, this section does not provide guidelines on how to choose between the different methods, for which a decision is application-specific.

The shown examples consider a CoAP group identified by the group hostname grp.example. Its members are CoAP servers listening to the associated IP multicast address ff35:30:2001:db8:f1:0:8000:1 and port number 5685.

Note that a group hostname is used in most examples to improve readability. In practice, as discussed in Section 2.2.1.2, using an IP address literal as the host subcomponent of the Group URI can reduce the size of the CoAP request message, in case the Uri-Host Option can be elided.

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) in the destination port field.

B.1. Group Naming using the URI Path Component

Figure 8 provides an example where the URI path component is used for naming application groups.

```
Application group name: gp1

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

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

Figure 8: Example of application group name in the URI path (1/2)

Figure 9 provides a different example, where an IPv6 literal address and the default CoAP port number 5683 are used in the authority component, which yields a compact CoAP request. Also, the resource structure is different in this example.

```
Application group name: gp1

Group URI: coap://[ff35:30:2001:db8:f1:0:8000:1]/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 9: Example of application group name in the URI path (2/2)

B.2. Group Naming using the URI Query Component

Figure 10 provides an example where the URI query component is used for naming application groups. In particular, it considers the first alternative discussed in Section 2.2.1.2, where the URI query component consists of only one parameter, which has no value and has the name of the application group as its own identifier.

```
Application group name: gp1

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

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

Figure 10: Example of application group name in the URI query (1/2)

Figure 11 provides another example, which considers the second alternative discussed in Section 2.2.1.2. In particular, the URI query component includes a query parameter "gp" as designated indicator, with value the name of the application group.

```
   Application group name: gp1

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

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

Figure 11: Example of application group name in the URI query (2/2)

B.3. Group Naming using the URI Authority Component

Figure 12 provides an example where the URI authority component as a whole is used for encoding the name of the application group. Note that, although the port information (5685) is not carried as a CoAP Option, it is still transported within the UDP message (in the UDP destination port field). So, effectively, the application group name is transported in the UDP message as two parts.

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

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

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

Figure 12: Example of application group name defined by the URI authority

Figure 13 provides an example where the URI host subcomponent of the URI authority component is used for encoding the name of the application group. Specifically, the first label of the DNS name is used.

```
Application group name: grp42

Group URI: coap://grp42.example:5685/light?foo=bar

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: "grp42.example"
   Uri-Path: "light"
   Uri-Query: "foo=bar"
```

Figure 13: Example of application group name encoded in the URI host

Figure 14 provides an example where the URI port subcomponent of the URI authority component is used for naming application groups. It uses a UDP port from the dynamic port range. Multiple application groups could be defined this way, each represented by a number and associated port in the dynamic port range.

```
Application group name: 55685

Group URI: coap://grp.example:55685/light?foo=bar

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

Figure 14: Example of application group name in the URI port

Appendix C. Examples of Group Discovery from CoAP Servers

This section provides examples for the different methods that a CoAP client can use to discover application groups and CoAP groups by interacting with CoAP servers, as defined in Section 2.2.3.2.

The examples build on the same assumptions considered in Section 2.2.3.2. In addition, a CoAP group is used and is identified by the URI authority grp.example:5685.

C.1. Application Groups Associated with a CoAP Group

Figure 15 provides an example where a CoAP client discovers all the application groups associated with a specific CoAP group.

As a result, 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.

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.

The example semantics "g.<GROUPTYPE>" is used for the values of the attribute "rt".

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

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

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

Figure 15: Discovery of application groups associated with a CoAP group

C.2. Members of a Given Application Group

Figure 16 provides an example where a CoAP client discovers the CoAP servers that are members of a specific application group and the CoAP group associated with the application group.

Note that, unlike in the example shown in Appendix C.1, now the servers need to respond with an absolute URI and not a relative URI. This is necessary because the responding CoAP endpoint serving the Link Format document (on port 5683) is a different CoAP endpoint from the one hosting the group resource "gp1" (on port 5685). Due to this situation, the responding server includes the full (absolute) URI in the Link Format response from which the client can conveniently gain knowledge of the CoAP group.

Also note that a server could equally well respond with the literal IPv6 multicast address within square brackets instead of the CoAP group name "grp.example". In that case, the client would still gain knowledge of the CoAP group, albeit in a different representation.

```
// 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:5685"
//   - The application group "gp1"
Res: 2.05 (Content)
Content-Format: 40 (application/link-format)
Payload:
<coap://grp.example:5685/gp/gp1>;rt=g.light

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

Figure 16: Discovery of members of an application group, together with the associated CoAP group

C.3. Members of any Application Group of a Given Type

Figure 17 provides an example where a CoAP client discovers the CoAP servers that are members of any application group of a specific type, and the CoAP group associated with those application groups.

```
// 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:5685"
//   - The application group "gp1" of type "g.temp"
Res: 2.05 (Content)
Content-Format: 40 (application/link-format)
Payload:
<coap://grp.example:5685/gp/gp1>;rt=g.temp

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

Figure 17: Discovery of members of application groups of a specified type, and of the associated CoAP group

C.4. Members of any Application Group in the Network

Figure 18 provides an example where a CoAP client discovers the CoAP servers that are members of any application group configured in the 6LoWPAN network of the client, and the CoAP group associated with each application group. In this example, the scope is realm-local to address all servers in the current 6LoWPAN network of the client.

The example semantics "g.<GROUPTYPE>" is used for the values of the attribute "rt".

```
// 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:5685" and "grp2.example"
//   - The application groups "gp1" and "gp5"
Res: 2.05 (Content)
Content-Format: 40 (application/link-format)
Payload:
<coap://grp.example:5685/gp/gp1>;rt=g.light,
<coap://grp2.example/gp/gp5>;rt=g.lock

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

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

Figure 18: Discovery of the resources and members of any application group, and of the associated CoAP group

Alternatively, some applications may use the "rt" attribute on a parent resource to denote support for a particular REST API to access child resources.

For instance, Figure 19 provides a different example where 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, in order 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:5685" and "grp2.example"
//   - The application groups "gp1" and "gp5"
Res: 2.05 (Content)
Content-Format: 40 (application/link-format)
Payload:
<coap://grp.example:5685/gp/gp1>;rt=oic.d.light;gpt=light,
<coap://grp2.example/gp/gp5>;rt=oic.d.smartlock;gpt=lock

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

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

Figure 19: Example of using a custom ’gpt’ link attribute to denote group type

Appendix D. Examples of Message Exchanges

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

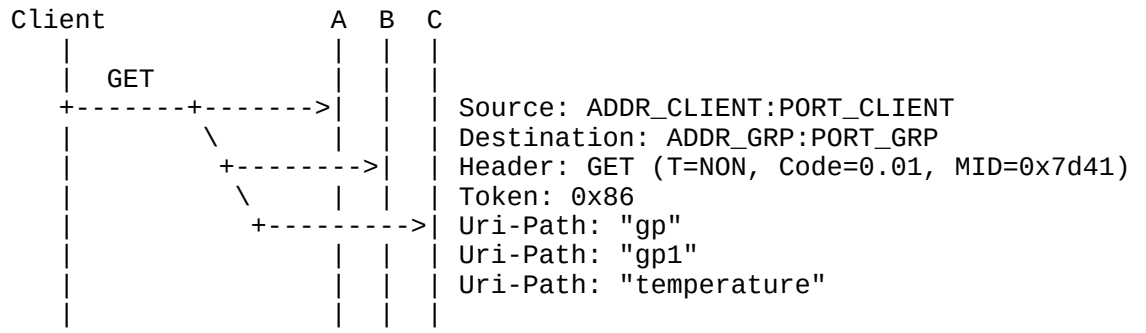
* A client with address ADDR_CLIENT and port number PORT_CLIENT.

* 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.

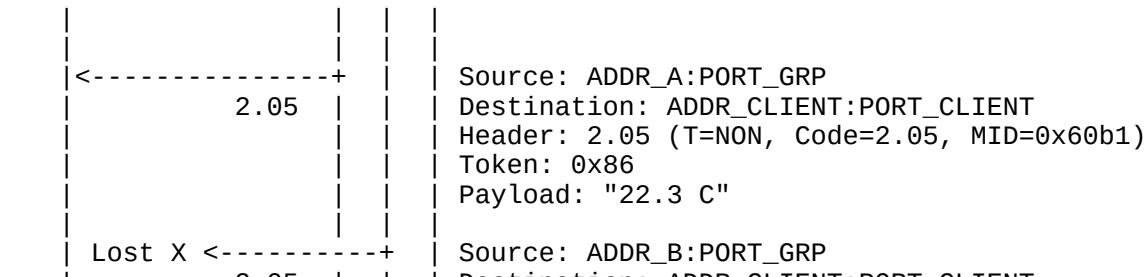
* Three servers A, B, and C, all of which are members of the CoAP group above and of the application group "gp1". Each server X (with X equal to A, B, or C): listens to its own address ADDR_X and port number PORT_X; and listens to the address ADDR_GRP and port number PORT_GRP. For each server its PORT_X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

In Figure 20, 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.



As source port number of its response, server C uses its own port number PORT_C.

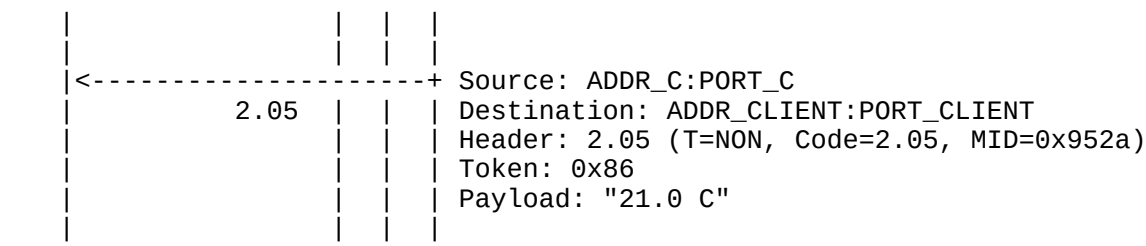
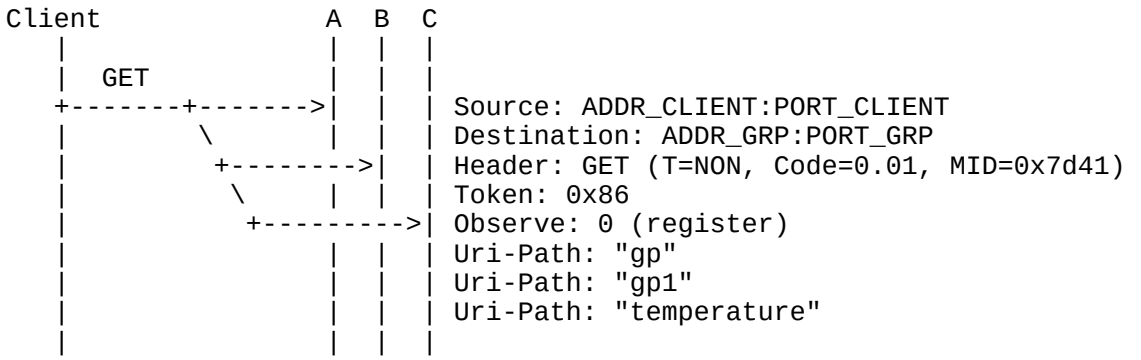


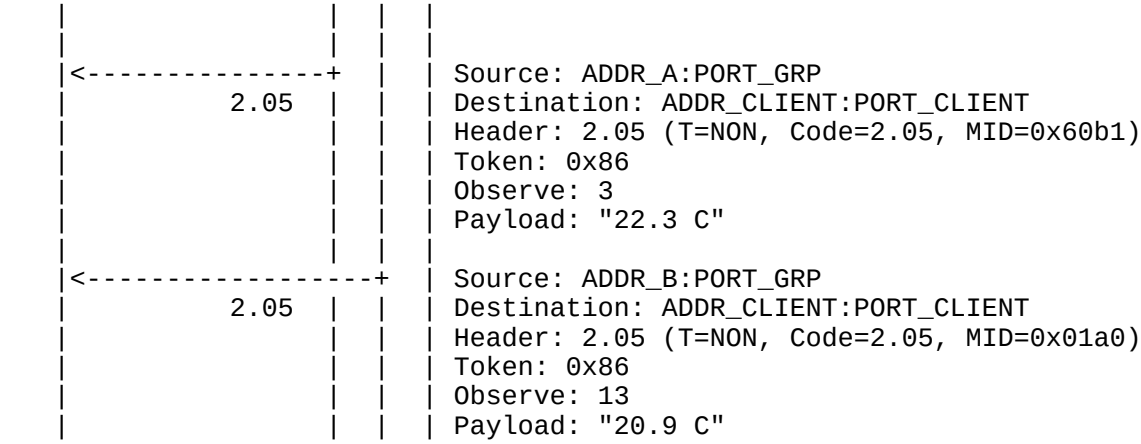
Figure 20: Example of Non-confirmable group request, followed by Non-confirmable Responses

In Figure 21, 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".

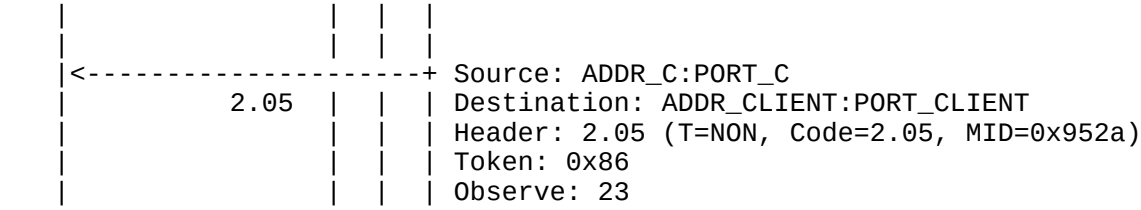


All servers reply with a 2.05 (Content) notification response.

As source port number of their response, servers A and B use the destination port number of the request, i.e., PORT_GRP.



As source port number of its response, server C uses its own port number PORT_C.



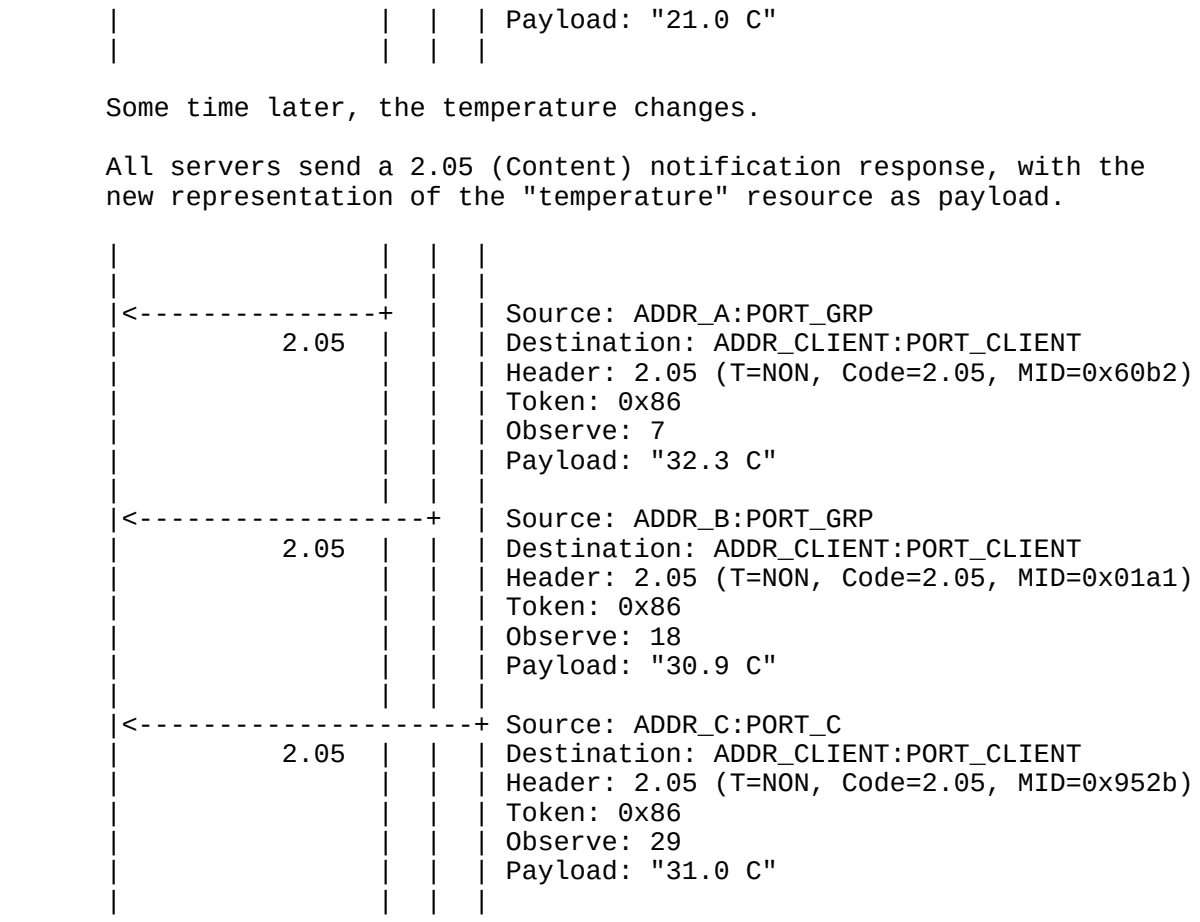
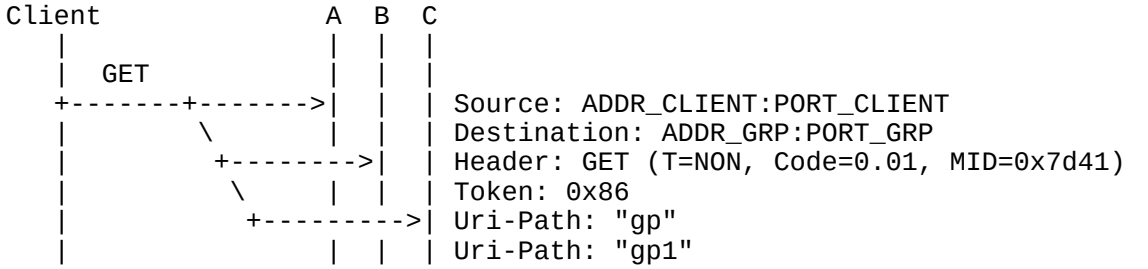
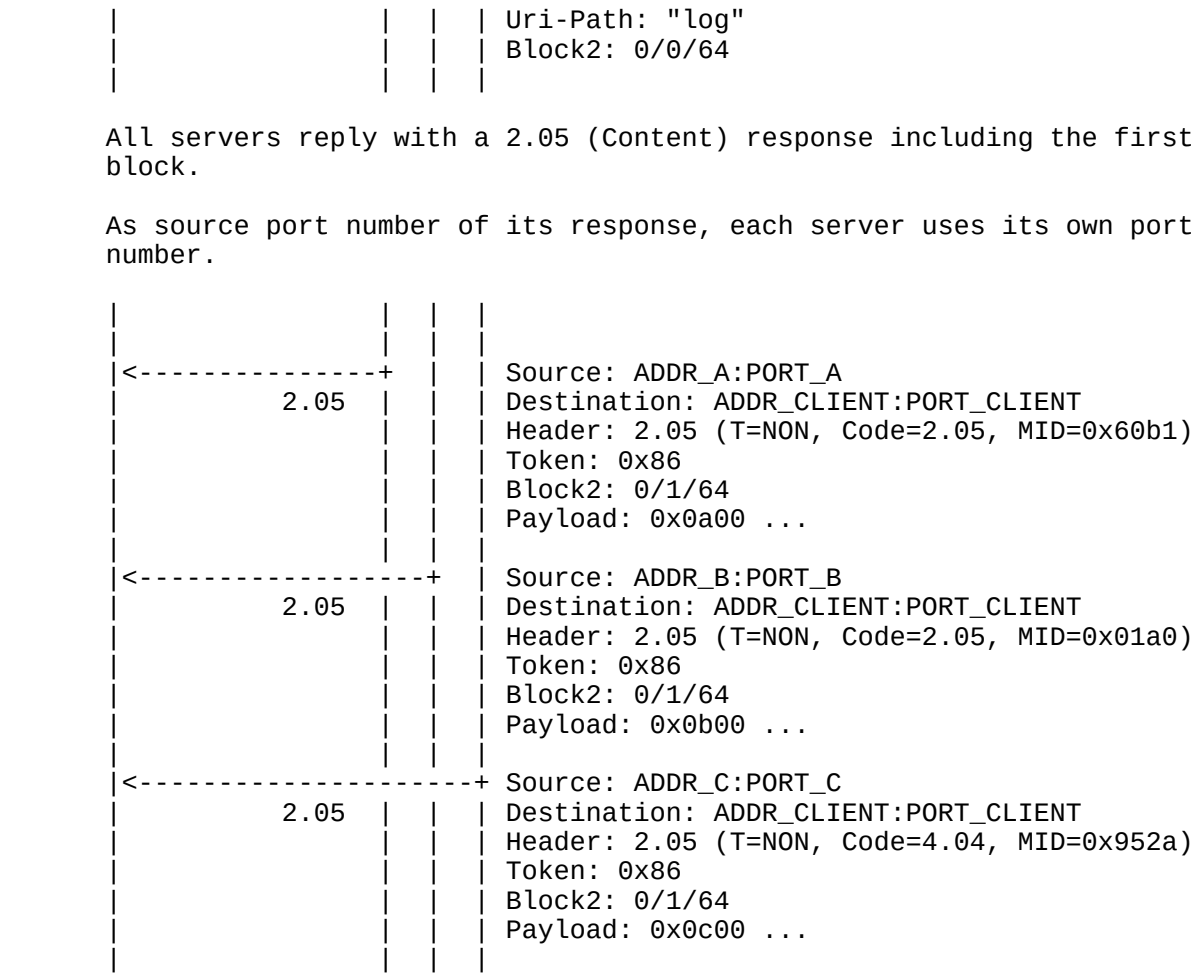


Figure 21: Example of Non-confirmable Observe group request, followed by Non- confirmable Responses as Observe notifications

In Figure 22, 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.



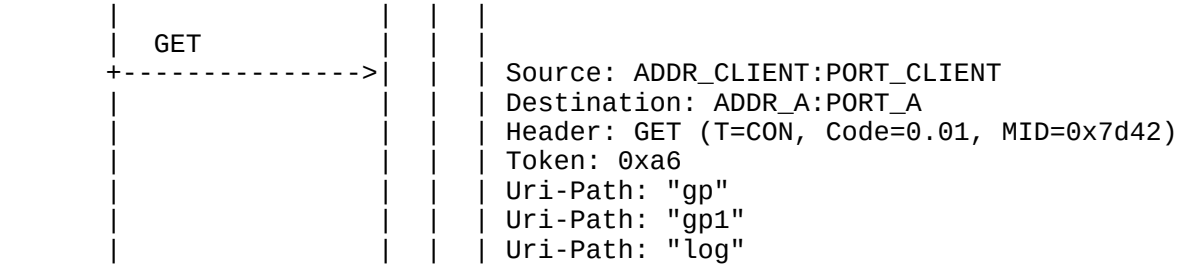


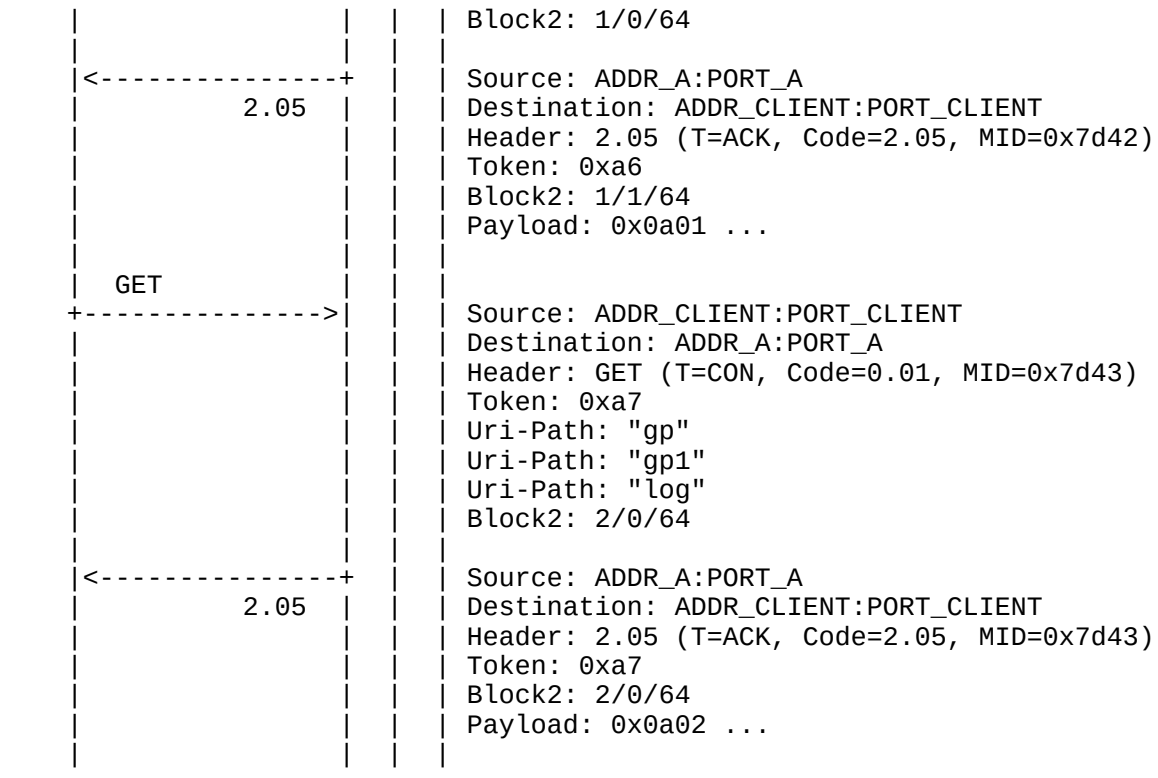
All servers reply with a 2.05 (Content) response including the first block.

As source port number of its response, each server uses its own port number.

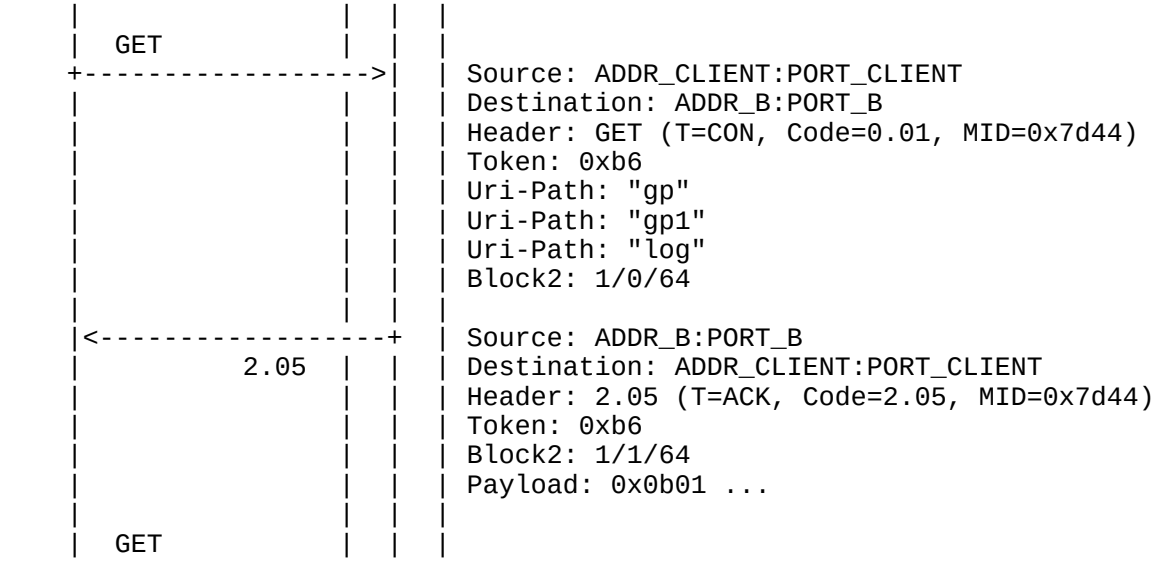
After obtaining the first block, the client requests the following blocks separately from each server, by means of unicast exchanges.

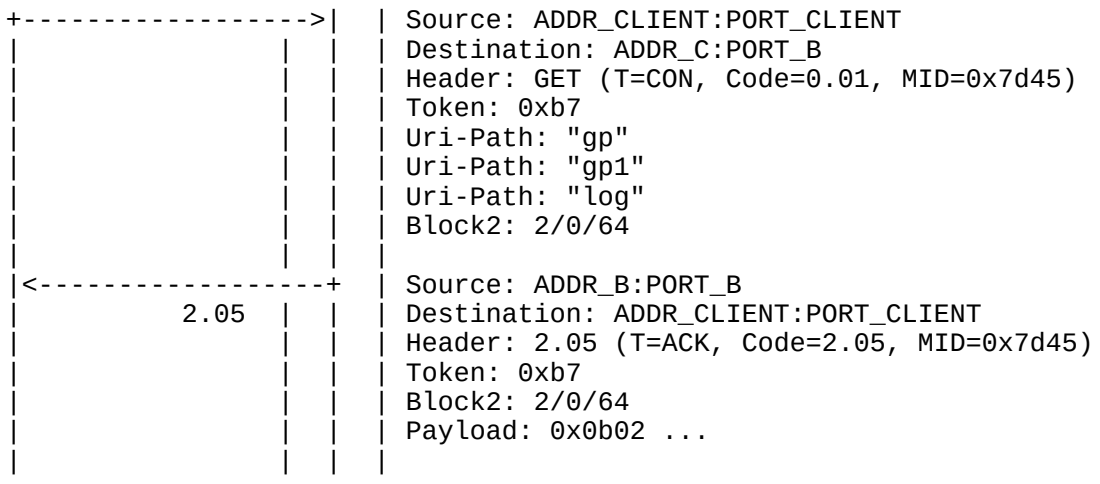
The client requests the following blocks from server A.



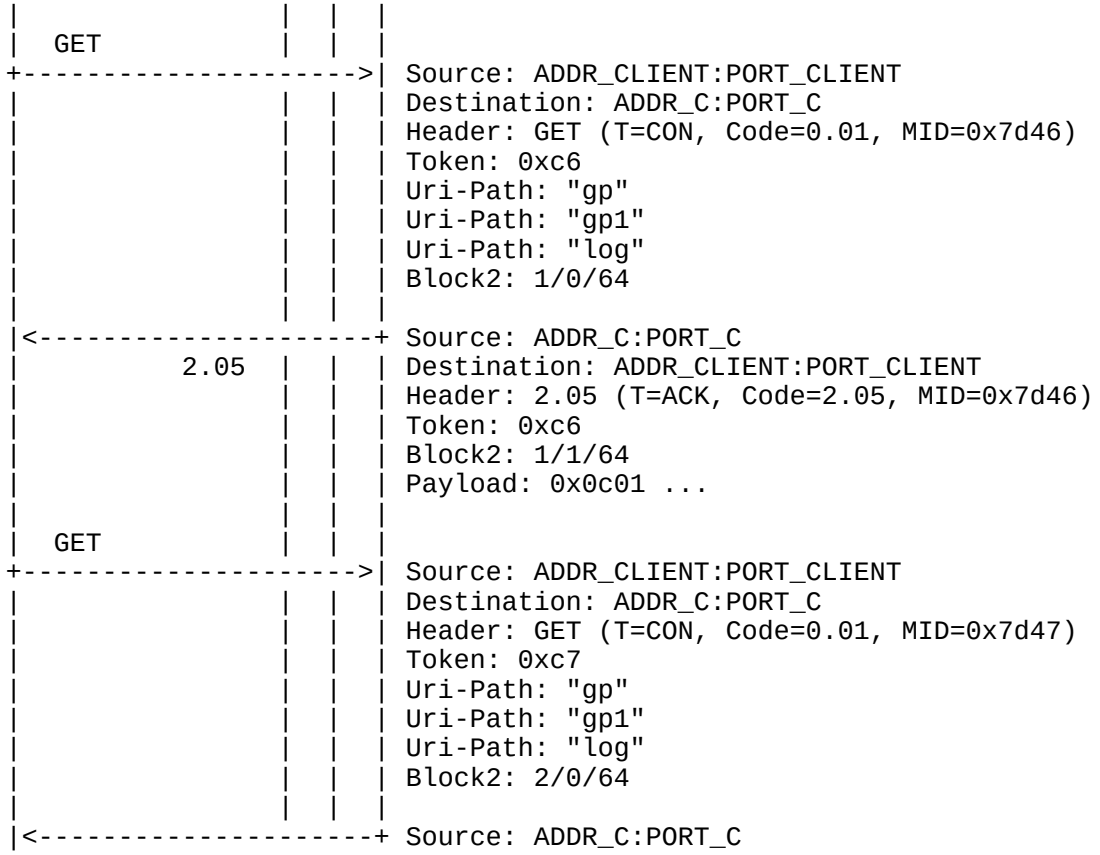


The client requests the following blocks from server B.





The client requests the following blocks from server 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 22: 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 E. Issues and Limitations with Forward-Proxies

Section 3.5.1 defines how a forward-proxy sends (e.g., multicasts) a CoAP group request to the group URI specified in the request from the client. After that, the proxy 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.

* Unlike a CoAP client, the proxy is likely to lack "application context". In particular, the proxy is not expected to know how many members there are in the CoAP group (not even the order of magnitude), how many members of that group will actually respond, or the minimal amount/percentage of those that will respond.

  Therefore, while still capable of forwarding the group request to the CoAP group and the corresponding responses to the client, the proxy does not know and cannot reliably determine for how long to collect responses, before it stops forwarding them to the client.

In principle, a CoAP client that is not using a proxy might face the same problems in collecting responses to a group request. However, unlike a CoAP proxy, the client itself would typically have application-specific rules or knowledge on how to handle this situation. For example, a CoAP client could monitor incoming responses and use this information to decide for how long to continue collecting responses

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].

Finally, Section 3.5.1 refers to [RFC8075] for the operation of HTTP-to-CoAP proxies for multicast CoAP requests. This relies on the "application/http" media type to let the proxy return multiple CoAP responses -- each translated to an 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 may still be able to identify the CoAP servers by other means such as application-specific information in the response payload.

Appendix F. Issues and Limitations with Reverse-Proxies

Section 3.5.2 defines how a reverse-proxy sends a group request to servers in a CoAP group, over a one-to-many transport such as IP/UDP multicast. This results in certain issues and limitations:

* The three issues and limitations defined in Section 3.5.1 for a forward proxy apply to a reverse-proxy as well.

* A reverse-proxy may have preconfigured time duration(s) that are used for collecting server responses and forwarding these back to the client. These duration(s) may be set as global configuration

or as 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.ietf-core-groupcomm-proxy] is not necessarily needed. Note that a reverse-proxy is in an explicit relationship with the origin servers it stands in for. Thus, compared to a forward-proxy, it has a much better basis for determining and configuring such time durations.

* 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.

Appendix G. Document Updates

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

G.1. Version -13 to -14

* More context information in the abstract and introduction.
* Imported examples from RFC 7390 into Appendix A "Use Cases".
* Early mentioning Group OSCORE and source authentication of messages.
* Updated references.
* Minor fixes and editorial improvements.

G.2. Version -12 to -13

* Updated author list.
* Registering Resource Type values is not strictly required.
* Highlighted what is used simply as an example.
* Added further alternative to follow-up with group requests using the Block2 Option.

* Justified importance of possibly repeating requests.

* Explicitly mentioned Observe as a reason for follow-on responses.

* Recommended use of a mitigation against abuses of the No-Response Option.

* Described the use of the Echo option like done in draft-ietf-core-oscore-groupcomm.

* Clarified possible use of CoAP over reliable transports.

* Given more context around prioritized packet forwarding in 6LoWPAN.

* Expanded considerations about replayed group requests.

* Clarifications and editorial improvements.

G.3. Version -11 to -12

* Clarified how a group name can be included in a URI path for reverse proxies; removed reference to URI templates for reverse proxies.

* Clarified the issue of identical ETags from multiple servers and added strategies to cope with this. Removed recommendation to not use such duplicate ETags.

* Simplified section on application group naming using URI authority and corresponding appendix examples; declaring new Option solution out of scope.

* Added application group resource URI path(s) in Figure 1.

* Clarified outcome of the RFC 7390 experiment on group membership configuration protocol.

* Clarified that rt=g.<GROUPTYPE> is used just as an example.

* Made RFC 7967 a normative reference.

* Generalized resending of a group request with different Message ID.

* Switched SHOULD to MAY on changing port number from group request to response.

* Further generalized the handling of multiple responses from the same server to the same request.

* Clarifications on response suppression.

* Issues and limitations with Forward-Proxies compiled in an appendix.

* Issues and limitations with Reverse-Proxies compiled in an appendix.

* Mentioned PROBING_RATE as a means to enforce congestion control.

* Consideration on how eventual consistency from Observe compensates for lost notifications.

* Admitted resource retrieval through consecutive group requests with the Block2 Option.

* Clarified relationship with TCP/TLS/WebSockets.

* Clarified security on the different legs with a proxy.

* Editorial improvements.

G.4. Version -10 to -11

* Updated references.

* Editorial improvements.

G.5. Version -09 to -10

* Clarified use of NoSec for ’early discovery’ and refer to cBRSKI.

* Clarified mandatory vs optional elements in CoAP group and CoAP group URI.

* Replaced "GROUPNAME" with "APPNAME".

* Explicitly mentioning of the type of group (CoAP/application/ security).

* Use of .example for example hostnames.

* The name of a security group is not necessarily a string.

* Changed "has to" to "should" for enforcing access control based on membership to security groups.

* Used normative language for policies about group rekeying.

* Further stressed that group communication ought to be secured.

* Avoid calling applications as "(non-)sensitive" and "(non-)critical".

* Clarification on source authentication, source addresses, and Echo Option.

* Editorial fixes and improvements.

G.6. Version -08 to -09

* Multiple responses in long exchanges are non-traditional responses.

* Updated references.

G.7. Version -07 to -08

* Updated references.

G.8. Version -06 to -07

* Updated list of changes to other documents.

* Added real-life context and clarifications to examples.

* Clarified aliasing of CoAP group names.

* Clarified use of security group names.

* Clarified response suppression.

* Clarified response revalidation.

* Clarified limitations and peculiarities when using proxies.

* Discussed the case of group request sent to multiple proxies at once.

* Discussed limited use of reliable transports with block-wise transfer.

* Revised text on joining CoAP groups and multicast routing.

* Clarified use/avoidance of the CoAP NoSec mode.

* Moved examples of application group naming and group discovery to appendix sections.

* Revised list of references.

* Updated list of implementations supporting group communication.

* Editorial improvements.

G.9. 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.

G.10. Version -04 to -05

* Clarified changes to other documents.

* Clarified relationship 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.

G.11. 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.

G.12. 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.

G.13. Version -01 to -02

* Clarified relationship 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.

G.14. 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.

Acknowledgements

The authors sincerely thank Christian Amsss, Mike Bishop, Carsten Bormann, Thomas Fossati, Rikard Hglund, Jaime Jimnez, John Preu Mattsson, Jim Schaad, and Jon Shallow for their comments and feedback.

The work on this document has been partly supported by the Sweden's Innovation Agency VINNOVA and the Celtic-Next projects CRITISEC and CYPRESS; and by the H2020 projects SIFIS-Home (Grant agreement 952652) and ARCADIAN-IoT (Grant agreement 101020259).

Authors' Addresses

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

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