CoRE Working Group
Internet-Draft
Obsoletes: 7390 (if approved)
Updates: 7252, 7641, 7959 (if approved)
Intended status: Standards Track
Expires: January 9, 2020

E. Dijk
IoTconsultancy.nl
C. Wang
InterDigital
M. Tiloca
RISE AB

July 08, 2019

Group Communication for the Constrained Application Protocol (CoAP) draft-dijk-core-groupcomm-bis-01

Abstract

This document specifies the use of the Constrained Application Protocol (CoAP) for group communication, using UDP/IP multicast as the underlying data transport. The target application area is any group communication use cases in resource-constrained networks. 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. Aspects of operation of using multicast CoAP in combination with CoAP block-wise transfers and CoAP observe are also specified.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2020.

Table of Contents

1. Introduction

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

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

Both unsecured and secured CoAP group communication over UDP/IP multicast 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) [I-D.ietf-core-object-security]. This method provides end-to-end application-layer security protection of CoAP messages, by using CBOR Object Signing and Encryption (COSE) [RFC8152] [RFC7049].

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 solutions that use CoAP over UDP/ multicast (both IPv6 and IPv4) are in scope. There are alternative methods to achieve group communication using CoAP, for example Publish-Subscribe [I-D.ietf-core-coap-pubsub] which uses a central broker server that CoAP clients access via unicast communication. The alternative methods may be usable for the same or similar use cases as are targeted in this document.

All guidelines in [RFC7390] are imported by this document which replaces [RFC7390] in this respect. This document furthermore adds the security solution using Group OSCORE as the default group communication security solution for CoAP, an updated request/response matching rule for multicast CoAP which updates [RFC7252], multicast use of CoAP Observe which updates [RFC7641] and extension of multicast use of CoAP block-wise transfers which updates [RFC7959].

Security solutions for group communication and configuration other than Group OSCORE are not in scope. General principles for secure group configuration are in scope. The experimental group configuration protocol in Section 2.6.2 of [RFC7390] is not in the scope of this document; thus, it remains an experimental protocol. Since application protocols defined on top of CoAP often define their own specific method of group configuration, the experimental protocol of [RFC7390] has not been subject to enough experimentation to warrant a change of this status.

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 [RFC7252] terminology.

2. General Group Communication Operation

The general operation of group communication, applicable for both unsecured and secured operation, is specified in this section by going through the stack from top to bottom. First, group configuration (e.g. group creation and maintenance which are usually done by an application, user or commissioning entity) is considered in Section 2.1. Then the use of CoAP for group communication including support for protocol extensions (block-wise, Observe, PATCH

method) follows in Section 2.2. How CoAP group messages are carried over various transport layers is the subject of Section 2.3. Finally, Section 2.4 covers the interworking of CoAP with other protocols at the layers below it.

2.1. Group Configuration

2.1.1. Group Definition

A CoAP group is defined as a set of CoAP endpoints, where each endpoint is configured to receive CoAP multicast requests that are sent to the group's associated IP multicast address and UDP port. An endpoint may be a member of multiple CoAP groups. Group membership(s) of an endpoint may dynamically change over time. A device sending a CoAP request to a group is not necessarily itself a member of this group: it is only a member if it also has a CoAP server endpoint listening to requests for this CoAP group. For secure group communication, a receiver also requires the security context to decrypt and/or verify group messages in order to be a group member.

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

Besides CoAP groups, that have relevance at the level of networked devices, there can also be application groups defined. 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. There can be a one-to-one or a one-to-many relation between CoAP groups and application groups.

2.1.2. Group Naming

For clients, it is RECOMMENDED to use by default an IP multicast address literal in a configured Group URI, instead of a hostname. This is because DNS infrastructure may not be deployed in many constrained networks. In case a group hostname is used in the Group URI, it can be uniquely mapped to an IP multicast address via DNS resolution - if DNS client functionality is available in the clients and the DNS service is supported in the network. Some examples of hierarchical group FQDN naming (and scoping) for a building control application are shown in Section 2.2 of [RFC7390].

Application groups can be named in many ways, e.g. numbers, IDs, strings or URIs. An application group identifier, if used, is typically included in the path component or query component of a Group URI. Appendix A of [I-D.ietf-core-resource-directory] shows registration of application groups into a Resource Directory, along with the CoAP group it maps to.

2.1.3. Group Creation and Membership

Group membership may be (factory-)preconfigured in devices or dynamically configured in a system on-site.

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 5683. Then, it configures one or more devices as listeners to that IP multicast address, with a CoAP server listening on the specific port. These devices are the group members. The configuring entity can be a local application with preconfiguration, a user, a cloud service, or a local commissioning tool for example. Also, the devices sending requests to the group in the role of CoAP clients 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 CoAP Group URI.

For unsecure group communication, any CoAP endpoint may become a group member at any time: there is no (central) configuring entity that needs to provide the security material for the group. This means that group creation and membership cannot be tightly controlled.

The IETF does not define a mandatory, standardized protocol to accomplish these steps. For secure group communication, the part of the process that involves secure distribution of group keys MAY use standardized communication with a Group Manager as defined in Section 4. [RFC7390] defines an experimental protocol for configuration of group membership for unsecured group communication, based on JSON-formatted configuration resources. This protocol remains experimental.

2.1.4. Group Maintenance

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

For unsecured group communication (see Section 3), addition/removal of group members is simply done by configuring these devices to start/stop listening to the group IP multicast address, and to start/stop the CoAP server listening to the group IP multicast address and port.

For secured group communication (see Section 4), the protocol Group OSCORE [I-D.ietf-core-oscore-groupcomm] is mandatory to implement. When using Group OSCORE, CoAP endpoints participating to group communication are also members of a corresponding OSCORE group, and thus share a common set of cryptographic material. Additional maintenance operations are discussed in Section 4.1.

2.2. CoAP Usage

2.2.1. Request/Response Model

All CoAP requests that are sent via IP multicast MUST be Non-confirmable (Section 8.1 of [RFC7252]). The Message ID in an IP multicast CoAP message is used for optional message deduplication as detailed in Section 4.5 of [RFC7252].

A server MAY send back a unicast response to the CoAP group communication request - whether it does this or not is selected by the server application. The unicast responses received by the CoAP client may be a mixture of success (e.g., 2.05 Content) and failure (e.g., 4.04 Not Found) codes depending on the individual server processing results.

TBD: the CoAP Option for No Server Response [RFC7967] can be used by the client to influence response suppression on the server side. Possibly we can include this information here; it specifically targets use for multicast use cases also.

The client can distinguish the origin of multiple server responses by the source IP address of the UDP message containing the CoAP response or any other available unique identifier (e.g., contained in the CoAP response payload). In case a client has sent multiple group requests with concurrent CoAP transactions ongoing, the responses are matched to a request using the Token value. The source endpoint of the response is not matched to the destination endpoint of the request, since for a multicast request these will never match. This is also specified in Section 8.2 of [RFC7252]. As an update to the [RFC7252] matching rule, a client MAY, in addition to the Token, match the source port of the request to the destination port of the response, since these will match in any correctly formatted CoAP response. This can help a client to more easily meet the below constraint on Token reuse or to more efficiently filter received responses.

For multicast CoAP requests, there are additional constraints on the reuse of Token values, compared to the unicast case. In the unicast case, if the Observe Option [RFC7641] is not used in a request, receiving a response effectively frees up its Token value for reuse since no more responses will follow. However, for multicast CoAP, the number of responses is not bounded a priori. Therefore, the reception of a response cannot be used as a trigger to "free up" a Token value for reuse. Reusing a Token value too early could lead to incorrect response/request matching in the client and would be a protocol error. Therefore, the time between reuse of Token values used in multicast requests MUST be greater than:

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 in [RFC7390], that is: the expected maximum response delay over all servers that the client can send a multicast 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 multicast 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 500 seconds.

2.2.2. Port and URI Path Selection

A CoAP server that is a member of a group listens for CoAP messages on the group's IP multicast address, usually on the CoAP default UDP port 5683, or another non-default UDP port if configured. Regardless of the method for selecting the port number, the same port number MUST be used across all CoAP servers that are members of a group and across all CoAP clients performing the group requests to that group. The URI Path used in the request is preferably a path that is known to be supported across all group members. However there are valid use cases where a request is known to be successful for a subset of the group and those group members for which the request is unsuccessful either ignore the multicast request or respond with an error status code.

Using different ports with the same IP multicast address is an efficient way to create multiple CoAP groups in constrained devices, in case the device's stack only supports a limited number of IP

multicast group memberships. However, it must be taken into account that this incurs additional processing overhead on each CoAP server participating in at least one of these groups: messages to groups that are not of interest to the node are only discarded at the higher transport (UDP) layer instead of directly at the network (IP) layer.

Port 5684 is reserved for DTLS-secured 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 5683 MUST be supported (see Section 7.1 of [RFC7252]) for the "All CoAP Nodes" multicast group. THis implies that the receiving server when correctly operating does not send a "ICMP Destination Unreachable - Port Unreachable" in response to a resource discovery request.

2.2.3. Proxy Operation

CoAP (Section 5.7.2 of [RFC7252]) allows a client to request a forward-proxy to process its CoAP request. For this purpose, the client specifies either the request group URI as a string in the Proxy-URI option or alternatively it uses the Proxy-Scheme option with the group URI constructed from the usual Uri-* options. This approach works well for unicast requests. However, there are certain issues and limitations of processing the (unicast) responses to a CoAP group communication request made in this manner through a proxy.

A proxy may buffer all the individual (unicast) responses to a CoAP group communication request and then send back only a single (aggregated) response to the client. However, there are some issues with this aggregation approach:

- Aggregation of (unicast) responses to a CoAP group communication request in a proxy is difficult. This is because the proxy does not know how many members there are in the group or how many group members will actually respond. Also, the proxy does not know how long to wait before deciding to send back the aggregated response to the client.

- There is no default format defined in CoAP for aggregation of multiple responses into a single response. Such a format could be defined based on the multipart content-format [I-D.ietf-core-multipart-ct] but is not standardized yet currently.

Alternatively, if a proxy does not aggregate responses and follows the CoAP Proxy specification (Section 5.7.2 of [RFC7252]), the proxy would forward all the individual (unicast) responses to a CoAP group

communication request to the client. There are also issues with this approach:

- The client may be confused as it may not have known that the Proxy-URI contained a group URI target. That is, the client that sent a unicast CoAP request to the proxy may be expecting only one (unicast) response. Instead, it receives multiple (unicast) responses, potentially leading to fault conditions in the application.

- Each individual CoAP response 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.

Due to the above issues, a CoAP Proxy SHOULD NOT support processing an IP multicast CoAP request but rather return a 501 (Not Implemented) response in such case. The exception case here (i.e., to support it) is when all the following conditions are met:

- The CoAP Proxy MUST be explicitly configured (whitelist) to allow proxied IP multicast requests by specific client(s).

- The proxy SHOULD return individual (unicast) CoAP responses to the client (i.e., not aggregated). If a (future) standardized aggregation format is being used, then aggregated responses may be sent.

- It MUST be known to the person/entity doing the configuration of the proxy, or otherwise verified in some way, that the client configured in the whitelist supports receiving multiple responses to a proxied unicast CoAP request.

The operation of HTTP-to-CoAP proxies for multicast CoAP requests is specified in Section 8.4 and 10.1 of [RFC8075]. In this case, the "application/http" media type can be used to let the proxy return multiple CoAP responses - each translated to a HTTP response - back to the HTTP client. Of course the HTTP client in this case needs to be aware that it is receiving this format and needs to be able to decode from it the responses of multiple servers. The above restrictions listed for CoAP Proxies still apply to HTTP-to-CoAP proxies: specifically, the IP address of the sender of each CoAP response cannot be determined from the application/http response.

2.2.4. Congestion Control

CoAP group communication requests may result in a multitude of responses from different nodes, potentially causing congestion. Therefore, both the sending of IP multicast requests and the sending of the unicast CoAP responses to these multicast 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's nothing useful to respond to (e.g., error or empty response); see Section 8.2 of [RFC7252].

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

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

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

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

Additional guidelines to reduce congestion risks defined in this document are as follows:

- A server in an LLN should only support group communication GET for resources that are small. For example, the payload of the response is limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size, so it fits into a single link-layer frame in case IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) (see Section 4 of [RFC4944]) is used.

- A server SHOULD minimize the payload length in response to a multicast GET 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 length of a response to a multicast GET (e.g., on "/.well-known/core") 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 an IP multicast CoAP request to "/.well-known/core" SHOULD support block-wise transfers.

- o A client SHOULD use CoAP group communication 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.

2.2.5. Observing Resources

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

This section updates [RFC7641] with the use of the Observe Option in a CoAP multicast GET request. This is a useful way to start observing a particular resource on all members of a (multicast) group at the same time. Group members that do not have this resource or do not allow the GET method on it will either respond with an error status - 4.04 Not Found or 4.05 Method Not Allowed respectively - or will silently ignore the request, depending on server-specific configuration.

A client that sends a multicast GET request with the Observe Option MAY repeat this request using the same Token value and same Observe Option value, in order to ensure that enough (or all) group members have been reached with the request. This is useful in case a number of group members did not respond to the initial request. This client MAY also use the same Message ID to avoid that group members that had already received the initial request would respond again. If the client uses a different, fresh Message ID then all group members that receive this new message will respond again.

A client that sends a multicast GET request with the Observe Option MAY send a new unicast request with the same Token value and same Observe Option value, in order to ensure that the specific server receives the request. This is useful in case a specific group member, that was expected to respond to the initial group request, did not respond to the initial request. The client in this case always uses a Message ID that differs from the initial message.

In the above client behaviors, the Token value is kept identical to the initial request to avoid that the client is included in more than one entry in the list of observers (Section 4.1 of [RFC7641]). While a Token value is in use for observing a group, this Token value cannot be reused by the same client endpoint for other purposes. Another endpoint on the client i.e. using a different UDP source port MAY re-use the Token value but only if the client implements the optional updated matching rule of Section 2.2.1.

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 allow the server the time to respond.

For observing a group of servers through a CoAP-to-CoAP proxy or HTTP-CoAP proxy, the limitations stated in Section 2.2.3 apply.

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

TBD: below solution for multicast block-wise Block1 is used e.g. for efficiently distributing large data/software updates using multicast. It is non-trivial to do right and needs testing. For this reason, we may decide to move this into a separate draft.

This section specifies in addition the use of CoAP block-wise transfers for multicast PUT/POST/PATCH/iPATCH requests in order to efficiently distribute a large request payload as multiple blocks to all members of a CoAP group. The Block1 Option [RFC7959] is then used by the client in each block-wise request and a server uses the Block1 Option in its response to confirm reception of a block and optionally to indicate in the first block-wise response that it prefers a smaller block size.

Prior to starting a block-wise multicast request, the client SHOULD already store a list of those members of the CoAP group that need to properly receive the request payload. These members are expected to support block-wise CoAP and are also expected to support the specific resource to which the request will be sent. Obtaining such list can

be achieved in various ways such as by group configuration, and/or CoAP discovery, and/or first sending one or more non-block-wise multicast requests to the same group and collect the responses.

The reason that the client should be aware of these group members is the following: after sending the first block (0), the client SHOULD first collect all group member responses to the first block before proceeding with further blocks. One or more of the group members MAY indicate a preference for a smaller block size in the Block1 Option in its first response. The client SHOULD use the smallest value over all collected responses as the block size to use for the remaining block-wise messages.

Since not all group member responses may be received, due to message loss, the client MAY resend the multicast request (with the same Message ID and Token) to collect the missing responses, or it MAY resend the block 0 request as a Confirmable or Non-Confirmable unicast request (with the same Message ID and Token) directly to the non-responsive group member(s), or it uses a combination of these. The reason to use the same Message ID here is to avoid that a group member server processes the request more than once.

TBD: open point - the server needs to treat a unicast message (with token T and MID M) as a duplicate of a prior multicast message (with token T and MID M). The deduplication rules allow this; however to be checked if a practical implementation also allows this?

TBD: open point - the time that the process takes to collect all "missing" responses for the first block (0), might take longer than the "operation timeout time" of the entire blockwise request per [RFC7959]. So for this case, the operation timeout time needs to be set longer than usual, or alternatively, the stateless-server mode of update needs to be mandated. In this case each block that is written produces a 2.04 not 2.31. First block with PUT may respond a 2.01.

TBD: if strict order of blocks is required by a server, the protocol must wait and collect again all responses after each block.

TBD: a protocol may be more efficient that first sends all blocks (without waiting for all responses every step) and then later checks which blocks are missing with all servers individually. These can be resent then (in unicast or multicast if many servers miss that block).

2.3. Transport

TBD: Mark [RFC8323] (TCP, TLS, WebSockets) as not applicable for this form of groupcomm, as well as CoAP-over-SMS.

2.3.1. UDP/IPv6 Multicast Transport

TBD: include the "Exceptions" cases here of RFC 7390 Section 2.10. State that IPv6 multicast is prerequisite. Also mention the All-CoAP-nodes IPv6 addresses.

2.3.2. UDP/IPv4 Multicast Transport

TBD: includes the "Exceptions" cases here of RFC 7390 2.10. State that IPv4 multicast is prerequisite. mention All-CoAP-nodes IPv4 addresses and the like

2.3.3. 6LoWPAN

TBD: 6lowpan-specific considerations to go here. Specifically, a multicast request should preferably fit in one L2 frame to avoid the strong performance drop that comes with 6LoWPAN-fragmentation and reassembly. Also reference [RFC7346] for the realm-local scope.

2.4. Interworking with Other Protocols

2.4.1. MLD/MLDv2/IGMP

TBD: see Section 4.2 of [RFC7390] and include the content here or refer to it.

2.4.2. RPL

TBD: see Section 4.3 of [RFC7390] and include the content here or refer to it.

2.4.3. MPL

TBD: see Section 4.4. [RFC7390] and include the content here or refer to it.

3. Unsecured Group Communication

CoAP group communication can operate in CoAP NoSec (No Security) mode, without using application-layer and transport-layer security mechanisms. The NoSec mode uses the "coap" scheme, and is defined in Section 9 of [RFC7252]. Before using this mode of operation, the security implications (Section 5.1) must be well understood.

4. Secured Group Communication using Group OSCORE

The application-layer protocol Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] 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 [RFC8152] to perform encryption, signing and Message Authentication Code operations, and to efficiently encode the result as a COSE object. In particular, OSCORE takes as input an unprotected CoAP message and transforms it into a protected CoAP message, by using Authenticated Encryption Algorithms with Additional Data (AEAD).

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

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

In particular, Group OSCORE protects CoAP requests sent over IP multicast by a CoAP client, as well as multiple corresponding CoAP responses sent over IP unicast by different CoAP servers. However, the same keying 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 uses digital signatures to ensure source authentication of all messages exchanged within the OSCORE group. That is, sender devices sign their outgoing messages by means of their own private key, and embed the signature in the protected CoAP message.

A Group Manager is responsible for one or multiple OSCORE groups. In particular, the Group Manager acts as repository of public keys of

group members; manages, renews and provides keying material in the group; and drives the join process for new group members.

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

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

If security is required, CoAP group communication as described in this specification MUST use Group OSCORE. In particular, a CoAP group as defined in Section 2.1.1 and using secure group communication is associated to an OSCORE group, which includes:

- All members of the CoAP group, i.e. the CoAP endpoints configured (also) as CoAP servers and listening to the group's multicast IP address.

- All further CoAP endpoints configured only as CoAP clients, that send (multicast) CoAP requests to the CoAP group.

4.1. Secure Group Maintenance

Additional key management operations on the OSCORE group are required, depending also on the security requirements of the application (see Section 5.2). That is:

- Adding new members to a CoAP group or enabling new client-only endpoints to interact with that group require also that each of such members/endpoints join the corresponding OSCORE group. By doing so, they are securely provided with the necessary cryptographic material. In case backward security is needed, this also requires to first renew such material and distribute it to the current members/endpoints, before new ones are added and join the OSCORE group.

- In case forward security is needed, 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 cryptographic material is generated and securely distributed only to the remaining members/endpoints. This ensures that only the members/endpoints intended to remain are able to continue participating to

secure group communication, while the evicted ones are not able to.

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 according to the approach described in [I-D.ietf-ace-key-groupcomm-oscore].

5. Security Considerations

This section provides security considerations for CoAP group communication using IP multicast.

5.1. CoAP NoSec Mode

CoAP group communication, if not protected, is vulnerable to all the attacks mentioned in Section 11 of [RFC7252] for IP multicast.

Thus, for sensitive and mission-critical applications (e.g., health monitoring systems and alarm monitoring systems), it is NOT RECOMMENDED to deploy CoAP group communication in NoSec mode.

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

Discovery of devices and resources is a typical use case where NoSec mode is applied, since the devices involved do not have yet configured any mutual security relations at the time the discovery takes place.

5.2. Group OSCORE

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

For sensitive and mission-critical applications, CoAP group communication MUST be protected by using Group OSCORE as specified in

[I-D.ietf-core-oscore-groupcomm]. The same security considerations from Section 8 of [I-D.ietf-core-oscore-groupcomm] hold for this specification.

5.2.1. Group Key Management

A key management scheme for secure revocation and renewal of group keying material, namely group rekeying, should be adopted in OSCORE groups. In particular, the key management scheme should preserve backward and forward security in the OSCORE group, if the application requires so (see Section 2.1 of [I-D.ietf-core-oscore-groupcomm]).

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

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

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

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

5.2.2. Source Authentication

CoAP endpoints using Group OSCORE countersign their outgoing messages, by means of the countersignature algorithm used in the OSCORE group. This ensures source authentication of messages exchanged by CoAP endpoints through CoAP group communication. In fact, it allows to verify that a received message has actually been originated by a specific and identified member of the OSCORE group.

Appendix F of [I-D.ietf-core-oscore-groupcomm] discusses a number of cases where a recipient CoAP endpoint may skip the verification of countersignatures, possibly on a per-message basis. However, this is NOT RECOMMENDED. That is, a CoAP endpoint receiving a message secured with Group OSCORE SHOULD always verify the countersignature.

5.2.3. Counteraction of Attacks

Group OSCORE addresses security attacks mentioned in Sections 11.2-11.6 of [RFC7252], with particular reference to their execution over IP multicast. That is: it provides confidentiality and integrity of request/response data through proxies also in multicast settings; it prevents amplification attacks carried out through responses to injected requests over IP multicast; it limits the impact of attacks based on IP spoofing; it prevents cross-protocol attacks; it derives the group key material from, among other things, a Master Secret securely generated by the Group Manager and provided to CoAP endpoints upon their joining of the OSCORE group; countersignatures assure source authentication of exchanged CoAP messages, and hence prevent a group member to be used for subverting security in the whole group.

5.3. 6LoWPAN

Editor Note, TBD: identify if multi-fragment multicast requests have a negative effect on security and, if so, advice here on trying to avoid such requests. Also an attacker could use multi-fragment to occupy reassembly buffers of many routing 6LoWPAN nodes.

5.4. Wi-Fi

TBD: Wi-Fi specific security considerations; see also Section 5.3.1 of [RFC7390].

5.5. Monitoring

TBD: see Section 5.4 of [RFC7390].

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. Normative References

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-16 (work in progress), March 2019.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-04 (work in progress), March 2019.

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

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

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

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

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

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

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

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

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

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

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

7.2. Informative References

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

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

[I-D.ietf-ace-key-groupcomm-oscore] Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-01 (work in progress), March 2019.

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-08 (work in progress), March 2019.

[I-D.ietf-core-multipart-ct]
Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for CoAP", draft-ietf-core-multipart-ct-03 (work in progress), March 2019.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-22 (work in progress), July 2019.

[I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-02 (work in progress), March 2019.

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

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

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

Appendix A. Use Cases

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

A.1. Discovery

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

A.1.1. Distributed Device Discovery

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

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

A.1.2. Distributed Service Discovery

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

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

A.1.3. Directory Discovery

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

A.2. Operational Phase

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

A.2.1. Actuator Group Control

Group communication can be beneficial to control actuators that need to act in synchrony, as a group, with strict timing (latency) requirements. Examples are office lighting, stage lighting, street lighting, or audio alert/Public Address systems. Sections 3.4 and 3.5 of [RFC7390] show examples of lighting control of a group of 6LoWPAN-connected lights.

A.2.2. Device Group Status Request

To properly monitor the status of systems, there may be a need for ad-hoc, unplanned status updates. Group communication can be used to quickly send out a request to a (potentially large) number of devices for specific information. Each device then responds back with the requested data. Those devices that did not respond to the request can optionally be polled again via reliable unicast communication to complete the dataset. The device group may be defined e.g. as "all

temperature sensors on floor 3", or "all lights in wing B". For example, it could be a status request for device temperature, most recent sensor event detected, firmware version, network load, and/or battery level.

A.2.3. Network-wide Query

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

A.2.4. Network-wide / Group Notification

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

A.3. Software Update

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

Acknowledgments

The authors sincerely thank Thomas Fossati and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC.

Authors' Addresses

Esko Dijk
IoTconsultancy.nl

Utrecht
The Netherlands

Email: esko.dijk@iotconsultancy.nl

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

Email: Chonggang.Wang@InterDigital.com

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

Email: marco.tiloca@ri.se