

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2014

C. Bormann
Universitaet Bremen TZI
October 22, 2013

CoRE Roadmap and Implementation Guide
draft-bormann-core-roadmap-05

Abstract

The CoRE set of protocols, in particular the CoAP protocol, is defined in draft-ietf-core-coap in conjunction with a number of specifications that are currently nearing completion. There are also several dozen more individual Internet-Drafts in various states of development, with various levels of WG review and interest.

Today, this is simply a bewildering array of documents. Beyond the main four documents, it is hard to find relevant information and assess the status of proposals. At the level of Internet-Drafts, the IETF has only adoption as a WG document to assign status - too crude an instrument to assess the level of development and standing for anyone who does not follow the daily proceedings of the WG.

With a more long-term perspective, as additional drafts mature and existing specifications enter various levels of spec maintenance, the entirety of these specifications may become harder to understand, pose specific implementation problems, or be simply inconsistent.

The present guide aims to provide a roadmap to these documents as well as provide specific advice how to use these specifications in combination. In certain cases, it may provide clarifications or even corrections to the specifications referenced.

This guide is intended as a continued work-in-progress, i.e. a long-lived Internet-Draft, to be updated whenever new information becomes available and new consensus on how to handle issues is formed. Similar to the ROHC implementation guide, RFC 4815, it might be published as an RFC at some future time later in the acceptance curve of the specifications.

This document does not describe a new protocol or attempt to set a new standard of any kind - it mostly describes good practice in using the existing specifications, but it may also document emerging consensus where a correction needs to be made.

(TODO: The present version does not completely cover the new Internet-Drafts submitted concurrently with it; it is to be updated by the start of IETF88.)

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 <http://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 April 25, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. The Main Four	3
2.1. The CoAP protocol	4
2.2. Discovery	5
2.3. Further reading	6
3. Informational Drafts	6
3.1. Implementation	6
3.2. Multicast and Group Communication	7
3.3. Security	8

3.4. Intermediaries	9
3.5. Congestion Control	9
4. CoAP over X	9
5. Optional components of CoRE	10
5.1. CoAP-misc	10
5.2. Generalizing Media Types	11
5.3. Patience, Leisure, Pledge, or: Timing extensions	11
5.4. Extending Observe	11
5.5. Service discovery	11
5.6. Server discovery, Naming, etc.	12
5.7. More support for sleepy nodes	12
6. Replaced drafts	14
7. IANA Considerations	15
8. Security Considerations	15
9. Acknowledgements	15
10. References	15
10.1. Normative References	15
10.2. Informative References	16
Author's Address	22

1. Introduction

(To be written - for now please see the Abstract.)

1.1. Terminology

This document is a guide. However, it might evolve to make specific recommendations on how to use standards-track specifications. Therefore: The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. They indicate requirement levels for compliant CoRE implementations [RFC2119]. Note that these keywords are not only used where a correction or clarification is intended; the latter are explicitly identified as such.

The term "byte" is used in its now customary sense as a synonym for "octet".

2. The Main Four

The main component of the CoRE architecture is the Constrained Application Protocol (CoAP). It aims to provide a RESTful transfer service, not unlike HTTP, but radically simplified for the use on constrained devices on constrained networks. REST is the architectural style that informed the design of HTTP [REST]. The terms "constrained device" and "constrained network" refer to limited-capability devices such as sensors operating on networks such

as the IEEE 802.15.4 based 6LoWPAN [RFC4919].

[I-D.ietf-lwig-terminology] provides a more detailed discussion of what we mean by these terms.

2.1. The CoAP protocol

The CoAP protocol is defined in three specifications:

- o [I-D.ietf-core-coap]
- o [I-D.ietf-core-block]
- o [I-D.ietf-core-observe]

The first specification, [I-D.ietf-core-coap], provides the core transfer protocol, including the means to provide communication security using the DTLS protocol [RFC6347] (compare this to the way [RFC2616] and [RFC2818] define HTTP and HTTPS). The protocol is structured into a message layer, which provides duplicate detection and optional message reliability on top of UDP, and a request/response layer, which provides the usual REST operations GET, PUT, POST, and DELETE. A highly efficient protocol encoding carries the 4-byte base header, a sequence of `_Options_`, and the payload (body) of a message. The main extension points of CoAP are its Options, similar to the way new header fields are used to extend HTTP.

Since CoAP is a very simple protocol running on top of UDP, it is limited in its transfer size by the datagram sizes provided by UDP. As a further constraint, many constrained networks do not provide good reliability of delivery once their small frame sizes are exceeded and the adaptation layer is forced to fragment [WEI]. This may lead to a practical limitation to payload sizes as small as 64 bytes. [I-D.ietf-core-block] extends the base CoAP protocol with three options that enable `_blockwise_` transfer, i.e., splitting up a larger transfer into a sequence of smaller transactions, as well as the early determination of the overall size of the resource representation.

In HTTP, transactions are always client initiated, and it is the responsibility of the client to perform GET operations again and again (polling) if it wants to stay up to date about the status of a resource. This "pull model" becomes expensive in an environment with limited power, limited network resources, and nodes that sleep most of the time. Some more or less savory workarounds have been developed for HTTP [RFC6202], but, as a new protocol, CoAP can do better. [I-D.ietf-core-observe] extends the base CoAP protocol with an option that a client can use to indicate its interest in further updates from a resource. If the server accepts this option, the

client becomes an `_Observer_` of this resource and receives an asynchronous notification message each time it changes. Each such notification message is identical in structure to the response to the initial GET request.

While the "Block" and "Observe" specifications are optional additions to the CoAP protocol (just as the core specification already defines 14 options most of which will not need to be used in every message), they together form what is now generally considered to be the CoAP protocol.

The CoRE Working Group has completed its work on the base CoAP protocol specification [I-D.ietf-core-coap] and it has been approved by the IESG for publication as a Standards-Track RFC on 2013-07-15. The completed document is currently waiting in the RFC editor queue for two of its normative references in the security area, [I-D.mcgregw-tls-aes-ccm-ecc] and [I-D.ietf-tls-oob-pubkey], to be completed and approved.

The other two CoAP specifications are, at the time of this writing, in the process of being updated based on the comments to the first Working-Group Last-Call [RFC2418], and in the second Working-Group Last-Call, respectively; these are prerequisites to submitting them to the IESG for publication as a Standards-Track RFC.

The specifications, together with link-format (below), have been widely implemented in highly interoperable implementations: an ETSI "plugtest" event in March 2012 was attended by 15 organizations with 20 implementations; in over 3000 tests performed only about 6 % failed; a second plugtest was conducted in November 2012 and led to some final adjustments of some details in the specifications. Another plugtest is planned for November 2013 [COAP3].

2.2. Discovery

The fourth specification in the main set now nearing completion does not extend the CoAP protocol but addresses a different problem.

In the Web, a number of methods for discovery of resources are common. Initially, Web discovery was just performed by humans based on an entry resource to a server (e.g., `/index.html`). This resource then includes links that directly or indirectly allow a human to reach the other Web resources that make up the Web site.

Web discovery can be performed by machines if standardized interfaces and resource descriptions are available. Among the component mechanisms for Web discovery that are standardized in the IETF are the well-known resource path `"/.well-known/..."` [RFC5785] and the

HTTP link header [RFC5988]. Several related techniques are in common use today.

Clearly, in the machine-to-machine environments that will be typical of CoAP applications, it is important to enable devices to discover each other and their resources. Autonomous devices and embedded systems necessitate uniform, interoperable resource discovery.

A basic component for this is provided by a standardized description format for the resources a server provides, the `_link-format_`. Unless other methods of discovery are available, CoAP servers should provide such a description via the well-known URI `"/.well-known/core"`, available for access via a GET request on that URI. (More advanced resource discovery schemes might make the same description available by other means, e.g. by posting it to a resource directory.)

The description format has been adapted from the format used in the HTTP link header [RFC5988], which is simple and easy to parse. In contrast to the HTTP specification, link-format is specified as an Internet media type (what used to be called "MIME type") and intended to be carried around in the payload [RFC6690].

[RFC6690] was the first RFC of the CoRE working group.

2.3. Further reading

A recent article provides a more detailed overview over the CoRE documents nearing completion [SB].

While the specification documents themselves have to go into meticulous details on every aspect of their protocols, they are the ultimate reference source and are the recommended reading if this basic overview is not sufficient.

3. Informational Drafts

3.1. Implementation

In the IETF, a separate working group is working on informational documents concerning guidance in lightweight implementation of protocols, the LWIG working group. LWIG has several drafts pertinent here:

[I-D.ietf-lwig-terminology] provides some common terms that are useful for discussing implementations and specification in the constrained node network space. Section 2 and 3 of this document are quite stable at this time; a new section 4 is in preparation that

will include discussion of power-related terminology.

[I-D.ietf-lwig-cellular] provides a well-founded discussion of methods for power conservation in CoAP nodes connected via cellular networks, from which some of the material will be used.

[I-D.ietf-lwig-guidance] was originally intended as the main working document of the WG. It contains some discussion about CoAP implementation in its section 3.4.2, including the efficient representation of managing duplicate detection state.

[I-D.kovatsch-lwig-class1-coap] contains additional considerations that, over time, might move into [I-D.ietf-lwig-guidance].

[I-D.castellani-lwig-coap-separate-responses] contains some examples for message exchanges, focusing on elaborating exchanges involving separate responses. Since IETF86, work is under way to merge the CoAP-related information from these three drafts into a new document, [I-D.kovatsch-lwig-coap].

A new working group has been established in the IETF Security Area to address the use of DTLS In Constrained Environments (DICE); several drafts are available for discussion at IETF88 in Vancouver. On the implementation side, two drafts show how to build minimal implementations of security protocols relevant for CoAP:

[I-D.ietf-lwig-tls-minimal] for TLS, which is relevant for CoAP's use of DTLS; and [I-D.ietf-lwig-ikev2-minimal] for IKEv2, the protocol for setting up IPsec security associations. Similarly, [I-D.hartke-core-codtls] looks specifically into the use of DTLS in constrained networks. It raises issues that pertain both to the LWIG and CoRE working groups of the IETF.

Further drafts submitted to LWIG address energy efficient implementation [I-D.hex-lwig-energy-efficient] and recent developments in operating systems for constrained devices [I-D.hahm-lwig-painless-constrained-programming].

After a somewhat slow start, LWIG is now picking up considerable energy.

3.2. Multicast and Group Communication

As it is based on UDP, CoAP easily supports the use of IP multicast to confer messages. However, there are difficult issues around making the desirable multicast applications actually work well.

This led to an additional milestone on the CoRE charter:

Nov 2012: Using CoAP for group communications to IESG as Informational

The informational WG draft [I-D.ietf-core-groupcomm] discusses fundamentals and use cases for group communication with CoAP. This is now very close to Working Group last call.

[I-D.dijk-core-groupcomm-misc] gives some additional considerations, listing requirements, providing some taxonomy, proposing deployment guidelines, and discussing approaches that are not (yet?) in the focus of the WG. Its section 5 can serve as an overview over the status of multicast in constrained node/networks.

3.3. Security

Several individual drafts analyze the issues around the security of constrained devices in constrained networks.

[I-D.garcia-core-security] in particular describes the "Thing Lifecycle" and discusses resulting architectural considerations.

[I-D.sarikaya-core-secure-bootsolution] documents the approach taken in the ZigBee IP specification (used in Smart Energy Profile 2.0); the CoRE WG currently is not working on replicating this specification as an IETF document.

[I-D.jennings-core-transitive-trust-enrollment] demonstrates a specific approach to securing the Thing Lifecycle based on defined roles of security players, including a Manufacturer, an Introducer, and a Transfer Agent. There is considerable interest in the CoRE working group to complete one or more specifications in this space.

Further work around Thing Lifecycles was expected to occur in the SOLACE initiative (Smart Object Lifecycle Architecture for Constrained Environments), with its early mailing list at solace@ietf.org -- developed after the model of the COMAN initiative (Management for Constrained Management Networks and Devices, coman@ietf.org, [I-D.ersue-constrained-mgmt]).

Besides [I-D.garcia-core-security], recently, more work has been focused on the Authentication and Authorization aspects of CoRE:

- o [I-D.gerdes-core-dcaf-authorize]
- o [I-D.greevenbosch-core-authreq]
- o [I-D.pporamba-dtls-certkey]
- o [I-D.urien-core-racs]
- o [I-D.schmitt-two-way-authentication-for-iot]

- o [I-D.seitz-core-sec-usecases]
- o [I-D.selander-core-access-control]
- o [I-D.zhu-core-groupauth]

3.4. Intermediaries

[I-D.castellani-core-http-mapping] discusses some ideas about what HTTP/CoAP intermediaries could do beyond the basic mapping defined in [I-D.ietf-core-coap]; in the IETF86 WG meeting, this document was agreed as a future working group item (with validation of the adoption on the mailing list still pending). An earlier version of this draft was split into the current document describing best practices for mapping between HTTP and CoAP (beyond what is already described in [I-D.ietf-core-coap]), and one additional document that describes usages that serve as additional useful examples for more advanced forms of mapping, a first draft of the latter is available in [I-D.castellani-core-advanced-http-mapping].

3.5. Congestion Control

[I-D.ietf-core-coap] only defines a very basic congestion control scheme that is focused on being safe in a wide variety of applications. Additional documents will define more advanced congestion control schemes that can provide more optimized performance in exchange for more implementation complexity and/or a narrower field of application.

Several drafts are contributing to this active subject of discussion in the WG:

draft-bormann-core-congestion-control	-02	2012-08-01	
draft-bormann-core-cocoa	-00	2012-08-13	

[I-D.greevenbosch-core-minimum-request-interval] proposes adding an option that allows a server to indicate its desire for some pacing of the requests sent to it by one client; enabling a form of server load control.

4. CoAP over X

[I-D.becker-core-coap-sms-gprs] shows how to run CoAP over cellular SMS and in mixed SMS/GPRS environments. This draft optionally makes use of an SMS-oriented encoding for CoAP that is described in [I-D.bormann-coap-misc]. [I-D.silverajan-core-coap-alternative-transports] discusses how to indicate the alternative transport in a URI.

[I-D.li-core-coap-payload-length-option] defines a way to indicate the length of the payload in case the underlying transport does not provide a suitable definite length indication.

5. Optional components of CoRE

Additional sub-protocols are being discussed in the IETF that may become optional protocols in CoREs.

The present document will track these sub-protocols and be amended once the sub-protocols reach formal status in the IETF.

Since the WG is cautious in adopting additional work while the main specifications near completion, none of the additional protocols proposed have become WG documents yet.

5.1. CoAP-misc

One draft is a little different from the other drafts in this category: [I-D.bormann-coap-misc] is a running document capturing CoAP extensions that are in various states of being cooked.

Some of these extensions may finally be adopted for the WG documents and then vanish from CoAP-misc. For other extensions, we may decide that they are not very good ideas. Instead of deleting them from CoAP-misc, they are moved to an appendix. This documents the approach, the best implementation of that approach that was reached, and the reasons why it was not adopted. This documentation should spare the WG and its contributors from the continuous reinvention of bad ideas.

As of the time of writing, the main body of CoAP-misc is almost empty, as most urgent developments have found their way into the WG documents, and many other ideas wait in the "nursery" section of the document.

5.2. Generalizing Media Types

CoAP defines a registry for combinations of an Internet Media Type ("MIME type") and a Content Encoding (e.g. some form of compression), enabling its compact encoding of this information in one or two bytes. Each entry in the registry defines a single, fixed set of media type parameters (as in ";charset=utf-8"), if any. This does not work well with media types that rely on more complex combinations of parameter settings. [I-D.doi-core-parameter-option] proposes to add an option to carry parameters for media types.

[I-D.fossati-core-multipart-ct] defines a new media type that can carry multiple embedded representations employing different media types using a binary type-length-value format.

5.3. Patience, Leisure, Pledge, or: Timing extensions

Several proposals intend to extend the amount of information available during an exchange about the timing requirements of the participants.

| draft-li-core-coap-patience-option | -01 | 2012-10-22 |

Another discussion is in Appendix B.4 of [I-D.bormann-coap-misc].

The question of whether some of this functionality should be introduced into the main WG documents now is currently also the subject of an active issue tracker ticket [CoRE204].

5.4. Extending Observe

5.5. Service discovery

Basic service discovery is defined in [RFC6690]. A JSON representation of the same information is defined in [I-D.ietf-core-links-json]. The intention is to make this information available in an equivalent format that is more accessible to classic Web servers, both as a file format (Internet media type) and as a format that can be used in e.g. a JavaScript API.

[I-D.arkko-core-dev-urn] defines a new Uniform Resource Name (URN) namespace that can be used to provide hardware device identifiers in resource descriptions.

[I-D.ietf-core-interfaces] provides additional semantics that can be used to make resource descriptions more directly machine-interpretable. This ties in to a more general discussion about CoRE profiles that has only just begun.

[I-D.greevenbosch-core-profile-description] ties into this and defines a basic JSON format for indicating what CoAP Options and what Content-Formats (still called media-types there) are available for a resource. At IETF86 there was fairly good consensus in the CoRE WG that we should be working on something addressing the underlying problem statement, while there was not yet agreement on the specific solution.

[I-D.fossati-core-fp-link-format-attribute] defines a link-format attribute that indicates a certain resource is best reached via a specific proxy.

5.6. Server discovery, Naming, etc.

On the boundary between service and server discovery, resource directory servers provide a way to collect resource descriptions from multiple servers into one accessible location.

[I-D.bormann-core-simple-server-discovery] provided a basic way to discover such servers in a constrained node/network without necessarily having to resort to multicast. It has been merged into [I-D.ietf-core-resource-directory], which defines protocol elements that can be used for setting up such a resource directory.

An attempt to merge mDNS/DNS-SD-based discovery (colloquially known as zeroconf or Bonjour), including recent approaches to extend these for constrained networks, into the picture is documented in [I-D.vanderstok-core-dna]; at IETF86 the authors showed interest to continue work on this.

5.7. More support for sleepy nodes

The basic communication model of CoAP was imported from the Web. This applies well to some communication requirements in constrained node/networks, but leaves some other requirements open.

The assumption underlying the current set of WG documents is that the communication layers below the application provide support functions for sleeping nodes. Adding support at the application layer might be able to further reduce the power requirements of "sleepy nodes" that can sleep most of the time.

[I-D.rahman-core-sleepy-problem-statement] summarizes the overall problem statement for sleepy nodes without getting into any specific solution.

A number of drafts aim to extend the CoAP communication model towards more support for sleepy nodes.

The base CoAP spec [I-D.ietf-core-coap] already provides some rudimentary support of sleepy nodes by supporting caching in intermediaries: resources from a sleepy node may be available from a caching proxy (if previously retrieved) even though the node is asleep. [I-D.ietf-core-observe] enhances this support by enabling sleepy nodes to update caching intermediaries on their own schedule.

A number of drafts more extensively extend the concept of an intermediary by introducing an additional kind of server that is hosting the resources of the sleepy node:

The approach of [I-D.vial-core-mirror-server] is to store the actual resource representations in a special type of Resource Directory called the Mirror Server. Communicating devices can then fetch the resource from the Mirror Server regardless of the state of the sleepy server. ([I-D.vial-core-mirror-proxy] simply appears to be a previous version of this draft.)

Similar to the above, the approach of [I-D.fossati-core-publish-option] is to temporarily delegate authority of its resources (when it is sleeping) to a proxy server that is always on.

Also, the approach of [I-D.giacomin-core-sleepy-option] is to define a proxy that acts as a store-and-forward agent for a sleepy node.

Other drafts introduce a variety of signaling based approaches to facilitate communicating with sleepy nodes: The approach of [I-D.castellani-core-alive] is to define a new CoAP message type (called "Alive") which the sleepy node multicasts to all interested devices when it wakes up. The approach of [I-D.rahman-core-sleepy] is to introduce storing of sleep characteristics in the Resource Directory. Communicating devices can then query the RD to learn the sleep status of the sleepy node before attempting communications.

Finally, some drafts build on the concept of the Observe mechanism to help keep track of the sleepy node information. The approach of [I-D.fossati-core-monitor-option] is to extend the Observe pattern to handle the scenario when both server and clients are sleepy nodes. Note that some of the other drafts (e.g., [I-D.vial-core-mirror-server], [I-D.rahman-core-sleepy]) include

using/extending the Observe mechanism as part of their overall approach.

Support for sleepy nodes is currently a very active subject of discussion in the WG; it is clear that there is a high level of interest in the WG in addressing application-level support for sleepy nodes in future specifications. See also the discussion of [I-D.ietf-lwig-cellular] in Section 3.1 above.

6. Replaced drafts

Internet-Drafts often get replaced by merged drafts or get promoted to WG drafts. As the relationships between drafts are not always accurately captured by the secretariat tools, this table provides a mapping from current drafts to any previous drafts they are replacing:

current draft	replaced draft
[I-D.ietf-core-coap]	draft-shelby-core-coap
[I-D.ietf-core-block]	draft-bormann-core-coap-block
	draft-li-core-coap-size-option
[I-D.ietf-core-observe]	draft-hartke-coap-observe
[RFC6690]	draft-shelby-core-link-format
[I-D.ietf-core-groupcomm]	draft-rahman-core-groupcomm
[I-D.becker-core-coap-sms-gprs]	draft-li-core-coap-over-sms
[I-D.vanderstok-core-dna]	draft-vanderstok-core-bc
[I-D.ietf-core-resource-directory]	draft-bormann-core-simple-server-discovery
[I-D.greevenbosch-core-minimum-request-interval]	draft-greevenbosch-core-block-minimum-time

Note that draft-scim-core-schema is just named against the naming conventions and actually unrelated to the CoRE working group.

7. IANA Considerations

This document has no actions for IANA.

8. Security Considerations

(None so far; this section will certainly grow as additional security considerations beyond those listed in the base specifications become known.)

9. Acknowledgements

(The concept for this document is borrowed from [RFC4815], which was invented by Lars-Erik Jonsson. Thanks!)

Akbar Rahman contributed text to this roadmap.

10. References

10.1. Normative References

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-13 (work in progress), October 2013.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-11 (work in progress), October 2013.

[I-D.ietf-tls-oob-pubkey]

Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", draft-ietf-tls-oob-pubkey-10 (work in progress), October 2013.

[I-D.mcgreww-tls-aes-ccm-ecc]

McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM ECC Cipher Suites for TLS", draft-mcgreww-tls-aes-ccm-ecc-07 (work in progress), August 2013.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

10.2. Informative References

- [COAP3] ETSI plugtests, "CoAP 3 & OMA Lightweight M2M", 2013, <<http://www.etsi.org/coap-oma-lightweight-m2m>>.
- [CoRE204] Bormann, C., "Introduce a minimal version of Pledge", CoRE ticket #204, 2012, <<http://trac.tools.ietf.org/wg/core/trac/ticket/204>>.
- [I-D.arkko-core-dev-urn]
Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-03 (work in progress), July 2012.
- [I-D.becker-core-coap-sms-gprs]
Becker, M., Li, K., Poetsch, T., and K. Kuladinithi, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-04 (work in progress), August 2013.
- [I-D.bormann-coap-misc]
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-25 (work in progress), May 2013.
- [I-D.bormann-core-simple-server-discovery]
Bormann, C., "CoRE Simple Server Discovery", draft-bormann-core-simple-server-discovery-01 (work in progress), March 2012.
- [I-D.castellani-core-advanced-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-advanced-http-mapping-02 (work in progress), July 2013.

- [I-D.castellani-core-alive]
Castellani, A. and S. Loreto, "CoAP Alive Message", draft-castellani-core-alive-00 (work in progress), March 2012.
- [I-D.castellani-core-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-http-mapping-07 (work in progress), February 2013.
- [I-D.castellani-lwig-coap-separate-responses]
Castellani, A., "Learning CoAP separate responses by examples", draft-castellani-lwig-coap-separate-responses-00 (work in progress), March 2012.
- [I-D.dijk-core-groupcomm-misc]
Dijk, E. and A. Rahman, "Miscellaneous CoAP Group Communication Topics", draft-dijk-core-groupcomm-misc-04 (work in progress), June 2013.
- [I-D.doi-core-parameter-option]
Doi, Y. and K. Lynn, "CoAP Content-Type Parameter Option", draft-doi-core-parameter-option-03 (work in progress), August 2013.
- [I-D.ersue-constrained-mgmt]
Ersue, M., Romascanu, D., and J. Schoenwaelder, "Management of Networks with Constrained Devices: Problem Statement, Use Cases and Requirements", draft-ersue-constrained-mgmt-03 (work in progress), February 2013.
- [I-D.fossati-core-fp-link-format-attribute]
Fossati, T. and S. Loreto, "Resource Discovery through Proxies", draft-fossati-core-fp-link-format-attribute-00 (work in progress), July 2012.
- [I-D.fossati-core-monitor-option]
Fossati, T., Giacomini, P., and S. Loreto, "Monitor Option for CoAP", draft-fossati-core-monitor-option-00 (work in progress), July 2012.
- [I-D.fossati-core-multipart-ct]
Fossati, T., "Multipart Content-Format Encoding for CoAP", draft-fossati-core-multipart-ct-03 (work in progress), October 2013.
- [I-D.fossati-core-publish-option]

Fossati, T., Giacomini, P., and S. Loreto, "Publish Option for CoAP", draft-fossati-core-publish-option-02 (work in progress), October 2013.

[I-D.garcia-core-security]

Garcia-Morchon, O., Kumar, S., Keoh, S., Hummen, R., and R. Struik, "Security Considerations in the IP-based Internet of Things", draft-garcia-core-security-06 (work in progress), September 2013.

[I-D.gerdes-core-dcaf-authorize]

Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authorization Function (DCAF)", draft-gerdes-core-dcaf-authorize-00 (work in progress), July 2013.

[I-D.giacomini-core-sleepy-option]

Fossati, T., Giacomini, P., Loreto, S., and M. Rossini, "Sleepy Option for CoAP", draft-giacomini-core-sleepy-option-00 (work in progress), February 2012.

[I-D.greevenbosch-core-authreq]

Greevenbosch, B., "Use cases and requirements for authentication and authorisation in CoAP", draft-greevenbosch-core-authreq-00 (work in progress), September 2013.

[I-D.greevenbosch-core-minimum-request-interval]

Greevenbosch, B., "CoAP Minimum Request Interval", draft-greevenbosch-core-minimum-request-interval-01 (work in progress), April 2013.

[I-D.greevenbosch-core-profile-description]

Greevenbosch, B., Hoebeke, J., Ishaq, I., and F. Abeele, "CoAP Profile Description Format", draft-greevenbosch-core-profile-description-02 (work in progress), June 2013.

[I-D.hahm-lwig-painless-constrained-programming]

Hahm, O., Baccelli, E., and K. Schleiser, "Painless Class 1 Devices Programming", draft-hahm-lwig-painless-constrained-programming-00 (work in progress), March 2013.

[I-D.hartke-core-codtls]

Hartke, K. and O. Bergmann, "Datagram Transport Layer Security in Constrained Environments", draft-hartke-core-codtls-02 (work in progress), July 2012.

[I-D.hex-lwig-energy-efficient]

Cao, Z., He, X., Kovatsch, M., Tian, H., and C. Gomez,
"Energy Efficient Implementation of IETF Constrained
Protocol Suite", draft-hex-lwig-energy-efficient-02 (work
in progress), October 2013.

[I-D.ietf-core-groupcomm]

Rahman, A. and E. Dijk, "Group Communication for CoAP",
draft-ietf-core-groupcomm-16 (work in progress), October
2013.

[I-D.ietf-core-interfaces]

Shelby, Z. and M. Vial, "CoRE Interfaces", draft-ietf-
core-interfaces-00 (work in progress), June 2013.

[I-D.ietf-core-links-json]

Bormann, C., "Representing CoRE Link Collections in JSON",
draft-ietf-core-links-json-00 (work in progress), June
2013.

[I-D.ietf-core-resource-directory]

Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource
Directory", draft-ietf-core-resource-directory-00 (work in
progress), June 2013.

[I-D.ietf-lwig-cellular]

Arkkio, J., Eriksson, A., and A. Keranen, "Building Power-
Efficient CoAP Devices for Cellular Networks", draft-ietf-
lwig-cellular-00 (work in progress), August 2013.

[I-D.ietf-lwig-guidance]

Bormann, C., "Guidance for Light-Weight Implementations of
the Internet Protocol Suite", draft-ietf-lwig-guidance-03
(work in progress), February 2013.

[I-D.ietf-lwig-ikev2-minimal]

Kivinen, T., "Minimal IKEv2", draft-ietf-lwig-
ikev2-minimal-01 (work in progress), October 2013.

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and A. Keranen, "Terminology for
Constrained Node Networks", draft-ietf-lwig-terminology-05
(work in progress), July 2013.

[I-D.ietf-lwig-tls-minimal]

Kumar, S., Keoh, S., and H. Tschofenig, "A Hitchhiker's Guide to the (Datagram) Transport Layer Security Protocol for Smart Objects and Constrained Node Networks", draft-ietf-lwig-tls-minimal-00 (work in progress), September 2013.

[I-D.jennings-core-transitive-trust-enrollment]

Jennings, C., "Transitive Trust Enrollment for Constrained Devices", draft-jennings-core-transitive-trust-enrollment-01 (work in progress), October 2012.

[I-D.kovatsch-lwig-class1-coap]

Kovatsch, M., "Implementing CoAP for Class 1 Devices", draft-kovatsch-lwig-class1-coap-00 (work in progress), October 2012.

[I-D.kovatsch-lwig-coap]

Kovatsch, M., Bergmann, O., Dijk, E., He, X., and C. Bormann, "CoAP Implementation Guidance", draft-kovatsch-lwig-coap-01 (work in progress), July 2013.

[I-D.li-core-coap-payload-length-option]

Li, K., "CoAP Payload-Length Option Extension", draft-li-core-coap-payload-length-option-02 (work in progress), August 2013.

[I-D.pporamba-dtls-certkey]

Porambage, P., Kumar, P., Gurtov, A., Ylianttila, M., and E. Harjula, "Certificate based keying scheme for DTLS secured IoT", draft-pporamba-dtls-certkey-00 (work in progress), June 2013.

[I-D.rahman-core-sleepy-problem-statement]

Rahman, A., Fossati, T., Loreto, S., and M. Vial, "Sleepy Devices in CoAP - Problem Statement", draft-rahman-core-sleepy-problem-statement-01 (work in progress), October 2012.

[I-D.rahman-core-sleepy]

Rahman, A., "Enhanced Sleepy Node Support for CoAP", draft-rahman-core-sleepy-04 (work in progress), October 2013.

[I-D.sarikaya-core-secure-bootsolution]

Sarikaya, B., "Security Bootstrapping Solution for Resource-Constrained Devices", draft-sarikaya-core-secure-bootsolution-00 (work in progress), February 2013.

- [I-D.schmitt-two-way-authentication-for-iot]
Schmitt, C., Stiller, B., Kothmayr, T., and W. Hu, "DTLS-based Security with two-way Authentication for IoT", draft-schmitt-two-way-authentication-for-iot-01 (work in progress), October 2013.
- [I-D.seitz-core-sec-usecases]
Seitz, L., Gerdes, S., and G. Selander, "Use cases for CoRE security", draft-seitz-core-sec-usecases-00 (work in progress), September 2013.
- [I-D.selander-core-access-control]
Selander, G., Sethi, M., and L. Seitz, "Access Control Framework for Constrained Environments", draft-selander-core-access-control-01 (work in progress), October 2013.
- [I-D.silverajan-core-coap-alternative-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-03 (work in progress), October 2013.
- [I-D.urien-core-racs]
Urien, P., "Remote APDU Call Secure (RACS)", draft-urien-core-racs-00 (work in progress), August 2013.
- [I-D.vanderstok-core-dna]
Stok, P., Lynn, K., and A. Brandt, "CoRE Discovery, Naming, and Addressing", draft-vanderstok-core-dna-02 (work in progress), July 2012.
- [I-D.vial-core-mirror-proxy]
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-proxy-01 (work in progress), July 2012.
- [I-D.vial-core-mirror-server]
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-server-01 (work in progress), April 2013.
- [I-D.zhu-core-groupauth]
Zhu, J. and M. Qi, "Group Authentication", draft-zhu-core-groupauth-01 (work in progress), September 2013.
- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

- [RFC2418] Bradner, S., "IETF Working Group Guidelines and Procedures", BCP 25, RFC 2418, September 1998.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC4815] Jonsson, L-E., Sandlund, K., Pelletier, G., and P. Kremer, "RObust Header Compression (ROHC): Corrections and Clarifications to RFC 3095", RFC 4815, February 2007.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.
- [SB] Bormann, C., Castellani, A., and Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes", DOI 10.1109/MIC.2012.29, 2012.
- [WEI] Shelby, Z. and C. Bormann, "6LoWPAN: the Wireless Embedded Internet", ISBN 9780470747995, 2009.

Author's Address

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 2, 2014

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
KoanLogic
E. Dijk
Philips Research
July 1, 2013

Best Practices for HTTP-CoAP Mapping Implementation
draft-castellani-core-advanced-http-mapping-02

Abstract

This draft describes advanced features for HTTP-CoAP proxy implementors. It details deployment options, discusses possible approaches for URI mapping, and provides useful considerations related to protocol translation.

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 <http://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 2, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Terminology and Conventions	3
2. Introduction	3
3. Use Case: HTTP/IPv4-CoAP/IPv6 Proxy	3
4. Multiple Message Exchanges Mapping	6
4.1. Relevant Features of Existing Standards	6
4.1.1. Multipart Messages	6
4.1.2. Immediate Message Delivery	6
4.1.3. Detailing Source Information	7
4.2. Multicast Mapping	7
4.2.1. URI Identification and Mapping	7
4.2.2. Request Handling	8
4.2.3. Examples	8
4.3. Multicast Response Caching	10
4.4. Observe Mapping	11
4.4.1. Identification	11
4.4.2. Notification(s) Mapping	13
4.4.3. Examples	14
5. HTML5 Scheme Handler Registration	20
6. Placement and Deployment	20
7. Examples	21
8. Acknowledgements	23
9. IANA Considerations	23
10. Security Considerations	24
10.1. Cross-protocol Security Policy Mapping	24
10.2. Subscription	24
11. References	24
11.1. Normative References	24
11.2. Informative References	26
Appendix A. Internal Mapping Functions (from an Implementer's Perspective)	26
A.1. URL Map Algorithm	27
A.2. Security Policy Map Algorithm	28
A.3. Content-Type Map Algorithm	29
Authors' Addresses	29

1. Terminology and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document assumes readers are familiar with the terms and concepts that are used in [I-D.ietf-core-coap] . In addition, this document defines the following terminology:

A device providing cross-protocol HTTP-CoAP mapping is called an HTTP-CoAP cross-protocol proxy (HC proxy).

At least two different kinds of HC proxies exist:

- o One-way cross-protocol proxy (1-way proxy): This proxy translates from a client of a protocol to a server of another protocol but not vice-versa.
- o Two-way (or bidirectional) cross-protocol proxy (2-way proxy): This proxy translates from a client of both protocols to a server supporting one protocol.

2. Introduction

RESTful protocols, such as HTTP [RFC2616] and CoAP [I-D.ietf-core-coap], can interoperate through an intermediary proxy which performs cross-protocol mapping.

A base reference for the mapping process is provided in [I-D.ietf-core-coap]. However, depending on the involved application, deployment scenario, or network topology, such mapping can be realized using a wide range of intermediaries.

Moreover, the process of implementing such a proxy can be complex, and details regarding its internal procedures and design choices deserve further discussion, which is provided in this document.

This draft itself is an evolution of the mapping features covered in [I-D.ietf-core-http-mapping].

3. Use Case: HTTP/IPv4-CoAP/IPv6 Proxy

This section covers the expected common use case regarding an HTTP/IPv4 client accessing a CoAP/IPv6 resource.

While HTTP and IPv4 are today widely adopted communication protocols in the Internet, a pervasive deployment of constrained nodes exploiting the IPv6 address space is expected: enabling direct interoperability of such technologies is a valuable goal.

An HC proxy supporting IPv4/IPv6 mapping is said to be a v4/v6 proxy.

An HC v4/v6 proxy SHOULD always try to resolve the URI authority, and SHOULD prefer using the IPv6 resolution if available. The authority part of the URI is used internally by the HC proxy and SHOULD NOT be mapped to CoAP.

Figure 1 shows an HTTP client on IPv4 (C) accessing a CoAP server on IPv6 (S) through an HC proxy on IPv4/IPv6 (P). The DNS has an A record for "node.coap.something.net" resolving to the IPv4 address of the HC proxy, and an AAAA record with the IPv6 address of the CoAP server.

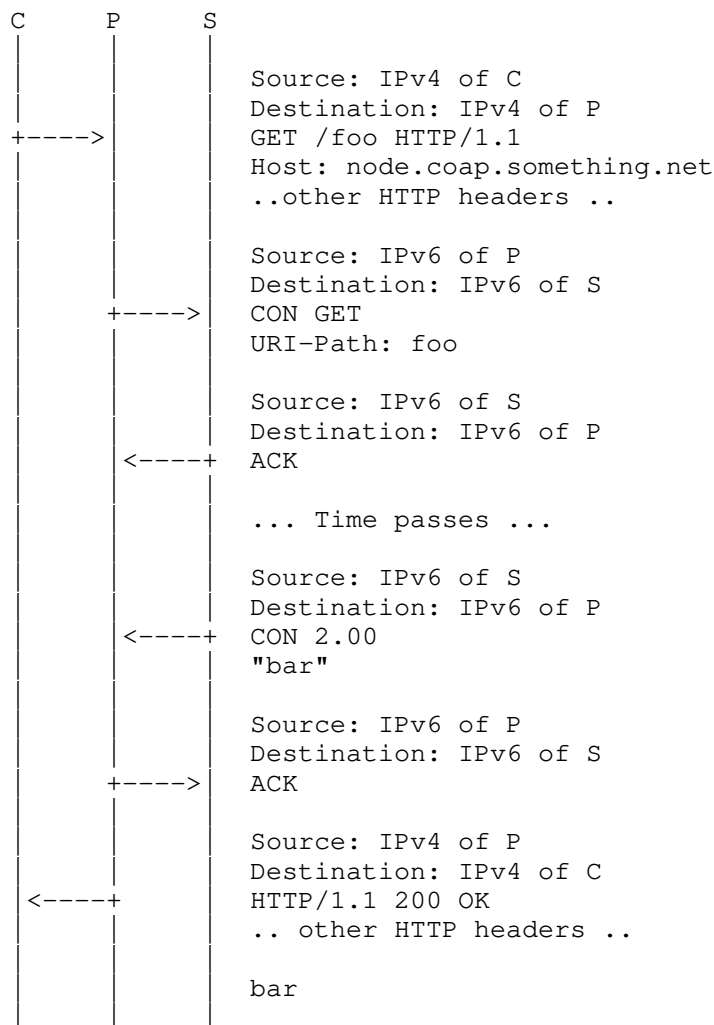


Figure 1: HTTP/IPv4 to CoAP/IPv6 Mapping

The proposed example shows the HC proxy operating also the mapping between IPv4 to IPv6 using the authority information available in any HTTP 1.1 request. This way, IPv6 connectivity is not required at the HTTP client when accessing a CoAP server over IPv6 only, which is a typical expected use case.

When P is an interception HC proxy, the CoAP request SHOULD have the IPv6 address of C as source (IPv4 can always be mapped into IPv6).

The described solution takes into account only the HTTP/IPv4 clients accessing CoAP/IPv6 servers; this solution does not provide a full fledged mapping from HTTP to CoAP.

In order to obtain a working deployment for HTTP/IPv6 clients, a different HC proxy access method may be required, or Internet AAAA records should not point to the node anymore (the HC proxy should use a different DNS database pointing to the node).

When an HC interception proxy deployment is used this solution is fully working even with HTTP/IPv6 clients.

4. Multiple Message Exchanges Mapping

This section discusses the mapping of the multicast and observe features of CoAP, which have no corresponding primitive in HTTP, and as such are not immediately translatable.

The mapping, which must be considered in both the arrow directions (H->C, C->H) may involve multi-part responses, as in the multicast use case, asynchronous delivery through HTTP bidirectional techniques, and HTTP Web Linking in order to reduce the semantics lost in the translation.

4.1. Relevant Features of Existing Standards

Various features provided by existing standards are useful to efficiently represent sessions involving multiple messages.

4.1.1. Multipart Messages

In particular, the "multipart/*" media type, defined in Section 5.1 of [RFC2046], is a suitable solution to deliver multiple CoAP responses within a single HTTP payload. Each part of a multipart entity SHOULD be represented using "message/http" media type containing the full mapping of a single CoAP response as previously described.

4.1.2. Immediate Message Delivery

An HC proxy may prefer to transfer each CoAP response immediately after its reception. This is possible thanks to the HTTP Transfer-Encoding "chunked", that enables transferring single responses without any further delay.

A detailed discussion on the use of chunked Transfer-Encoding to stream data over HTTP can be found in [RFC6202]. Large delays

between chunks can lead the HTTP session to timeout, more details on this issue can be found in [I-D.thomson-hybi-http-timeout].

An HC proxy MAY prefer (e.g. to avoid buffering) to transfer each response related to a multicast request as soon as it comes in from the server. One possible way to achieve this result is using the "chunked" Transfer-Encoding in the HTTP response, to push individual responses until some trigger is fired (timeout, max number of messages, etc.).

An example showing immediate delivery of CoAP responses using HTTP chunks will be provided in Section 4.4, while describing its application to an observe session.

4.1.3. Detailing Source Information

Under some circumstances, responses may come from different sources (i.e. responses to a multicast request); in this case details about the actual source of each CoAP response MAY be provided to the client. Source information can be represented using HTTP Web Linking as defined in [RFC5988], by adding the actual source URI into each response using Link option with "via" relation type.

4.2. Multicast Mapping

In order to establish a multicast communication such a feature should be offered either by the network (i.e. IP multicast, link-layer multicast, etc.) or by a gateway (i.e. the HC proxy). Rationale on the methods available to obtain such a feature is out-of-scope of this document, and extensive discussion of group communication techniques is available in [I-D.ietf-core-groupcomm].

Additional considerations related to handling multicast requests mapping are detailed in the following sections.

4.2.1. URI Identification and Mapping

In order to successfully handle a multicast request, the HC proxy MUST successfully perform the following tasks on the URI:

Identification: The HC proxy MUST understand whether the requested URI identifies a group of nodes.

Mapping: The HC proxy MUST know how to distribute the multicast request to involved servers; this process is specific of the group communication technology used.

When using IPv6 multicast paired with DNS, the mapping to IPv6

multicast is simply done using DNS resolution. If the group management is performed at the proxy, the URI or part of it (i.e. the authority) can be mapped using some static or dynamic table available at the HC proxy. In Section 3.5 of [I-D.ietf-core-groupcomm] discusses a method to build and maintain a local table of multicast authorities.

4.2.2. Request Handling

When the HC proxy receives a request to a URI that has been successfully identified and mapped to a group of nodes, it SHOULD start a multicast proxying operation, if supported by the proxy.

Multicast request handling consists of the following steps:

Multicast TX: The HC proxy sends out the request on the CoAP side by using the methods offered by the specific group communication technology used in the constrained network;

Collecting RXs: The HC proxy collects every response related to the request;

Timeout: The HC proxy has to pay special attention in multicast timing, detailed discussion about timing depends upon the particular group communication technology used;

Distributing RXs to the client: The HC proxy can distribute the responses in two different ways: batch delivering them at the end of the process or on timeout, or immediately delivering them as they are available. Batch requires more caching and introduces delays but may lead to lower TCP overhead and simpler processing. Immediate delivery is the converse. A trade-off solution of partial batch delivery may also be feasible and efficient in some circumstances.

4.2.3. Examples

Figure 2 shows an HTTP client (C) requesting the resource "/foo" to a group of CoAP servers (S1/S2/S3) through an HC proxy (P) which uses IP multicast to send the corresponding CoAP request.

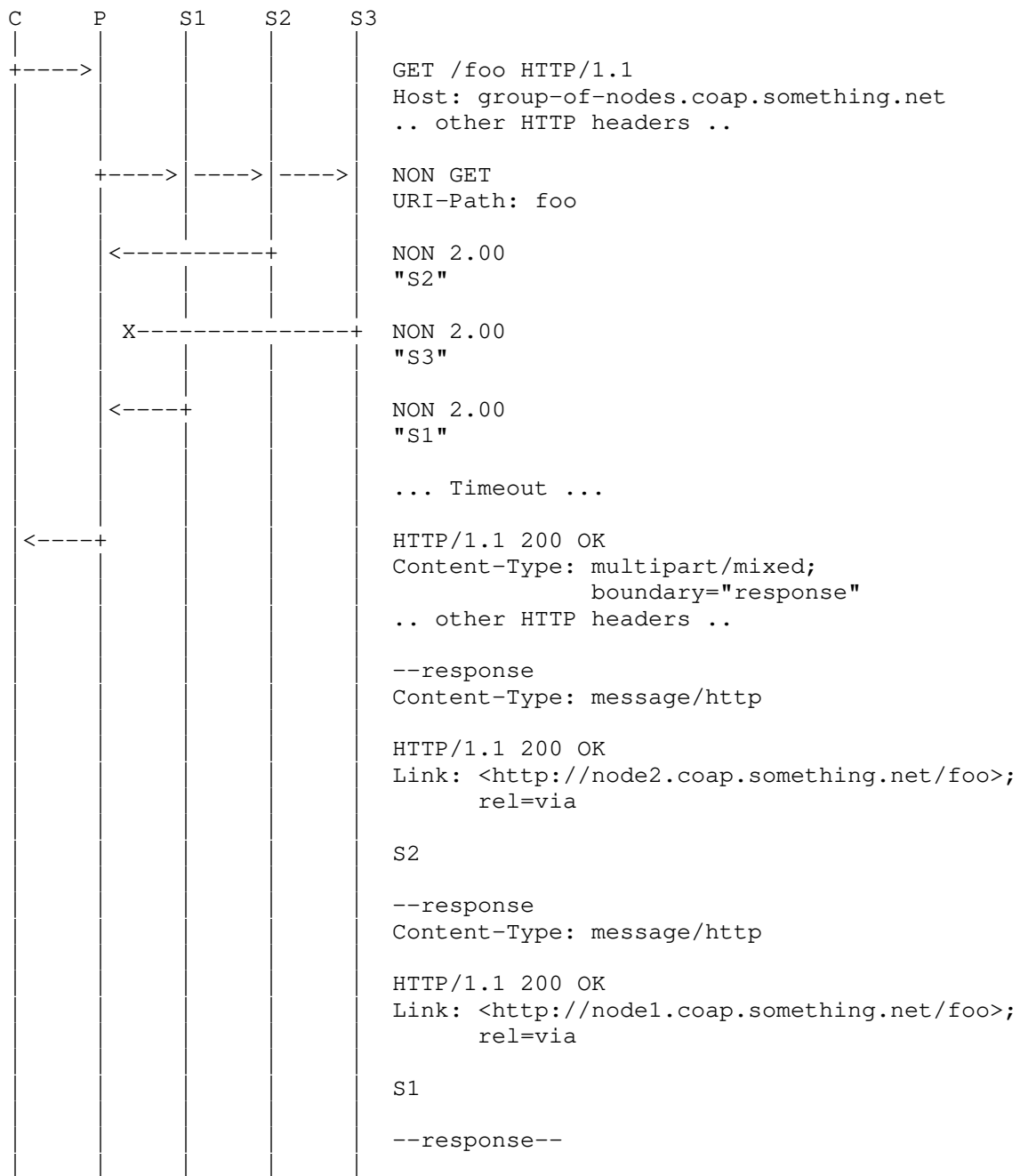


Figure 2: Unicast HTTP to Multicast CoAP Mapping

The example proposed in the above diagram does not make any assumption on which underlying group communication technology is available in the constrained network. Some detailed discussion is provided about it along the following lines.

C makes a GET request to `group-of-nodes.coap.something.net`. This domain name MAY either resolve to the address of P, or to the IPv6 multicast address of the nodes (if IP multicast is supported and P is an interception proxy), or the proxy P is specifically known by the client that sends this request to it.

To successfully start multicast proxying operation, the HC proxy MUST know that the destination URI involves a group of CoAP servers, e.g. the authority `group-of-nodes.coap.something.net` is known to identify a group of nodes either by using an internal lookup table, using DNS paired with IPv6 multicast, or by using some other special technique.

A specific implementation option is proposed to further explain the proposed example. Assume that DNS is configured such that all subdomain queries to `coap.something.net`, such as `group-of-nodes.coap.something.net`, resolve to the address of P. P performs the HC URI mapping by removing the 'coap' subdomain from the authority and by switching the scheme from 'http' to 'coap' (result: `"coap://group-of-node.something.net/foo"`); `"group-of-nodes.something.net"` is resolved to an IPv6 multicast address to which S1, S2 and S3 belong. The proxy handles this request as multicast and sends the request `"GET /foo"` to the multicast group .

4.3. Multicast Response Caching

We call perfect caching when the proxy uses only the cached representations to provide a response to the HTTP client. In the case of a multicast CoAP request, perfect caching is not adequate. This section updates the general caching and congestion control guidelines of with specific guidelines for the multicast use case.

Due to the inherent unreliable nature of the NON messages involved and since nodes may have dynamic membership in multicast groups, responding only with previously cached responses without issuing a new multicast request is not recommended. This perfect caching behaviour leads to miss responses of nodes that later joined the multicast group, and/or to repeatedly serve partial representations due to message losses. Therefore a multicast CoAP request SHOULD be sent by a HC proxy for each incoming request addressed to a multicast group.

Caching of multicast responses is still a valuable goal to pursue reduce network congestion, battery consumption and response latency.

Some considerations to be performed when adopting a multicast caching behaviour are outlined in the following paragraph.

Caching of multicast GET responses MAY be implemented by adopting some technique that takes into account either knowledge about dynamic characteristics of group membership (occurrence or frequency of group changes) or even better its full knowledge (list of nodes currently part of the group).

When using a technique exploiting this knowledge, valid cached responses SHOULD be served from cache.

4.4. Observe Mapping

By design, and certainly not without a good rationale, HTTP lacks a publish-subscriber facility. This implies that the mapping of the CoAP observe semantics has to be created ad hoc, perhaps by making use of one of the well-known HTTP techniques currently employed to establish an HTTP bidirectional connection with the target resource - as documented in [RFC6202].

In the following sections we will describe some of the approaches that can be used to identify an observable resource and to create the communication bridging needed to set up an end to end HTTP-CoAP observation.

4.4.1. Identification

In order to appropriately process an observe request, the HC proxy needs to know whether a given request is intended to establish an observation on the target resource, instead of triggering a regular request-response exchange.

At least two different approaches to identify such special requests exist, as discussed below.

4.4.1.1. Observable URI Mapping

An URI is said to be observable whenever every request to it implicitly requires the establishment of an HTTP bidirectional connection to the resource.

Such subscription to the resource is always paired, if possible, to a CoAP observe session to the actual resource being observed. In general, multiple connections that are active with a single observable resource at the same time, are multiplexed to the single observe session opened by the intermediary. Its notifications are then de-multiplexed by the HC proxy to every HTTP subscriber.

An intermediary MAY pair a couple of distinct HTTP URIs to a single CoAP observable resource: one providing the usual request-response mediated access to the resource, and the other that always triggers a CoAP observe session.

4.4.1.1.1. Discovery

As shown in Figure 3, in order to know whether an URI is observable, an HTTP UA MAY do a pre-flight request to the target resource using the HTTP OPTIONS method (see section 6.2 of [I-D.ietf-httpbis-p2-semantics]) to discover the communication options available for that resource.

If the resource supports observation, the proxy adds a Link Header [RFC5988] with the "obs" attribute as link-param (see Section 7 of [I-D.ietf-core-observe]).

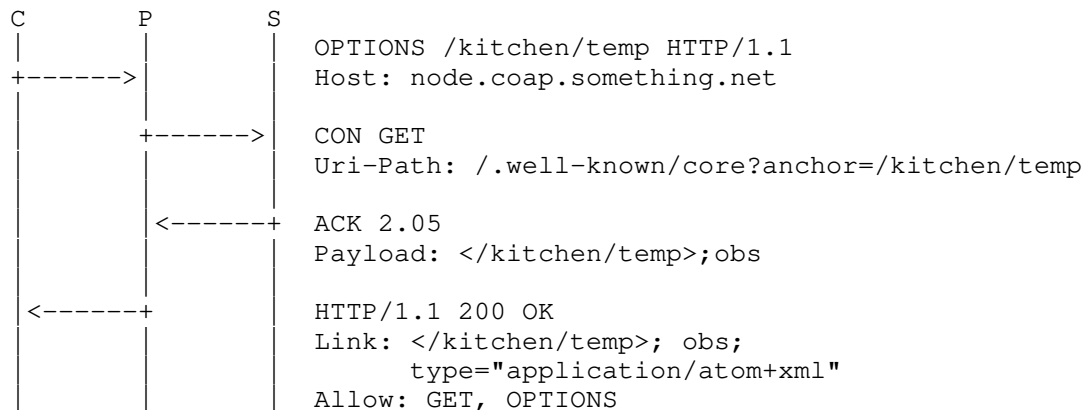


Figure 3: Discover Observability with HTTP OPTIONS

4.4.1.2. Differentiation Using HTTP Header

Discerning an observation request through in-protocol means, e.g. via the presence and values of some HTTP metadata, avoids introducing static "observable" URIs in the HC proxy namespace. Though ideally the former should be preferred, there seems to be no standard way to use one of the established HTTP headers to convey the observe semantics.

Standardizing such methods is out-of-scope of this document, so we just point out some possible approaches that in the future may be used to differentiate observation requests from regular requests.

4.4.1.2.1. Expect Header

The first method involves the use of the Expect header as defined in Section 9.3 of [I-D.ietf-httpbis-p2-semantics]. Whenever an HC proxy receives a request with a "206-partial-content" expectation, the proxy MUST fulfill this expectation by pairing this request to either a new or existing observe session to the resource.

If the proxy is unable to observe the resource, or if the observation establishment fails, the proxy MUST reply to the client with "417 Expectation Failed" status code.

Given that the Expect header is processed hop-by-hop, this method will fail immediately in case a proxy not supporting this expectation is traversed. For this reason, at present, the said approach can't be used in the public Internet.

4.4.1.2.2. Prefer Header

A second, very similar, approach involves the use of the Prefer header, defined in [I-D.snell-http-prefer]. The HTTP user agent expresses the preference to establish an observation with the target resource by including a "streaming" preference to request an HTTP Streaming session, or a "long-polling" preference to signal to the proxy its intended polling behaviour (see [RFC6202]).

A compliant HC proxy will try to fulfill the preference, and manifest observation establishment success by responding with a status code of "206 Partial Content". The observation request fails, falling back to a single response, whenever the status code is different from 206.

This approach will never fail immediately, differently from the previous one, even across a chain of unaware proxies; however, as documented in [RFC6202], caching intermediaries may interfere, delay or block the HTTP bidirectional connection, making this approach unacceptable when no weak consistency of the resource can be tolerated by the requesting UA.

4.4.2. Notification(s) Mapping

Multiplexing notifications using a single HTTP bidirectional session needs some further considerations about the selection of the media type that best fits this specific use case.

The usage of two different content-types that are suitable for carrying multiple notifications in a single session, is discussed in the following sections.

4.4.2.1. Multipart Messaging

As already discussed in Section 4.1.1 for multicasting, the "multipart/*" media type is a suitable solution to deliver multiple CoAP notifications within a single HTTP payload.

As in the multicast case, each part of the multipart entity MAY be represented using a "message/http" media type, containing the full mapping of the single CoAP notification mapped, so that CoAP envelope information are preserved (e.g. the response code).

A more sophisticated mapping could use multipart/mixed with native or translated media type.

4.4.2.2. Using ATOM Feeds

Popular observable resources with refresh rates higher than a couple of seconds may be treated as Atom feeds [RFC4287], especially with delay tolerant user agents and where persistence is required.

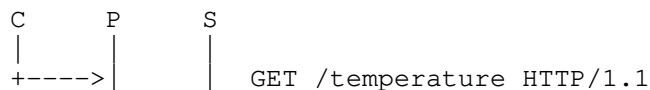
Figure 3 shows a resource supporting 'application/atom+xml' media-type. In such case clients can listen to update notification by regularly polling the resource via opportunely spaced GETs, i.e. driven by the advertised max-age value.

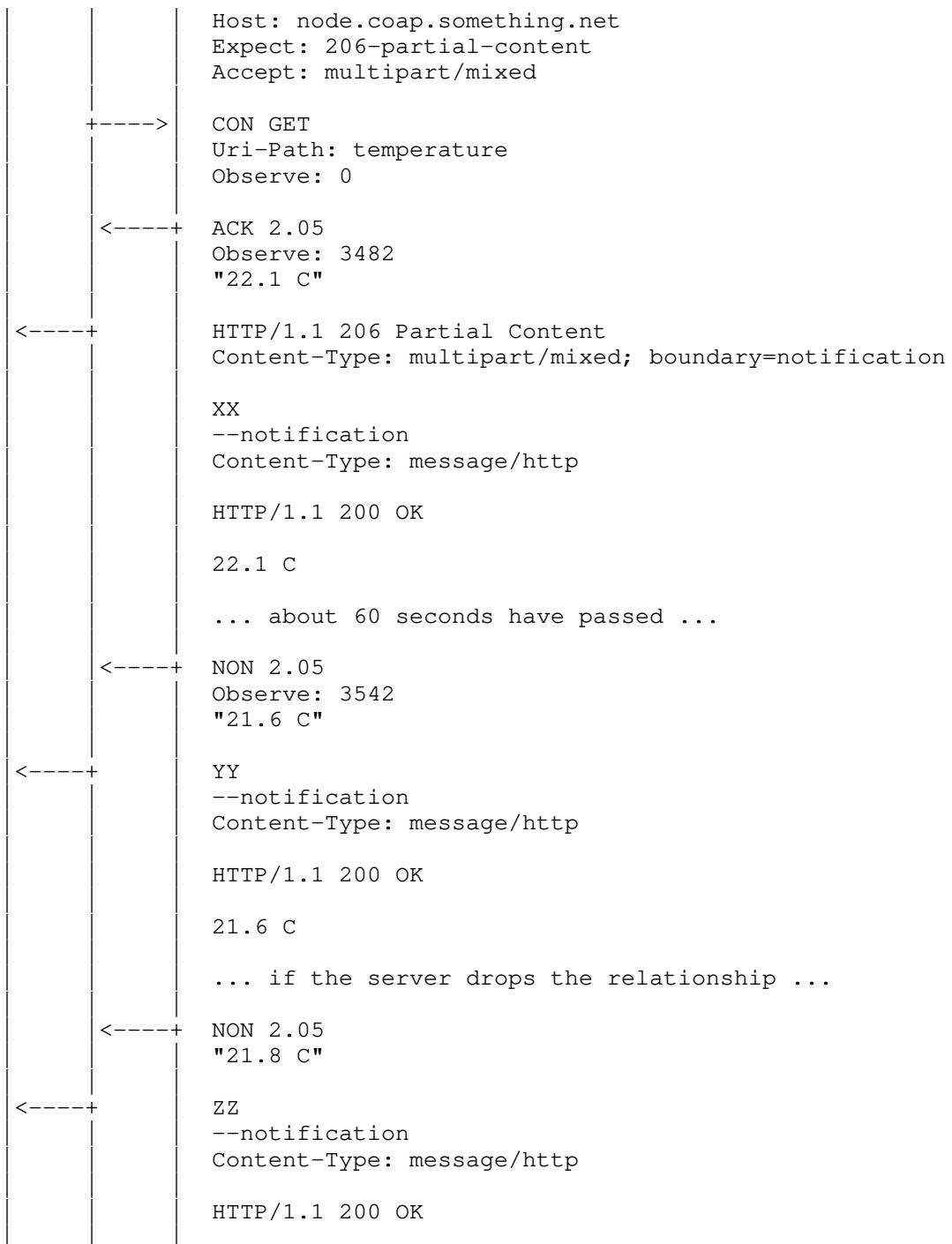
4.4.3. Examples

Figure 4 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

C manifests its intention to observe T by including the Expect Header in the request; if P or S do not support this interaction, the request MUST fail with "417 Expectation Failed" return code. In the presented example, both P and C support this interaction, and the subscription is successful, as stated by the "206 Partial Content" return code.

At every notification corresponds the emission of a HTTP chunk containing a single part, which contains a "message/http" payload containing the full mapping of the notification. When the observation is dropped by the CoAP server, the HTTP streaming session is closed.





		21.8 C
		--notification--
		0

Figure 4: HTTP Streaming to CoAP Observe

Figure 5 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

C manifests its intention to observe T by including the Prefer Header in the request; if P or S do not support this interaction, the request silently fails if a status code "200 OK" is returned, which means that no further notification is expected on that session.

In the presented example, both P and C support this interaction, and the subscription is successful, as stated by the "206 Partial Content" status code. At every notification a new response is sent to the pending client, always containing the "206 Partial Content" status code, to indicate that the observe session is still active, so that C can issue a new long-polling request immediately after this notification.

If the observation relationship is dropped by S, P notifies the last received content using the "200 OK" status code, indicating that no further notification is expected on this observe session.

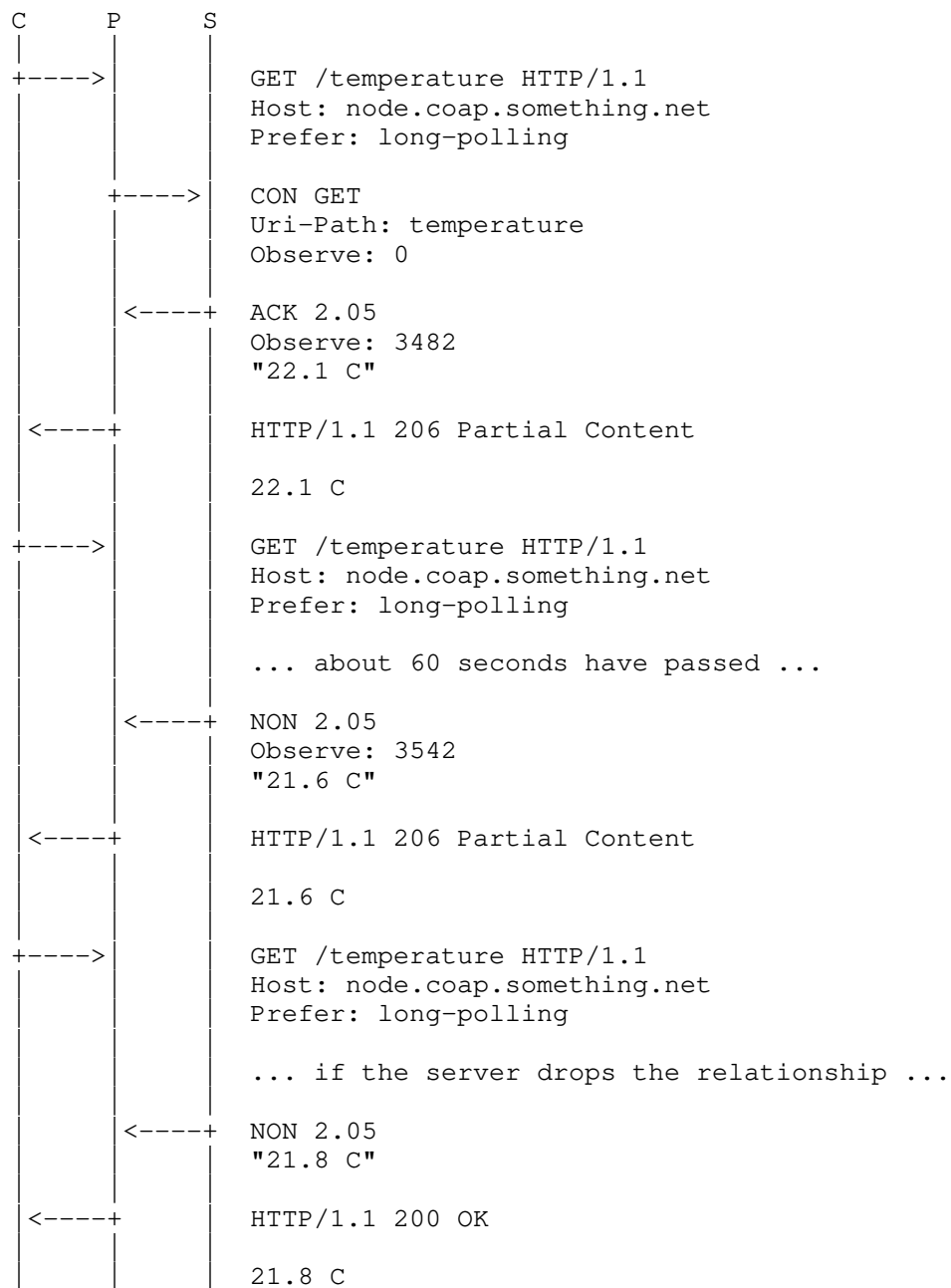


Figure 5: HTTP Long Polling to CoAP Observe

Figure 6 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "kitchen/temp" (T) available on S.

It is assumed that the HC proxy knows that the requested resource is observable (since perhaps being asked beforehand to discover its properties as described in Figure 3.) When asked by the HTTP client to retrieve the resource, it requests an observation - in case it weren't already in place - and then sends the collected data to the client as an Atom feed. The data coming through in the constrained network is stored locally on the proxy, and forwarded when further requests are received on the HTTP side. As already said, using the Atom format has two main advantages: first, there is always a "current" feed, but there may also be a complete log made available to HTTP clients; secondly, the HTTP intermediaries can play a substantial role in absorbing a fair amount of the load on the HC proxy. The latter is a very important property when the requested resource is or becomes very popular.

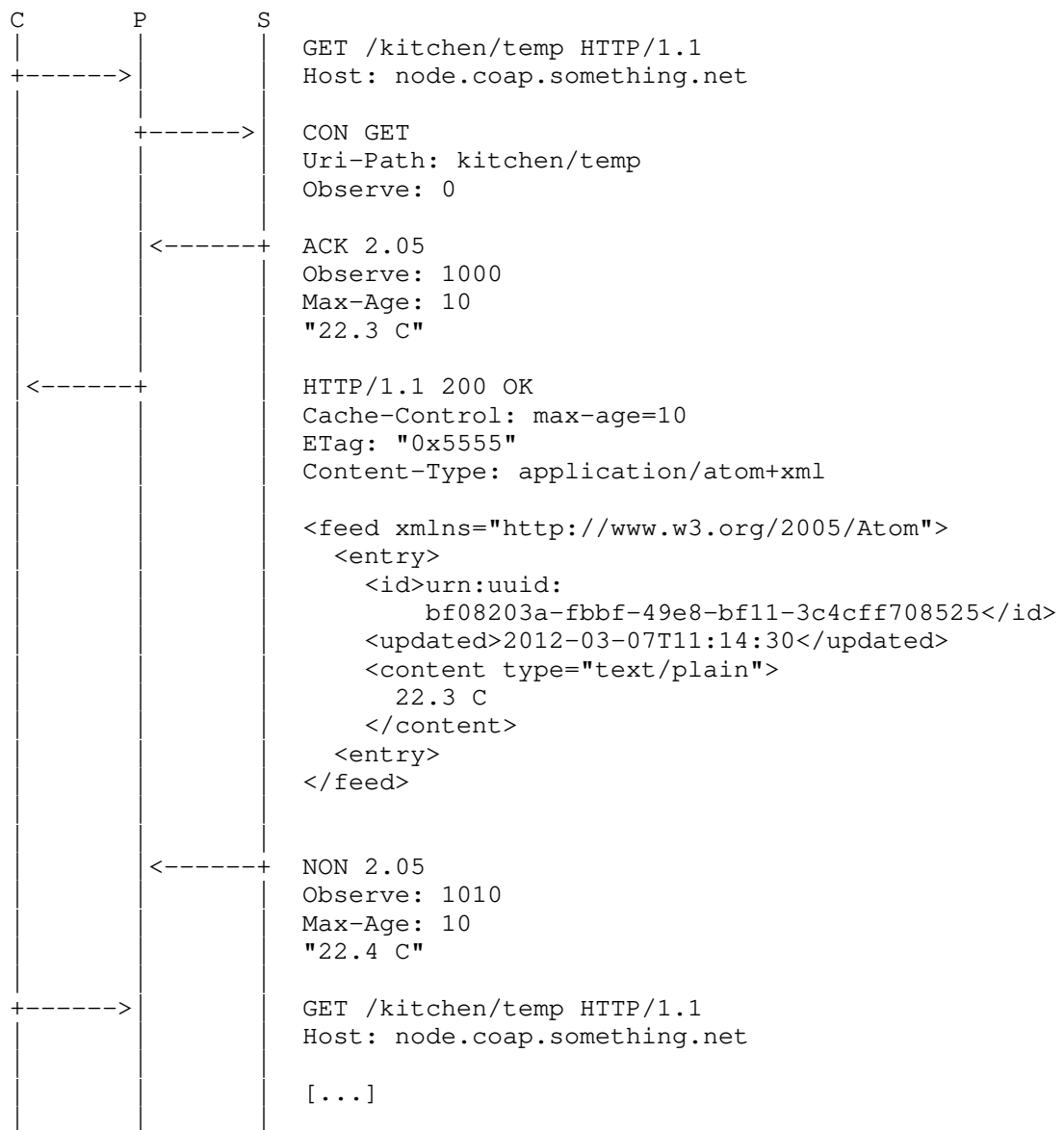


Figure 6: Observation via Atom feeds

5. HTML5 Scheme Handler Registration

The draft HTML5 standard offers a mechanism that allows an HTTP user agent to register a custom scheme handler through an HTML5 web page. This feature permits to an HC proxy to be registered as "handler" for URIs with the 'web+coap' or 'web+coaps' schemes using an HTML5 web page which embeds the custom scheme handler registration call `registerProtocolHandler()` described in Section 6.5.1.2 of [W3C.HTML5].

Example: the HTML5 homepage of a HC proxy at `h2c.example.org` could include the method call:

```
registerProtocolHandler('web+coap','proxy?url=%s','example HC proxy')
```

This registration call will prompt the HTTP user agent to ask for the user's permission to register the HC proxy as a handler for all 'web+coap' URIs. If the user accepts, whenever a 'web+coap' link is requested, the request will be fulfilled through the HC proxy: URI "web+coap://foo.org/a" will be transformed into URI "http://h2c.example.org/proxy?url=web+coap://foo.org/a".

6. Placement and Deployment

In typical scenarios, for communication from a CoAP client to an HTTP origin server, the HC proxy is expected to be located on the client-side (CS). Specifically, the HC proxy is expected to be deployed at the edge of the constrained network as shown in Figure 7.

The arguments supporting CS placement are as follows:

Client/Proxy/Network configuration overhead: CoAP clients require either static proxy configuration or proxy discovery support. This overhead is simplified if the proxy is placed on the same network domain of the client.

TCP/UDP: Translation between CoAP and HTTP requires also UDP to TCP mapping; UDP performance over the unconstrained Internet may not be adequate. In order to minimize the number of required retransmissions on the constrained part of the network and the overall reliability, TCP/UDP conversion SHOULD be performed as soon as possible in the network path.

Caching: Efficient caching requires that all the CoAP traffic is intercepted by the same proxy, thus a CS placement, collecting all the traffic, is strategic for this need.

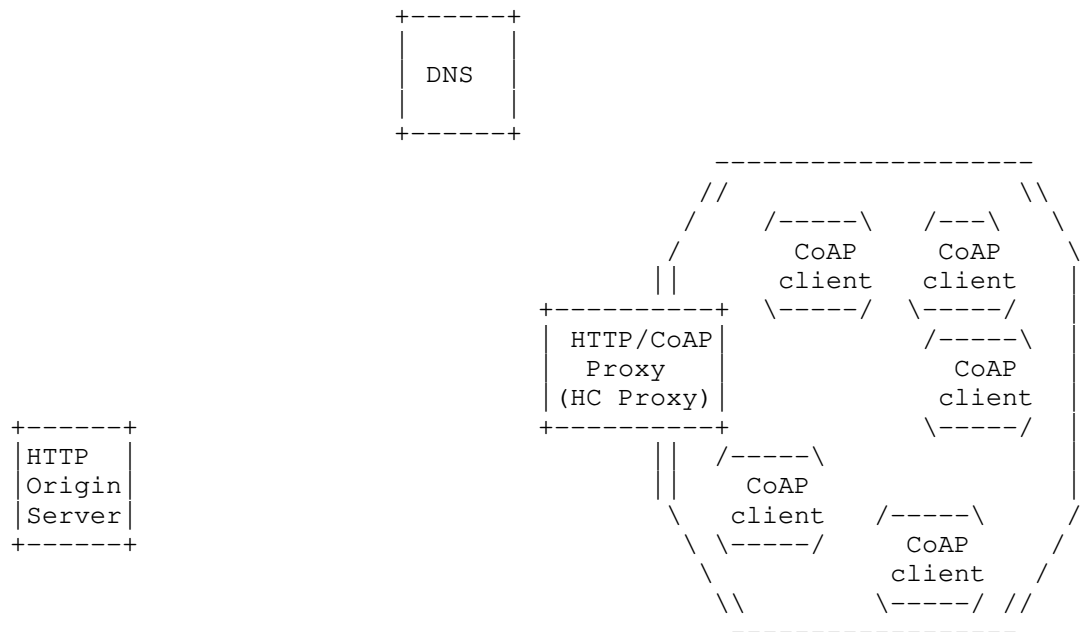


Figure 7: Client-side HC Proxy Deployment Scenario

7. Examples

Figure 8 shows an example implementation of a basic CoAP GET request with an HTTP URI as the value of a Proxy-URI option. The proxy retrieves a representation of the target resource from the HTTP origin server. It converts the payload to a UTF-8 charset, calculates the Max-Age Option from the Expires header field, and derives an entity-tag from the ETag header field.

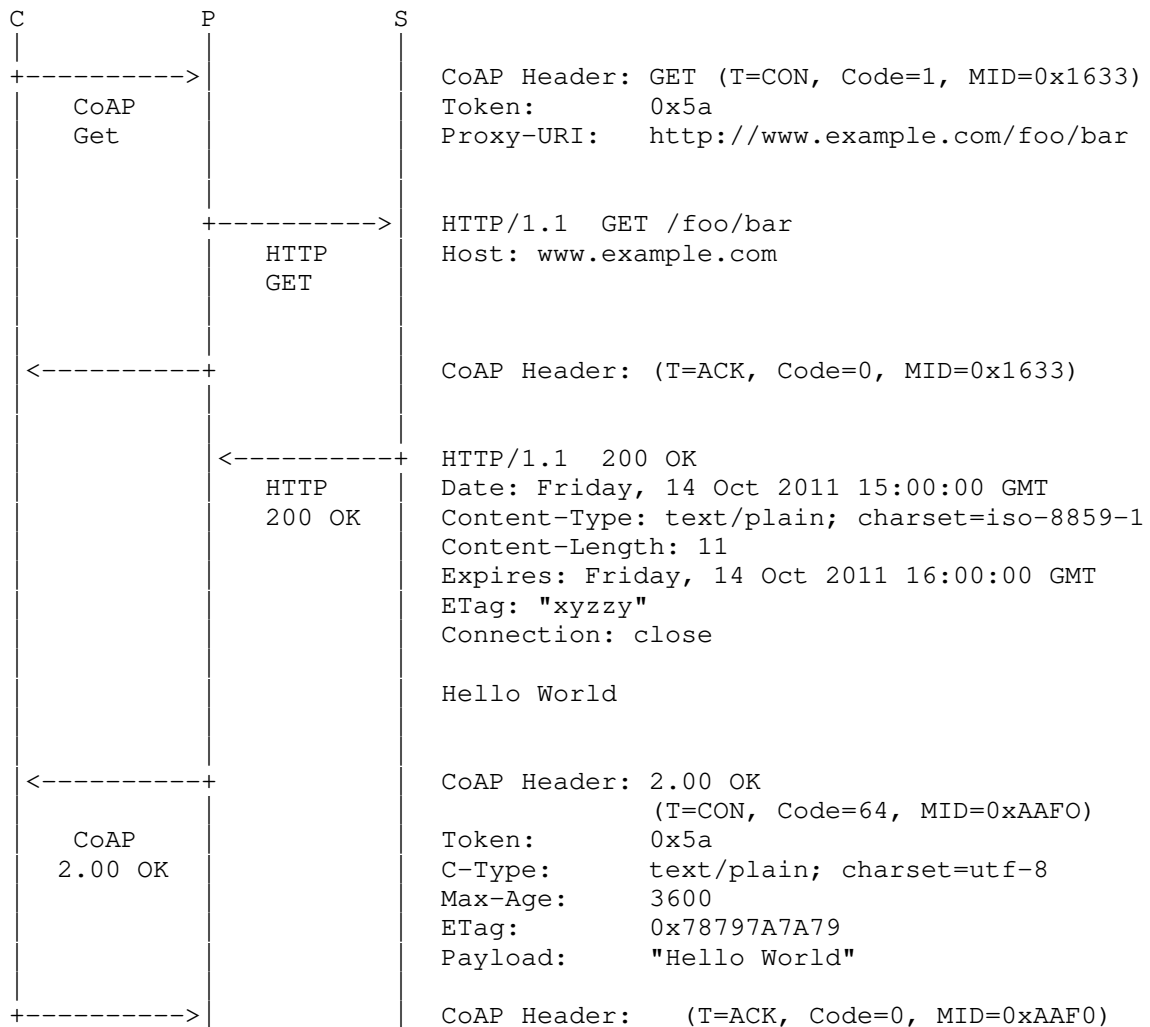


Figure 8: A Basic CoAP-HTTP GET Request

The example in Figure 9 builds on the previous example and shows an implementation of a GET request that includes a previously returned ETag Option. The proxy makes a Conditional Request to the HTTP origin server by including an If-None-Match header field in the HTTP GET Request. The CoAP response indicates that the response stored by the client is fresh. It includes a Max-Age Option calculated from the HTTP response's Expires header field.

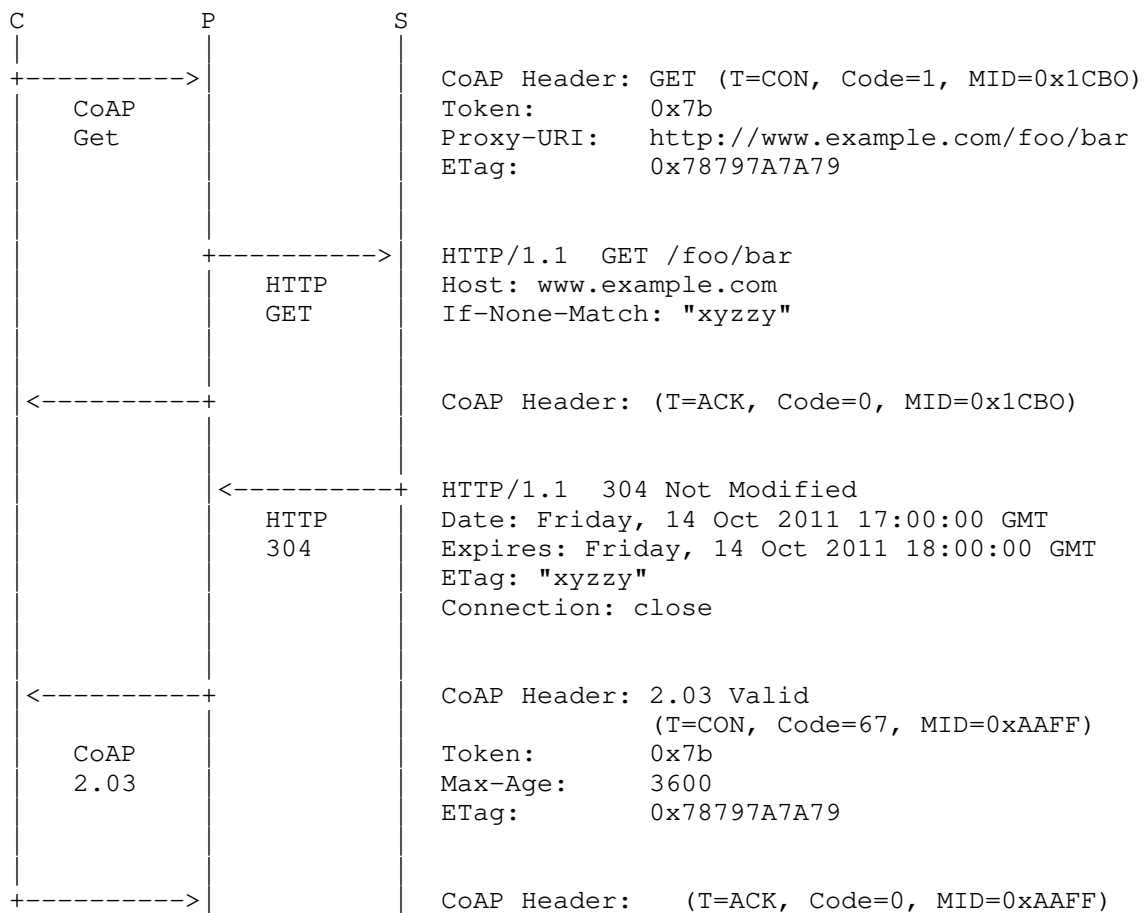


Figure 9: A CoAP-HTTP GET Request with an ETag Option

8. Acknowledgements

TBD.

9. IANA Considerations

This memo includes no request to IANA.

10. Security Considerations

10.1. Cross-protocol Security Policy Mapping

At the moment of this writing, CoAP and HTTP are missing any cross-protocol security policy mapping.

The HC proxy SHOULD flexibly support security policies between the two protocols, possibly as part of the HC URI mapping function, in order to statically map HTTP and CoAP security policies at the proxy (see Appendix A.2 for an example.)

10.2. Subscription

As noted in Section 7 of [I-D.ietf-core-observe], when using the observe pattern, an attacker could easily impose resource exhaustion on a naive server who's indiscriminately accepting observer relationships establishment from clients. The converse of this problem is also present, a malicious client may also target the HC proxy itself, by trying to exhaust the HTTP connection limit of the proxy by opening multiple subscriptions to some CoAP resource.

Effective strategies to reduce success of such a DoS on the HTTP side (by forcing prior identification of the HTTP client via usual web authentication mechanisms), must always be weighted against an acceptable level of usability of the exposed CoAP resources.

11. References

11.1. Normative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-12 (work in progress), June 2013.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-09 (work in progress), May 2013.
- [I-D.ietf-core-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping"

Implementation", draft-ietf-core-http-mapping-00 (work in progress), June 2013.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP",
draft-ietf-core-observe-08 (work in progress),
February 2013.

[I-D.ietf-httpbis-p1-messaging]

Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
(HTTP/1.1): Message Syntax and Routing",
draft-ietf-httpbis-p1-messaging-22 (work in progress),
February 2013.

[I-D.ietf-httpbis-p2-semantics]

Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
(HTTP/1.1): Semantics and Content",
draft-ietf-httpbis-p2-semantics-22 (work in progress),
February 2013.

[I-D.thomson-hybi-http-timeout]

Thomson, M., Loreto, S., and G. Wilkins, "Hypertext
Transfer Protocol (HTTP) Keep-Alive Header",
draft-thomson-hybi-http-timeout-03 (work in progress),
July 2012.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail
Extensions (MIME) Part Two: Media Types", RFC 2046,
November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, January 2005.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom
Syndication Format", RFC 4287, December 2005.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

11.2. Informative References

- [I-D.bormann-core-simple-server-discovery]
Bormann, C., "CoRE Simple Server Discovery",
draft-bormann-core-simple-server-discovery-01 (work in
progress), March 2012.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource
Directory", draft-ietf-core-resource-directory-00 (work in
progress), June 2013.
- [I-D.snell-http-prefer]
Snell, J., "Prefer Header for HTTP",
draft-snell-http-prefer-18 (work in progress),
January 2013.
- [I-D.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building
Control", draft-vanderstok-core-bc-05 (work in progress),
October 2011.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web
Replication and Caching Taxonomy", RFC 3040, January 2001.
- [RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-
Service Considerations", RFC 4732, December 2006.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins,
"Known Issues and Best Practices for the Use of Long
Polling and Streaming in Bidirectional HTTP", RFC 6202,
April 2011.
- [W3C.HTML5]
Hickson, I., "HTML5", World Wide Web Consortium WD (work
in progress) WD-html5-20111018, October 2011,
<<http://dev.w3.org/html5/spec/>>.

Appendix A. Internal Mapping Functions (from an Implementer's
Perspective)

At least three mapping functions have been identified, which take
place at different stages of the HC proxy processing chain, involving
the URL, Content-Type and Security Policy translation.

All these maps are required to have at least URL granularity so that,
in principle, each and every requested URL may be treated as an

independent mapping source.

In the following, the said map functions are characterized via their expected input and output, and a simple, yet sufficiently rich, configuration syntax is suggested.

In the spirit of a document providing implementation guidance, the specification of a map grammar aims at putting the basis for a reusable software component (e.g. a stand-alone C library) that many different proxy implementations can link to, and benefit from.

A.1. URL Map Algorithm

In case the HC proxy is a reverse proxy, i.e. it acts as the origin server in face of the served network, the URL of the resource requested by its clients (perhaps having an 'http' scheme) shall be mapped to the real resource origin (perhaps in the 'coap' scheme).

In case HC is a forward proxy, no URL translation is needed since the client already knows the "real name" of the resource.

An interception HC proxy, instead, MAY use the homogeneous mapping strategy to operate without any pre-configuration need.

As noted in Appendix B of [RFC3986] any correctly formatted URL can be matched by a POSIX regular expression. By leveraging on this property, we suggest a syntax that describes the URL mapping in terms of substituting the regex-matching portions of the requested URL into the mapped URL template.

E.g.: given the source regular expression
'^http://example.com/coap/.*\$' and destination template 'coap://\$1'
(where \$1 stands for the first - and only in this specific case -
substring matched by the regex pattern in the source), the input URL
"http://example.com/coap/node1/resource2" translates to
"coap://node1/resource2".

This is a well established technique used in many today's web components (e.g. Django URL dispatcher, Apache mod_rewrite, etc.), which provides a compact and powerful engine to implement what essentially is an URL rewrite function.

INPUT
* requested URL

OUTPUT
* target URL

SYNTAX
url_map [rule name] {
 requested_url <regex>
 mapped_url <regex match subst template>
}

EXAMPLE 1
url_map homogeneous {
 requested_url '^http://.*\$'
 mapped_url 'coap//\$1'
}

EXAMPLE 2
url_map embedded {
 requested_url '^http://example.com/coap/.*\$'
 mapped_url 'coap//\$1'
}

Note that many different url_map records may be given in order to build the whole mapping function. Each of these records can be queried (in some predefined order) by the HC proxy until a match is found, or the list is exhausted. In the latter case, depending on the mapping policy (only internal, internal then external, etc.) the original request can be refused, or the same mapping query is forwarded to one or more external URL mapping components.

A.2. Security Policy Map Algorithm

In case the "incoming" URL has been successfully translated, the HC proxy must lookup the security policy, if any, that needs to be applied to the request/response transaction carried on the "outgoing" leg.

INPUT

- * target URL (after URL map has been applied)
- * original requester identity (given by cookie, or IP address, or crypto credentials/security context, etc.)

OUTPUT

- * security context that will be applied to access the target URL

SYNTAX

```
sec_map [rule name] {
    target_url      <regex>          -- one or more
    requester_id    <TBD>
    sec_context     <TBD>
}
```

EXAMPLE

```
<TBD>
```

A.3. Content-Type Map Algorithm

In case a set of destination URLs is known as being limited in handling a narrow subset of mime types, a content-type map can be configured in order to let the HC proxy transparently handle the compatible/lossless format translation.

INPUT

- * destination URL (after URL map has been applied)
- * original content-type

OUTPUT

- * mapped content-type

SYNTAX

```
ct_map {
    target_url <regex>          -- one or more targetURLs
    ct_switch  <source_ct, dest_ct> -- one or more CTs
}
```

EXAMPLE

```
ct_map {
    target_url  '^coap://class-1-device/.*$'
    ct_switch  */xml    application/exi
}
```

Authors' Addresses

Angelo P. Castellani
University of Padova
Via Gradenigo 6/B
Padova 35131
Italy

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal H3A 3G4
canada

Phone: +1 514 585 0761
Email: Akbar.Rahman@InterDigital.com

Thomas Fossati
KoanLogic
Via di Sabbiano 11/5
Bologna 40136
Italy

Phone: +39 051 644 82 68
Email: tho@koanlogic.com

Esko Dijk
Philips Research

Email: esko.dijk@philips.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: December 19, 2013

E. Dijk, Ed.
Philips Research
A. Rahman, Ed.
InterDigital Communications, LLC
June 17, 2013

Miscellaneous CoAP Group Communication Topics
draft-dijk-core-groupcomm-misc-04

Abstract

This document contains miscellaneous text around the topic of group communication for the Constrained Application Protocol (CoAP). The first part contains, for reference, text that was removed from the WG version of Group Communication for CoAP draft. The second part describes group communication and multicast functionality that may be input to future standardization in the CoRE WG.

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 <http://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 December 19, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Potential Solutions for Group Communication	3
3. Use Cases	3
4. Requirements	4
4.1. Background	4
4.2. General Requirements	5
4.3. Security Requirements	6
5. Group Communication Solutions	8
5.1. IP Multicast Transmission Methods	8
5.1.1. Serial unicast	8
5.1.2. Unreliable IP Multicast	8
5.1.3. Reliable IP Multicast	8
5.2. Overlay Multicast	9
5.3. CoAP Application Layer Group Management	10
6. DNS-SD Based Group Resource Manipulation	12
7. Group Discovery and Member Discovery	13
7.1. DNS-SD	13
7.2. CoRE Resource Directory	14
8. Deployment Guidelines	14
8.1. Overview	14
8.2. Implementation in Target Network Topologies	14
8.2.1. Single LLN Topology	15
8.2.2. Single LLN with Backbone Topology	17
8.2.3. Multiple LLNs with Backbone Topology	19
8.2.4. LLN(s) with Multiple 6LBRs	19
8.2.5. Conclusions	19
8.3. Implementation Considerations	19
8.3.1. MLD Implementation on LLNs and MLD alternatives	20
8.3.2. 6LBR Implementation	21
8.3.3. Backbone IP Multicast Infrastructure	21
9. Miscellaneous Topics	22
9.1. CoAP Multicast and HTTP Unicast Interworking	22
10. Acknowledgements	23
11. IANA Considerations	23
12. Security Considerations	24
13. References	24
13.1. Normative References	24
13.2. Informative References	25
Appendix A. Multicast Listener Discovery (MLD)	27
Appendix B. CoAP-Observe Alternative to Group Communication	28
Authors' Addresses	28

1. Introduction

This document contains miscellaneous text around the topic of group communication for the Constrained Application Protocol, CoAP [I-D.ietf-core-coap]. The first part of the document (Section 5) contains, for reference, text that was removed from the Group Communication for CoAP [I-D.ietf-core-groupcomm] draft and its predecessor [I-D.rahman-core-groupcomm]. The second part of the document (Section 9) contains text and/or functionality that may be considered for inclusion in [I-D.ietf-core-groupcomm] or otherwise may be input to future standardization in the CoRE WG.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Potential Solutions for Group Communication

The classic concept of group communications is that of a single source distributing content to multiple destination recipients that are all part of a group. Before content can be distributed, there is a separate process to form the group. The source may be either a member or non-member of the group.

Group communication solutions have evolved from "bottom" to "top", i.e., from layer 2 (Media Access Control broadcast/multicast) and layer 3 (IP multicast) to application layer group communication, also referred to as application layer multicast. A study published in 2005 [Lao05] identified new solutions in the "middle" (referred to as overlay multicast) that utilize an infrastructure based on proxies.

Each of these classes of solutions may be compared [Lao05] using metrics such as link stress and level of host complexity [Banerjee01]. The results show for a realistic internet topology that IP Multicast is the most resource-efficient, with the downside being that it requires the most effort to deploy in the infrastructure. IP Multicast is the solution adopted by this draft for CoAP group communication.

3. Use Cases

CoAP group communication can be applied in the context of the following use cases:

- o Discovery of Resource Directory: discovering the local CoRE RD which contains links (URIs) to resources stored on other servers [RFC6690].

- o Lighting Control: synchronous operation of a group of IPv6-connected lights (e.g., 6LoWPAN [RFC4944] lights).
- o Parameter Update: updating parameters/settings simultaneously in a large group of devices in a building/campus control ([I-D.vanderstok-core-bc]) application.
- o Firmware Update: efficiently updating firmware simultaneously in a large group of devices in a building/campus control ([I-D.vanderstok-core-bc]) application. Here, the use of CoAP group communication could be realized via a multicast extension of CoAP blockwise transfer [I-D.ietf-core-block]. This use case and use of multicast is especially valuable if there are time constraints related to the software update for large groups of devices.
- o Group Status Report: requesting status information or event reports from a group of devices in a building/campus control application. In this use case, conditional reporting is required: only device that have events to report (as indicated by the request query) respond, others remain silent. This use case requires reliable CoAP group communication, which is currently not in CoRE WG scope.

4. Requirements

Requirements that a CoAP group communication solution should fulfill can be found in existing documents ([RFC5867], [I-D.ietf-6lowpan-routing-requirements], [I-D.vanderstok-core-bc], and [I-D.shelby-core-coap-req]). Below, a set of high-level requirements is listed that a group communication solution should ideally fulfill. In practice, all these requirements can never be satisfied at once in an LLN context. Furthermore, different use cases will have different needs i.e. an elaboration of a subset of below requirements.

4.1. Background

The requirements for CoAP are documented in [I-D.shelby-core-coap-req]. In this draft, we focus and expand discussions on the requirements pertaining to CoAP "group communication" and "multicast" support as stated in [I-D.shelby-core-coap-req]:

REQ 9: CoAP will support a non-reliable IP multicast message to be sent to a group of Devices to manipulate a resource on all the Devices simultaneously. The use of multicast to query and advertise descriptions must be supported, along with the support of unicast responses.

Currently, the CoAP protocol [I-D.ietf-core-coap] supports unreliable IP multicast using UDP. It defines the unreliable multicast operation as follows in Section 4.5:

"CoAP supports sending messages to multicast destination addresses. Such multicast messages MUST be Non-Confirmable. Some mechanisms for avoiding congestion from multicast requests are being considered in [I-D.eggert-core-congestion-control]."

Additional requirements were introduced in [I-D.vanderstok-core-bc] driven by quality of experience issues in commercial lighting; the need for large numbers of devices to respond with near simultaneity to a command (multicast PUT), and for that command to be received reliably (reliable multicast).

4.2. General Requirements

A CoAP group communication solution should (ideally) meet the following general requirements:

- GEN-REQ 1: Optional Reliability: the application can select between unreliable group communication and reliable group communication.
- GEN-REQ 2: Efficiency: delivers messages more efficiently than a "serial unicast" solution. Provides a balance between group data traffic and control overhead.
- GEN-REQ 3: Low latency: deliver a message as quickly as possible.
- GEN-REQ 4: Synchrony: allows near-simultaneous modification of a resource on all devices in a target group, providing a perceived effect of synchrony or simultaneity. For example a specified time span D such that a message is delivered to all destinations in a time interval $[t, t+D]$.
- GEN-REQ 5: Ordering: message ordering may be required for reliable group communication use cases.
- GEN-REQ 6: Security: see Section 4.3 for security requirements for group communication.

- GEN-REQ 7: Flexibility: support for one or many source(s), both dense and sparse networks, for high or low listener density, small or large number of groups, and multi-group membership.
- GEN-REQ 8: Robust group management: functionality to join groups, leave groups, view group membership, and persistent group membership in failure or sleeping node situations.
- GEN-REQ 9: Network layer independence: a solution is independent from specific unicast and/or IP multicast routing protocols.
- GEN-REQ 10: Minimal specification overhead: a group communication solution should preferably re-use existing/established (IETF) protocols that are suitable for LLN deployments, instead of defining new protocols from scratch.
- GEN-REQ 11: Minimal implementation overhead: e.g. a solution allows to re-use existing (software) components that are already present on constrained nodes such as (typical) 6LoWPAN/CoAP nodes.
- GEN-REQ 12: Mixed backbone/LLN topology support: a solution should work within a single LLN, and in combined LLN/backbone network topologies, including multi-LLN topologies. Both the senders and receivers of CoAP group messages may be attached to different network links or be part of different LLNs, possibly with routers or switches in between group members. In addition, different routing protocols may operate on the LLN and backbone networks. Preferably a solution also works with existing, common backbone IP infrastructure (e.g. switches or routers).
- GEN-REQ 13: CoAP Proxying support: a CoAP proxy can handle distribution of a message to a group on behalf of a (constrained) CoAP client.
- GEN-REQ 14: Suitable for operation on LLNs with constrained nodes.

4.3. Security Requirements

Security for group communications at the IP level has been studied extensively in the IETF MSEC (Multicast Security) WG, and to a lesser extent in the IRTF SAMRG (Scalable Adaptive Multicast Research Group). In particular, [RFC3740], [RFC5374] and [RFC4046] are very instructive. A set of requirements for securing group communications in CoAP were derived from a study of these previous investigations as well as understanding of CoAP specific needs. These are listed below.

A CoAP group communication solution should (ideally) meet the following security requirements:

SEC-REQ 1: Group communications data encryption: Important CoAP group communications shall be encrypted (using a group key) to preserve confidentiality. It shall also be possible to send CoAP group communications in the clear (i.e. unencrypted) for low value data.

SEC-REQ 2: Group communications source data authentication: Important CoAP group communications shall be authenticated by verifying the source of the data (i.e. that it was generated by a given and trusted group member). It shall also be possible to send unauthenticated CoAP group communications for low value data.

SEC-REQ 3: Group communications limited data authentication: Less important CoAP group communications shall be authenticated by simply verifying that it originated from one of the group members (i.e. without explicitly identifying the source node). This is a weaker requirement (but simpler to implement) than REQ2. It shall also be possible to send unauthenticated CoAP group communications for low value data.

SEC-REQ 4: Group key management: There shall be a secure mechanism to manage the cryptographic keys (e.g. generation and distribution) belonging to the group; the state (e.g. current membership) associated with the keys; and other security parameters.

SEC-REQ 5: Use of Multicast IPSec: The CoAP protocol [I-D.ietf-core-coap] allows IPSec to be used as one option to secure CoAP. If IPSec is used as a way to security CoAP communications, then multicast IPSec [RFC5374] should be used for securing CoAP group communications.

SEC-REQ 6: Independence from underlying routing security: CoAP group communication security shall not be tied to the security of underlying routing and distribution protocols such as PIM [RFC4601] and RPL [RFC6550]. Insecure or inappropriate routing (including IP multicast routing) may cause loss of data to CoAP but will not affect the authenticity or secrecy of CoAP group communications.

SEC-REQ 7: Interaction with HTTPS: The security scheme for CoAP group communications shall account for the fact that it may need to interact with HTTPS (Hypertext Transfer Protocol Secure) when a transaction involves a node in the general Internet (non-constrained network) communicating via a HTTP-CoAP proxy.

5. Group Communication Solutions

This section includes the text that describes the solutions of IP multicast, overlay multicast, and application layer group communication which were removed from [I-D.rahman-core-groupcomm] version 07 when the text was transferred to [I-D.ietf-core-groupcomm].

5.1. IP Multicast Transmission Methods

5.1.1. Serial unicast

Even in systems that generally support IP Multicast, there may be certain data links (or transports) that don't support IP multicast. For those links a serial unicast alternative must be provided. This implies that it should be possible to enumerate the members of a group, in order to determine the correct unicast destinations.

5.1.2. Unreliable IP Multicast

The CoRE WG charter specified support for non-reliable IP multicast. In the current CoAP protocol design [I-D.ietf-core-coap], unreliable multicast is realized by the source sending Non-Confirmable messages to a multicast IP address. IP Multicast (using UDP) in itself is unreliable, unless specific reliability features are added to it.

5.1.3. Reliable IP Multicast

[TBD: This is a difficult problem. Need to investigate the benefits of repeating MGET and MPUT requests (saturation) to get "Pretty Good Reliability". Use the same MID or a new MID for repeated requests? Carsten suggests the use of bloom filters to suppress duplicate responses.

One could argue that non-idempotent operations (POST) cannot be supported without a *truly* reliable multicast protocol. However, is this the case? If a multicast POST request is sent repeatedly with the same Message ID (MID), then CoAP nodes that already received it once will ignore duplicates. Sending with Message ID is supported in CoAP for Non-Confirmable messages (thus including multicast messages) as per [I-D.ietf-core-coap] section 4.2.]

Reliable multicast supports guaranteed delivery of messages to a group of nodes. The following specifies the requirements as was proposed originally in version 01 of [I-D.vanderstok-core-bc]:

- o Validity - If sender sends a message, m, to a group, g, of destinations, a path exists between sender and destinations, and

the sender and destinations are correct, all destinations in *g* eventually receive *m*.

- o Integrity - destination receives *m* at most once from sender and only if sender sent *m* to a group including destination.
- o Agreement - If a correct destination of *g* receives *m*, then all correct destinations of *g* receive *m*.
- o Timeliness - For real-time control of devices, there is a known constant *D* such that if *m* is sent at time *t*, no correct destination receives *m* after *t*+*D*.

There are various approaches to achieve reliability, such as

- o Destination node sends response: a destination sends a CoAP Response upon multicast Request reception (it SHOULD be a Non-Confirmable response). The source node may retry a request to destination nodes that did not respond in time with a CoAP response.
- o Route redundancy
- o Source node transmits multiple times (destinations do not respond)

5.2. Overlay Multicast

An alternative group communication solution (to IP Multicast) is an "overlay multicast" approach. We define an overlay multicast as one that utilizes an infrastructure based on proxies (rather than an IP router based IP multicast backbone) to deliver IP multicast packets to end devices. MLD ([RFC3810]) has been selected as the basis for multicast support by the ROLL working group for the RPL routing protocol. Therefore, it is proposed that "IGMP/MLD Proxying" [RFC4605] be used as a basis for an overlay multicast solution for CoAP.

Specifically, a CoAP proxy [I-D.ietf-core-coap] may also contain an MLD Proxy function. All CoAP devices that want to join a given IP multicast group would then send an MLD Join to the CoAP (MLD) proxy. Thereafter, the CoAP (MLD) proxy would be responsible for delivering any IP multicast message to the subscribed CoAP devices. This will require modifications to the existing [RFC4605] functionality.

Note that the CoAP (MLD) proxy may or may not be connected to an external IP multicast enabled backbone. The key function for the CoAP (MLD) proxy is to distribute CoAP generated multicast packets even in the absence of router support for multicast.

5.3. CoAP Application Layer Group Management

Another alternative solution (to IP Multicast and Overlay Multicast) is to define CoAP application level group management primitives. Thus, CoAP can support group management features without need for any underlying IP multicast support.

Interestingly, such group management primitives could also be offered even if there is underlying IP multicast support. This is useful because IP multicast inherently does not support the concept of a group with managed members, while a managed group may be required for some applications.

The following group management primitives are in general useful:

- o discover groups;
- o query group properties (e.g. related resource descriptions);
- o create a group;
- o remove a group;
- o add a group member;
- o remove a group member;
- o enumerate group members;
- o security and access control primitives.

In this proposal a (at least one) CoAP Proxy node is responsible for group membership management. A constrained node can specify which group it intends to join (or leave) using a CoAP request to the appropriate CoAP Proxy. To Join, the group name will be included in optional request header fields (explained below). These header fields will be included in a PUT request to the Proxy. The Proxy-URI is set to the Group Management URI of the Proxy (found previously through the "/.well-known/" resource discovery mechanism). Note that in this solution also CoAP Proxies may exist in a network that are not capable of CoAP group operations.

Group names may be defined as arbitrary strings with a predefined maximum length (e.g. 268 characters or the maximum string length in a CoAP Option), or as URIs.

[TBD: how can a client send a request to a group? Does it only need to know the group name (string or URI) or also an IP multicast

address? One way is to send a CoAP request to the CoAP Proxy with a group URI directly in the Proxy-URI field. This avoids having to know anything related to IP multicast addresses.]

This solution in principle supports both unreliable and reliable group communication. A client would indicate unreliable communication by sending a CoAP Non-Confirmable request to the CoAP Proxy, or reliable communication by sending a CoAP Confirmable request.

It is proposed that CoAP supports two Header Options for group "Join" and "Leave". These Options are Elective so they should be assigned an even number. Assuming the Type for "join" is x (value TBD), the Header Options are illustrated by the table in Figure 1:

Type	C/E	Name	Data type	Length	Default
x	E	Group Join	String	1-270 B	""
x+2	E	Group Leave	String	1-270 B	""

Figure 1: CoAP Header Options for Group Management

Figure 2 illustrates how a node can join or leave a group using the Header Options in a CoAP message:

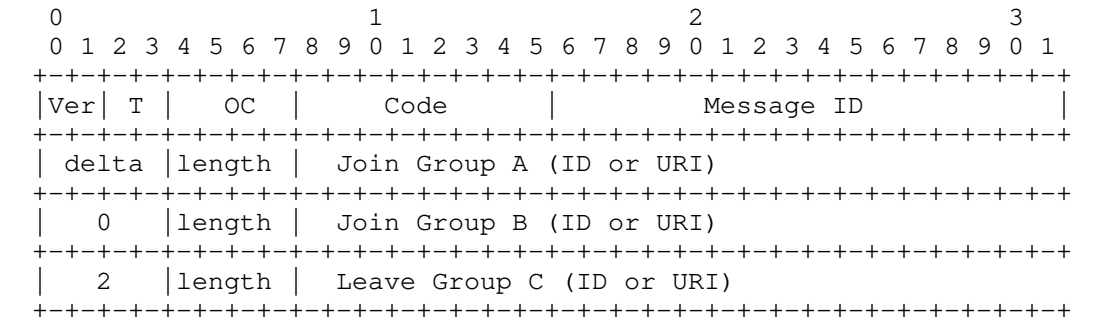


Figure 2: CoAP Message for Group Management

Header Fields for the above example:

Ver: 2-bit unsigned integer for CoAP Version. Set to 1 by implementation as defined by the CoAP specification.

T: 2-bit unsigned integer for CoAP Transaction Type. Either '0' Confirmation or '1' Non-Confirmable can be used for group "join" or "leave" request.

OC: 4-bit unsigned integer for Option Count. For this example, the value should be "3" since there are three option fields.

Code: 8-bit unsigned integer to indicate the Method in a Request or a Response Code in a Response message. Any Code can be used so the group management can be piggy-backed in either Request or Response message.

Message ID: 16-bit value assigned by the source to uniquely identify a pair of Request and Response.

CoAP defines a delta encoding for header options. The first delta is the "Type" for group join in this specific example. If the type for group join is x as illustrated in Figure 2, delta will be x. In the second header option, it is also a group join so the delta is 0. The third header option is a group leave so the delta is 2.

An alternative solution to using Header Options (explained above) is to use designated parameters in the query part of the URI in the Proxy-URI field of a POST (TBD: or PUT?) request to a Proxy's group management service resource advertized by DNS-SD. For example, to join group1 and leave group2:

```
coap://proxy1.bld2.example.com/groupmgt?j=group1&l=group2
```

6. DNS-SD Based Group Resource Manipulation

Ideally, all nodes in a given group (defined by its multicast IP address) must receive the same request with high probability. This will not be the case if there is diversity in the authority port (i.e. a diversity of dynamic port addresses across the group) or if the targeted resource is located at different paths on different nodes. Extending the definition of group membership to include port and path discovery is not desirable.

Therefore, some measures must be present to ensure uniformity in port number and resource name/location within a group.

A first solution in this respect is to couple groups to service descriptions in DNS (using DNS-SD as in [I-D.vanderstok-core-bc]). A service description for a multicast group may have a TXT record in DNS defining a schema X (e.g. "schema=DALI"), which defines by service standard X (e.g. "DALI") which resources a node supporting X MUST have. Therefore a multicast source can safely refer to all resources with corresponding operations as prescribed by standard X. For port numbers (which can be found using DNS-SD also) the same holds. Alternatively, only the default CoAP port may be used in all CoAP multicast requests.

7. Group Discovery and Member Discovery

CoAP defines a resource discovery capability, but does not yet specify how to discover groups (e.g. find a group to join or send a multicast message to) or to discover members of a group (e.g. to address selected group members by unicast). These topics are elaborated in more detail in [I-D.vanderstok-core-dna] including examples for using DNS-SD and CoRE Resource Directory.

7.1. DNS-SD

DNS-based Service Discovery [I-D.cheshire-dnsext-dns-sd] defines a conventional way to configure DNS PTR, SRV, and TXT records to enable enumeration of services, such as services offered by CoAP nodes, or enumeration of all CoAP nodes, within specified subdomains. A service is specified by a name of the form <Instance>.<ServiceType>.<Domain>, where the service type for CoAP nodes is _coap._udp and the domain is a DNS domain name that identifies a group as in the examples above. For each CoAP end-point in a group, a PTR record with the name _coap._udp and/or a PTR record with the name _coap._udp.<Domain> is defined and it points to an SRV record having the <Instance>.<ServiceType>.<Domain> name.

All CoAP nodes in a given subdomain may be enumerated by sending a query for PTR records named _coap._udp to the authoritative DNS server for that zone. A list of SRV records is returned. Each SRV record contains the port and host name (AAAA record) of a CoAP node. The IP address of the node is obtained by resolving the host name. DNS-SD also specifies an optional TXT record, having the same name as the SRV record, which can contain "key=value" attributes. This can be used to store information about the device, e.g. schema=DALI, type=switch, group=lighting.bldg6, etc.

Another feature of DNS-SD is the ability to specify service sub-types using PTR records. For example, one could represent all the CoAP groups in a subdomain by PTR records with the name `_group._sub._coap._udp` or alternatively `_group._sub._coap._udp.<Domain>.`

7.2. CoRE Resource Directory

CoRE Resource Directory [I-D.shelby-core-resource-directory] defines the concept of a Resource Directory (RD) server where CoAP servers can register their resources offered and CoAP clients can discover these resources by querying the RD server. RD syntax can be mapped to DNS-SD syntax and vice versa [I-D.lynn-core-discovery-mapping], such that the above approach can be reused for group discovery and group member discovery.

Specifically, the Domain (d) parameter can be set to the group URI by an end-point registering to the RD. If an end-point wants to join multiple groups, it has to repeat the registration process for each group it wants to join.

8. Deployment Guidelines

8.1. Overview

We recommend to use IP multicast as the base solution for CoAP Group Communication, provided that the use case and network characteristics allow this. It has the advantage that it re-uses the IP multicast suite of protocols and can operate even if group members are distributed over both constrained and un-constrained network segments. Still, this approach may require specifying or implementing additional IP Multicast functionality in an LLN, in a backbone network, or in both - this will be evaluated in more detail in this section.

8.2. Implementation in Target Network Topologies

This section looks in more detail how an IP Multicast based solution can be deployed onto the various network topologies that we consider important for group communication use cases. Note that the chosen solution of IP Multicast for CoAP group communication works mostly independently from the underlying network topology and its specific IP multicast implementation.

Starting from the simplest case of a single LLN topology, we move to more complex topologies involving a backbone network or multiple LLNs. With "backbone" we refer here typically to a corporate LAN or VLAN, which constitutes a single broadcast domain by design. It

could also be an in-home network. A multi-link backbone is also possible, if there is proper IP multicast routing or forwarding configured between these links. (The term 6LoWPAN Border Router or "6LBR" is used here for a border router, though our evaluation is not necessarily restricted to 6LoWPAN networks.)

8.2.1. Single LLN Topology

The simplest topology is a single LLN, where all the IP multicast source(s) and destinations are constrained nodes within this same LLN. Possible implementations of IP multicast routing and group administration for this topology are listed below.

8.2.1.1. Mesh-Under Multicast Routing

The LLN may be set up in either a mesh-under or a route-over configuration. In the former case, the mesh routing protocol should take care of routing IP multicast messages throughout the LLN.

Because conceptually all nodes in the LLN are attached to a single link, there is in principle no need for nodes to announce their interest in multicast IP addresses via MLD (see Appendix A). A multicast message to a specific IP destination, which is delivered to all 6LoWPAN nodes by the mesh routing algorithm, is accepted by the IP network layer of that node only if it is listening on that specific multicast IP address and port.

8.2.1.2. RPL Multicast Routing

The RPL routing protocol for LLNs provides support for routing to multicast IP destinations (Section 12 of [RFC6550]). Like regular unicast destinations, multicast destinations are advertised by nodes using RPL DAO messages. This functionality requires "Storing mode with multicast support" (Mode Of Operation, MOP is 3) in the RPL network.

Once all RPL routing tables in the network are populated, any RPL node can send packets to an IP multicast destination. The RPL protocol performs distribution of multicast packet both upward towards the DODAG root and downwards into the DODAG.

The text in Section 12 of the RPL specification clearly implies that IP multicast packets are distributed using link-layer unicast transmissions, looking at the use of the word "copied" in this section. Specifically in 6LoWPAN networks, this behavior conflicts with the requirement that IP multicast packets MUST be carried as link-layer 802.15.4 broadcast frames [RFC4944].

Assuming that link-layer unicast is indeed meant, this approach seems efficient only in a balanced, sparse tree network topology, or in situations where the fraction of nodes listening to a specific multicast IP address is low, or in duty cycled LLNs where link-layer broadcast is a very expensive operation.

8.2.1.3. RPL Routers with Non-RPL Hosts

Now we consider the case that hosts exist in a RPL network that are not RPL-aware themselves, but rely on RPL routers for their IP connectivity beyond link-local scope. Note that the current RPL specification [RFC6550] leaves this case for future specification (see Section 16.4). Non-RPL hosts cannot advertise their IP multicast groups of interest via RPL DAO messages as defined above. Therefore in that case MLD could be used for such advertisements (State Change Report messages), with all or a subset of RPL routers acting in the role of MLD Routers as defined in [RFC3810]. However, as the MLD protocol is not designed specifically for LLNs it may be a burden for the constrained RPL router nodes to run the full MLD protocol. Alternatives are therefore proposed in Section 8.3.1.

8.2.1.4. Trickle Multicast Forwarding

Trickle Multicast Forwarding [I-D.ietf-roll-trickle-mcast] is an IP multicast routing protocol suitable for LLNs, that uses the Trickle algorithm as a basis. It is a simple protocol in the sense that no topology maintenance is required. It can deal especially well with situations where the node density is a-priori unknown.

Nodes from anywhere in the LLN can be the multicast source, and nodes anywhere in the LLN can be multicast destinations.

Using Trickle Multicast Forwarding it is not required for IP multicast destinations (listeners) to announce their interest in a specific multicast IP address, e.g. by means of MLD. Instead, all multicast IP packets regardless of IP destination address are stored and forwarded by all routers. Because forwarding is always done by multicast, both hosts and routers will be able to receive all multicast IP packets. Routers that receive multicast packets they are not interested in, will only buffer these for a limited time until retransmission can be stopped as specified by the protocol. Hosts that receive multicast packets they are not interested in, will discard multicast packets that are not of interest. Above properties seem to make Trickle especially efficient for cases where the multicast listener density is high and the number of distinct multicast groups relatively low.

8.2.1.5. Other Route-Over Methods

Other known IP multicast routing methods may be used, for example flooding or other to be defined methods suitable for LLNs. An important design consideration here is whether multicast listeners need to advertise their interest in specific multicast addresses, or not. If they do, MLD is a possible option but also protocol-specific means (as in RPL) is an option. See Section 8.3.1 for more efficient substitutes for MLD targeted towards a LLN context.

8.2.2. Single LLN with Backbone Topology

A LLN may be connected via a Border Router (e.g. 6LBR) to a backbone network, on which IP multicast listeners and/or sources may be present. This section analyzes cases in which IP multicast traffic needs to flow from/to the backbone, to/from the LLN.

8.2.2.1. Mesh-Under Multicast Routing

Because in a mesh routing network conceptually all nodes in the LLN are attached to a single link, a multicast IP packet originating in the LLN is typically delivered by the mesh routing algorithm to the 6LBR as well, although there is no guaranteed delivery. The 6LBR may be configured to accept all IP multicast traffic from the LLN and then may forward such packets onto its backbone link. Alternatively, the 6LBR may act in an MLD Router or MLD Snooper role on its backbone link and decide whether to forward a multicast packet or not based on information learned from previous MLD Reports received on its backbone link.

Conversely, multicast packets originating on the backbone network will reach the 6LBR if either the backbone is a single link (LAN/VLAN) or IPv6 multicast routing is enabled on the backbone. Then, the 6LBR could simply forward all IP multicast traffic from the backbone onto the LLN. However, in practice this situation may lead to overload of the LLN caused by unnecessary multicast traffic. Therefore the 6LBR SHOULD only forward traffic that one or more nodes in the LLN have expressed interest in, effectively filtering inbound LLN multicast traffic.

To realize this "filter", nodes on the LLN may use MLD to announce their interest in specific multicast IP addresses to the 6LBR. One option is for the 6LBR to act in an MLD Router role on its LLN interface. However, this may be too much of a "burden" for constrained nodes. Light-weight alternatives for MLD are discussed in Section 8.3.1.

8.2.2.2. RPL Multicast Routing

For RPL routing within the 6LoWPAN, we first consider the case of an IP multicast source on the backbone network with one or more IP multicast listeners on the RPL LLN. Typically, the 6LBR would be the root of a DODAG so that the 6LBR can easily forward the IP multicast packet received on its backbone interface to the right RPL nodes in the LLN down along this DODAG (based on previously DAO-advertized destinations).

Second, a multicast source may be in the RPL LLN and listeners may be both on the LLN and on the backbone. For this case RPL defines that the multicast packet will propagate both up and down the DODAG, eventually reaching the DODAG root (typically a 6LBR) from which the packet can be routed onto the backbone in a manner specified in the previous section.

8.2.2.3. RPL Routers with Non-RPL Hosts

For the case that a RPL LLN contains non-RPL hosts, the solutions from the previous section can be used if in addition RPL routers implement MLD or "MLD like" functionality similar to as described in Section 8.2.1.3.

8.2.2.4. Trickle Multicast Forwarding

First, we consider the case of an IP multicast source node on the LLN (where all 6LRs support Trickle Multicast Forwarding) and IP multicast listeners that may be on the LLN and on the backbone. As Trickle will eventually deliver multicast packets also to a 6LBR, which acts as a Trickle Multicast router as well, the 6LBR can then forward onto the backbone in the ways described earlier in Section 8.2.2.1.

Second, for the case of an IP multicast source on the backbone and multicast listeners on both backbone and/or LLN, the 6LBR needs to forward multicast traffic from the backbone onto the LLN. Here, the aforementioned problem (Section 8.2.2.1) of potentially overloading the LLN with unwanted backbone IP multicast traffic appears again.

A possible solution to this is (again) to let multicast listeners advertise their interest using MLD as described in Section 8.2.2.1 or to use an MLD alternative suitable for LLNs as described in Section 8.3.1. However, following this approach requires possibly an extension to Trickle Multicast Forwarding: the protocol should ensure that MLD-advertised information is somehow communicated to the 6LBR, possibly over multiple hops. MLD itself supports link-local communication only.

8.2.2.5. Other Route-Over Methods

For other multicast routing methods used on the LLN, there are similar considerations to the ones in sections above: the strong need to filter IP multicast traffic coming into the LLN, the need for reporting multicast listener interest (e.g. with MLD or a to-be-defined MLD alternative) by constrained (6LoWPAN) nodes, and the need for LLN-internal routing as identified in the previous section such that the MLD communicated information can reach the 6LBR to be used there in multicast traffic filtering decisions.

8.2.3. Multiple LLNs with Backbone Topology

Now the case of a single backbone network with two or more LLNs attached to it via 6LBRs is considered. For this case all the considerations and solutions of the previous section can be applied.

For the specific case that a source on a backbone network has to send to a very large number of destination located on many LLNs, the use of IGMP/MLD Proxying [RFC4605] with a leaf IGMP/MLD Proxy located in each 6LBR may be useful. This method only is defined for a tree topology backbone network with the IP multicast source at the root of the tree.

8.2.4. LLN(s) with Multiple 6LBRs

[TBD: an LLN with multiple 6LBRs may require some additional consideration. Any need to synchronize mutually on multicast listener information?]

8.2.5. Conclusions

For all network topologies that were evaluated, CoAP group communication can be in principle supported with IP Multicast, making use of existing protocols. For the case of Trickle Multicast Forwarding, it appears that an addition to the protocol is required such that information about multicast listeners can be distributed towards the 6LBR. Opportunities were identified for an "MLD-like" or "MLD-lightweight" protocol specifically suitable for LLNs, which should inter-work with regular MLD on the backbone network. Such MLD variants are further analyzed in Section 8.3.1.

8.3. Implementation Considerations

In this section various implementation aspects are considered such as required protocol implementations, additional functionality of the 6LBR and backbone network equipment.

8.3.1. MLD Implementation on LLNs and MLD alternatives

In previous sections, it was mentioned that the MLDv2 protocol [RFC3810] may be too costly for use in a LLN. MLD relies on periodic link-local multicast operations to maintain state. Also it is optimized to fairly dynamic situations where multicast listeners may come and go over time. Such dynamic situations are less frequently found in typical LLN use cases such as building control, where multicast group membership can remain constant over longer periods of time (e.g. months) after commissioning.

Hence, a viable strategy is to implement a subset of MLD functionality in 6LoWPAN nodes which is just enough for the required functionality. A first option is that 6LoWPAN Routers, like MLD Snoopers, passively listen to MLD State Change Report messages and handle the learned ("snooped") IP multicast destinations in the way defined by the multicast routing protocol they are running (e.g. for RPL, Routers advertise these destinations using DAO messages).

A second option is to use MLD as-is but adapt the recommended parameter values such that operation on a LLN becomes more efficient. [RFC6636] could be a guideline in this case.

A third option is to standardize a new protocol, taking a subset of MLD functionality into a "MLD for 6LoWPAN" protocol to support constrained nodes optimally.

A fourth option is now presented, which seems attractive in that it minimizes standardization, implementation and network communication overhead all at the same time. This option is to specify a new Multicast Listener Option (MLO) as an addition to the 6LoWPAN-ND [RFC6775] protocol communication that is anyway ongoing between a 6LoWPAN host and router(s). This MLO is preferably designed to be maximally similar to the Address Registration Option (ARO), which minimizes the need for additional program code on constrained nodes. With an MLO, instead of registering a host's unicast IP address as with ARO, a host "registers" its interest in a multicast IPv6 address. Unlike the ARO, multiple MLO can be used in the same ND packet. A registration period is also defined in the MLO just like in the ARO. MLO allows a host to persistently register as a listener to IP multicast traffic and to avoid the overhead of periodic multicast communication which is required for the regular MLD protocol.

[TBD: consider what aspects are needed/not needed for CoAP/LLN applications. Will MLDv1 suffice? What to do with options like 'source specific' and include/exclude. Source-specific can also be dealt with at the destination host by filtering? Do we need limits

on number of records per packet? Do we need a higher MLD reliability setting - see the parameters in the MLD RFC]

8.3.2. 6LBR Implementation

To support mixed backbone/LLN scenarios in CoAP group communication, it is RECOMMENDED that a 6LowPAN Border Router (6LBR) will act in an MLD Router role on the backbone link. If this is not possible then the 6LBR SHOULD be configured to act as an MLD Multicast Address Listener and/or MLD Snooper on the backbone link.

8.3.3. Backbone IP Multicast Infrastructure

For corporate/professional applications, most routing and switching equipment that is currently on the market is IPv6 capable. For that reason backbone infrastructure operating IPv4 only is considered out of scope in this document, at least for the backbone network segment(s) where IP multicast destinations are present. What is still in scope is for example an IPv4-only HTTP client that wants to send a group communication message via a HTTP-CoAP proxy as considered in [I-D.castellani-core-advanced-http-mapping].

The availability of, and requirements for, IP multicast support may depend on the specific installation use case. For example, the following cases may be relevant for new IP based building control installations:

1. System deployed on existing IP (Ethernet/WiFi/...) infrastructure, shared with existing IP devices (PCs)
2. Newly designed and deployed IP (Ethernet/WiFi/...) infrastructure, to be shared with other IP devices (PCs)
3. Newly designed and deployed IP (Ethernet/WiFi/...) infrastructure, exclusively used for building control.

Besides physical separation the building control backbone can be separated from regular (PC) infrastructure by using a different VLAN. A typical corporate installation will have many LAN switches and/or routing switches, which pass through IP multicast traffic but on the other hand do not support acting in the Router role of MLD/IGMP. Perhaps for case 2) and 3) above it is acceptable to add a MLD/IGMP capable router somewhere in the network, while for case 1) this may not be the case.

[TBD: consider the influence of WiFi based backbone networks. What if 6LBRs are at the same time also WiFi routers? What if 6LBRs have an Ethernet connection to legacy WiFi routers? Check if equivalent with Ethernet backbone.]

9. Miscellaneous Topics

This section collects miscellaneous text, topics or proposals related to CoAP group communication which do not directly fit into any of the preceding sections.

9.1. CoAP Multicast and HTTP Unicast Interworking

CoAP supports operation over UDP multicast, while HTTP does not. For use cases where it is required that CoAP group communication is initiated from an HTTP end-point, it would be advantageous if the HTTP-CoAP Proxy supports mapping of HTTP unicast to CoAP group communication based on IP multicast. One possible way of operation of such HTTP-CoAP Proxy is illustrated in Figure 3. Note that this topic is covered in more detail in [I-D.castellani-core-advanced-http-mapping].

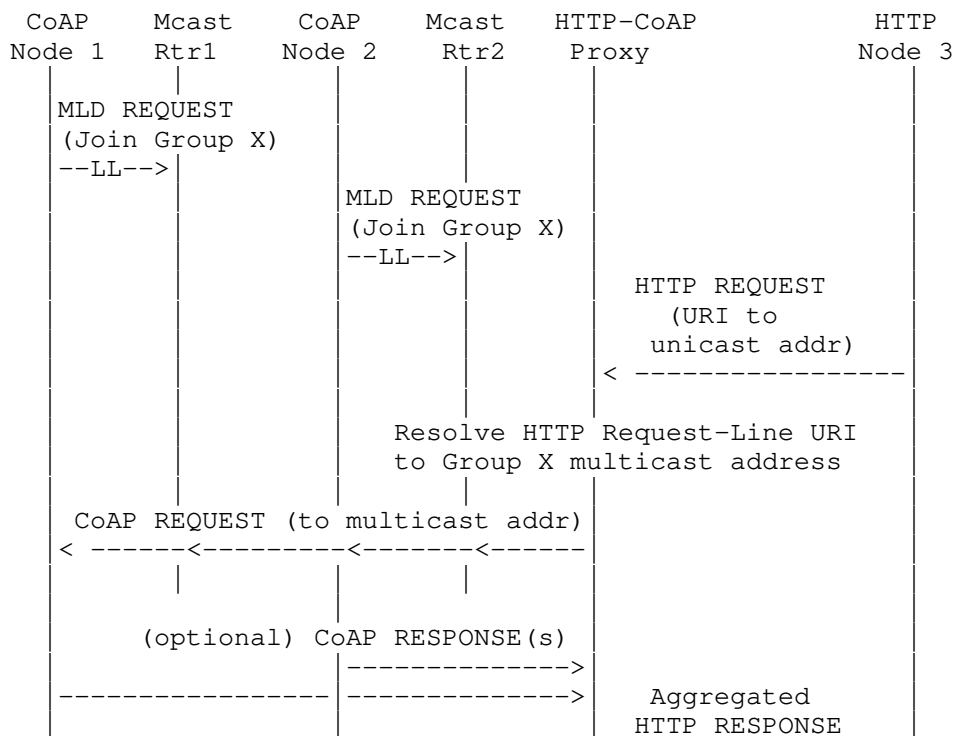




Figure 3: CoAP Multicast and HTTP Unicast Interworking

Note that Figure 3 illustrates the case of IP multicast as the underlying group communications mechanism. MLD denotes the Multicast Listener Discovery protocol ([RFC3810], Appendix A) and LL denotes a Link-Local multicast.

A key point in Figure 3 is that the incoming HTTP Request (from node 3) will carry a Host request-header field that resolves in the general Internet to the proxy node. At the proxy node, this hostname and/or the Request-Line URI will then possibly be mapped (as detailed in [I-D.castellani-core-advanced-http-mapping]) and again resolved (with the CoAP scheme) to an IP multicast address. This may be accomplished, for example, by using DNS or DNS-SD (Section 7). The proxy node will then IP multicast the CoAP Request (corresponding to the received HTTP Request) via multicast routers to the appropriate nodes (i.e. nodes 1 and 2).

In terms of the HTTP Response, Figure 3 illustrates that it will be generated by the proxy node based on aggregated responses of the CoAP nodes and sent back to the client in the general Internet that sent the HTTP Request (i.e. node 1). In

[I-D.castellani-core-advanced-http-mapping] the HTTP Response that the Proxy may use to aggregate multiple CoAP responses is described in more detail. So in terms of overall operation, the CoAP proxy can be considered to be a "non-transparent" proxy according to [RFC2616]. Specifically, [RFC2616] states that a "non-transparent proxy is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction or anonymity filtering."

An alternative to the above is using a Forward Proxy. In this case, the CoAP request URI is carried in the HTTP Request-Line (as defined in [I-D.ietf-core-coap] Section 10.2) in a HTTP request sent to the IP address of the Proxy.

10. Acknowledgements

Thanks to all CoRE WG members who participated in the IETF 82 discussions, which was the trigger to initiate this document.

11. IANA Considerations

This memo includes no request to IANA.

12. Security Considerations

Security aspects of group communication for CoAP are discussed in [I-D.ietf-core-groupcomm]. The current document contains no new proposals yet, for which security considerations have to be analyzed here.

13. References

13.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-17 (work in progress), May 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, March 2004.
- [RFC3810] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.
- [RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, April 2005.
- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.
- [RFC4605] Fenner, B., He, H., Haberman, B., and H. Sandick, "Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP /MLD Proxying")", RFC 4605, August 2006.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

- [RFC5374] Weis, B., Gross, G., and D. Ignjatic, "Multicast Extensions to the Security Architecture for the Internet Protocol", RFC 5374, November 2008.
- [RFC5867] Martocci, J., De Mil, P., Riou, N., and W. Vermeylen, "Building Automation Routing Requirements in Low-Power and Lossy Networks", RFC 5867, June 2010.
- [RFC6550] Winter, T., Thubert, P., 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, March 2012.
- [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, May 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.

13.2. Informative References

- [Banerjee01] Banerjee, B. and B. Bhattacharjee, "A Comparative Study of Application Layer Multicast Protocols ", 2001, <<http://wmedia.grnet.gr/P2PBackground/a-comparative-study-ofALM.pdf>>.
- [I-D.castellani-core-advanced-http-mapping] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-advanced-http-mapping-01 (work in progress), January 2013.
- [I-D.cheshire-dnsext-dns-sd] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-11 (work in progress), December 2011.
- [I-D.eggert-core-congestion-control]

Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.

[I-D.ietf-6lowpan-routing-requirements]

Kim, E., Kaspar, D., Gomez, C., and C. Bormann, "Problem Statement and Requirements for 6LoWPAN Routing", draft-ietf-6lowpan-routing-requirements-10 (work in progress), November 2011.

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-11 (work in progress), March 2013.

[I-D.ietf-core-groupcomm]

Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-09 (work in progress), May 2013.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.

[I-D.ietf-roll-trickle-mcast]

Hui, J. and R. Kelsey, "Multicast Protocol for Low power and Lossy Networks (MPL)", draft-ietf-roll-trickle-mcast-04 (work in progress), February 2013.

[I-D.lynn-core-discovery-mapping]

Lynn, K. and Z. Shelby, "CoRE Link-Format to DNS-Based Service Discovery Mapping", draft-lynn-core-discovery-mapping-02 (work in progress), October 2012.

[I-D.rahman-core-groupcomm]

Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-rahman-core-groupcomm-07 (work in progress), October 2011.

[I-D.shelby-core-coap-req]

Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R. Kelsey, "CoAP Requirements and Features", draft-shelby-core-coap-req-02 (work in progress), October 2010.

[I-D.shelby-core-resource-directory]

Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-shelby-core-resource-directory-05 (work in progress), February 2013.

[I-D.vanderstok-core-bc]

Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-05 (work in progress), October 2011.

[I-D.vanderstok-core-dna]

Stok, P., Lynn, K., and A. Brandt, "CoRE Discovery, Naming, and Addressing", draft-vanderstok-core-dna-02 (work in progress), July 2012.

[Lao05] Lao, L., Cui, J., Gerla, M., and D. Maggiorini, "A Comparative Study of Multicast Protocols: Top, Bottom, or In the Middle? ", 2005, <http://www.cs.ucla.edu/NRL/hpi/AggMC/papers/comparison_gi_2005.pdf>.

Appendix A. Multicast Listener Discovery (MLD)

In order to extend the scope of IP multicast beyond link-local scope, an IP multicast routing protocol has to be active in routers on an LLN. To achieve efficient multicast routing (i.e. avoid always flooding multicast IP packets), routers have to learn which hosts need to receive packets addressed to specific IP multicast destinations.

The Multicast Listener Discovery (MLD) protocol [RFC3810] (or its IPv4 pendant IGMP) is today the method of choice used by an (IP multicast enabled) router to discover the presence of multicast listeners on directly attached links, and to discover which multicast addresses are of interest to those listening nodes. MLD was specifically designed to cope with fairly dynamic situations in which multicast listeners may join and leave at any time.

IGMP/MLD Snooping is a technique implemented in some corporate LAN routing/switching devices. An MLD snooping switch listens to MLD State Change Report messages from MLD listeners on attached links. Based on this, the switch learns on what LAN segments there is interest for what IP multicast traffic. If the switch receives at some point an IP multicast packet, it uses the stored information to decide onto which LAN segment(s) to send the packet. This improves network efficiency compared to the regular behavior of forwarding every incoming multicast packet onto all LAN segments. An MLD snooping switch may also send out MLD Query messages (which is normally done by a device in MLD Router role) if no MLD Router is present.

[RFC6636] discusses optimal tuning of the parameters of MLD for routers for mobile and wireless networks. These guidelines may be useful when implementing MLD in LLNs.

Appendix B. CoAP-Observe Alternative to Group Communication

The CoAP Observation extension [I-D.ietf-core-observe] can be used as a simple (but very limited) alternative for group communication. A group in this case consists of a CoAP server hosting a specific resource, plus all CoAP clients observing that resource. The server is the only group member that can send a group message. It does this by modifying the state of a resource under observation and subsequently notifying its observers of the change. Serial unicast is used for sending the notifications. This approach can be a simple alternative for networks where IP multicast is not available or too expensive.

The CoAP-Observe approach is unreliable in the sense that, even though Confirmable CoAP messages may be used, there are no guarantees that an update will be received. For example, a client may believe it is observing a resource while in reality the server rebooted and lost its listener state.

Authors' Addresses

Esko Dijk (editor)
Philips Research

Email: esko.dijk@philips.com

Akbar Rahman (editor)
InterDigital Communications, LLC

Email: Akbar.Rahman@InterDigital.com

CoRE
Internet-Draft
Intended status: Informational
Expires: March 15, 2014

O. Garcia-Morchon
S. Kumar
Philips Research
S. Keoh
University of Glasgow
R. Hummen
RWTH Aachen
R. Struik
Struik Consultancy
September 11, 2013

Security Considerations in the IP-based Internet of Things
draft-garcia-core-security-06

Abstract

A direct interpretation of the Internet of Things concept refers to the usage of standard Internet protocols to allow for human-to-thing or thing-to-thing communication. Although the security needs are well-recognized, it is still not fully clear how existing IP-based security protocols can be applied to this new setting. This Internet-Draft first provides an overview of security architecture, its deployment model and general security needs in the context of the lifecycle of a thing. Then, it presents challenges and requirements for the successful roll-out of new applications and usage of standard IP-based security protocols when applied to get a functional Internet of Things.

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 <http://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 September 12, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Conventions and Terminology Used in this Document	4
2. Introduction	4
3. The Thing Lifecycle and Architectural Considerations	5
3.1. Threat Analysis	6
3.2. Security Aspects	10
4. State of the Art	13
4.1. IP-based Security Solutions	13
4.2. Wireless Sensor Network Security and Beyond	15
5. Challenges for a Secure Internet of Things	16
5.1. Constraints and Heterogeneous Communication	16
5.1.1. Tight Resource Constraints	16
5.1.2. Denial-of-Service Resistance	18
5.1.3. Protocol Translation and End-to-End Security	18
5.2. Bootstrapping of a Security Domain	20
5.2.1. Distributed vs. Centralized Architecture and Operation	20
5.2.2. Bootstrapping a thing's identity and keying materials	21
5.2.3. Privacy-aware Identification	22
5.3. Operation	23
5.3.1. End-to-End Security	23
5.3.2. Group Membership and Security	23
5.3.3. Mobility and IP Network Dynamics	24
6. Security Suites for the IP-based Internet of Things	25
6.1. Security Architecture	29
6.2. Security Model	30
6.3. Security Bootstrapping and Management	31
6.4. Network Security	33
6.5. Application Security	34
7. Next Steps towards a Flexible and Secure Internet of Things .	36
8. Security Considerations	40
9. IANA Considerations	40
10. Acknowledgements	40
11. References	40
11.1. Informative References	40
Authors' Addresses	45

1. Conventions and Terminology Used in this Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

2. Introduction

The Internet of Things (IoT) denotes the interconnection of highly heterogeneous networked entities and networks following a number of communication patterns such as: human-to-human (H2H), human-to-thing (H2T), thing-to-thing (T2T), or thing-to-things (T2Ts). The term IoT was first coined by the Auto-ID center [AUTO-ID] in 1999. Since then, the development of the underlying concepts has ever increased its pace. Nowadays, the IoT presents a strong focus of research with various initiatives working on the (re)design, application, and usage of standard Internet technology in the IoT.

The introduction of IPv6 and web services as fundamental building blocks for IoT applications [RFC6568] promises to bring a number of basic advantages including: (i) a homogeneous protocol ecosystem that allows simple integration with Internet hosts; (ii) simplified development of very different appliances; (iii) an unified interface for applications, removing the need for application-level proxies. Such features greatly simplify the deployment of the envisioned scenarios ranging from building automation to production environments to personal area networks, in which very different things such as a temperature sensor, a luminaire, or an RFID tag might interact with each other, with a human carrying a smart phone, or with backend services.

This Internet Draft presents an overview of the security aspects of the envisioned all-IP architecture as well as of the lifecycle of an IoT device, a thing, within this architecture. In particular, we review the most pressing aspects and functionalities that are required for a secure all-IP solution.

With this, this Internet-Draft pursues several goals. First, we aim at presenting a comprehensive view of the interactions and relationships between an IoT application and security. Second, we aim at describing challenges for a secure IoT in the specific context of the lifecycle of a resource-constrained device. The final goal of this draft is to discuss the next steps towards a secure IoT.

The rest of the Internet-Draft is organized as follows. Section 3 depicts the lifecycle of a thing and gives general definitions for

the main security aspects within the IoT domain. In Section 4, we review existing protocols and work done in the area of security for wireless sensor networks. Section 5 identifies general challenges and needs for an IoT security protocol design and discusses existing protocols and protocol proposals against the identified requirements. Section 6 proposes a number of illustrative security suites describing how different applications involve distinct security needs. Section 7 includes final remarks and conclusions.

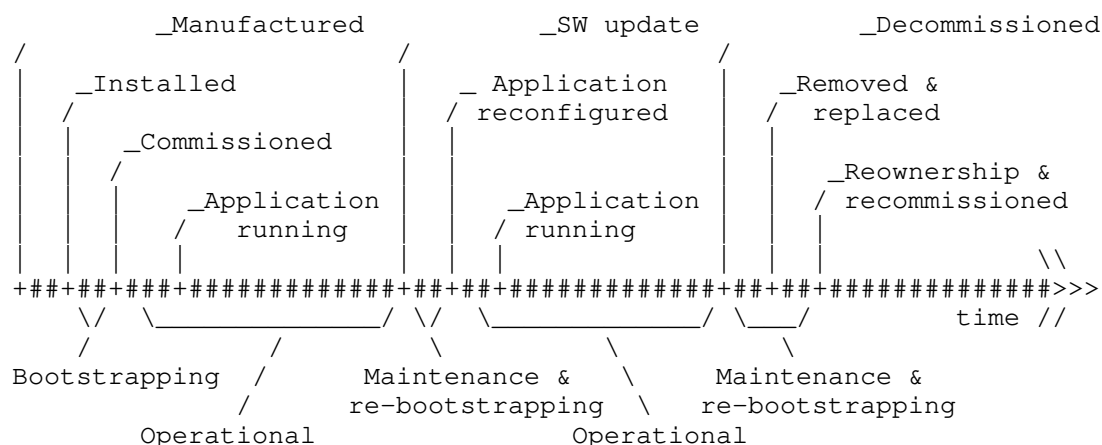
3. The Thing Lifecycle and Architectural Considerations

We consider the installation of a Building Automation and Control (BAC) system to illustrate the lifecycle of a thing in a BAC scenario. A BAC system consists of a network of interconnected nodes that perform various functions in the domains of HVAC (Heating, Ventilating, and Air Conditioning), lighting, safety etc. The nodes vary in functionality and a majority of them represent resource constrained devices such as sensors and luminaries. Some devices may also be battery operated or battery-less nodes, demanding for a focus on low energy consumption and on sleeping devices.

In our example, the life of a thing starts when it is manufactured. Due to the different application areas (i.e., HVAC, lighting, safety) nodes are tailored to a specific task. It is therefore unlikely that one single manufacturer will create all nodes in a building. Hence, interoperability as well as trust bootstrapping between nodes of different vendors is important. The thing is later installed and commissioned within a network by an installer during the bootstrapping phase. Specifically, the device identity and the secret keys used during normal operation are provided to the device during this phase. Different subcontractors may install different IoT devices for different purposes. Furthermore, the installation and bootstrapping procedures may not be a defined event but may stretch over an extended period of time. After being bootstrapped, the device and the system of things are in operational mode and run the functions of the BAC system. During this operational phase, the device is under the control of the system owner. For devices with lifetimes spanning several years, occasional maintenance cycles may be required. During each maintenance phase, the software on the device can be upgraded or applications running on the device can be reconfigured. The maintenance tasks can thereby be performed either locally or from a backend system. Depending on the operational changes of the device, it may be required to re-bootstrap at the end of a maintenance cycle. The device continues to loop through the operational phase and the eventual maintenance phase until the device is decommissioned at the end of its lifecycle. However, the end-of-life of a device does not necessarily mean that it is defective but

rather denotes a need to replace and upgrade the network to next-generation devices in order to provide additional functionality. Therefore the device can be removed and re-commissioned to be used in a different network under a different owner by starting the lifecycle over again. Figure 1 shows the generic lifecycle of a thing. This generic lifecycle is also applicable for IoT scenarios other than BAC systems.

At present, BAC systems use legacy building control standards such as BACNet [BACNET] or DALI [DALI] with independent networks for each subsystem (HVAC, lighting, etc.). However, this separation of functionality adds further complexity and costs to the configuration and maintenance of the different networks within the same building. As a result, more recent building control networks employ IP-based standards allowing seamless control over the various nodes with a single management system. While allowing for easier integration, this shift towards IP-based standards results in new requirements regarding the implementation of IP security protocols on constrained devices and the bootstrapping of security keys for devices across multiple manufacturers.



The lifecycle of a thing in the Internet of Things.

Figure 1

3.1. Threat Analysis

This section explores the security threats and vulnerabilities of a network of things in the IoTs. Security threats have been analyzed in related IP protocols including HTTPS [RFC2818], 6LoWPAN [RFC4919], ANCP [RFC5713], DNS security threats [RFC3833], SIP [RFC3261], IPv6

ND [RFC3756], and PANA [RFC4016]. Nonetheless, the challenge is about their impacts on scenarios of the IoTs. In this section, we specifically discuss the threats that could compromise an individual thing, or network as a whole, with regard to different phases in the thing's lifecycle. Note that these set of threats might go beyond the scope of Internet protocols but we gather them here for the sake of completeness.

- 1 Cloning of things: During the manufacturing process of a thing, an untrusted manufacturer can easily clone the physical characteristics, firmware/software, or security configuration of the thing. Subsequently, such a cloned thing may be sold at a cheaper price in the market, and yet be still able to function normally, as a genuine thing. For example, two cloned devices can still be associated and work with each other. In the worst case scenario, a cloned device can be used to control a genuine device. One should note here, that an untrusted manufacturer may also change functionality of the cloned thing, resulting in degraded functionality with respect to the genuine thing (thereby, inflicting potential reputational risk to the original thing manufacturer). Moreover, it can implement additional functionality with the cloned thing, such as a backdoor.
- 2 Malicious substitution of things: During the installation of a thing, a genuine thing may be substituted with a similar variant of lower quality without being detected. The main motivation may be cost savings, where the installation of lower-quality things (e.g., non-certified products) may significantly reduce the installation and operational costs. The installers can subsequently resell the genuine things in order to gain further financial benefits. Another motivation may be to inflict reputational damage on a competitor's offerings.
- 3 Eavesdropping attack: During the commissioning of a thing into a network, it may be susceptible to eavesdropping, especially if operational keying materials, security parameters, or configuration settings, are exchanged in clear using a wireless medium. After obtaining the keying material, the attacker might be able to recover the secret keys established between the communicating entities (e.g., H2T, T2Ts, or Thing to the backend management system), thereby compromising the authenticity and confidentiality of the communication channel, as well as the authenticity of commands and other traffic exchanged over this communication channel. When the network is in operation, T2T communication may be eavesdropped upon if the communication channel is not sufficiently protected or in the event of session key compromise due to a long period of usage without key renewal or updates.

- 4 **Man-in-the-middle attack:** The commissioning phase may also be vulnerable to man-in-the-middle attacks, e.g., when keying material between communicating entities is exchanged in the clear and the security of the key establishment protocol depends on the tacit assumption that no third party is able to eavesdrop on or sit in between the two communicating entities during the execution of this protocol. Additionally, device authentication or device authorization may be nontrivial, or may need support of a human decision process, since things usually do not have a priori knowledge about each other and can, therefore, not always be able to differentiate friends and foes via completely automated mechanisms. Thus, even if the key establishment protocol provides cryptographic device authentication, this knowledge on device identities may still need complementing with a human-assisted authorization step (thereby, presenting a weak link and offering the potential of man-in-the-middle attacks this way).
- 5 **Firmware Replacement attack:** When a thing is in operation or maintenance phase, its firmware or software may be updated to allow for new functionality or new features. An attacker may be able to exploit such a firmware upgrade by replacing the thing's with malicious software, thereby influencing the operational behaviour of the thing. For example, an attacker could add a piece of malicious code to the firmware that will cause it to periodically report the energy usage of the lamp to a data repository for analysis.
- 6 **Extraction of security parameters:** A thing deployed in the ambient environment (such as sensors, actuators, etc.) is usually physically unprotected and could easily be captured by an attacker. Such an attacker may then attempt to extract security information such as keys (e.g., device's key, private-key, group key) from this thing or try and re-program it to serve his needs. If a group key is used and compromised this way, the whole network may be compromised as well. Compromise of a thing's unique key has less security impact, since only the communication channels of this particular thing in question are compromised. Here, one should caution that compromise of the communication channel may also compromise all data communicated over this channel. In particular, one has to be weary of, e.g., compromise of group keys communicated over this channel (thus, leading to transitive exposure ripple effects).
- 7 **Routing attack:** As highlighted in [ID-Daniel], routing information in IoT can be spoofed, altered, or replayed, in order to create routing loops, attract/repel network traffic, extend/shorten source routes, etc. Other relevant routing attacks

include 1) Sinkhole attack (or blackhole attack), where an attacker declares himself to have a high-quality route/path to the base station, thus allowing him to do anything to all packets passing through it. 2) Selective forwarding, where an attacker may selectively forward packets or simply drop a packet. 3) Wormhole attack, where an attacker may record packets at one location in the network and tunnel them to another location, thereby influencing perceived network behaviour and potentially distorting statistics, thus greatly impacting the functionality of routing. 4) Sybil attack, whereby an attacker presents multiple identities to other things in the network.

- 8 Privacy threat: The tracking of a thing's location and usage may pose a privacy risk to its users. An attacker can infer information based on the information gathered about individual things, thus deducing behavioural patterns of the user of interest to him. Such information can subsequently be sold to interested parties for marketing purposes and targeted advertizing.
- 9 Denial-of-Service attack: Typically, things have tight memory and limited computation, they are thus vulnerable to resource exhaustion attack. Attackers can continuously send requests to be processed by specific things so as to deplete their resources. This is especially dangerous in the IoTs since an attacker might be located in the backend and target resource-constrained devices in an LLN. Additionally, DoS attack can be launched by physically jamming the communication channel, thus breaking down the T2T communication channel. Network availability can also be disrupted by flooding the network with a large number of packets.

The following table summarizes the security threats we identified above and the potential point of vulnerabilities at different layers of the communication stack. We also include related RFCs that include a threat model that might apply to the IoTs.

	Manufacturing	Installation/ Commissioning	Operation
Thing's Model	Device Cloning	Substitution	Privacy threat Extraction of security params
Application Layer		RFC2818 RFC4016	RFC2818, Firmware replacement
Transport Layer		Eavesdropping & Man-in-the-middle attack	Eavesdropping Man-in-the-middle
Network Layer		RFC4919, RFC5713 RFC3833, RFC3756	RFC4919, DoS attack Routing attack RFC3833
Physical Layer			DoS attack

The security threat analysis

Figure 2

3.2. Security Aspects

The term security subsumes a wide range of different concepts. In the first place, it refers to the basic provision of security services including confidentiality, authentication, integrity, authorization, non-repudiation, and availability, and some augmented services, such as duplicate detection and detection of stale packets (timeliness). These security services can be implemented by a combination of cryptographic mechanisms, such as block ciphers, hash functions, or signature algorithms, and non-cryptographic mechanisms, which implement authorization and other security policy enforcement aspects. For each of the cryptographic mechanisms, a solid key management infrastructure is fundamental to handling the required cryptographic keys, whereas for security policy enforcement, one needs to properly codify authorizations as a function of device roles and a security policy engine that implements these authorization checks and that can implement changes hereto throughout the system's lifecycle.

In the context of the IoT, however, the security must not only focus on the required security services, but also how these are realized in the overall system and how the security functionalities are executed.

To this end, we use the following terminology to analyze and classify security aspects in the IoT:

- 1 The security architecture refers to the system elements involved in the management of the security relationships between things and the way these security interactions are handled (e.g., centralized or distributed) during the lifecycle of a thing.
- 2 The security model of a node describes how the security parameters, processes, and applications are managed in a thing. This includes aspects such as process separation, secure storage of keying materials, etc.
- 3 Security bootstrapping denotes the process by which a thing securely joins the IoT at a given location and point in time. Bootstrapping includes the authentication and authorization of a device as well as the transfer of security parameters allowing for its trusted operation in a given network.
- 4 Network security describes the mechanisms applied within a network to ensure trusted operation of the IoT. Specifically, it prevents attackers from endangering or modifying the expected operation of networked things. Network security can include a number of mechanisms ranging from secure routing to data link layer and network layer security.
- 5 Application security guarantees that only trusted instances of an application running in the IoT can communicate with each other, while illegitimate instances cannot interfere.

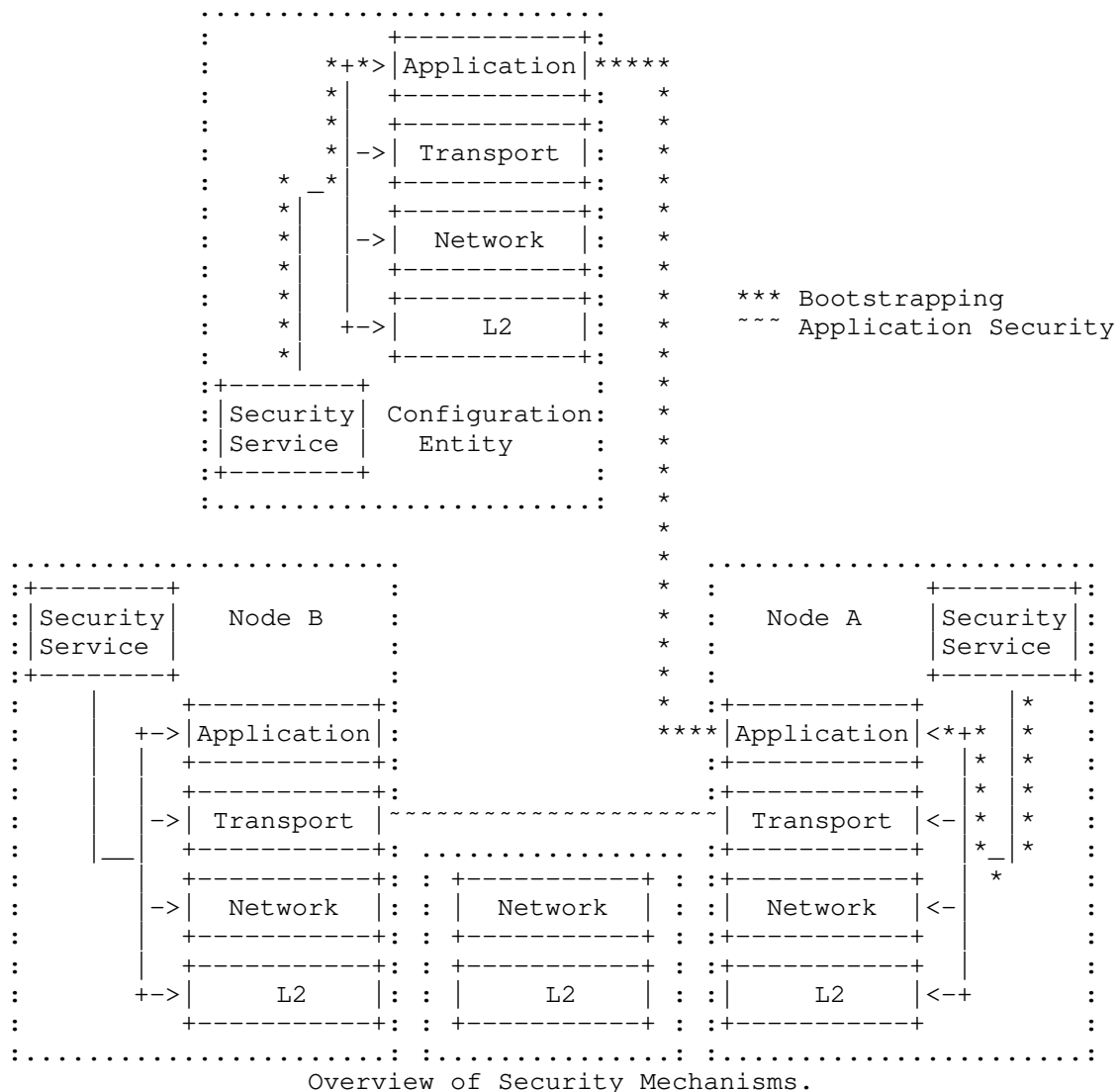


Figure 3

We now discuss an exemplary security architecture relying on a configuration entity for the management of the system with regard to the introduced security aspects (see Figure 2). Inspired by the security framework for routing over low power and lossy network [ID-Tsao], we show an example of security model and illustrates how different security concepts and the lifecycle phases map to the Internet communication stack. Assume a centralized architecture in

which a configuration entity stores and manages the identities of the things associated with the system along with their cryptographic keys. During the bootstrapping phase, each thing executes the bootstrapping protocol with the configuration entity, thus obtaining the required device identities and the keying material. The security service on a thing in turn stores the received keying material for the network layer and application security mechanisms for secure communication. Things can then securely communicate with each other during their operational phase by means of the employed network and application security mechanisms.

4. State of the Art

Nowadays, there exists a multitude of control protocols for the IoT. For BAC systems, the ZigBee standard [ZB], BACNet [BACNET], or DALI [DALI] play key roles. Recent trends, however, focus on an all-IP approach for system control.

In this setting, a number of IETF working groups are designing new protocols for resource constrained networks of smart things. The 6LoWPAN working group [WG-6LoWPAN] concentrates on the definition of methods and protocols for the efficient transmission and adaptation of IPv6 packets over IEEE 802.15.4 networks [RFC4944]. The CoRE working group [WG-CoRE] provides a framework for resource-oriented applications intended to run on constrained IP network (6LoWPAN). One of its main tasks is the definition of a lightweight version of the HTTP protocol, the Constrained Application Protocol (CoAP) [ID-CoAP], that runs over UDP and enables efficient application-level communication for things.

4.1. IP-based Security Solutions

In the context of the IP-based IoT solutions, consideration of TCP/IP security protocols is important as these protocols are designed to fit the IP network ideology and technology. While a wide range of specialized as well as general-purpose key exchange and security solutions exist for the Internet domain, we discuss a number of protocols and procedures that have been recently discussed in the context of the above working groups. The considered protocols are IKEv2/IPsec [RFC4306], TLS/SSL [RFC5246], DTLS [RFC5238], HIP [RFC5201][ID-Moskowitz], PANA [RFC5191], and EAP [RFC3748] in this Internet-Draft. Application layer solutions such as SSH [RFC4251] also exist, however, these are currently not considered. Figure 3 depicts the relationships between the discussed protocols in the context of the security terminology introduced in Section 3.1.

Transport Layer Security (TLS) and its datagram-oriented variant DTLS secure transport-layer connections. TLS provides security for TCP and requires a reliable transport, while DTLS secures and uses datagram-oriented protocols such as UDP. Both protocols are intentionally kept similar and share the same ideology and cipher suites.

The Extensible Authentication Protocol (EAP) is an authentication framework supporting multiple authentication methods. EAP runs directly over the data link layer and, thus, does not require the deployment of IP. It supports duplicate detection and retransmission, but does not allow for packet fragmentation. The Protocol for Carrying Authentication for Network Access (PANA) is a network-layer transport for EAP that enables network access authentication between clients and the network infrastructure. In EAP terms, PANA is a UDP-based EAP lower layer that runs between the EAP peer and the EAP authenticator.

4.2. Wireless Sensor Network Security and Beyond

A variety of key agreement and privacy protection protocols that are tailored to IoT scenarios have been introduced in the literature. For instance, random key pre-distribution schemes [PROC-Chan] or more centralized solutions, such as SPINS [JOURNAL-Perrig], have been proposed for key establishment in wireless sensor networks. The ZigBee standard [ZB] for sensor networks defines a security architecture based on an online trust center that is in charge of handling the security relationships within a ZigBee network. Personal privacy in ubiquitous computing has been studied extensively, e.g., in [THESIS-Langheinrich]. Due to resource constraints and the specialization to meet specific requirements, these solutions often implement a collapsed cross layer optimized communication stack (e.g., without task-specific network layers and layered packet headers). Consequently, they cannot directly be adapted to the requirements of the Internet due to the nature of their design.

Despite important steps done by, e.g., Gupta et al. [PROC-Gupta], to show the feasibility of an end-to-end standard security architecture for the embedded Internet, the Internet and the IoT domain still do not fit together easily. This is mainly due to the fact that IoT security solutions are often tailored to the specific scenario requirements without considering interoperability with Internet protocols. On the other hand, the direct use of existing Internet security protocols in the IoT might lead to inefficient or insecure operation as we show in our discussion below.

5. Challenges for a Secure Internet of Things

In this section, we take a closer look at the various security challenges in the operational and technical features of the IoT and then discuss how existing Internet security protocols cope with these technical and conceptual challenges through the lifecycle of a thing. Table 1 summarizes which requirements need to be met in the lifecycle phases as well as the considered protocols. The structure of this section follows the structure of the table. This discussion should neither be understood as a comprehensive evaluation of all protocols, nor can it cover all possible aspects of IoT security. Yet, it aims at showing concrete limitations of existing Internet security protocols in some areas rather than giving an abstract discussion about general properties of the protocols. In this regard, the discussion handles issues that are most important from the authors' perspectives.

5.1. Constraints and Heterogeneous Communication

Coupling resource constrained networks and the powerful Internet is a challenge because the resulting heterogeneity of both networks complicates protocol design and system operation. In the following we briefly discuss the resource constraints of IoT devices and the consequences for the use of Internet Protocols in the IoT domain.

5.1.1. Tight Resource Constraints

The IoT is a resource-constrained network that relies on lossy and low-bandwidth channels for communication between small nodes, regarding CPU, memory, and energy budget. These characteristics directly impact the threats to and the design of security protocols for the IoT domain. First, the use of small packets, e.g., IEEE 802.15.4 supports 127-byte sized packets at the physical layer, may result in fragmentation of larger packets of security protocols. This may open new attack vectors for state exhaustion DoS attacks, which is especially tragic, e.g., if the fragmentation is caused by large key exchange messages of security protocols. Moreover, packet fragmentation commonly downgrades the overall system performance due to fragment losses and the need for retransmissions. For instance, fate-sharing packet flight as implemented by DTLS might aggravate the resulting performance loss.

	Bootstrapping phase	Operational Phase
Requirements	Incremental deployment Identity and key management Privacy-aware identification Group creation	End-to-End security Mobility support Group membership management
Protocols	IKEv2 TLS/DTLS HIP/Diet-HIP PANA/EAP	IKEv2/MOBIKE TLS/DTLS HIP/Diet-HIP

Relationships between IP-based security protocols.

Figure 5

The size and number of messages should be minimized to reduce memory requirements and optimize bandwidth usage. In this context, layered approaches involving a number of protocols might lead to worse performance in resource-constrained devices since they combine the headers of the different protocols. In some settings, protocol negotiation can increase the number of exchanged messages. To improve performance during basic procedures such as, e.g., bootstrapping, it might be a good strategy to perform those procedures at a lower layer.

Small CPUs and scarce memory limit the usage of resource-expensive cryptoprimitives such as public-key cryptography as used in most Internet security standards. This is especially true, if the basic cryptoblocks need to be frequently used or the underlying application demands a low delay.

Independently from the development in the IoT domain, all discussed security protocols show efforts to reduce the cryptographic cost of the required public-key-based key exchanges and signatures with ECC[RFC5246][RFC5903][ID-Moskowitz][ID-HIP]. Moreover, all protocols have been revised in the last years to enable crypto agility, making cryptographic primitives interchangeable. Diet HIP takes the reduction of the cryptographic load one step further by focusing on cryptographic primitives that are to be expected to be enabled in hardware on IEEE 802.15.4 compliant devices. For example, Diet HIP does not require cryptographic hash functions but uses a CMAC [NIST] based mechanism, which can directly use the AES hardware available in standard sensor platforms. However, these improvements are only a first step in reducing the computation and communication overhead of Internet protocols. The question remains if other approaches can be

applied to leverage key agreement in these heavily resource-constrained environments.

A further fundamental need refers to the limited energy budget available to IoT nodes. Careful protocol (re)design and usage is required to reduce not only the energy consumption during normal operation, but also under DoS attacks. Since the energy consumption of IoT devices differs from other device classes, judgments on the energy consumption of a particular protocol cannot be made without tailor-made IoT implementations.

5.1.2. Denial-of-Service Resistance

The tight memory and processing constraints of things naturally alleviate resource exhaustion attacks. Especially in unattended T2T communication, such attacks are difficult to notice before the service becomes unavailable (e.g., because of battery or memory exhaustion). As a DoS countermeasure, DTLS, IKEv2, HIP, and Diet HIP implement return routability checks based on a cookie mechanism to delay the establishment of state at the responding host until the address of the initiating host is verified. The effectiveness of these defenses strongly depends on the routing topology of the network. Return routability checks are particularly effective if hosts cannot receive packets addressed to other hosts and if IP addresses present meaningful information as is the case in today's Internet. However, they are less effective in broadcast media or when attackers can influence the routing and addressing of hosts (e.g., if hosts contribute to the routing infrastructure in ad-hoc networks and meshes).

In addition, HIP implements a puzzle mechanism that can force the initiator of a connection (and potential attacker) to solve cryptographic puzzles with variable difficulties. Puzzle-based defense mechanisms are less dependent on the network topology but perform poorly if CPU resources in the network are heterogeneous (e.g., if a powerful Internet host attacks a thing). Increasing the puzzle difficulty under attack conditions can easily lead to situations, where a powerful attacker can still solve the puzzle while weak IoT clients cannot and are excluded from communicating with the victim. Still, puzzle-based approaches are a viable option for sheltering IoT devices against unintended overload caused by misconfigured or malfunctioning things.

5.1.3. Protocol Translation and End-to-End Security

Even though 6LoWPAN and CoAP progress towards reducing the gap between Internet protocols and the IoT, they do not target protocol specifications that are identical to their Internet pendants due to

performance reasons. Hence, more or less subtle differences between IoT protocols and Internet protocols will remain. While these differences can easily be bridged with protocol translators at gateways, they become major obstacles if end-to-end security measures between IoT devices and Internet hosts are used.

Cryptographic payload processing applies message authentication codes or encryption to packets. These protection methods render the protected parts of the packets immutable as rewriting is either not possible because a) the relevant information is encrypted and inaccessible to the gateway or b) rewriting integrity-protected parts of the packet would invalidate the end-to-end integrity protection.

There are essentially four solutions for this problem:

- 1 Sharing symmetric keys with gateways enables gateways to transform (e.g., de-compress, convert, etc.) packets and re-apply the security measures after transformation. This method abandons end-to-end security and is only applicable to simple scenarios with a rudimentary security model.
- 2 Reusing the Internet wire format in the IoT makes conversion between IoT and Internet protocols unnecessary. However, it leads to poor performance because IoT specific optimizations (e.g., stateful or stateless compression) are not possible.
- 3 Selectively protecting vital and immutable packet parts with a MAC or with encryption requires a careful balance between performance and security. Otherwise, this approach will either result in poor performance (protect as much as possible) or poor security (compress and transform as much as possible).
- 4 Message authentication codes that sustain transformation can be realized by considering the order of transformation and protection (e.g., by creating a signature before compression so that the gateway can decompress the packet without recalculating the signature). This enables IoT specific optimizations but is more complex and may require application-specific transformations before security is applied. Moreover, it cannot be used with encrypted data because the lack of cleartext prevents gateways from transforming packets.

To the best of our knowledge, none of the mentioned security protocols provides a fully customizable solution in this problem space. In fact, they usually offer an end-to-end secured connection. An exception is the usage layered approach as might be PANA and EAP. In such a case, this configuration (i) allows for a number of configurations regarding the location of, e.g., the EAP authenticator

and authentication server and (ii) the layered architecture might allow for authentication at different places. The drawback of this approach, however, lies in its high signaling traffic volume compared to other approaches. Hence, future work is required to ensure security, performance and interoperability between IoT and the Internet.

5.2. Bootstrapping of a Security Domain

Creating a security domain from a set of previously unassociated IoT devices is a key operation in the lifecycle of a thing and in the IoT network. In this section, we discuss general forms of network operation, how to communicate a thing's identity and the privacy implications arising from the communication of this identity.

5.2.1. Distributed vs. Centralized Architecture and Operation

Most things might be required to support both centralized and distributed operation patterns. Distributed thing-to-thing communication might happen on demand, for instance, when two things form an ad-hoc security domain to cooperatively fulfill a certain task. Likewise, nodes may communicate with a backend service located in the Internet without a central security manager. The same nodes may also be part of a centralized architecture with a dedicated node being responsible for the security management for group communication between things in the IoT domain. In today's IoT, most common architectures are fully centralized in the sense that all the security relationships within a segment are handled by a central party. In the ZigBee standard, this entity is the trust center. Current proposals for 6LoWPAN/CoRE identify the 6LoWPAN Border Router (6LBR) as such a device.

A centralized architecture allows for central management of devices and keying materials as well as for the backup of cryptographic keys. However, it also imposes some limitations. First, it represents a single point of failure. This is a major drawback, e.g., when key agreement between two devices requires online connectivity to the central node. Second, it limits the possibility to create ad-hoc security domains without dedicated security infrastructure. Third, it codifies a more static world view, where device roles are cast in stone, rather than a more dynamic world view that recognizes that networks and devices, and their roles and ownership, may change over time (e.g., due to device replacement and hand-over of control).

Decentralized architectures, on the other hand, allow creating ad-hoc security domains that might not require a single online management entity and are operative in a much more stand-alone manner. The ad-hoc security domains can be added to a centralized architecture at a

later point in time, allowing for central or remote management.

5.2.2. Bootstrapping a thing's identity and keying materials

Bootstrapping refers to the process by which a device is associated to another one, to a network, or to a system. The way it is performed depends upon the architecture: centralized or distributed. It is important to realize that bootstrapping may involve different types of information, ranging from network parameters and information on device capabilities and their presumed functionality, to management information related to, e.g., resource scheduling and trust initialization/management. Furthermore, bootstrapping may occur in stages during the lifecycle of a device and may include provisioning steps already conducted during device manufacturing (e.g., imprinting a unique identifier or a root certificate into a device during chip testing), further steps during module manufacturing (e.g., setting of application-based configurations, such as temperature read-out frequencies and push-thresholds), during personalization (e.g., fine-tuned settings depending on installation context), during hand-over (e.g., transfer of ownership from supplier to user), and, e.g., in preparation of operation in a specific network. In what follows, we focus on bootstrapping of security-related information, since bootstrapping of all other information can be conducted as ordinary secured communications, once a secure and authentic channel between devices has been put in place.

In a distributed approach, a Diffie-Hellman type of handshake can allow two peers to agree on a common secret. In general, IKEv2, HIP, TLS, DTLS, can perform key exchanges and the setup of security associations without online connections to a trust center. If we do not consider the resource limitations of things, certificates and certificate chains can be employed to securely communicate capabilities in such a decentralized scenario. HIP and Diet HIP do not directly use certificates for identifying a host, however certificate handling capabilities exist for HIP and the same protocol logic could be used for Diet HIP. It is noteworthy, that Diet HIP does not require a host to implement cryptographic hashes. Hence, some lightweight implementations of Diet HIP might not be able to verify certificates unless a hash function is implemented by the host.

In a centralized architecture, preconfigured keys or certificates held by a thing can be used for the distribution of operational keys in a given security domain. A current proposal [ID-OFlynn] refers to the use of PANA for the transport of EAP messages between the PANA client (the joining thing) and the PANA Authentication Agent (PAA), the 6LBR. EAP is thereby used to authenticate the identity of the joining thing. After the successful authentication, the PANA PAA

provides the joining thing with fresh network and security parameters.

IKEv2, HIP, TLS, and DTLS could be applied as well for the transfer of configuration parameters in a centralized scenario. While HIP's cryptographic secret identifies the thing, the other protocols do not represent primary identifiers but are used instead to bind other identifiers such as the operation keys to the public-key identities.

In addition to the protocols, operational aspects during bootstrapping are of key importance as well. Many other standard Internet protocols assume that the identity of a host is either available by using secondary services like certificate authorities or secure name resolution (e.g., DNSsec) or can be provided over a side channel (entering passwords via screen and keyboard). While these assumptions may hold in traditional networks, intermittent connectivity, localized communication, and lack of input methods complicate the situation for the IoT.

The order in which the things within a security domain are bootstrapped plays an important role as well. In [RFC6345], the PANA relay element is introduced, relaying PANA messages between a PaC (joining thing) and PAA of a segment [ID-OFlynn]. This approach forces commissioning based on distance to PAA, i.e., things can only be bootstrapped hop-by-hop starting from those closer to the PAA, all things that are 1-hop away are bootstrapped first, followed by those that are 2-hop away, and so on. Such an approach might impose important limitations on actual use cases in which, e.g., an installer without technical background has to roll-out the system.

5.2.3. Privacy-aware Identification

During the last years, the introduction of RFID tags has raised privacy concerns because anyone might access and track tags. As the IoT involves not only passive devices, but also includes active and sensing devices, the IoT might irrupt even deeper in people's privacy spheres. Thus, IoT protocols should be designed to avoid these privacy threats during bootstrapping and operation where deemed necessary. In H2T and T2T interactions, privacy-aware identifiers might be used to prevent unauthorized user tracking. Similarly, authentication can be used to prove membership of a group without revealing unnecessary individual information.

TLS and DTLS provide the option of only authenticating the responding host. This way, the initiating host can stay anonymous. If authentication for the initiating host is required as well, either public-key certificates or authentication via the established encrypted payload channel can be employed. Such a setup allows to

only reveal the responder's identity to possible eavesdroppers.

HIP and IKEv2 use public-key identities to authenticate the initiator of a connection. These identities could easily be traced if no additional protection were in place. IKEv2 transmits this information in an encrypted packet. Likewise, HIP provides the option to keep the identity of the initiator secret from eavesdroppers by encrypting it with the symmetric key generated during the handshake. However, Diet HIP cannot provide a similar feature because the identity of the initiator simultaneously serves as static Diffie-Hellman key. Note that all discussed solutions could use anonymous public-key identities that change for each communication. However, such identity cycling may require a considerable computational effort for generating new asymmetric key pairs. In addition to the built-in privacy features of the here discussed protocols, a large body of anonymity research for key exchange protocols exists. However, the comparison of these protocols and protocol extensions is out of scope for this work.

5.3. Operation

After the bootstrapping phase, the system enters the operational phase. During the operational phase, things can relate to the state information created during the bootstrapping phase in order to exchange information securely and in an authenticated fashion. In this section, we discuss aspects of communication patterns and network dynamics during this phase.

5.3.1. End-to-End Security

Providing end-to-end security is of great importance to address and secure individual T2T or H2T communication within one IoT domain. Moreover, end-to-end security associations are an important measure to bridge the gap between the IoT and the Internet. IKEv2 and HIP, TLS and DTLS provide end-to-end security services including peer entity authentication, end-to-end encryption and integrity protection above the network layer and the transport layer respectively. Once bootstrapped, these functions can be carried out without online connections to third parties, making the protocols applicable for decentralized use in the IoT. However, protocol translation by intermediary nodes may invalidate end-to-end protection measures (see Section 5.1).

5.3.2. Group Membership and Security

In addition to end-to-end security, group key negotiation is an important security service for the T2Ts and Ts2T communication patterns in the IoT as efficient local broadcast and multicast relies

on symmetric group keys.

All discussed protocols only cover unicast communication and therefore do not focus on group-key establishment. However, the Diffie-Hellman keys that are used in IKEv2 and HIP could be used for group Diffie-Hellman key-negotiations. Conceptually, solutions that provide secure group communication at the network layer (IPsec/IKEv2, HIP/Diet HIP) may have an advantage regarding the cryptographic overhead compared to application-focused security solutions (TLS/DTLS). This is due to the fact that application-focused solutions require cryptographic operations per group application, whereas network layer approaches may allow to share secure group associations between multiple applications (e.g., for neighbor discovery and routing or service discovery). Hence, implementing shared features lower in the communication stack can avoid redundant security measures.

A number of group key solutions have been developed in the context of the IETF working group MSEC in the context of the MIKEY architecture [WG-MSEC][RFC4738]. These are specifically tailored for multicast and group broadcast applications in the Internet and should also be considered as candidate solutions for group key agreement in the IoT. The MIKEY architecture describes a coordinator entity that disseminates symmetric keys over pair-wise end-to-end secured channels. However, such a centralized approach may not be applicable in a distributed environment, where the choice of one or several coordinators and the management of the group key is not trivial.

5.3.3. Mobility and IP Network Dynamics

It is expected that many things (e.g., wearable sensors, and user devices) will be mobile in the sense that they are attached to different networks during the lifetime of a security association. Built-in mobility signaling can greatly reduce the overhead of the cryptographic protocols because unnecessary and costly re-establishments of the session (possibly including handshake and key agreement) can be avoided. IKEv2 supports host mobility with the MOBIKE [RFC4555][RFC4621] extension. MOBIKE refrains from applying heavyweight cryptographic extensions for mobility. However, MOBIKE mandates the use of IPsec tunnel mode which requires to transmit an additional IP header in each packet. This additional overhead could be alleviated by using header compression methods or the Bound End-to-End Tunnel (BEET) mode [ID-Nikander], a hybrid of tunnel and transport mode with smaller packet headers.

HIP offers a simple yet effective mobility management by allowing hosts to signal changes to their associations [RFC5206]. However, slight adjustments might be necessary to reduce the cryptographic

costs, for example, by making the public-key signatures in the mobility messages optional. Diet HIP does not define mobility yet but it is sufficiently similar to HIP to employ the same mechanisms. TLS and DTLS do not have standards for mobility support, however, work on DTLS mobility exists in the form of an Internet draft [ID-Williams]. The specific need for IP-layer mobility mainly depends on the scenario in which nodes operate. In many cases, mobility support by means of a mobile gateway may suffice to enable mobile IoT networks, such as body sensor networks. However, if individual things change their point of network attachment while communicating, mobility support may gain importance.

6. Security Suites for the IP-based Internet of Things

Different applications have different security requirements and needs and, depending on various factors, such as device capability, availability of network infrastructure, security services needed, usage, etc., the required security protection may vary from "no security" to "full-blown security". For example, applications may have different needs regarding authentication and confidentiality. While some application might not require any authentication at all, others might require strong end-to-end authentication. In terms of secure bootstrapping of keys, some applications might assume the existence and online availability of a central key-distribution-center (KDC) within the 6LoWPAN network to distribute and manage keys; while other applications cannot rely on such a central party or their availability.

Thus, it is essential to define security profiles to better tailor security solutions for different applications with the same characteristics and requirements. This provides a means of grouping applications into profiles and then defines the minimal required security primitives to enable and support the security needs of the profile. The security elements in a security profile can be classified according to Section 3.1, namely:

- 1 Security architecture,
- 2 Security model,
- 3 Security bootstrapping,
- 4 Network security, and

5 Application security.

In order to (i) guide the design process by identifying open gaps; (ii) allow for later interoperability; and (iii) prevent possible security misconfigurations, this section defines a number of generic security profiles with different security needs. Each security profile is identified by:

- 1 a short description,
- 2 an exemplary application that might use/require such a security policy,
- 3 the security requirements for each of the above security aspects according to our classification in Section 3.1.

These security profiles can serve to guide the standardization process, since these explicitly describe the basic functionalities and protocols required to get different use cases up and running. It can allow for later interoperability since different manufacturers can describe the implemented security profile in their products. Finally, the security profiles can avoid possible security misconfigurations, since each security profile can be bound to a different application area so that it can be clearly defined which security protocols and approaches can be applied where and under which circumstances.

Note that each of these security profiles aim at summarizing the required security requirements for different applications and at providing a set of initial security features. In other words, these profiles reflect the need for different security configurations, depending on the threat and trust models of the underlying applications. In this sense, this section does not provide an overview of existing protocols as done in previous sections of the Internet Draft, but it rather explicitly describes what should be in place to ensure secure system operation. Observe also that this list of security profiles is not exhaustive and that it should be considered just as an example not related to existing legal regulations for any existing application. These security profiles are summarized in the table below:

	Application	Description
SecProf_0	No security needs	6LoWPAN/CoAP is used without security
SecProf_1	Home usage	Enables operation between home things without interaction with central device
SecProf_2	Managed Home usage	Enables operation between home things. Interaction with a central and local device is possible
SecProf_3	Industrial usage	Enables operation between things. Relies on central (local or backend) device for security
SecProf_4	Advanced Industrial usage	Enables ad-hoc operation between things and relies on central device or on a collection of control devices

Security profiles and application areas.

Figure 6

The classification in the table considers different potential applications and situations in which their security needs change due to different operational features (network size, existence of a central device, connectivity to the Internet, importance of the exchanged information, etc) or threat model (what are the assets that an attacker looks for). As already pointed out, this set of scenarios is exemplary and they should be further discussed based on a broader consensus.

SecProf_0 is meant for any application that does not require security. Examples include applications during system development, system testing, or some very basic applications in which security is not required at all.

The second security suite (SecProf_1) is catered for environments in which 6LoWPAN/CoAP can be used to enable communication between things in an ad-hoc manner and the security requirements are minimal. An example, is a home application in which two devices should exchange information and no further connection with other devices (local or with a backend) is required. In this scenario, value of the exchanged information is low and that it usually happen in a confined room, thus, it is possible to have a short period of time during

which initial secrets can be exchanged in the clear. Due to this fact, there is no requirement to enable devices from different manufacturers to interoperate in a secure way (keys are just exchanged). The expected network size of applications using this profile is expected to be small such that the provision of network security, e.g., secure routing, is of low importance.

The next security suite (SecProf_2) represents an evolution of SecProf_1 in which, e.g., home devices, can be managed locally. A first possibility for the securing domain management refers to the creation of a centrally managed security domain without any connectivity to the Internet. The central device used for management can serve as, e.g., a key distribution center including policies for key update, storage, etc. The presence of a central device can help in the management of larger networks. Network security becomes more relevant in this scenario since the 6LoWPAN/CoAP network can be prone to Denial of Service attacks (e.g., flooding if L2 is not protected) or routing attacks.

SecProf_3 considers that a central device is always required for network management. Example applications of this profile include building control and automation, sensor networks for industrial use, environmental monitoring, etc. As before, the network manager can be located in the 6LoWPAN/CoAP network and handle key management. In this case, the first association of devices to the network is required to be done in a secure way. In other words, the threat model requires measurements to protect against any vulnerable period of time. This step can involve the secure transmission of keying materials used for network security at different layers. The information exchanged in the network is considered to be valuable and it should be protected in the sense of pairwise links. Commands should be secured and broadcast should be secured with entity authentication [ID-CoAPMulticast]. Network should be protected from attacks. A further extension to this use case is to allow for remote management. A "backend manager" is in charge of managing SW or information exchanged or collected within the 6LoWPAN/CoAP network. This requires connection of devices to the Internet over a 6LBR involving a number of new threats that were not present before. A list of potential attacks include: resource-exhaustion attacks from the Internet; amplification attacks; trust issues related a HTTP-CoAP proxy [ID-proHTTPCoAP], etc. This use case requires protecting the communication from a device in the backend to a device in the 6LoWPAN/CoAP network, end-to-end. This use case also requires measures to provide the 6LBR with the capability of dropping fake requests coming from the Internet. This becomes especially challenging when the 6LBR is not trusted and access to the exchanged information is limited; and even more in the case of a HTTP-CoAP proxy since protocol translation is required. This use case should

take care of protecting information accessed from the backend due to privacy issues (e.g., information such as type of devices, location, usage, type and amount of exchanged information, or mobility patterns can be gathered at the backend threatening the privacy sphere of users) so that only required information is disclosed.

The last security suite (SecProf_4) essentially represents interoperability of all the security profiles defined previously. It considers applications with some additional requirements regarding operation such as: (i) ad-hoc establishment of security relationships between things (potentially from different manufacturers) in non-secure environments or (ii) dynamic roaming of things between different 6LoWPAN/CoAP security domains. Such operational requirements pose additional security requirements, e.g., in addition to secure bootstrapping of a device within a 6LoWPAN/CoAP security domain and the secure transfer of network operational key, there is a need to enable inter-domains secure communication to facilitate data sharing.

The above description illustrates how different applications of 6LoWPAN/CoAP networks involve different security needs. In the following sections, we summarize the expected security features or capabilities for each the security profile with regards to "Security Architecture", "Security Model", "Security Bootstrapping", "Network Security", and "Application Security".

6.1. Security Architecture

The choice of security architecture has many implications regarding key management, access control, or security scope. A distributed (or ad-hoc) architecture means that security relationships between things are setup on the fly between a number of objects and kept in a decentralized fashion. A locally centralized security architecture means that a central device, e.g., the 6LBR, handles the keys for all the devices in the security domain. Alternatively, a central security architecture could also refer to the fact that smart objects are managed from the backend. The security architecture for the different security profiles is classified as follows.

	Description
SecProf_0	-
SecProf_1	Distributed
SecProf_2	Distributed able to move centralized (local)
SecProf_3	Centralized (local &/or backend)
SecProf_4	Distributed & centralized (local &/or backend)

Security architectures in different security profiles.

Figure 7

In "SecProf_1", management mechanisms for the distributed assignment and management of keying materials is required. Since this is a very simple use case, access control to the security domain can be enabled by means of a common secret known to all devices. In the next security suite (SecProf_2), a central device can assume key management responsibilities and handle the access to the network. The last two security suites (SecProf_3 and SecProf_4) further allow for the management of devices or some keying materials from the backend.

6.2. Security Model

While some applications might involve very resource-constrained things such as, e.g., a humidity, pollution sensor, other applications might target more powerful devices aimed at more exposed applications. Security parameters such as keying materials, certificates, etc must be protected in the thing, for example by means of tamper-resistant hardware. Keys may be shared across a thing's networking stack to provide authenticity and confidentiality in each networking layer. This would minimize the number of key establishment/agreement handshake and incurs less overhead for constrained thing. While more advance applications may require key separation at different networking layers, and possibly process separation and sandboxing to isolate one application from another. In this sense, this section reflects the fact that different applications require different sets of security mechanisms.

	Description
SecProf_0	-
SecProf_1	No tamper resistant Sharing keys between layers
SecProf_2	No tamper resistant Sharing keys between layers
SecProf_3	Tamper resistant Key and process separation
SecProf_4	(no) Tamper resistant Sharing keys between layers/Key and process separation Sandbox

Thing security models in different security profiles.

Figure 8

6.3. Security Bootstrapping and Management

Bootstrapping refers to the process by which a thing initiates its life within a security domain and includes the initialization of secure and/or authentic parameters bound to the thing and at least one other device in the network. Here, different mechanisms may be used to achieve confidentiality and/or authenticity of these parameters, depending on deployment scenario assumptions and the communication channel(s) used for passing these parameters. The simplest mechanism for initial set-up of secure and authentic parameters is via communication in the clear using a physical interface (USB, wire, chip contact, etc.). Here, one commonly assumes this communication channel is secure, since eavesdropping and/or manipulation of this interface would generally require access to the physical medium and, thereby, to one or both of the devices themselves. This mechanism was used with the so-called original "resurrecting duckling" model, as introduced in [PROC-Stajano]. This technique may also be used securely in wireless, rather than wired, set-ups, if the prospect of eavesdropping and/or manipulating this channel are dim (a so-called "location-limited" channel [PROC-Smetters-04, PROC-Smetters-02]). Examples hereof include the communication of secret keys in the clear using near field communication (NFC) - where the physical channel is purported to have very limited range (roughly 10cm), thereby thwarting eavesdropping by

far-away adversarial devices, and in-the-clear communication during a small time window (triggered by, e.g., a button-push) - where eavesdropping is presumed absent during this small time window. With the use of public-key based techniques, assumptions on the communication channel can be relaxed even further, since then the cryptographic technique itself provides for confidentiality of the channel set-up and the location-limited channel - or use of certificates - rules out man-in-the-middle attacks, thereby providing authenticity [PROC-Smetters-02]. The same result can be obtained using password-based public-key protocols [SPEKE], where authenticity depends on the (weak) password not being guessed during execution of the protocol. It should be noted that while most of these techniques realize a secure and authentic channel for passing parameters, these generally do not provide for explicit authorization. As an example, with use of certificate-based public-key based techniques, one may obtain hard evidence on whom one shares secret and/or authentic parameters with, but this does not answer the question as to whether one wishes to share this information at all with this specifically identified device (the latter usually involves a human-decision element). Thus, the bootstrapping mechanisms above should generally be complemented by mechanisms that regulate (security policies for) authorization. Furthermore, the type of bootstrapping is very related to the required type of security architecture. Distributed bootstrapping means that a pair of devices can setup a security relationship on the fly, without interaction with a central device elsewhere within the system. In many cases, it is handy to have a distributed bootstrapping protocol based on existing security protocols (e.g., DTLS in CoAP) required for other purposes: this reduces the amount of required software. A centralized bootstrapping protocol is one in which a central device manages the security relationships within a network. This can happen locally, e.g., handled by the 6LBR, or remotely, e.g., from a server connected via the Internet. The security bootstrapping for the different security profiles is as follows.

	Description
SecProf_0	-
SecProf_1	* Distributed, (e.g., Resurrecting duckling) * First key distribution happens in the clear
SecProf_2	* Distributed, (e.g., Resurrecting duckling) * Centralized (local), 6LBR acts as KDC * First key distribution occurs in the clear, if the KDC is available, the KDC can manage network access
SecProf_3	* 6LBR acts as KDC. It handles node joining, provides them with keying material from L2 to application layers * Bootstrapping occurs in a secure way - either in secure environment or the security mechanisms ensure that eavesdropping is not possible. * KDC and backend can implement secure methods for network access
SecProf_4	* As in SecProf_3.

Security bootstrapping methods in different security profiles

Figure 9

6.4. Network Security

Network security refers to the mechanisms used to ensure the secure transport of 6LoWPAN frames. This involves a multitude of issues ranging from secure discovery, frame authentication, routing security, detection of replay, secure group communication, etc. Network security is important to thwart potential attacks such as denial-of-service (e.g., through message flooding) or routing attacks.

The Internet Draft [ID-Tsao] presents a very good overview of attacks and security needs classified according to the confidentiality, integrity, and availability needs. A potential limitation is that there exist no differentiation in security between different use cases and the framework is limited to L3. The security suites gathered in the present ID aim at solving this by allowing for a more flexible selection of security needs at L2 and L3.

	Description
SecProf_0	-
SecProf_1	<ul style="list-style-type: none"> * Network key creating a home security domain at L2 ensuring authentication and freshness of exchanged data * Secure multicast does not ensure origin authentication * No need for secure routing at L3
SecProf_2	<ul style="list-style-type: none"> * Network key creating a home security domain at L2 ensuring authentication and freshness of exchanged data * Secure multicast does not ensure origin authentication * No need for secure routing at L3
SecProf_3	<ul style="list-style-type: none"> * Network key creating an industry security domain at L2 ensuring authentication and freshness of exchanged data * Secure routing needed (integrity & availability) at L3 within 6LoWPAN/CoAP * Secure multicast requires origin authentication
SecProf_4	<ul style="list-style-type: none"> * Network key creating an industry security domain at L2 ensuring authentication and freshness of exchanged data * Inter-domain authentication/secure handoff * Secure routing needed at L3 * Secure multicast requires origin authentication * 6LBR (HTTP-CoAP proxy) requires verification of forwarded messages and messages leaving or entering the 6LoWPAN/CoAP network.

Network security needs in different security profiles

Figure 10

6.5. Application Security

In the context of 6LoWPAN/CoAP networks, application security refers firstly to the configuration of DTLS used to protect the exchanged information. It further refers to the measures required in potential translation points (e.g., a (HTTP-CoAP) proxy) where information can be collected and the privacy sphere of users in a given security domain is endangered. Application security for the different security profiles is as follows.

	Description
SecProf_0	-
SecProf_1	-
SecProf_2	<ul style="list-style-type: none"> * DTLS is used for end-to-end application security between management device and things and between things * DTLS ciphersuites configurable to provide confidentiality and/or authentication and/or freshness * Key transport and policies for generation of session keys are required
SecProf_3	<ul style="list-style-type: none"> * Requirements as in SecProf_2 and * DTLS is used for end-to-end application security between management device and things and between things * Communication between KDC and each thing secured by pairwise keys * Group keys for communication in a group distributed by KDC * Privacy protection should be provided in translation points
SecProf_4	<ul style="list-style-type: none"> * Requirements as in SecProf_3 and * TLS or DTLS can be used to send commands from the backend to the 6LBR or things in a 6LoWPAN/CoAP network * End-to-end secure connectivity from backend required * Secure broadcast in a network from backend required

Application security methods in different security profiles

Figure 11

The first two security profiles do not include any security at the application layer. The reason is that, in the first case, security is not provided and, in the second case, it seems reasonable to provide basic security at L2. In the third security profile (SecProf_2), DTLS becomes the way of protecting messages at application layer between things and with the KDC running on a 6LBR. A key option refers to the capability of easily configuring DTLS to provide a subset of security services (e.g., some applications do not require confidentiality) to reduce the impact of security in the system operation of resource-constrained things. In addition to basic key management mechanisms running within the KDC, communication protocols for key transport or key update are required. These

protocols could be based on DTLS. The next security suite (SecProf_3) requires pairwise keys for communication between things within the security domain. Furthermore, it can involve the usage of group keys for group communication. If secure multicast is implemented, it should provide origin authentication. Finally, privacy protection should be taken into account to limit access to valuable information -- such as identifiers, type of collected data, traffic patterns -- in potential translation points (proxies) or in the backend. The last security suite (SecProf_4) further extends the previous set of requirements considering security mechanisms to deal with translations between TLS and DTLS or for the provision of secure multicast within a 6LoWPAN/CoAP network from the backend.

7. Next Steps towards a Flexible and Secure Internet of Things

This Internet Draft included an overview of both operational and security requirements of things in the Internet of Things, discussed a general threat model and security issues, and introduced a number of potential security suites fitting different types of IoT deployments.

We conclude this document by giving our assessment of the current status of CoAP security with respect to addressing the IP security challenges we identified, so as to facilitate discussion of next steps towards workable security design concepts suitable for IP-based IoT in the broader community. Hereby, we focus on the employed security protocols and the type of security architecture.

With current status, we refer to the feasibility of realizing secure deployments with existing CoAP protocols and the practicality of creating comprehensive security architectures based on those protocols:

- 1 DTLS has been defined as the basic building block for protecting CoAP. At the time it was first proposed, no DTLS implementation for small, constrained devices was available. In the mean-time, TinyDTLS [TinyDTLS] has been developed offering the first open-source implementation of the protocol for small devices. However, more experience with the protocol is required. In particular, a performance evaluation and comparison should be made with a well-defined set of standard node platforms/networks. The results will help understand the limitations and the benefits of DTLS as well as to give recommended usage scenarios for this security protocol.

- 2 (D)TLS was designed for traditional computer networks and, thus, some of its features may not be optimal for resource-constrained networks. This includes:
 - a Basic DTLS features that are, in our view, not ideal for resource-constrained devices. For instance, the loss of a message in-flight requires the retransmission of all messages in-flight. On the other hand, if all messages in-flight are transmitted together in a single UDP packet, more resources are required for handling of large buffers. As pointed out in [ID-Hartke] , the number of flights in the DTLS handshake should be reduced, so that a faster setup of a secure channel can be realized. This would definitely improve the performance of DTLS significantly.
 - b Fragmentation of messages due to smaller MTUs in resource-constrained networks is problematic. This implies that the node must have a large buffer to store all the fragments and subsequently perform re-ordering and reassembly in order to construct the entire DTLS message. The fragmentation of the handshake messages can, e.g., allow for a very simple method to carry out a denial of service attack.
 - c The completion of the DTLS handshake is based on the successful verification of the Finished message by both client and server. As the Finished message is computed based on the hash of all handshake messages in the correct order, the node must allocate a large buffer to queue all handshake messages.
 - d DTLS is thought to offer end-to-end security; however, end-to-end security also has to be considered from the point of view of LLN protection, so that end-to-end exchanges can still be verified and the LLN protected from, e.g., DoS attacks.
- 3 Raw public-key in DTLS has been defined as mandatory. However, memory-optimized public-key libraries still require several KB of flash and several hundreds of B of RAM. Although Moore's law still applies and an increase of platform resources is expected, many IoT scenarios are cost-driven, and in many use cases, the same work could be done with symmetric-keys. Thus, a key question is whether the choice for raw public-key is the best one. In addition, using raw public keys rather than certified public keys hard codes identities to public keys, thereby inhibiting public key updates and potentially complicating initial configuration.

- 4 Performance of DTLS from a system perspective should be evaluated involving not just the cryptographic constructs and protocols, but should also include implementation benchmarks for security policies, since these may impact overall system performance and network traffic (an example of this would be policies on the frequency of key updates, which would necessitate securely propagating these to all devices in the network).
- 5 Protection of lower protocol layers is a must in networks of any size to guarantee resistance against routing attacks such as flooding or wormhole attacks. The wireless medium that is used by things to communicate is broadcast in nature and allows anybody on the right frequency to overhear and even inject packets at will. Hence, IP-only security solutions may not suffice in many IoT scenarios. At the time of writing the document, comprehensive methods are either not in place or have not been evaluated yet. This limits the deployment of large-scale systems and makes the secure deployment of large scale networks rather infeasible.
- 6 The term "bootstrapping" has been discussed in many occasions. Although everyone agrees on its importance, finding a good solution applicable to most use cases is rather challenging. While usage of existing methods for network access might partially address bootstrapping in the short-term and facilitate integration with legacy back-end systems, we feel that, in the medium-term, this may lead to too large of an overhead and imposes unnecessary constraints on flexible deployment models. The bootstrapping protocol should be reusable and light-weight to fit with small devices. Such a standard bootstrapping protocol must allow for commissioning of devices from different manufacturers in both centralized and ad-hoc scenarios and facilitate transitions of control amongst devices during the device's and system's lifecycle. Examples of the latter include scenarios that involve hand-over of control, e.g., from a configuration device to an operational management console and involving replacement of such a control device. A key challenge for secure bootstrapping of a device in a centralized architecture is that it is currently not feasible to commission a device when the adjacent devices have not been commissioned yet. In view of the authors, a light-weight approach is still required that allows for the bootstrapping of symmetric-keys and of identities in a certified public-key setting.
- 7 Secure resource discovery has not been discussed so far. However, this issue is currently gaining relevance. The IoT, comprising sensors and actuators, will provide access to many resources to sense and modify the environment. The usage of DNS presents

well-known security issues, while the application of secure DNS may not be feasible on small devices. In general, security issues and solutions related to resource discovery are still unclear.

- 8 A security architecture involves, beyond the basic protocols, many different aspects such as key management and the management of evolving security responsibilities of entities during the lifecycle of a thing. This document discussed a number of security suites and argued that different types of security architectures are required. A flexible IoT security architecture should incorporate the properties of a fully centralized architecture as well as allow devices to be paired together initially without the need for a trusted third party to create ad-hoc security domains comprising a number of nodes. These ad-hoc security domains could then be added later to the Internet via a single, central node or via a collection of nodes (thus, facilitating implementation of a centralized or distributed architecture, respectively). The architecture should also facilitate scenarios, where an operational network may be partitioned or merged, and where hand-over of control functionality of a single device or even of a complete subnetwork may occur over time (if only to facilitate smooth device repair/replacement without the need for a hard "system reboot" or to realize ownership transfer). This would allow the IoT to transparently and effortlessly move from an ad-hoc security domain to a centrally-managed single security domain or a heterogeneous collection of security domains, and vice-versa. However, currently, these features still lack validation in real-life, large-scale deployments.
- 9 Currently, security solutions are layered, in the sense that each layer takes care of its own security needs. This approach fits well with traditional computer networks, but it has some limitations when resource-constrained devices are involved and these devices communicate with more powerful devices in the back-end. We argue that protocols should be more interconnected across layers to ensure efficiency as resource limitations make it challenging to secure (and manage) all layers individually. In this regard, securing only the application layer leaves the network open to attacks, while security focused only at the network or link layer might introduce possible inter-application security threats. Hence, the limited resources of things may require sharing of keying material and common security mechanisms between layers. It is required that the data format of the keying material is standardized to facilitate cross-layer interaction. Additionally, cross-layer concepts should be considered for an IoT-driven re-design of Internet security

protocols.

8. Security Considerations

This document reflects upon the requirements and challenges of the security architectural framework for Internet of Things.

9. IANA Considerations

This document contains no request to IANA.

10. Acknowledgements

We gratefully acknowledge feedback and fruitful discussion with Tobias Heer and Robert Moskowitz.

11. References

11.1. Informative References

[RFC6568]Kim, E., Kaspar, D., and JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6568, April 2012.

[RFC2818]Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC6345]Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., Ed., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA) Relay Element", RFC 6345, August 2011.

[ID-CoAP]Z. Shelby, K. Hartke, C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18, June 2013.

[ID-CoAPMulticast]Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-12 (work in progress), July 2013.

[ID-Daniel]Park, S., Kim, K., Haddad, W., Chakrabarti, S., and J. Laganier, "IPv6 over Low Power WPAN Security Analysis", Internet Draft draft-daniel-6lowpan-security-analysis-05, Mar 2011.

[ID-HIP]Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-06 (work in progress), May 2012.

[ID-Hartke]Hartke, K. and O. Bergmann, "Datagram Transport Layer Security in Constrained Environments", draft-hartke-core-codtls-02 (work in progress), July 2012.

[ID-Moskowitz]Moskowitz, R., Heer, T., Jokela, P., and Henderson, T., "Host Identity Protocol Version 2", draft-ietf-hip-rfc5201-bis-13 (work in progress), Sep 2013.

[ID-Nikander]Nikander, P. and J. Melen, "A Bound End-to-End Tunnel (BEET) mode for ESP", draft-nikander-esp-beet-mode-09, Aug 2008.

[ID-OFlynn]O'Flynn, C., Sarikaya, B., Ohba, Y., Cao, Z., and R. Cragie, "Security Bootstrapping of Resource-Constrained Devices", draft-oflynn-core-bootstrapping-03 (work in progress), Nov 2010.

[ID-Tsao]Tsao, T., Alexander, R., Dohler, M., Daza, V., and A. Lozano, "A Security Framework for Routing over Low Power and Lossy Networks", draft-ietf-roll-security-framework-07, Jan 2012.

[ID-Williams]Williams, M. and J. Barrett, "Mobile DTLS", draft-barrett-mobile-dtls-00, Mar 2009.

[ID-proHTTPCoAP]Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best practices for HTTP-CoAP mapping implementation", draft-castellani-core-http-mapping-07 (work in progress), Feb 2013.

[RFC3261]Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3748]Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC3756]Nikander, P., Ed., Kempf, J., and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats", RFC 3756, May 2004.

[RFC3833]Atkins, D. and R. Austein, "Threat Analysis of the Domain Name System (DNS)", RFC 3833, August 2004.

[RFC4016]Parthasarathy, M., "Protocol for Carrying Authentication and Network Access (PANA) Threat Analysis and Security Requirements", RFC 4016, March 2005.

[RFC5246]Dierks, T. and E. Rescorla, "The Transport Layer Security

(TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC4251]Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[RFC4306]Kaufman, C., Ed., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC4555]Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, June 2006.

[RFC4621]Kivinen, T. and H. Tschofenig, "Design of the IKEv2 Mobility and Multihoming (MOBIKE) Protocol", RFC 4621, August 2006.

[RFC4738]Ignjatic, D., Dondeti, L., Audet, F., and P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", RFC 4738, November 2006.

[RFC4919]Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.

[RFC4944]Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

[RFC5191]Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.

[RFC5201]Moskowitz, R., Nikander, P., Jokela, P., Ed., and T. Henderson, "Host Identity Protocol", RFC 5201, April 2008.

[RFC5206]Nikander, P., Henderson, T., Ed., Vogt, C., and J. Arkko, "End-Host Mobility and Multihoming with the Host Identity Protocol", RFC 5206, April 2008.

[RFC5238]Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, May 2008.

[RFC5246]Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5713]Moustafa, H., Tschofenig, H., and S. De Cnodder, "Security Threats and Security Requirements for the Access Node Control Protocol (ANCP)", RFC 5713, January 2010.

[RFC5903]Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime

(ECP Groups) for IKE and IKEv2", RFC 5903, June 2010.

[AUTO-ID]"AUTO-ID LABS", Web <http://www.autoidlabs.org/>, Sept 2010.

[BACNET]"BACnet", Web <http://www.bacnet.org/>, Feb 2011.

[DALI]"DALI", Web <http://www.dalibydesign.us/dali.html>, Feb 2011.

[JOURNAL-Perrig]Perrig, A., Szewczyk, R., Wen, V., Culler, D., and J. Tygar, "SPINS: Security protocols for Sensor Networks", Journal Wireless Networks, Sept 2002.

[NIST]Dworkin, M., "NIST Specification Publication 800-38B", 2005.

[PROC-Chan]Chan, H., Perrig, A., and D. Song, "Random Key Predistribution Schemes for Sensor Networks", Proceedings IEEE Symposium on Security and Privacy, 2003.

[PROC-Gupta]Gupta, V., Wurm, M., Zhu, Y., Millard, M., Fung, S., Gura, N., Eberle, H., and S. Shantz, "Sizzle: A Standards-based End-to-End Security Architecture for the Embedded Internet", Proceedings Pervasive Computing and Communications (PerCom), 2005.

[PROC-Smetters-02]Balfanz, D., Smetters, D., Steward, P., and H. Chi Wong, "Talking To Strangers: Authentication in Ad-Hoc Wireless Networks", Paper NDSS, 2002.

[PROC-Smetters-04]Balfanz, D., Durfee, G., Grinter, R., Smetters, D., and P. Steward, "Network-in-a-Box: How to Set Up a Secure Wireless Network in Under a Minute", Paper USENIX, 2004.

[PROC-Stajano-99]Stajano, F. and R. Anderson, "Resurrecting Duckling - Security Issues for Adhoc Wireless Networks", 7th International Workshop Proceedings, Nov 1999.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[THESIS-Langheinrich]Langheinrich, M., "Personal Privacy in Ubiquitous Computing", PhD Thesis ETH Zurich, 2005.

[TinyDTLS "TinyDTLS", Web <http://tinydtls.sourceforge.net/>, Feb 2012.

[WG-6LoWPAN]"IETF 6LoWPAN Working Group", Web <http://tools.ietf.org/wg/6lowpan/>, Feb 2011.

[WG-CORE]"IETF Constrained RESTful Environment (CoRE) Working Group",
Web <https://datatracker.ietf.org/wg/core/charter/>, Feb 2011.

[WG-MSEC]"MSEC Working Group", Web
<http://datatracker.ietf.org/wg/msec/>.

[ZB]"ZigBee Alliance", Web <http://www.zigbee.org/>, Feb 2011.

Authors' Addresses

Oscar Garcia-Morchon
Philips Research
High Tech Campus
Eindhoven, 5656 AA
The Netherlands

Email: oscar.garcia@philips.com

Sandeep S. Kumar
Philips Research
High Tech Campus
Eindhoven, 5656 AA
The Netherlands

Email: sandeep.kumar@philips.com

Sye Loong Keoh
University of Glasgow Singapore
Republic PolyTechnic, 9 Woodlands Ave 9
Singapore 838964
SG

Email: SyeLoong.Keoh@glasgow.ac.uk

Rene Hummen
RWTH Aachen University
Templergraben 55
Aachen, 52056
Germany

Email: rene.hummen@cs.rwth-aachen.de

Rene Struik
Struik Security Consultancy
Toronto,
Canada

Email: rstruik.ext@gmail.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2014

S. Gerdes
O. Bergmann
C. Bormann
Universitaet Bremen TZI
October 21, 2013

Delegated CoAP Authentication and Authorization Framework (DCAF)
draft-gerdes-core-dcaf-authorize-01

Abstract

This specification defines a protocol for delegating client authentication and authorization in a constrained environment for establishing a Datagram Transport Layer Security (DTLS) channel between resource-constrained nodes. The protocol relies on DTLS to transfer authorization information and shared secrets for symmetric cryptography between entities in a constrained network. A resource-constrained node can use this protocol to delegate authentication of communication peers and management of authorization information to a trusted host with less severe limitations regarding processing power and memory.

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 <http://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 April 24, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. Features	4
1.2. Terminology	4
1.2.1. Roles	5
1.2.2. Other terms	5
2. System Overview	6
3. Protocol	7
3.1. Overview	7
3.2. Unauthorized Resource Request Message	8
3.3. AS Information Message	9
3.4. Access Request	10
3.5. Ticket Request Message	11
3.6. Ticket Grant Message	12
3.7. Ticket Transfer Message	13
3.8. DTLS Channel Setup Between C and RS	13
3.9. Authorized Resource Request Message	14
4. Ticket	16
4.1. Face	16
4.2. Verifier	16
4.3. Revocation	17
4.3.1. Lifetime	17
4.3.2. Revocation Messages	17
5. Payload Format and Encoding (application/dcaf+json)	18
5.1. Examples	19
6. DTLS PSK Generation Methods	22
6.1. DTLS PSK Transfer	22
6.2. Distributed Key Derivation	22
7. Authorization Configuration	24
8. Trust Relationships	25
9. Listing Authorization Server Information in a Resource Directory	26
10. Examples	27
10.1. Access Granted	27
10.2. Access Denied	29
10.3. Access Restricted	29
10.4. Implicit Authorization	30
11. Security Considerations	31
12. IANA Considerations	32
12.1. dcaf+json Media Type Registration	32
12.2. CoAP Content Format Registration	33
13. References	34
13.1. Normative References	34
13.2. Informative References	34
Authors' Addresses	36

1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a transfer protocol similar to HTTP which is designed for the special requirements of constrained environments. A serious problem with constrained devices is the realization of secure communication. The devices only have limited resources such as memory, stable storage (such as disk space) and transmission capacity and often lack input/output devices such as keyboards or displays. Therefore, they are not readily capable of using common protocols. Especially authentication mechanisms are difficult to realize, because the lack of stable storage severely limits the number of keys the system can store. Moreover, CoAP has no mechanism to distinguish access rights for different clients (authorization).

The DCAF architecture is designed to relieve the constrained nodes from managing keys for numerous devices by introducing authorization servers which conduct the authentication and authorization for their nodes. To achieve this, access tokens are used. A device which wants to access a constrained node's resource first has to gain permission in the form of a token from the node's authorization server.

As fine-grained authorization is not always needed on constrained devices, DCAF supports an implicit authorization mode where no authorization information is exchanged.

The main goals of DCAF are the setup of a Datagram Transport Layer Security (DTLS) [RFC6347] channel with symmetric pre-shared keys (PSK) [RFC4279] and to securely transmit authorization tickets.

1.1. Features

- o Utilize DTLS communication with pre-shared keys.
- o Authenticated exchange of authorization information.
- o Simplified authorization mechanism for cases where implicit authorization is sufficient.
- o Using only symmetric encryption on constrained nodes.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2.1. Roles

Resource Server (RS): A constrained device that hosts resources the Client wants to access.

Client (C): A device that wants to access resources on the Resource Server.

Authorization Server (AS): The node that conducts authentication and authorization for a Resource Server. An Authorization Server can be responsible for a single or multiple devices or even for a whole network. A Resource Server can have multiple Authorization Servers.

Authentication Manager (AM): The node that conducts authentication on behalf of the Client.

Resource Owner: The principal that owns the resource and controls its access permissions.

1.2.2. Other terms

Access ticket: Contains the authentication and, if necessary, the authorization information needed to access a resource.

Authorization information: Contains all information needed by RS to decide if C is privileged to access a resource in a specific way.

Authentication information: Contains all information needed by RS to decide if the entity claiming to be C is to be trusted.

Access information: Contains authentication information and, if necessary, authorization information.

Explicit authorization: The Authorization Server informs the Resource Server in detail which privileges are granted to the Client.

Implicit authorization: The Authorization Server informs the Resource Server that the Client is authorized to access any resource on RS in any way, without specifying the privileges in detail.

2. System Overview

Within the DCAF Architecture each Resource Server (RS) has one or more Authorization Servers (AS) which conduct the authentication and authorization for RS. RS and AS share a symmetric key which has to be exchanged initially to provide for a secure channel. The mechanism used for this is not in the scope of this document.

To gain access to a specific resource on a Resource Server, a client (C) has to request an access ticket from one of the Authorization Servers serving RS either directly or, if it is a constrained device, using its Authentication Manager (AM). In the following, we always discuss the AM role separately, even if that is co-located within a (more powerful) C.

If AS decides that C is allowed to access the resource, it generates a DTLS pre-shared key (PSK) for the communication between C and RS and wraps it into an access ticket. For explicit access control, AS adds the detailed access permissions to the ticket in a way that RS can interpret. After presenting the ticket to RS, C and RS can communicate securely.

To be able to provide for the authentication and authorization services, the Authorization Servers have to fulfill several requirements:

- o An AS must have enough stable storage (such as disk space) to store the necessary number of credentials (matching the number of clients and Resource Servers).
- o An AS must possess means for user interaction, for example directly or indirectly connected input/output devices like keyboard and display, to allow for configuration of authorization information by the Resource Owner.
- o An AS must have enough processing power to handle the authorization requests for all RS devices it is responsible for.

3. Protocol

The DCAF protocol comprises three parts:

1. transfer of authentication and, if necessary, authorization information between C and RS;
2. transfer of access requests and the respective ticket grants between C and AM; and
3. transfer of access requests and the respective ticket grants between AS and AM.

3.1. Overview

In Figure 1, a DCAF protocol flow is depicted (messages in square brackets are optional):

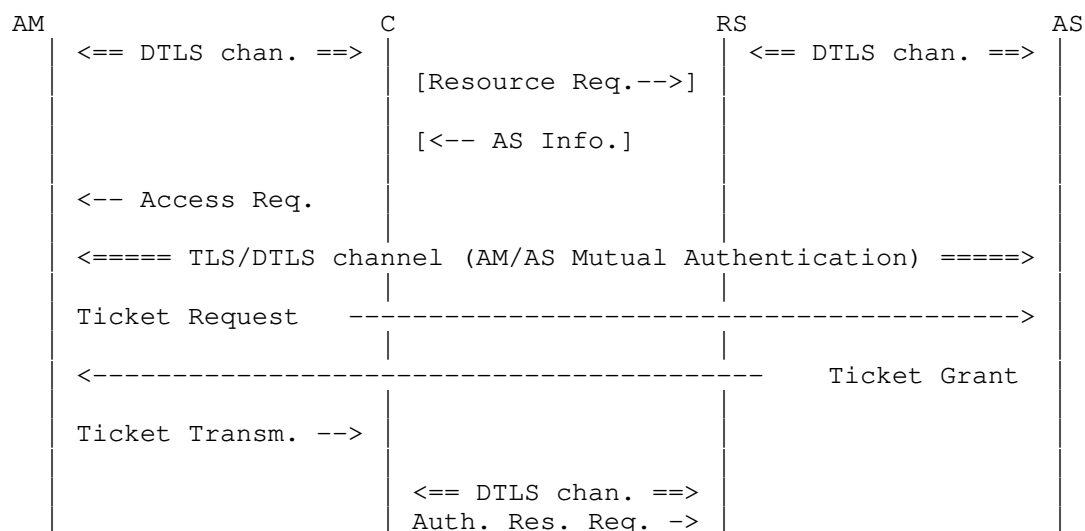


Figure 1: Protocol Overview

To determine the Authorization Server in charge of a resource hosted at the Resource Server (RS), the Client (C) MAY send an initial Unauthorized Resource Request message to RS. RS then denies the request and sends the address of its Authorization Server (AS) back to the Client.

Instead of the initial Unauthorized Resource Request message, C MAY look up the desired resource in a resource directory (cf.

[I-D.ietf-core-resource-directory]) that lists RS's resources as discussed in Section 9.

Once C knows AS' address, it can send a request for authorization to AS using its own Authentication Manager (AM). AS authenticates AM, who serves as a trusted introducer for C, and decides if C is allowed to communicate with RS and access the requested resource. If it is, AS generates an access ticket for C. The ticket contains keying material for the establishment of a secure channel and, if necessary, a representation of the permissions C has for the resource. C keeps one part of the access ticket and presents the other part to RS to prove its right to access. With their respective parts of the ticket, C and RS are able to establish a secure channel.

The following sections specify how CoAP is used to interchange access-related data between RS and AS so that AS can provide C and RS with sufficient information to establish a secure channel, and simultaneously convey authorization information specific for this communication relationship to RS.

This document uses JavaScript Object Notation (JSON, [RFC4627]) to express authorization information as set of attributes passed in CoAP payloads. Notation and encoding options are discussed in Section 5.

3.2. Unauthorized Resource Request Message

The optional Unauthorized Resource Request message is a request for a resource hosted by RS for which no proper authorization is granted. RS MUST treat any CoAP request as Unauthorized Resource Request message when any of the following holds:

- o The request has been received on an insecure channel.
- o RS has no valid access information for the sender of the request regarding the requested action on that resource.
- o RS has valid access information for the sender of the request, but this does not allow the requested action on the requested resource.

Note: These conditions ensure that RS can handle requests autonomously once access was granted and a secure channel has been established between C and RS.

Unauthorized Resource Request messages MUST be denied with a client error response. In this response, the Resource Server MUST provide proper AS Information to enable the Client to request an access

ticket from RS's Authorization Server as described in Section 3.3.

The response code MUST be 4.01 (Unauthorized) in case the sender of the Unauthorized Resource Request message is not authenticated, or if RS has no valid access ticket for C. If RS has authorization information for C but not for the resource that C has requested, RS MUST reject the request with a 4.03 (Forbidden). If RS has authorization information for C but they do not cover the action C requested on the resource, RS MUST reject the request with a 4.05 (Method Not Allowed).

Editor's Note: The use of the response codes 4.03 and 4.05 is intended to prevent infinite loops where a dumb Client optimistically tries to access a requested resource with any access token received from the AS. As malicious clients could pretend to be C to determine C's privileges, these detailed response codes must be used only when a certain level of security is already available which can be achieved only when the Client is authenticated.

3.3. AS Information Message

The AS Information Message is sent by RS as a response to an Unauthorized Resource Request message (see Section 3.2) to point the sender of the Unauthorized Resource Request message to RS's Authorization Server. The AS information is a set of attributes containing an absolute URI (see Section 4.3 of [RFC3986]) that specifies the Authorization Server in charge of RS.

The message MAY also contain a timestamp generated by RS.

Figure 2 shows an example for an AS Information message payload using JSON. (Refer to Section 5 for a detailed description of the available attributes and their semantics.)

```
4.01 Unauthorized
Content-Format: application/dcaf+json
{
  "AS": "coaps://as-rs.example.com/authorize",
  "TS": 168537
}
```

Figure 2: AS Information Payload Example

In this example, the attribute AS points the receiver of this message to the URI "coaps://as-rs.example.com/authorize" to request access permissions. The originator of the AS Information payload (i.e. RS) uses a local clock that is loosely synchronized with a time scale

common between RS and AS (e.g., wall clock time). Therefore, it has included a time stamp on its own time scale that is used as a nonce for replay attack prevention. Refer to Section 4.1 for more details concerning the usage of time stamps to ensure freshness of access tickets.

3.4. Access Request

To retrieve an access ticket for the resource that C wants to access, C sends an Access Request to its authentication manager AM. The Access Request is constructed as follows:

1. The request method is POST.
2. The request URI is set as described below.
3. The message payload contains a data structure that describes the action and resource for which C requests an access ticket.

The request URI identifies a resource at AM for handling authorization requests from C. The URI SHOULD be announced by AM in its resource directory as described in Section 9.

Note: Where capacity limitations of C do not allow for resource directory lookups, the request URI in Access Requests could be hard-coded during provisioning or set in a specific device configuration profile.

The message payload is constructed from the AS information that RS has returned in its AS Information message (see Section 3.3) and information that C provides to describe its intended request(s). The Access Request MUST contain the following attributes:

1. Contact information for the AS to use.
2. An identifier of C that can be used by AS to distinguish Access Requests from different Clients.
3. An absolute URI of the resource that C wants to access.
4. The actions that C wants to perform on the resource.
5. Any time stamp generated by RS.

The identifier of C included in the Access Request is used to distinguish the Clients within AM's namespace. To decide if a Client is allowed to access the requested resource, AS must rely on AM's verification of that Client's identity because AS cannot authenticate

the Client by itself. (Refer to Section 8 for a detailed discussion of the trust relationship between authentication managers and authorization servers.)

Note: Whenever the Resource Owner wants to specify access permissions for an individual client, the Authorization Server must be able to relate those permissions to that respective client using an identifier that is globally unique.

An example Access Request from C to AM is depicted in Figure 3. (Refer to Section 5 for a detailed description of the available attributes and their semantics.)

```
POST client-authorize
Content-Format: application/dcaf+json
{
  "AS": coaps://as-rs.example.com/authorize",
  "CI": "node-588",
  "M": [ "GET", "PUT" ],
  "R": "coaps://temp451.example.com/s/tempC",
  "TS": 168537
}
```

Figure 3: Access Request Message Example

The example shows an Access Request message payload for the resource `/s/tempC` on the Resource Server `temp451.example.com`. Requested operations in attribute M are GET and PUT. The requesting client is identified as `node-588`.

The attributes AS (that denotes the Authorization Server to use) and TS (a nonce generated by RS) are taken from the AS Information message from RS.

The response to an Authorization Request is delivered by AM back to C in a Ticket Transfer message.

3.5. Ticket Request Message

When AM receives an Access Request message from C it MAY return a cached response if it is known to be fresh. Otherwise, it checks whether the request payload is of type `application/dcaf+json` and contains at least the fields AS, CI, M, and R. AM MUST respond with 4.00 (Bad Request) if the type is `application/dcaf+json` and any of these fields is missing or does not conform to the format described in Section 5. Content formats other than `application/dcaf+json` are out of scope of this specification.

When the payload is correct, AM creates a Ticket Request message from the Access Request received from C as follows:

1. The destination of the Ticket Request message is derived from the authority information in the URI contained in field "AS" of the Access Request message payload.
2. The request method is POST.
3. The request URI is constructed from the AS field received in the Access Request message payload.
4. The payload is copied from the Access Request sent by C.

To send the Ticket Request message to AS a secure channel between AM and AS MUST be used. Depending on the URI scheme used in the AS field of the Access Request message payload, this could be, e.g., a DTLS channel (for "coaps") or a TLS connection (for "https"). AM and AS MUST be able to mutually authenticate each other, e.g. based on a public key infrastructure. (Refer to Section 8 for a detailed discussion of the trust relationship between authentication managers and authorization servers.)

3.6. Ticket Grant Message

When AS has received a Ticket Request message it has to evaluate the access request information contained therein. First, it checks whether the request payload is of type "application/dcaf+json" and contains at least the fields AS, CI, M, and R. AS MUST respond with 4.00 (Bad Request) for CoAP (or 400 for HTTP) if the type is "application/dcaf+json" and any of these fields is missing or does not conform to the format described in Section 5.

AS decides whether or not access is granted to the requested resource and then creates a Ticket Grant message that reflects the result. To grant access to the requested resource, AS creates an access ticket comprised of a Face and a Verifier as described in Section 4.1.

The Ticket Grant message then is constructed as a success response indicating attached content, i.e. 2.05 for CoAP, or 200 for HTTP, respectively. The payload of the Ticket Grant message is a data structure that contains the result of the access request. When access is granted, the data structure contains the ticket's Face, the Verifier and the Session Key Generation Method.

The Ticket Grant message MAY provide cache-control options to enable intermediaries to cache the response. The message MAY be cached according to the rules defined in [I-D.ietf-core-coap] to facilitate

ticket retrieval when C has crashed and wants to recover the DTLS session with RS.

AS sets Max-Age according to the ticket lifetime in its response (Ticket Grant Message).

Figure 4 shows an example Ticket Grant message using CoAP. The Face/Verifier information is transferred as a JSON data structure as specified in Section 5. The Max-Age option tells the receiving AM how long this ticket will be valid.

2.05 Content

Content-Format: application/dcaf+json

Max-Age: 86400

```
{ "F": {  
    "AI": { "Role" : 3 },  
    "CI": "2001:db8:ab9:1234:7920:3133:ae5f:87",  
    "TS": "2013-07-10T10:04:12.391",  
    "L": 86400,  
    "G": "hmac_sha256"  
  },  
  "V": "w+ZeJx5MxIEkt7yBMWjX6ztSYcIBTz+sv4z98m+PUEY="
```

Figure 4: Example Ticket Grant Message

A Ticket Grant message that declines any operation on the requested resource is illustrated in Figure 5. As no ticket needs to be issued, an empty payload is included with the response.

2.05 Content

Content-Format: application/dcaf+json

Figure 5: Example Ticket Grant Message With Reject

3.7. Ticket Transfer Message

A Ticket Transfer message delivers the access information sent by AS in a Ticket Grant message to the requesting client C. The Ticket Transfer message is the response to the Access Request message sent from C to AM and includes any access information from AS contained in the Ticket Grant message.

3.8. DTLS Channel Setup Between C and RS

Using the information contained in a positive response to its Access Request (i.e. a Ticket Transfer message that contains a Face and a Verifier), C can initiate establishment of a new DTLS channel with

RS. To use DTLS with pre-shared keys, C follows the PSK key exchange algorithm specified in Section 2 of [RFC4279], with the following additional requirements:

1. C sets the `psk_identity` field of the ClientKeyExchange message to the ticket Face received in the Ticket Transfer message.
2. C uses the ticket Verifier as PSK when constructing the premaster secret.

Note1: As RS cannot provide C with a meaningful PSK identity hint in response to C's ClientHello message, RS SHOULD NOT send a ServerKeyExchange message.

Note2: According to [I-D.ietf-core-coap], CoAP implementations MUST support the ciphersuite `TLS_PSK_WITH_AES_128_CCM_8` [RFC6655]. C is therefore expected to offer at least this ciphersuite to RS.

Note3: The ticket is constructed by AS such that RS can derive the authorization information as well as the PSK (refer to Section 6 for details).

3.9. Authorized Resource Request Message

Successful establishment of the DTLS channel between C and RS ties the authorization information contained in the `psk_identity` field to this channel. Any request that RS receives on this channel is checked against these authorization rules. Incoming CoAP requests that are not Authorized Resource Requests MUST be rejected by RS with 4.01 response as described in Section 3.2.

RS SHOULD treat an incoming CoAP request as Authorized Resource Request if the following holds:

1. The message was received on a secure channel that has been established using the procedure defined in Section 3.8.
2. The authorization information tied to the secure channel is valid.
3. The request is destined for RS.
4. The resource URI specified in the request is covered by the authorization information.
5. The request method is an authorized action on the resource with respect to the authorization information.

Note that the authorization information is not restricted to a single resource URI. For example, role-based authorization can be used to authorize a collection of semantically connected resources simultaneously. Implicit authorization also provides access rights to authenticated clients for all actions on all resources that RS offers. As a result, C can use the same DTLS channel not only for subsequent requests for the same resource (e.g. for block-wise transfer as defined in [I-D.ietf-core-block] or refreshing observe-relationships [I-D.ietf-core-observe]) but also for requests to distinct resources.

Incoming CoAP requests received on a secure channel according to the procedure defined in Section 3.8 MUST be rejected

1. with response code 4.03 (Forbidden) when the resource URI specified in the request is not covered by the authorization information, and
2. with response code 4.05 (Method Not Allowed) when the resource URI specified in the request covered by the authorization information but not the requested action.

Since AS may limit the set of requested actions in its Ticket Grant message, C cannot know a priori if a an Authorized Resource Request will succeed.

4. Ticket

Access tokens in DCAF are tickets that consist of two parts, namely the Face and the Verifier.

4.1. Face

Face is the part of the ticket generated for RS. Face MUST contain all information needed for authorized access to a resource:

- o Authorization Information
- o Client Identifier
- o A timestamp generated by AS

Optionally, Face MAY also contain:

- o A lifetime (optional)
- o A DTLS pre-shared key (optional)

RS MUST verify the integrity of Face, i.e. the information contained in Face stems from AS and was not manipulated by anyone else.

Face MUST contain a timestamp to verify that the contained information is fresh. As constrained devices may not have a clock, timestamps MAY be generated using the clock ticks since the last reboot. To circumvent synchronization problems the timestamp MAY be generated by RS and included in the first AS Information message. Alternatively, AS MAY generate the timestamp. In this case, AS and RS MUST use a time synchronization mechanism to make sure that RS interprets the timestamp correctly.

Face MAY be encrypted. If Face contains a DTLS PSK, the whole content of Face MUST be encrypted.

Note: The integrity of Face can be ensured by various means. Face may be encrypted by AS with a key it shares with RS. Alternatively, RS can use a mechanism to generate the DTLS PSK which includes Face and is only able to calculate the correct key with the correct Face (refer to Section 6 for details).

4.2. Verifier

The Verifier part of the ticket is generated for C. It contains the DTLS PSK for C. The Verifier MUST NOT be transmitted over insecure channels.

4.3. Revocation

The existence of access tickets SHOULD be limited in time. This can be achieved either by explicit Revocation Messages to invalidate a ticket or implicitly by attaching a lifetime to the ticket.

4.3.1. Lifetime

Tickets MAY have a lifetime. AS is responsible for defining the ticket lifetime. If AS sets a lifetime for a ticket, AS and RS MUST use a time synchronization method to ensure that RS is able to interpret the lifetime correctly. RS SHOULD end the DTLS connection to C if the lifetime of a ticket has run out and it MUST NOT accept new requests. RS MUST NOT accept tickets with an invalid lifetime.

Note: Defining reasonable ticket lifetimes is difficult to accomplish. How long a client needs to access a resource depends heavily on the application scenario and may be difficult to decide for AS.

4.3.2. Revocation Messages

AS MAY revoke tickets by sending a ticket revocation message to RS. If RS receives a ticket revocation message, it MUST end the DTLS connection to C and MUST NOT accept any further requests from C.

If ticket revocation messages are used, RS MUST check regularly if AS is still available. If RS cannot contact AS, it MUST end all DTLS connections and reject any further requests from C.

Note: The loss of the connection between RS and AS prevents all access to RS. This might especially be a severe problem if AS is responsible for several Resource Servers or even a whole network.

5. Payload Format and Encoding (application/dcaf+json)

Various messages types of the DCAF protocol carry payloads to express authorization information and parameters for generating the DTLS PSK to be used by C and RS. In this section, a representation in JavaScript Object Notation (JSON, [RFC4627]) is defined.

The following attributes are defined:

- AS: Authorization Server. This attribute denotes the authorization server that is in charge of the resource specified in attribute R. The attribute's value is a string that contains an absolute URI according to Section 4.3 of [RFC3986].
- AI: Authorization Information. A data structure used to convey authorization information from AS to RS (see below).
- CI: Client Identity. A string that identifies the initiator of the authorization request. This label MAY be a fully qualified domain name, an IP address, or any other character literal that is used by the Authorization Server to decide whether or not access is granted to the requesting entity.
- E: Encrypted Ticket Face. A string containing an encrypted ticket Face encoded as base64 according to Section 4 of [RFC4648].
- K: Key. A string that identifies the shared key between RS and AS that can be used to decrypt the contents of E. If the attribute E is present and no attribute K has been specified, the default is to use the current session key for the secured channel between RS and AS.
- M: Methods. The list of actions to be performed on the resource R, encoded as an array of strings. In an authorization request, this list contains the actions that the initiator of the request wants to perform. In an authorization ticket, this attribute denotes the actions that are permitted.
- R: Resource. A string that denotes the absolute URI of the resource to be accessed. As the access ticket is requested in order to establish a DTLS connection with the server that hosts this resource, the URI scheme typically is "coaps".
- TS: Time Stamp. An optional time stamp that indicates the instant when the access ticket request was formed. This attribute can be used by the resource server in an AS Information message to convey a time stamp in its local time scale (e.g. when it does not have a real time clock with synchronized global time). When the

attribute's value is encoded as a string, it MUST contain a valid UTC timestamp without time zone information. When encoded as integer, TS contains a system timestamp relative to the local time scale of its generator, usually RS.

- L: Lifetime. A lifetime of the ticket. When encoded as a string, L MUST denote the ticket's expiry time as a valid UTC timestamp without time zone information. When encoded as an integer, L MUST denote the ticket's validity period in seconds relative to TS.
- G: DTLS PSK Generation Method. A string that identifies the method that RS MUST use to derive the DTLS PSK from the ticket Face. This attribute MUST NOT be used when attribute V is present within the contents of F.
- F: Ticket Face. An object containing the fields AI, CI, TS, and optionally G, L and V.
- V: Ticket Verifier. A string containing the shared secret between C and RS, encoded as base64 according to Section 4 of [RFC4648].

The exact specification of AI is out of scope for this document. AI may, e.g., contain a single role identifier known by RS, or an array of pairs (M, R) with M and R defined as above.

As AI is used to authorize resource access as defined in Section 3.9, RS MUST be able to interpret the contained information.

5.1. Examples

The following example specifies an Authorization Server that will be accessed using HTTP over TLS. The request URI is set to "/a?ep=%5B2001:DB8::dcaf:1234%5D" (hence denoting the endpoint address to authorize). TS denotes a local timestamp in UTC.

```
POST /a?ep=%5B2001:DB8::dcaf:1234%5D HTTP/1.1
Host: as-rs.example.com
Content-Type: application/dcaf+json

{
  "AS": "https://as-rs.example.com/a?ep=%5B2001:DB8::dcaf:1234%5D",
  "CI": "2001:DB8::dcaf:1234",
  "M": [ "GET" ],
  "R": "coaps://temp451.example.com/s/tempC",
  "TS": "2013-07-14T11:58:22.923"
}
```

The following example shows a ticket for the distributed key

generation method (cf. Section 6.2), comprised of a Face (F) and a Verifier (V). The Face data structure contains authorization information AI with an application-specific role identifier, a client identifier, a timestamp using the local time scale of RS, and a lifetime relative to RS's time scale.

The DTLS PSK Generation Method is set to "hmac_sha256" denoting that the distributed key derivation is used as defined in Section 6.2 with SHA-256 as HMAC function.

The Verifier V contains a shared secret to be used as DTLS PSK between C and RS.

HTTP/1.1 200 OK
Content-Type: application/dcaf+json

```
{
  "F": {
    "AI": { "Role" : 3 },
    "CI": "2001:db8:ab9:1234:7920:3133:ae5f:87",
    "TS": 2938749,
    "L": 3600,
    "G": "hmac_sha256"
  },
  "V": "zrPOuc6x zr/Pjc+Bz4TOuSDOvM6lIM68zq3Ou865Cg=="
}
```

The Face may be encrypted as illustrated in the following example. Here, the field E carries an encrypted and base64-encoded Face data structure that contains the same information as the previous example, and an additional Verifier. Encryption was done with a secret shared by AS and RS. (This example uses AES128_CCM with the secret { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f } and RS's timestamp { 0x00, 0x2C, 0xD7, 0x7D } as nonce.) Line breaks have been inserted to improve readability.

The attribute K describes the identity of the key to be used by RS to decrypt the contents of attribute E. Here, The value "key0" in this example is used to indicate that the shared session key between RS and AS was used for encrypting E.

```
{
  "E": "rjtolfjyX9q7Emxgsnz+nf0xTQhelMjzZBRoIEW4vmSVlyJdW4KDgVtW
LyBnQSVX0lmVpxUYbdNuk/5PkCOJBeex0obiEBC1UmKoJfJfjy7bLQhq
k9HuJD7cvjHNOVZtNZf5qrxt7xJSofFe6j/SJuxGNH/72SPDrdrMQeXJI
pX6vCJB698FcRDOXh/ipi9KT8YWeo/ljUMgJc+LI",
  "K": "key0",
  "V": "zrPOuc6x zr/Pjc+Bz4TOuSDOvM6lIM68zq3Ou865Cg=="
}
```

```
}
```

The decrypted contents of E are depicted below (whitespace has been added to improve readability). The presence of the attribute V indicates that the DTLS PSK Transfer is used to convey the session key (cf. Section 6.1).

```
"F":{"AI":{"Role":3},  
"CI":"2001:db8:ab9:1234:7920:3133:ae5f:87",  
"TS":2938749,  
"L":3600,  
"V":"zrPOuc6x zr/Pjc+Bz4TOuSDOvM6lIM68zq3Ou865Cg=="}
```

6. DTLS PSK Generation Methods

One goal of the DCAF protocol is to provide for a DTLS PSK shared between C and RS. AS and RS MUST negotiate the method for the DTLS PSK generation.

6.1. DTLS PSK Transfer

The DTLS PSK is generated by AS and transmitted to C and RS using a secure channel.

The DTLS PSK transfer method is defined as follows:

- o AS generates the DTLS PSK using an algorithm of its choice
- o AS MUST include a representation of the DTLS PSK in Face and encrypt it together with all other information in Face with a key $K(AS,RS)$ it shares with RS. How AS and RS exchange $K(AS,RS)$ is not in the scope of this document. AS and RS MAY use their preshared key as $K(AS,RS)$.
- o AS MUST include a representation of the DTLS PSK in the Verifier.
- o As AS and C do not have a shared secret, the Verifier MUST be transmitted to C using encrypted channels.
- o RS MUST decrypt Face using $K(AS,RS)$

6.2. Distributed Key Derivation

AS generates a DTLS PSK for C which is transmitted using a secure channel. RS generates its own version of the DTLS PSK using the information contained in Face (see also Section 4.1).

The distributed key derivation method is defined as follows:

- o AS and RS both generate the DTLS PSK using the information included in Face. They use an HMAC algorithm on Face with a shared key. The result serves as the DTLS PSK. How AS and RS negotiate the used HMAC algorithm is not in the scope of this document. They MAY however use the HMAC algorithm they use for their DTLS connection.
- o AS MUST include a representation of the DTLS PSK in the Verifier.
- o As AS and C do not have a shared secret, the Verifier MUST be transmitted to C using encrypted channels.

- o AS MUST NOT include a representation of the DTLS PSK in Face.
- o AS MUST NOT encrypt Face.

7. Authorization Configuration

For the protocol defined in this document, proper configuration of AS is crucial. The principal who owns the resources hosted by RS (i.e. the Resource Owner) needs to define permissions for the resources. The data representation of these permissions are not in the scope of this document.

8. Trust Relationships

C trusts AM, and RS trusts AS. Obviously, AM trusts C with the specific permissions it hands over to it. How this trust is established, is not in the scope of this document. It may be achieved by using a bootstrapping mechanism similar to [bergmann12].

Additionally, AS and AM need to have a trust relationship established. Its establishment is also not in the scope of this document. It fulfills the following conditions:

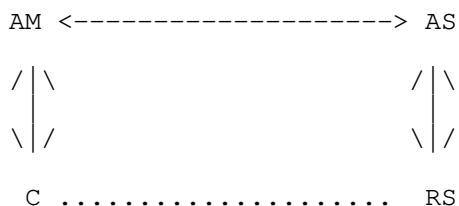
1. AS has means to authenticate AM (e.g. it has a certificate of AM or a PKI in which AM is included) and vice versa
2. As far as AS needs to rely on the different clients of AM to receive different permissions, it can be sure that AM correctly identifies these clients towards AS and does not leak tickets that have been generated for a specific client C to another client.

AS trusts C indirectly because it trusts AM and AM vouches for C. The DCAF Protocol does not provide any means for AS to validate that a resource requests stems from C.

C indirectly trusts AS with some potentially confidential information, and that AS correctly represents RS, because AM trusts AS.

AM trusts RS indirectly because it trusts AS and AS vouches for RS.

C implicitly trusts RS with some potentially confidential information because it trusts AM and because RS can prove that it shares a key with AS.



9. Listing Authorization Server Information in a Resource Directory

CoAP utilizes the Web Linking format [RFC5988] to facilitate discovery of services in an M2M environment. [RFC6690] defines specific link parameters that can be used to describe resources to be listed in a resource directory [I-D.ietf-core-resource-directory].

This section defines a resource type "auth-request" that can be used by clients to retrieve the request URI for a server's authorization service. When used with the parameter rt in a web link, "auth-request" indicates that the corresponding target URI can be used in a POST message to request authorization for the resource and action that are described in the request payload.

The Content-Format "application/dcaf+json" with numeric identifier TBD1 defined in this specification MAY be used to express access requests and their responses.

The following example shows the web link used by AM in this document to relay incoming Authorization Request messages to AS. (Whitespace is included only for readability.)

```
<client-authorize>;rt="auth-request";ct=TBD1
;title="Contact Remote Authorization Server"
```

The resource directory that hosts the resource descriptions of RS could list the following description. In this example, the URI "ep/node138/a/switch2941" is relative to the resource context "coaps://as-rs.example.com/", i.e. the authorization server AS.

```
<ep/node138/a/switch2941>;rt="auth-request";ct=TBD1;ep="node138"
;title="Request Client Authorization"
;anchor="coaps://as-rs.example.com/"
```

10. Examples

This section gives a number of short examples with message flows for the initial Unauthorized Resource Request and the subsequent retrieval of a ticket from AS. The notation here follows the role conventions defined in Section 1.2.1. The payload format is encoded as proposed in Section 5. The IP address of AS is 2001:DB8::1, the IP address of RS is 2001:DB8::dcaf:1234, and C's IP address is 2001:DB8::c.

10.1. Access Granted

This example shows an Unauthorized PUT request from C to RS that is answered with an AS Information message. C then sends a POST request to AM with a description of its intended request. AM forwards this request to AS using CoAP over a DTLS-secured channel. The response from AS contains an access ticket that is relayed back to AM.

```
C --> RS
PUT a/switch2941 [Mid=1234]
Content-Format: application/senml+json
{e: [{"bv": "1"}]}

C <-- RS
4.01 Unauthorized [Mid=1234]
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941"}

C --> AM
POST client-authorize [Mid=1235,Token="tok"]
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
 "CI": "2001:DB8::c",
 "M": [ "PUT" ],
 "R": "coaps://[2001:DB8::dcaf:1234]/a/switch2941"
}

AM --> AS [Mid=23146]
POST ep/node138/a/switch2941
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
 "CI": "2001:DB8::c",
 "M": [ "PUT" ],
 "R": "coaps://[2001:DB8::dcaf:1234]/a/switch2941"
}

AM <-- AS
2.05 Content [Mid=23146]
```

```

Content-Format: application/dcaf+json
{ "F": { "AI": { "R" : "a/switch2941", "M" : [ "GET", "PUT" ] },
          "CI": "2001:DB8::c",
          "TS": "2013-07-04T20:17:38.002",
          "G": "hmac_sha256"
        },
  "V": "yYVLyZZ5Nssbn0by3fqel9WK6jHdoYyNej2d/kSuBLw="
}

C <-- AM
2.05 Content [Mid=1235,Token="tok"]
Content-Format: application/dcaf+json
{ "F": { "AI": { "R" : "a/switch2941", "M" : [ "GET", "PUT" ] },
          "CI": "2001:DB8::c",
          "TS": "2013-07-04T20:17:38.002",
          "G": "hmac_sha256"
        },
  "V": "MR5TMrNngbSEAkF10akmsdbmzF0gqxGI/d3KjwT8GxI="
}

C --> RS
ClientHello (TLS_PSK_WITH_AES_128_CCM_8)

C <-- RS
ServerHello (TLS_PSK_WITH_AES_128_CCM_8)
ServerHelloDone

C --> RS
ClientKeyExchange
  psk_identity=' "F":{ "AI":{ "R": "a/switch2941", "M": [ "GET", "PUT" ] },
                    "CI": "2001:DB8::c",
                    "TS": "2013-07-04T20:17:38.002",
                    "G": "hmac_sha256" }' )

(C decodes the contents of V and uses the result as PSK)
ChangeCipherSpec
Finished

(RS calculates PSK from AI, CI, TS and its session key
HMAC_sha256("{ \"R\": \"a/switch2941\", \"M\": [ \"GET\", \"PUT\" ] }"+
             "2001:DB8::c"+"2013-07-04T20:17:38.002",
             "secret")
= 311e5332b36781b484024165d1a926b1d6e6cc5d20ab1188fdddca8f04fc1b12
)

C <-- RS
ChangeCipherSpec
Finished

```

10.2. Access Denied

This example shows a denied Authorization request for the DELETE operation.

```
C --> RS
DELETE a/switch2941
```

```
C <-- RS
4.01 Unauthorized
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941"}
```

```
C --> AM
POST client-authorize
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
 "CI": "2001:DB8::c",
 "M": [ "DELETE" ],
 "R": "coaps://[2001:DB8::dcaf:1234]/a/switch2941"
}
```

```
AM --> AS
POST ep/node138/a/switch2941
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
 "CI": "2001:DB8::c",
 "M": [ "DELETE" ],
 "R": "coaps://[2001:DB8::dcaf:1234]/a/switch2941"
}
```

```
AM <-- AS
2.05 Content
Content-Format: application/dcaf+json
```

```
C <-- AM
2.05 Content
Content-Format: application/dcaf+json
```

10.3. Access Restricted

This example shows a denied Authorization request for the operations GET, PUT, and DELETE. AS grants access for PUT only.

```
AM --> AS
POST ep/node138/a/switch2941
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
 "CI": "2001:DB8::c",
 "M": [ "GET", "PUT", "DELETE" ],
 "R": "coaps://[2001:DB8::dcaf:1234]/a/switch2941"
}

AM <-- AS
2.05 Content
Content-Format: application/dcaf+json
{ "F": { "AI": { "R" : "a/switch2941", "M" : [ "GET", "PUT" ] },
        "CI": "2001:DB8::c",
        "TS": "2013-07-04T21:33:11.930",
        "G": "hmac_sha256"
      },
  "V": "NZ8Q3o8P4eHOzkoscaUpoRvrn5d74Cscw/aXAiNmC/k="
}
```

10.4. Implicit Authorization

This example shows an Authorization request using implicit authorization. AM initially requests the actions GET and POST on the resource "coaps://[2001:DB8::dcaf:1234]/a/switch2941". AS returns a ticket that has no AI field in its ticket Face, hence implicitly authorizing C.

```
AM --> AS
POST ep/node138/a/switch2941
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
 "CI": "2001:DB8::c",
 "M": [ "GET", "POST" ],
 "R": "coaps://[2001:DB8::dcaf:1234]/a/switch2941"
}

AM <-- AS
2.05 Content
Content-Format: application/dcaf+json
{ "F": { "CI": "2001:DB8::c",
        "TS": "2013-07-16T10:15:43.663",
        "G": "hmac_sha256"
      },
  "V": "mCYtLG/oZIWki/mCFNJ4nxIOWMPw1DKAnXIo/ir2dtI="
}
```

11. Security Considerations

As this protocol builds on transitive trust between authorization servers as mentioned in Section 8, AS has no direct means to validate that a resource request originates from C. It has to trust AM that it correctly vouches for C and that it does not give authorization tickets meant for C to another client nor disclose the contained session key.

The Authorization Server also could constitute a single point of failure. If the Authorization Server fails, the resources on all Resource Servers it is responsible for cannot be accessed any more. Thus, it is crucial for large networks to use Authorization Servers in a redundant setup.

12. IANA Considerations

The following registrations are done following the procedure specified in [RFC6838].

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

12.1. dcaf+json Media Type Registration

Type name: application

Subtype name: dcaf+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC4627]. Specifically, only the primitive data types String and Number are allowed. The type Number is restricted to unsigned integers (i.e., no negative numbers, fractions or exponents are allowed). Encoding MUST be UTF-8. These restrictions simplify implementations on devices that have very limited memory capacity.

Security considerations: TBD

Interoperability considerations: TBD

Published specification: [RFC-XXXX]

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): dcaf

Macintosh file type code(s): none

Person & email address to contact for further information: TBD

Intended usage: COMMON

Restrictions on usage: None

Author: TBD

Change controller: IESG

12.2. CoAP Content Format Registration

This document specifies a new media type `application/dcaf+json` (cf. Section 12.1). For use with CoAP, a numeric Content-Format identifier is to be registered in the "CoAP Content-Formats" sub-registry within the "CoRE Parameters" registry.

Note to RFC Editor: Please replace all occurrences of "RFC-XXXX" with the RFC number of this specification.

Media type	Encoding	Id.	Reference
<code>application/dcaf+json</code>	-	TBD1	[RFC-XXXX]

13. References

13.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013.

13.2. Informative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-13 (work in progress), October 2013.
- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-11 (work in progress), October 2013.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-ietf-core-resource-directory-00 (work in progress), June 2013.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [bergmann12] Bergmann, O., Gerdes, S., Schaefer, S., Junge, F., and C. Bormann, "Secure Bootstrapping of Nodes in a CoAP Network", IEEE Wireless Communications and Networking Conference Workshops (WCNCW), April 2012.

Authors' Addresses

Stefanie Gerdes
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63906
Email: gerdes@tzi.org

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63904
Email: bergmann@tzi.org

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CoRE
Internet-Draft
Intended status: Informational
Expires: March 10, 2014

B. Greevenbosch
Huawei Technologies
September 06, 2013

Use cases and requirements for authentication and authorisation in CoAP
draft-greevenbosch-core-authreq-00

Abstract

This draft describes use cases and requirements for authenticated and authorised CoAP. The draft especially focuses on threats and their prevention.

Note

Discussion and suggestions for improvement are requested, and should be sent to core@ietf.org.

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 <http://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 March 10, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Requirements notation	2
2. Introduction	2
3. Use cases	3
3.1. Authorised and unauthorised devices	3
3.2. Home security	3
3.3. Illegal smart-meters	4
3.4. Maintaining and extending a network of sensors and actuators	4
3.5. Discovered compromised device	5
3.6. Vulnerability discovery in actuators in a chemical plant	5
3.7. Revocation of a non-compromised device	5
3.8. Mixing nodes from different vendors	6
3.9. Privacy of medical communications	6
4. Requirements	7
5. Discussion	8
5.1. Certificate authority	8
5.2. Expiry	8
5.3. Time of revocation	8
6. Security considerations	8
7. IANA considerations	9
8. Acknowledgements	9
9. References	9
9.1. Normative References	9
9.2. Informative References	9
Author's Address	9

1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

This draft describes use cases and requirements for secure authentication and authorisation, as well as their expiry and revocation, in CoAP.

The draft consists of the following parts:

- o The draft starts with several use cases.

- o A section with requirements related to the use cases follows.
- o Discussion of the various security trade-offs that need to be made can be found in Section 5.

The goal of the draft is to provide background material for usage when defining a solution for authorised CoAP.

3. Use cases

3.1. Authorised and unauthorised devices

Company A produces sensor devices. These devices are of high quality, and no vulnerabilities have been detected. As such, they have been certified to be used in a wide area of applications.

Company B produces also sensor devices. However, these devices are of low quality, and have known security issues. They failed the certification requirements.

Company C is oblivious of this fact, and since it needs this kind of sensors to monitor its industrial process, it buys some to test.

During installation of the sensors into Company C's monitoring network, the credentials of the sensors are verified by the system. The sensors from Company A install without problem. However, for the sensors from Company B the authentication fails, and the installation of the sensors is refused. The system informs the installation engineers about the reason of failure.

Fortunately the authentication mechanism revealed that the sensors from Company B are not to be used. This avoided a lot of trouble and potential security issues.

3.2. Home security

Henry has an advanced home security system. The security system provides protection against burglary, as well as against fire. It has sensors on doors, motion sensors, smoke detectors, cameras etc. It also has actuators for the electronic locks, a sprinkler system and actuators that can close the gas tap and cut the electricity.

The system comes with tokens. These tokens are used to turn on or off part of the system, and allow certain actions that need human interaction. One of these actions is to open or close the front door lock. Henry has provided a token to each of his family members.

The system has a solid authorisation and authentication model, ensuring that only Henry and his family's tokens can drive the system. Even though the tokens can be bought in a regular store, only tokens that Henry has approved can be used in the system.

Certain peripherals allow different access rights to different entities. For example, the electricity closure can only be set by Henry and the master system, whereas its on/off status can be read by all family members.

All peripherals are certified by an impartial certification body, which has specified minimum security requirements. In this way, Henry is assured that when he adds a new peripheral and it is accepted by the system, it can be deemed reliable.

3.3. Illegal smart-meters

An electricity company depends on smart-meters to measure energy usage of the households it serves. The gathered information is used for several purposes, billing being one of them.

On the black market, there appear illegal smart-meters that only report 75% of the actual electricity usage. These smart-meters are based on a clone of a valid public key.

Once the electricity company discovers this, it revokes the associated public key, thereby ensuring that the illegal meters cannot be installed anymore.

3.4. Maintaining and extending a network of sensors and actuators

An agricultural company uses an IP network to ensure an optimal climate for the vegetables they grow in their green houses. Sensors do measurements about e.g. humidity and sunlight, whereas actuators can drive artificial rain and supporting light. A central controller is responsible for processing the sensor readings and driving the actuators accordingly.

Sometimes, a sensor or actuator needs replacement as part of the normal maintenance cycle. This is a routine task for the associated engineer, and involves simply disconnecting the old apparatus and connecting a new one. The rest of the installation to the network happens automatically.

As the agricultural company is doing good business, it decides to expand. It buys another piece of land, and modernises the green house that was already built on the land. The modernisation includes installing new sensors and actuators, which are seamlessly integrated

into the already existent network, such that they can work with the central controller too.

The use case illustrates the need to be able to automatically install and update network nodes in an existing network. It is also important to note, that installation of the network nodes includes proper authentication and authorisation. After all, the agricultural company does not want outsiders to be able to influence the climate in the green houses, for example by driving the actuators or modifying the sensor readings.

3.5. Discovered compromised device

Company A has a certain type of actuators installed throughout its building. On a certain time, some of these actuators start behaving funny. It turns out that some hackers have been able to access the sensors, and drive them as they wish.

Company A can't de-install the actuators immediately, after all, they are installed everywhere in the building. Instead Company A has the actuators revoked, and then can replace them on a less hasty schedule.

3.6. Vulnerability discovery in actuators in a chemical plant

A chemical plant deploys sensors for the several properties of the substance being produced, and actuators that start certain processes when the substance is ready for the next step.

A vulnerability in certain of the actuators is discovered; it would allow unauthorised third parties to take over the actuators and start processes at their will.

After the discovery of the vulnerability, the chemical plant proactively de-activates the actuators and revokes their keys. It then makes sure the vulnerability is resolved as quickly as possible, such that normal production can resume.

3.7. Revocation of a non-compromised device

Jack worked at the IT department of company E.

However, due to a conflict with the company, Jack has been fired. When leaving, he smuggled out some tokens used to control several of the company's peripherals.

When the company realises it misses the tokens, it revokes them to ensure they cannot be used to control the peripherals anymore.

Jack fails to wreak havoc as his revenge, and neither can he sell the tokens to other adversaries.

3.8. Mixing nodes from different vendors

A weather analysis and forecast agency needs global coverage for collection of temperature and air-pressure data. It has contracts with several local authorities and companies for the placement of their sensors.

For both logistic and economic reasons, the weather agency does not want to rely on one particular type of sensor from a single vendor. Instead, it wants to allow different sensors from different vendors, as long as these sensors meet certain criteria concerning precision, response time and reliability.

To ensure the criteria are met, the weather agency performs several tests with new candidate sensors. When the sensors pass the tests, the agency allows their usage in its network. When the sensors fail the tests, the agency is ensured that they cannot be used for collecting data, lest the quality of the agency's analysis and forecast suffer from data of bad quality.

In this use case, the vendor pro-actively controls which sensor types can be used in their network. It uses an authentication and authorisation mechanism to automatically ensure that only those types it has approved can be installed. The use case illustrates the need for interoperability in authentication between nodes manufactured by different vendors, as well as the need to exclude nodes that are not authorised to join the network.

3.9. Privacy of medical communications

Mr P has developed a heart problem. To diagnose and monitor the condition of Mr P's heart, his cardiologist has requested Mr P to wear a sensor during the day. The sensor measures the heartbeat and other vital functions. The sensor transmits this information to the hospital, generally once every day. When needed, e.g. when a situation occurs that requires extra attention, the sensor can also send information ad-hoc.

Protecting the integrity of the sensor readings is important, even when it is unlikely that an adversary will tamper with the sensor readings. After all, doing so would constitute a serious crime. Protecting Mr P's privacy adds significantly to the value of a solid security model in this use case. In any case eavesdropping needs to be prevented, and that includes man-in-the-middle attacks.

4. Requirements

This section lists requirements associated with authentication and authorisation in CoAP:

1. It SHALL be possible to verify the binding between the key and the entity associated with it.
2. It SHALL be possible to verify whether an entity is authorised to establish the connection.
3. It SHALL be possible to specify authorisation for a specific resource.
4. It SHOULD be possible to specify authorisation based on the message type.
5. It SHALL NOT be possible for an unauthorised third party to establish a cryptographic relationship.
6. There SHALL be a mechanism that allows revocation of previously granted authorisation.
7. It SHALL be possible for a receiver to determine whether a key has been revoked.
8. It SHALL be possible to perform authentication, authorisation and revocation verification fully automatically.
9. The verification technology MUST NOT require much complexity on constrained entities.
10. The verification mechanism SHALL be scalable, allowing potentially millions of entities to verify authentication and authorisation.
11. It SHOULD be possible to specify an expiry date for keys and/or authorisation.
12. It SHALL be possible to revoke compromised keys.
13. Revocation SHALL NOT require physically unplugging the device.
14. There SHALL be protection against an unauthorised third party spoofing authorisation and/or revocation of keys and entities.
15. There SHOULD be protection against denial of service (DoS) attacks, as far as it is feasible.

5. Discussion

In this section, we discuss the various trade-offs that need to be made, and implications they may have.

5.1. Certificate authority

Much of a traditional Public Key Infrastructure depends on a certificate authority. The certificate authority (CA) signs the certificate of the device, or an intermediate certificate that signs the certificate of the device.

This creates islands of trust, in which the CA has the power to revoke any key on its island. Interoperability between devices of different CAs may still be possible, depending on which CAs the entities trust apart from their own CA.

5.2. Expiry

X.509 certificates [X.509] contain an expiry date. This means that the certificates automatically become invalid after a time has passed. Should the device's lifetime be longer than the validity period of the certificate, then the certificate has to be updated.

The expiry date has the advantage that there is no need to keep track of revoked certificates infinitely. After the certificate's expiration, the revocation status can be forgotten.

However a major draw-back is that a mechanism is needed to update expired certificates, provided that the entities holding them should continue to be used.

5.3. Time of revocation

Authentication and revocation are normally checked when two entities meet each other for the first time. But how about entities that are to be revoked later?

The dealings with this highly depends on the security requirements of the employed system. For example, home light-switches may have less stringent security requirements than actuators in a chemical plant. In the former, a revocation mechanism for deployed devices may not be needed, whereas in the latter it is essential.

6. Security considerations

This whole draft concerns security considerations. It indicates use cases and requirements for authentication, authorisation and

associated expiry and revocation. In addition it discusses several of the associated details and trade-offs.

We refer to the rest of the draft for the complete picture.

7. IANA considerations

No IANA requests are required for this document.

8. Acknowledgements

Thanks to Rene Struik and Kepeng Li for their valuable feedback.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2. Informative References

[X.509] , "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks. ", ITU-T Recommendation X.509, ISO/IEC 9594-8:2005, 2005.

Author's Address

Bert Greevenbosch
Huawei Technologies Co., Ltd.
Huawei Industrial Base
Bantian, Longgang District
Shenzhen 518129
P.R. China

Phone: +86-755-28978088
Email: bert.greevenbosch@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2014

C. Bormann
Universitaet Bremen TZI
Z. Shelby, Ed.
Sensinode
October 21, 2013

Blockwise transfers in CoAP
draft-ietf-core-block-14

Abstract

CoAP is a RESTful transfer protocol for constrained nodes and networks. Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. Occasionally, however, applications will need to transfer larger payloads -- for instance, for firmware updates. With HTTP, TCP does the grunt work of slicing large payloads up into multiple packets and ensuring that they all arrive and are handled in the right order.

CoAP is based on datagram transports such as UDP or DTLS, which limits the maximum size of resource representations that can be transferred without too much fragmentation. Although UDP supports larger payloads through IP fragmentation, it is limited to 64 KiB and, more importantly, doesn't really work well for constrained applications and networks.

Instead of relying on IP fragmentation, this specification extends basic CoAP with a pair of "Block" options, for transferring multiple blocks of information from a resource representation in multiple request-response pairs. In many important cases, the Block options enable a server to be truly stateless: the server can handle each block transfer separately, with no need for a connection setup or other server-side memory of previous block transfers.

In summary, the Block options provide a minimal way to transfer larger representations in a block-wise fashion.

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 <http://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 April 24, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Block-wise transfers	4
2.1. The Block2 and Block1 Options	5
2.2. Structure of a Block Option	6
2.3. Block Options in Requests and Responses	8
2.4. Using the Block2 Option	10
2.5. Using the Block1 Option	11
2.6. Combining Blockwise Transfers with the Observe Option . .	12
2.7. Combining Block1 and Block2	13
2.8. Combining Block2 with Multicast	13
2.9. Response Codes	14
2.9.1. 2.31 Continue	14
2.9.2. 4.08 Request Entity Incomplete	14
2.9.3. 4.13 Request Entity Too Large	14
3. Examples	14
3.1. Block2 Examples	15
3.2. Block1 Examples	18
3.3. Combining Block1 and Block2	20
3.4. Combining Observe and Block2	21
4. The Size2 and Size1 Options	24
5. HTTP Mapping Considerations	25
6. IANA Considerations	27
7. Security Considerations	28
7.1. Mitigating Resource Exhaustion Attacks	28

7.2. Mitigating Amplification Attacks	29
8. Acknowledgements	29
9. References	30
9.1. Normative References	30
9.2. Informative References	30
Authors' Addresses	30

1. Introduction

The CoRE WG is tasked with standardizing an Application Protocol for Constrained Networks/Nodes, CoAP. This protocol is intended to provide RESTful [REST] services not unlike HTTP [RFC2616], while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

This objective requires restraint in a number of sometimes conflicting ways:

- o reducing implementation complexity in order to minimize code size,
- o reducing message sizes in order to minimize the number of fragments needed for each message (in turn to maximize the probability of delivery of the message), the amount of transmission power needed and the loading of the limited-bandwidth channel,
- o reducing requirements on the environment such as stable storage, good sources of randomness or user interaction capabilities.

CoAP is based on datagram transports such as UDP, which limit the maximum size of resource representations that can be transferred without creating unreasonable levels of IP fragmentation. In addition, not all resource representations will fit into a single link layer packet of a constrained network, which may cause adaptation layer fragmentation even if IP layer fragmentation is not required. Using fragmentation (either at the adaptation layer or at the IP layer) to enable the transport of larger representations is possible up to the maximum size of the underlying datagram protocol (such as UDP), but the fragmentation/reassembly process burdens the lower layers with conversation state that is better managed in the application layer.

The present specification defines a pair of CoAP options to enable block-wise access to resource representations. The Block options provide a minimal way to transfer larger resource representations in a block-wise fashion. The overriding objective is to avoid the need for creating conversation state at the server for block-wise GET

requests. (It is impossible to fully avoid creating conversation state for POST/PUT, if the creation/replacement of resources is to be atomic; where that property is not needed, there is no need to create server conversation state in this case, either.)

In summary, this specification adds a pair of Block options to CoAP that can be used for block-wise transfers. Benefits of using these options include:

- o Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- o No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.
- o The transfer of each block is acknowledged, enabling retransmission if required.
- o Both sides have a say in the block size that actually will be used.
- o The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.
- o If needed, the Block options can also be used (without changes) to provide random access to power-of-two sized blocks within a resource representation.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119] and indicate requirement levels for compliant CoAP implementations.

In this document, the term "byte" is used in its now customary sense as a synonym for "octet".

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C, except that the operator "***" stands for exponentiation.

2. Block-wise transfers

As discussed in the introduction, there are good reasons to limit the size of datagrams in constrained networks:

- o by the maximum datagram size (~ 64 KiB for UDP)

- o by the desire to avoid IP fragmentation (MTU of 1280 for IPv6)
- o by the desire to avoid adaptation layer fragmentation (60-80 bytes for 6LoWPAN [RFC4919])

When a resource representation is larger than can be comfortably transferred in the payload of a single CoAP datagram, a Block option can be used to indicate a block-wise transfer. As payloads can be sent both with requests and with responses, this specification provides two separate options for each direction of payload transfer. In identifying these options, we use the number 1 to refer to the transfer of the resource representation that pertains to the request, and the number 2 to refer to the transfer of the resource representation for the response.

In the following, the term "payload" will be used for the actual content of a single CoAP message, i.e. a single block being transferred, while the term "body" will be used for the entire resource representation that is being transferred in a block-wise fashion. The Content-Format option applies to the body, not to the payload, in particular the boundaries between the blocks may be in places that are not separating whole units in terms of the structure, encoding, or content-coding used by the Content-Format.

In most cases, all blocks being transferred for a body will be of the same size. The block size is not fixed by the protocol. To keep the implementation as simple as possible, the Block options support only a small range of power-of-two block sizes, from 2^4 (16) to 2^{10} (1024) bytes. As bodies often will not evenly divide into the power-of-two block size chosen, the size need not be reached in the final block (but even for the final block, the chosen power-of-two size will still be indicated in the block size field of the Block option).

2.1. The Block2 and Block1 Options

Type	C	U	N	R	Name	Format	Length	Default
23	C	U	-	-	Block2	uint	0-3 B	(none)
27	C	U	-	-	Block1	uint	0-3 B	(none)

Table 1: Block Option Numbers

Both Block1 and Block2 options can be present both in request and response messages. In either case, the Block1 Option pertains to the request payload, and the Block2 Option pertains to the response payload.

Hence, for the methods defined in [I-D.ietf-core-coap], Block1 is useful with the payload-bearing POST and PUT requests and their responses. Block2 is useful with GET, POST, and PUT requests and their payload-bearing responses (2.01, 2.02, 2.04, 2.05 -- see section "Payload" of [I-D.ietf-core-coap]).

Where Block1 is present in a request or Block2 in a response (i.e., in that message to the payload of which it pertains) it indicates a block-wise transfer and describes how this block-wise payload forms part of the entire body being transferred ("descriptive usage"). Where it is present in the opposite direction, it provides additional control on how that payload will be formed or was processed ("control usage").

Implementation of either Block option is intended to be optional. However, when it is present in a CoAP message, it MUST be processed (or the message rejected); therefore it is identified as a critical option. It MUST NOT occur more than once.

2.2. Structure of a Block Option

Three items of information may need to be transferred in a Block (Block1 or Block2) option:

- o The size of the block (SZX);
- o whether more blocks are following (M);
- o the relative number of the block (NUM) within a sequence of blocks with the given size.

The value of the Block Option is a variable-size (0 to 3 byte) unsigned integer (uint, see Appendix A of [I-D.ietf-core-coap]). This integer value encodes these three fields, see Figure 1. (Due to the CoAP uint encoding rules, when all of NUM, M, and SZX happen to be zero, a zero-byte integer will be sent.)

```

0
0 1 2 3 4 5 6 7
+---+---+---+---+
|  NUM  |M| SZX |
+---+---+---+---+
```

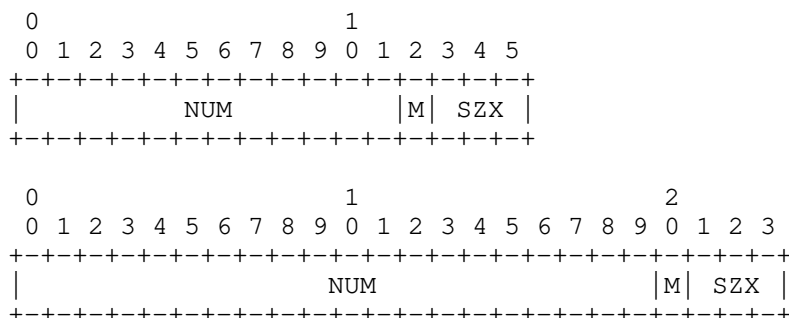


Figure 1: Block option value

The block size is encoded using a three-bit unsigned integer (0 for $2^{**}4$ to 6 for $2^{**}10$ bytes), which we call the "SZX" ("size exponent"); the actual block size is then $2^{**}(\text{SZX} + 4)$. SZX is transferred in the three least significant bits of the option value (i.e., $\text{val} \& 7$ where "val" is the value of the option).

The fourth least significant bit, the M or "more" bit ($\text{val} \& 8$), indicates whether more blocks are following or the current block-wise transfer is the last block being transferred.

The option value divided by sixteen (the NUM field) is the sequence number of the block currently being transferred, starting from zero. The current transfer is therefore about the "size" bytes starting at byte $\text{NUM} \ll (\text{SZX} + 4)$.

Implementation note: As an implementation convenience, $(\text{val} \& \sim 0xF) \ll (\text{val} \& 7)$, i.e., the option value with the last 4 bits masked out, shifted to the left by the value of SZX, gives the byte position of the block being transferred.

More specifically, within the option value of a Block1 or Block2 Option, the meaning of the option fields is defined as follows:

NUM: Block Number, indicating the block number being requested or provided. Block number 0 indicates the first block of a body (i.e., starting with the first byte of the body).

M: More Flag (not last block). For descriptive usage, this flag, if unset, indicates that the payload in this message is the last block in the body; when set it indicates that there are one or more additional blocks available. When a Block2 Option is used in a request to retrieve a specific block number ("control usage"), the M bit MUST be sent as zero and ignored on reception. (In a Block1 Option in a response, the M flag is used to indicate atomicity, see below.)

SZX: Block Size. The block size is represented as three-bit unsigned integer indicating the size of a block to the power of two. Thus block size = $2^{(SZX + 4)}$. The allowed values of SZX are 0 to 6, i.e., the minimum block size is $2^{(0+4)} = 16$ and the maximum is $2^{(6+4)} = 1024$. The value 7 for SZX (which would indicate a block size of 2048) is reserved, i.e. MUST NOT be sent and MUST lead to a 4.00 Bad Request response code upon reception in a request.

There is no default value for the Block1 and Block2 Options. Absence of one of these options is equivalent to an option value of 0 with respect to the value of NUM and M that could be given in the option, i.e. it indicates that the current block is the first and only block of the transfer (block number 0, M bit not set). However, in contrast to the explicit value 0, which would indicate an SZX of 0 and thus a size value of 16 bytes, there is no specific explicit size implied by the absence of the option -- the size is left unspecified. (As for any uint, the explicit value 0 is efficiently indicated by a zero-length option; this, therefore, is different in semantics from the absence of the option.)

2.3. Block Options in Requests and Responses

The Block options are used in one of three roles:

- o In descriptive usage, i.e., a Block2 Option in a response (such as a 2.05 response for GET), or a Block1 Option in a request (a PUT or POST):
 - * The NUM field in the option value describes what block number is contained in the payload of this message.
 - * The M bit indicates whether further blocks need to be transferred to complete the transfer of that body.
 - * The block size given by SZX MUST match the size of the payload in bytes, if the M bit is set. (SZX does not govern the payload size if M is unset). For Block2, if the request suggested a larger value of SZX, the next request MUST move SZX

down to the size given in the response. (The effect is that, if the server uses the smaller of (1) its preferred block size and (2) the block size requested, all blocks for a body use the same block size.)

- o A Block2 Option in control usage in a request (e.g., GET):
 - * The NUM field in the Block2 Option gives the block number of the payload that is being requested to be returned in the response.
 - * In this case, the M bit has no function and MUST be set to zero.
 - * The block size given (SZX) suggests a block size (in the case of block number 0) or repeats the block size of previous blocks received (in the case of a non-zero block number).
- o A Block1 Option in control usage in a response (e.g., a 2.xx response for a PUT or POST request):
 - * The NUM field of the Block1 Option indicates what block number is being acknowledged.
 - * If the M bit was set in the request, the server can choose whether to act on each block separately, with no memory, or whether to handle the request for the entire body atomically, or any mix of the two.
 - + If the M bit is also set in the response, it indicates that this response does not carry the final response code to the request, i.e. the server collects further blocks from the same endpoint and plans to implement the request atomically (e.g., acts only upon reception of the last block of payload). In this case, the response MUST NOT carry a Block2 option.
 - + Conversely, if the M bit is unset even though it was set in the request, it indicates the block-wise request was enacted now specifically for this block, and the response carries the final response to this request (and to any previous ones with the M bit set in the response's Block1 Option in this sequence of block-wise transfers); the client is still expected to continue sending further blocks, the request method for which may or may not also be enacted per-block.
 - * Finally, the SZX block size given in a control Block1 Option indicates the largest block size preferred by the server for

transfers toward the resource that is the same or smaller than the one used in the initial exchange; the client SHOULD use this block size or a smaller one in all further requests in the transfer sequence, even if that means changing the block size (and possibly scaling the block number accordingly) from now on.

Using one or both Block options, a single REST operation can be split into multiple CoAP message exchanges. As specified in [I-D.ietf-core-coap], each of these message exchanges uses their own CoAP Message ID.

2.4. Using the Block2 Option

When a request is answered with a response carrying a Block2 Option with the M bit set, the requester may retrieve additional blocks of the resource representation by sending further requests with the same options and a Block2 Option giving the block number and block size desired. In a request, the client MUST set the M bit of a Block2 Option to zero and the server MUST ignore it on reception.

To influence the block size used in a response, the requester also uses the Block2 Option, giving the desired size, a block number of zero and an M bit of zero. A server MUST use the block size indicated or a smaller size. Any further block-wise requests for blocks beyond the first one MUST indicate the same block size that was used by the server in the response for the first request that gave a desired size using a Block2 Option.

Once the Block2 Option is used by the requester and a first response has been received with a possibly adjusted block size, all further requests in a single block-wise transfer SHOULD ultimately use the same size, except that there may not be enough content to fill the last block (the one returned with the M bit not set). (Note that the client may start using the Block2 Option in a second request after a first request without a Block2 Option resulted in a Block2 option in the response.) The server SHOULD use the block size indicated in the request option or a smaller size, but the requester MUST take note of the actual block size used in the response it receives to its initial request and proceed to use it in subsequent requests. The server behavior MUST ensure that this client behavior results in the same block size for all responses in a sequence (except for the last one with the M bit not set, and possibly the first one if the initial request did not contain a Block2 Option).

Block-wise transfers can be used to GET resources the representations of which are entirely static (not changing over time at all, such as in a schema describing a device), or for dynamically changing

resources. In the latter case, the Block2 Option SHOULD be used in conjunction with the ETag Option, to ensure that the blocks being reassembled are from the same version of the representation: The server SHOULD include an ETag option in each response. If an ETag option is available, the client's reassembler, when reassembling the representation from the blocks being exchanged, MUST compare ETag Options. If the ETag Options do not match in a GET transfer, the requester has the option of attempting to retrieve fresh values for the blocks it retrieved first. To minimize the resulting inefficiency, the server MAY cache the current value of a representation for an ongoing sequence of requests. (The server may identify the sequence by the combination of the requesting end-point and the URI being the same in each block-wise request.) Note well that this specification makes no requirement for the server to establish any state; however, servers that offer quickly changing resources may thereby make it impossible for a client to ever retrieve a consistent set of blocks.

2.5. Using the Block1 Option

In a request with a request payload (e.g., PUT or POST), the Block1 Option refers to the payload in the request (descriptive usage).

In response to a request with a payload (e.g., a PUT or POST transfer), the block size given in the Block1 Option indicates the block size preference of the server for this resource (control usage). Obviously, at this point the first block has already been transferred by the client without benefit of this knowledge. Still, the client SHOULD heed the preference indicated and, for all further blocks, use the block size preferred by the server or a smaller one. Note that any reduction in the block size may mean that the second request starts with a block number larger than one, as the first request already transferred multiple blocks as counted in the smaller size.

To counter the effects of adaptation layer fragmentation on packet delivery probability, a client may want to give up retransmitting a request with a relatively large payload even before MAX_RETRANSMIT has been reached, and try restating the request as a block-wise transfer with a smaller payload. Note that this new attempt is then a new message-layer transaction and requires a new Message ID. (Because of the uncertainty whether the request or the acknowledgement was lost, this strategy is useful mostly for idempotent requests.)

In a blockwise transfer of a request payload (e.g., a PUT or POST) that is intended to be implemented in an atomic fashion at the server, the actual creation/replacement takes place at the time the

final block, i.e. a block with the M bit unset in the Block1 Option, is received. In this case, all success responses to non-final blocks carry the response code 2.31 (Continue, Section 2.9.1). If not all previous blocks are available at the server at the time of processing the final block, the transfer fails and error code 4.08 (Request Entity Incomplete, Section 2.9.2) MUST be returned. A server MAY also return a 4.08 error code for any (final or non-final) Block1 transfer that is not in sequence; clients that do not have specific mechanisms to handle this case therefore SHOULD always start with block zero and send the following blocks in order.

The error code 4.13 (Request Entity Too Large) can be returned at any time by a server that does not currently have the resources to store blocks for a block-wise request payload transfer that it would intend to implement in an atomic fashion. (Note that a 4.13 response to a request that does not employ Block1 is a hint for the client to try sending Block1, and a 4.13 response with a smaller SZX in its Block1 option than requested is a hint to try a smaller SZX.)

The Block1 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise request payload transfer (e.g., PUT or POST) operations to the same resource. Starting a new block-wise sequence of requests to the same resource (before an old sequence from the same endpoint was finished) simply overwrites the context the server may still be keeping. (This is probably exactly what one wants in this case - the client may simply have restarted and lost its knowledge of the previous sequence.)

2.6. Combining Blockwise Transfers with the Observe Option

The Observe Option provides a way for a client to be notified about changes over time of a resource [I-D.ietf-core-observe]. Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. The following rules apply to the combination of blockwise transfers with notifications.

Observation relationships always apply to an entire resource; the Block2 option does not provide a way to observe a single block of a resource.

As with basic GET transfers, the client can indicate its desired block size in a Block2 Option in the GET request establishing or renewing the observation relationship. If the server supports blockwise transfers, it SHOULD take note of the block size and apply it as a maximum size to all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the entry in that list is updated by the server receiving a new GET request for the resource from the client).

When sending a 2.05 (Content) notification, the server only sends the first block of the representation. The client retrieves the rest of the representation as if it had caused this first response by a GET request, i.e., by using additional GET requests with Block2 options containing NUM values greater than zero. (This results in the transfer of the entire representation, even if only some of the blocks have changed with respect to a previous notification.)

As with other dynamically changing resources, to ensure that the blocks being reassembled are from the same version of the representation, the server SHOULD include an ETag option in each response, and the reassembling client MUST compare the ETag options (Section 2.4).

See Section 3.4 for examples.

2.7. Combining Block1 and Block2

In PUT and particularly in POST exchanges, both the request body and the response body may be large enough to require the use of blockwise transfers. First, the Block1 transfer of the request body proceeds as usual. In the exchange of the last slice of this blockwise transfer, the response carries the first slice of the Block2 transfer (NUM is zero). To continue this Block2 transfer, the client continues to send requests similar to the requests in the Block1 phase, but leaves out the Block1 options and includes a Block2 request option with non-zero NUM.

Block2 transfers that retrieve the response body for a request that used Block1 MUST be performed in sequential order.

2.8. Combining Block2 with Multicast

A client can use the Block2 option in a multicast GET request with NUM = 0 to aid in limiting the size of the response.

Similarly, a response to a multicast GET request can use a Block2 option with NUM = 0 if the representation is large, or to further limit the size of the response.

In both cases, the client retrieves any further blocks using unicast exchanges; in the unicast requests, the client SHOULD heed any block size preferences indicated by the server in the response to the multicast request.

Other uses of the Block options in conjunction with multicast messages are for further study.

2.9. Response Codes

Two response codes are defined by this specification beyond those already defined in [I-D.ietf-core-coap], and another response code is extended in its meaning.

TODO: Add or appropriate useful response codes for Block2 problems (overshoot beyond end of representation, unsupported out-of-order access).

2.9.1. 2.31 Continue

This new success status code indicates that the transfer of this block of the request body was successful and that the server encourages sending further blocks, but that a final outcome of the whole block-wise request cannot yet be determined. No payload is returned with this response code.

2.9.2. 4.08 Request Entity Incomplete

This new client error status code indicates that the server has not received the blocks of the request body that it needs to proceed. The client has not sent all blocks, not sent them in the order required by the server, or has sent them long enough ago that the server has already discarded them.

2.9.3. 4.13 Request Entity Too Large

In [I-D.ietf-core-coap], section 5.9.2.9, the response code 4.13 (Request Entity Too Large) is defined to be like HTTP 413 "Request Entity Too Large". [I-D.ietf-core-coap] also recommends that this response SHOULD include a Size1 Option Section 4 to indicate the maximum size of request entity the server is able and willing to handle, unless the server is not in a position to make this information available.

The present specification allows the server to return this response code at any time during a Block1 transfer to indicate that it does not currently have the resources to store blocks for a transfer that it would intend to implement in an atomic fashion. It also allows the server to return a 4.13 response to a request that does not employ Block1 as a hint for the client to try sending Block1. Finally, a 4.13 response to a request with a Block1 option (control usage, see Section 2.3) where the response carries a smaller SZX in its Block1 option is a hint to try that smaller SZX.

3. Examples

This section gives a number of short examples with message flows for a block-wise GET, and for a PUT or POST. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and examples for the operation of the block size negotiation.

In all these examples, a Block option is shown in a decomposed way indicating the kind of Block option (1 or 2) followed by a colon, and then the block number (NUM), more bit (M), and block size exponent ($2^{*(SZX+4)}$) separated by slashes. E.g., a Block2 Option value of 33 would be shown as 2:2/0/32), or a Block1 Option value of 59 would be shown as 1:3/1/128.

3.1. Block2 Examples

The first example (Figure 2) shows a GET request that is split into three blocks. The server proposes a block size of 128, and the client agrees. The first two ACKs contain 128 bytes of payload each, and third ACK contains between 1 and 128 bytes.

CLIENT		SERVER
CON [MID=1234], GET, /status	----->	
<----- ACK [MID=1234], 2.05 Content, 2:0/1/128		
CON [MID=1235], GET, /status, 2:1/0/128	----->	
<----- ACK [MID=1235], 2.05 Content, 2:1/1/128		
CON [MID=1236], GET, /status, 2:2/0/128	----->	
<----- ACK [MID=1236], 2.05 Content, 2:2/0/128		

Figure 2: Simple blockwise GET

In the second example (Figure 3), the client anticipates the blockwise transfer (e.g., because of a size indication in the link-format description [RFC6690]) and sends a size proposal. All ACK messages except for the last carry 64 bytes of payload; the last one carries between 1 and 64 bytes.

CLIENT		SERVER
CON [MID=1234], GET, /status, 2:0/0/64	----->	
<----- ACK [MID=1234], 2.05 Content, 2:0/1/64		

```

| CON [MID=1235], GET, /status, 2:1/0/64          -----> |
| <----- ACK [MID=1235], 2.05 Content, 2:1/1/64    |
| :                                                    :
| :                  ...                               :
| :                                                    :
| CON [MID=1238], GET, /status, 2:4/0/64          -----> |
| <----- ACK [MID=1238], 2.05 Content, 2:4/1/64    |
|
| CON [MID=1239], GET, /status, 2:5/0/64          -----> |
| <----- ACK [MID=1239], 2.05 Content, 2:5/0/64    |

```

Figure 3: Blockwise GET with early negotiation

In the third example (Figure 4), the client is surprised by the need for a blockwise transfer, and unhappy with the size chosen unilaterally by the server. As it did not send a size proposal initially, the negotiation only influences the size from the second message exchange onward. Since the client already obtained both the first and second 64-byte block in the first 128-byte exchange, it goes on requesting the third 64-byte block ("2/0/64"). None of this is (or needs to be) understood by the server, which simply responds to the requests as it best can.

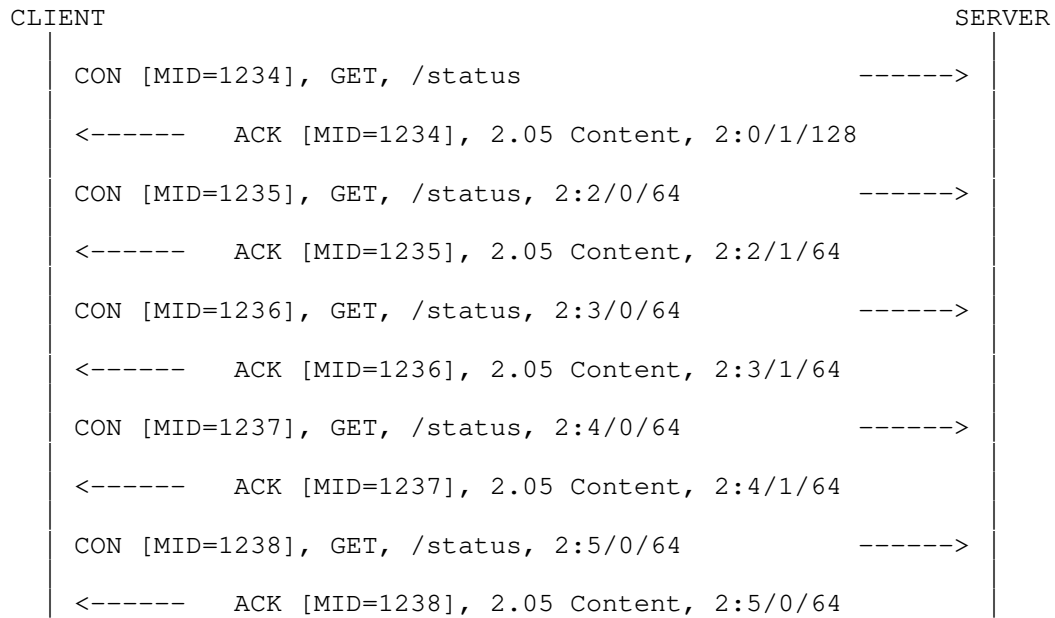


Figure 4: Blockwise GET with late negotiation

In all these (and the following) cases, retransmissions are handled by the CoAP message exchange layer, so they don't influence the block operations (Figure 5, Figure 6).

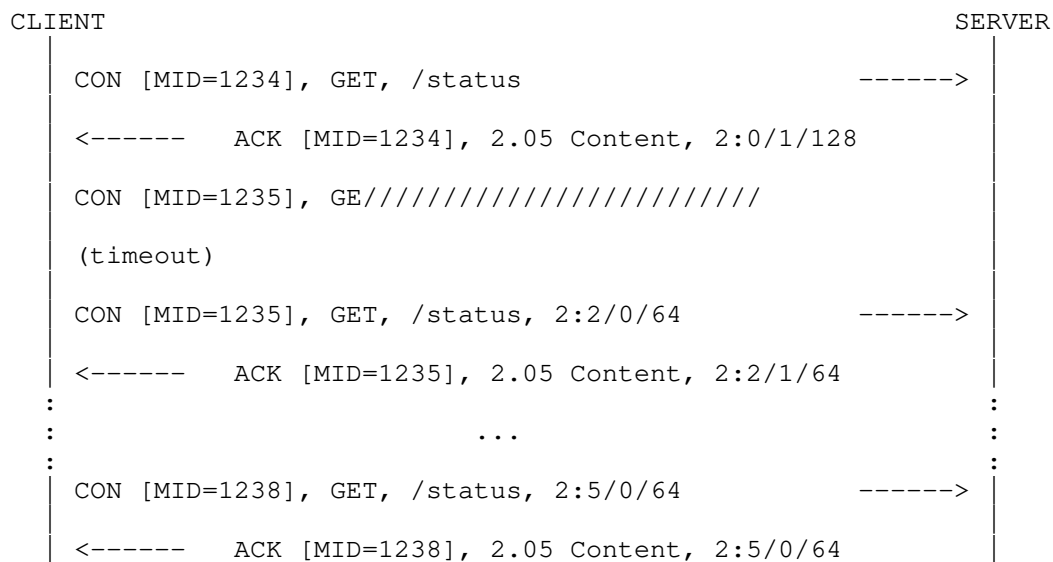


Figure 5: Blockwise GET with late negotiation and lost CON

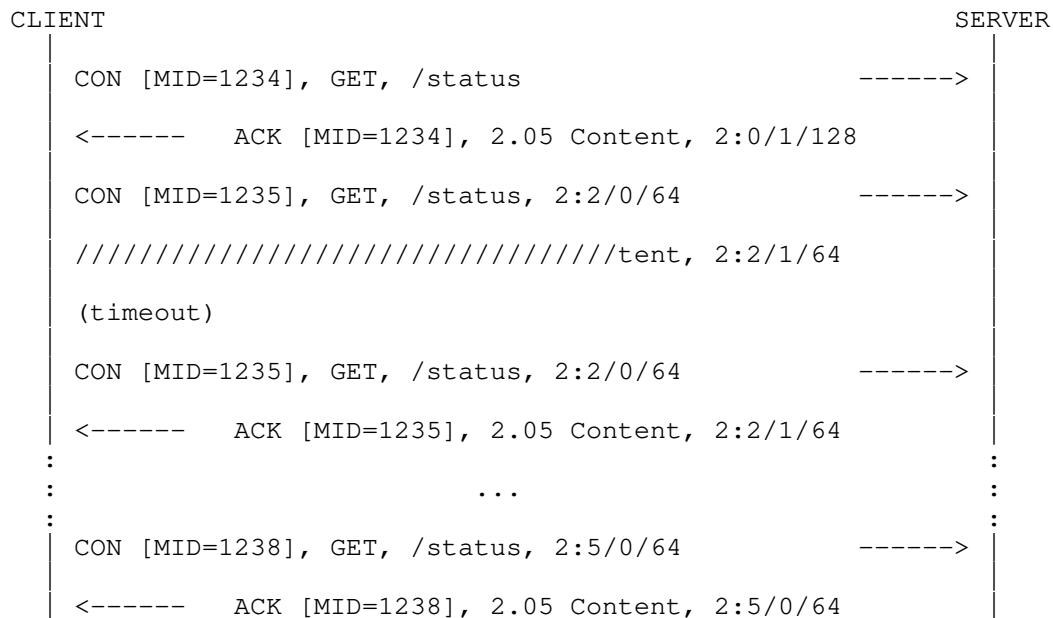


Figure 6: Blockwise GET with late negotiation and lost ACK

3.2. Block1 Examples

The following examples demonstrate a PUT exchange; a POST exchange looks the same, with different requirements on atomicity/idempotence. Note that, similar to GET, the responses to the requests that have a more bit in the request Block1 Option are provisional and carry the response code 2.31 (Continue); only the final response tells the client that the PUT did succeed.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1:0/1/128	----->
<----- ACK [MID=1234], 2.31 Continue, 1:0/1/128	
CON [MID=1235], PUT, /options, 1:1/1/128	----->
<----- ACK [MID=1235], 2.31 Continue, 1:1/1/128	
CON [MID=1236], PUT, /options, 1:2/0/128	----->
<----- ACK [MID=1236], 2.04 Changed, 1:2/0/128	

Figure 7: Simple atomic blockwise PUT

A stateless server that simply builds/updates the resource in place (statelessly) may indicate this by not setting the more bit in the response (Figure 8); in this case, the response codes are valid separately for each block being updated. This is of course only an acceptable behavior of the server if the potential inconsistency present during the run of the message exchange sequence does not lead to problems, e.g. because the resource being created or changed is not yet or not currently in use.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1:0/1/128	----->
<----- ACK [MID=1234], 2.04 Changed, 1:0/0/128	
CON [MID=1235], PUT, /options, 1:1/1/128	----->
<----- ACK [MID=1235], 2.04 Changed, 1:1/0/128	
CON [MID=1236], PUT, /options, 1:2/0/128	----->
<----- ACK [MID=1236], 2.04 Changed, 1:2/0/128	

Figure 8: Simple stateless blockwise PUT

Finally, a server receiving a blockwise PUT or POST may want to indicate a smaller block size preference (Figure 9). In this case, the client SHOULD continue with a smaller block size; if it does, it MUST adjust the block number to properly count in that smaller size.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1:0/1/128	----->
<-----	ACK [MID=1234], 2.04 Changed, 1:0/1/32
CON [MID=1235], PUT, /options, 1:4/1/32	----->
<-----	ACK [MID=1235], 2.04 Changed, 1:4/1/32
CON [MID=1236], PUT, /options, 1:5/1/32	----->
<-----	ACK [MID=1235], 2.04 Changed, 1:5/1/32
CON [MID=1237], PUT, /options, 1:6/0/32	----->
<-----	ACK [MID=1236], 2.04 Changed, 1:6/0/32

Figure 9: Simple atomic blockwise PUT with negotiation

3.3. Combining Block1 and Block2

Block options may be used in both directions of a single exchange. The following example demonstrates a blockwise POST request, resulting in a separate blockwise response.

CLIENT	SERVER
CON [MID=1234], POST, /soap, 1:0/1/128	----->
<-----	ACK [MID=1234], 2.31 Continue, 1:0/1/128
CON [MID=1235], POST, /soap, 1:1/1/128	----->
<-----	ACK [MID=1235], 2.31 Continue, 1:1/1/128
CON [MID=1236], POST, /soap, 1:2/0/128	----->
<-----	ACK [MID=1236], 2.04 Changed, 2:0/1/128, 1:2/0/128
CON [MID=1237], POST, /soap, 2:1/0/128	----->
(no payload for requests with Block2 with NUM != 0)	
(could also do late negotiation by requesting e.g. 2:2/0/64)	

<pre> <----- ACK [MID=1237], 2.04 Changed, 2:1/1/128 CON [MID=1238], POST, /soap, 2:2/0/128 -----> </pre>	<pre> </pre>
<pre> <----- ACK [MID=1238], 2.04 Changed, 2:2/1/128 CON [MID=1239], POST, /soap, 2:3/0/128 -----> </pre>	<pre> </pre>
<pre> <----- ACK [MID=1239], 2.04 Changed, 2:3/0/128 </pre>	<pre> </pre>

Figure 10: Atomic blockwise POST with blockwise response

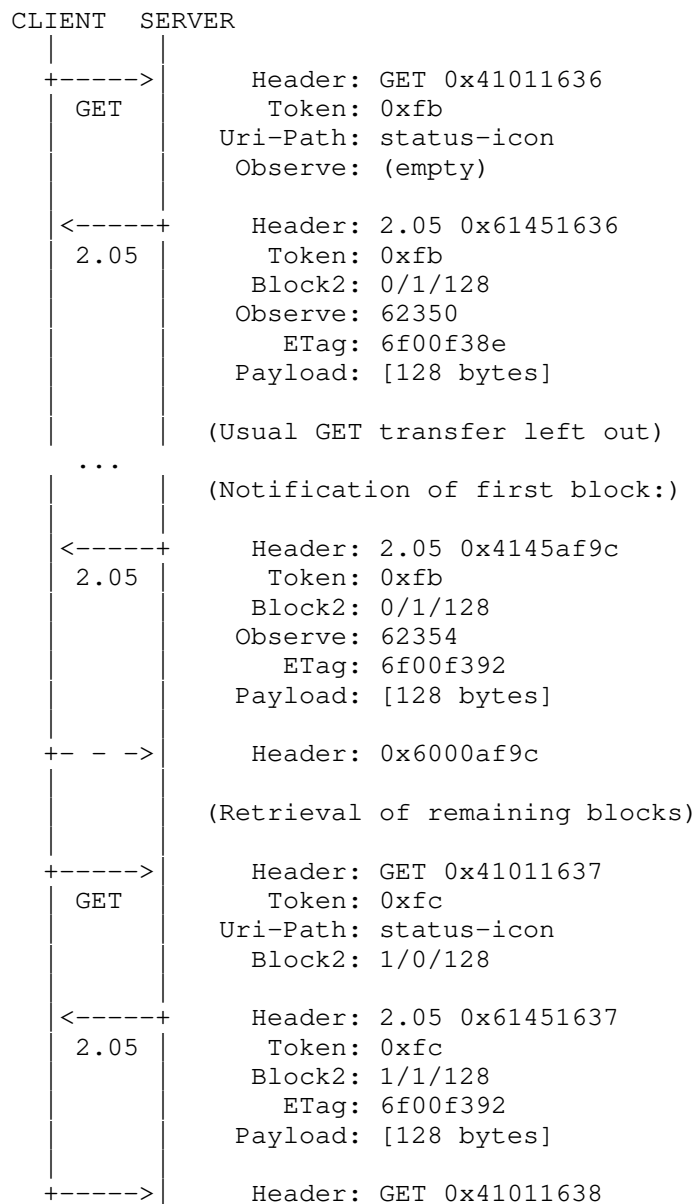
This model does provide for early negotiation input to the Block2 blockwise transfer, as shown below.

CLIENT	SERVER
<pre> CON [MID=1234], POST, /soap, 1:0/1/128 -----> </pre>	<pre> </pre>
<pre> <----- ACK [MID=1234], 2.31 Continue, 1:0/1/128 </pre>	<pre> </pre>
<pre> CON [MID=1235], POST, /soap, 1:1/1/128 -----> </pre>	<pre> </pre>
<pre> <----- ACK [MID=1235], 2.31 Continue, 1:1/1/128 </pre>	<pre> </pre>
<pre> CON [MID=1236], POST, /soap, 1:2/0/128, 2:0/0/64 -----> </pre>	<pre> </pre>
<pre> <----- ACK [MID=1236], 2.04 Changed, 1:2/0/128, 2:0/1/64 </pre>	<pre> </pre>
<pre> CON [MID=1237], POST, /soap, 2:1/0/64 -----> (no payload for requests with Block2 with NUM != 0) </pre>	<pre> </pre>
<pre> <----- ACK [MID=1237], 2.04 Changed, 2:1/1/64 </pre>	<pre> </pre>
<pre> CON [MID=1238], POST, /soap, 2:2/0/64 -----> </pre>	<pre> </pre>
<pre> <----- ACK [MID=1238], 2.04 Changed, 2:2/1/64 </pre>	<pre> </pre>
<pre> CON [MID=1239], POST, /soap, 2:3/0/64 -----> </pre>	<pre> </pre>
<pre> <----- ACK [MID=1239], 2.04 Changed, 2:3/0/64 </pre>	<pre> </pre>

Figure 11: Atomic blockwise POST with blockwise response, early negotiation

3.4. Combining Observe and Block2

In the following example, the server first sends a direct response (Observe sequence number 62350) to the initial GET request (the resulting blockwise transfer is as in Figure 4 and has therefore been left out). The second transfer is started by a 2.05 notification that contains just the first block (Observe sequence number 62354); the client then goes on to obtain the rest of the blocks.



GET	Token: 0xfc Uri-Path: status-icon Block2: 2/0/128
<-----+ 2.05	Header: 2.05 0x61451638 Token: 0xfc Block2: 2/0/128 ETag: 6f00f392 Payload: [53 bytes]

Figure 12: Observe sequence with blockwise response

In the following example, the client also uses early negotiation to limit the block size to 64 bytes.

CLIENT	SERVER
+-----> GET	Header: GET 0x41011636 Token: 0xfb Uri-Path: status-icon Observe: (empty) Block2: 0/0/64
<-----+ 2.05	Header: 2.05 0x61451636 Token: 0xfb Block2: 0/1/64 Observe: 62350 ETag: 6f00f38e Max-Age: 60 Payload: [64 bytes]
	(Usual GET transfer left out)
...	(Notification of first block:)
<-----+ 2.05	Header: 2.05 0x4145af9c Token: 0xfb Block2: 0/1/64 Observe: 62354 ETag: 6f00f392 Payload: [64 bytes]
+ - ->	Header: 0x6000af9c (Retrieval of remaining blocks)
+----->	Header: GET 0x41011637

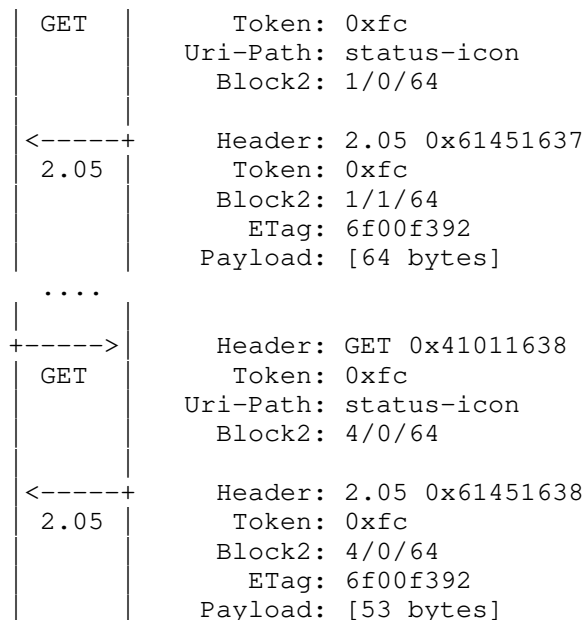


Figure 13: Observe sequence with early negotiation

4. The Size2 and Size1 Options

In many cases when transferring a large resource representation block by block, it is advantageous to know the total size early in the process. Some indication may be available from the maximum size estimate attribute "sz" provided in a resource description [RFC6690]. However, the size may vary dynamically, so a more up-to-date indication may be useful.

This specification defines two CoAP Options, Size1 for indicating the size of the representation transferred in requests, and Size2 for indicating the size of the representation transferred in responses.

The Size2 Option may be used for two purposes:

- o in a request, to ask the server to provide a size estimate along with the usual response ("size request"). For this usage, the value MUST be set to 0.
- o in a response carrying a Block2 Option, to indicate the current estimate the server has of the total size of the resource representation, measured in bytes ("size indication").

Similarly, the Size1 Option may be used for two purposes:

- o in a request carrying a Block1 Option, to indicate the current estimate the client has of the total size of the resource representation, measured in bytes ("size indication").
- o in a 4.13 response, to indicate the maximum size that would have been acceptable [I-D.ietf-core-coap], measured in bytes.

Apart from conveying/asking for size information, the Size options have no other effect on the processing of the request or response. If the client wants to minimize the size of the payload in the resulting response, it should add a Block2 option to the request with a small block size (e.g., setting SZX=0).

The Size Options are "elective", i.e., a client **MUST** be prepared for the server to ignore the size estimate request. The Size Options **MUST NOT** occur more than once.

Type	C	U	N	R	Name	Format	Length	Default
60			x		Size1	uint	0-4 B	(none)
28			x		Size2	uint	0-4 B	(none)

Table 2: Size Option Numbers

Implementation Notes:

- o As a quality of implementation consideration, blockwise transfers for which the total size considerably exceeds the size of one block are expected to include size indications, whenever those can be provided without undue effort (preferably with the first block exchanged). If the size estimate does not change, the indication does not need to be repeated for every block.
- o The end of a blockwise transfer is governed by the M bits in the Block Options, not by exhausting the size estimates exchanged.
- o As usual for an option of type uint, the value 0 is best expressed as an empty option (0 bytes). There is no default value.
- o The Size Options are neither critical nor unsafe, and are marked as No-Cache-Key.

5. HTTP Mapping Considerations

In this subsection, we give some brief examples for the influence the Block options might have on intermediaries that map between CoAP and HTTP.

For mapping CoAP requests to HTTP, the intermediary may want to map the sequence of block-wise transfers into a single HTTP transfer. E.g., for a GET request, the intermediary could perform the HTTP request once the first block has been requested and could then fulfill all further block requests out of its cache. A constrained implementation may not be able to cache the entire object and may use a combination of TCP flow control and (in particular if timeouts occur) HTTP range requests to obtain the information necessary for the next block transfer at the right time.

For PUT or POST requests, there is more variation in how HTTP servers might implement ranges. Some WebDAV servers do, but in general the CoAP-to-HTTP intermediary will have to try sending the payload of all the blocks of a block-wise transfer within one HTTP request. If enough buffering is available, this request can be started when the last CoAP block is received. A constrained implementation may want to relieve its buffering by already starting to send the HTTP request at the time the first CoAP block is received; any HTTP 408 status code that indicates that the HTTP server became impatient with the resulting transfer can then be mapped into a CoAP 4.08 response code (similarly, 413 maps to 4.13).

For mapping HTTP to CoAP, the intermediary may want to map a single HTTP transfer into a sequence of block-wise transfers. If the HTTP client is too slow delivering a request body on a PUT or POST, the CoAP server might time out and return a 4.08 response code, which in turn maps well to an HTTP 408 status code (again, 4.13 maps to 413). HTTP range requests received on the HTTP side may be served out of a cache and/or mapped to GET requests that request a sequence of blocks overlapping the range.

(Note that, while the semantics of CoAP 4.08 and HTTP 408 differ, this difference is largely due to the different way the two protocols are mapped to transport. HTTP has an underlying TCP connection, which supplies connection state, so a HTTP 408 status code can immediately be used to indicate that a timeout occurred during transmitting a request through that active TCP connection. The CoAP 4.08 response code indicates one or more missing blocks, which may be due to timeouts or resource constraints; as there is no connection state, there is no way to deliver such a response immediately; instead, it is delivered on the next block transfer. Still, HTTP 408 is probably the best mapping back to HTTP, as the timeout is the most likely cause for a CoAP 4.08. Note that there is no way to distinguish a timeout from a missing block for a server without creating additional state, the need for which we want to avoid.)

6. IANA Considerations

This draft adds the following option numbers to the CoAP Option Numbers registry of [I-D.ietf-core-coap]:

Number	Name	Reference
23	Block2	[RFCXXXX]
27	Block1	[RFCXXXX]
28	Size2	[RFCXXXX]
60	Size1	[RFCXXXX]

Table 3: CoAP Option Numbers

This draft adds the following response code to the CoAP Response Codes registry of [I-D.ietf-core-coap]:

Code	Description	Reference
2.31	Continue	[RFCXXXX]
4.08	Request Entity Incomplete	[RFCXXXX]

Table 4: CoAP Response Codes

7. Security Considerations

Providing access to blocks within a resource may lead to surprising vulnerabilities. Where requests are not implemented atomically, an attacker may be able to exploit a race condition or confuse a server by inducing it to use a partially updated resource representation. Partial transfers may also make certain problematic data invisible to intrusion detection systems; it is RECOMMENDED that an intrusion detection system (IDS) that analyzes resource representations transferred by CoAP implement the Block options to gain access to entire resource representations. Still, approaches such as transferring even-numbered blocks on one path and odd-numbered blocks on another path, or even transferring blocks multiple times with different content and obtaining a different interpretation of temporal order at the IDS than at the server, may prevent an IDS from seeing the whole picture. These kinds of attacks are well understood from IP fragmentation and TCP segmentation; CoAP does not add fundamentally new considerations.

Where access to a resource is only granted to clients making use of a specific security association, all blocks of that resource MUST be subject to the same security checks; it MUST NOT be possible for unprotected exchanges to influence blocks of an otherwise protected resource. As a related consideration, where object security is employed, PUT/POST should be implemented in the atomic fashion, unless the object security operation is performed on each access and the creation of unusable resources can be tolerated.

A stateless server might be susceptible to an attack where the adversary sends a Block1 (e.g., PUT) block with a high block number: A naive implementation might exhaust its resources by creating a huge resource representation.

Misleading size indications may be used by an attacker to induce buffer overflows in poor implementations, for which the usual considerations apply.

7.1. Mitigating Resource Exhaustion Attacks

Certain blockwise requests may induce the server to create state, e.g. to create a snapshot for the blockwise GET of a fast-changing resource to enable consistent access to the same version of a resource for all blocks, or to create temporary resource representations that are collected until pressed into service by a final PUT or POST with the more bit unset. All mechanisms that induce a server to create state that cannot simply be cleaned up create opportunities for denial-of-service attacks. Servers SHOULD avoid being subject to resource exhaustion based on state created by

untrusted sources. But even if this is done, the mitigation may cause a denial-of-service to a legitimate request when it is drowned out by other state-creating requests. Wherever possible, servers should therefore minimize the opportunities to create state for untrusted sources, e.g. by using stateless approaches.

Performing segmentation at the application layer is almost always better in this respect than at the transport layer or lower (IP fragmentation, adaptation layer fragmentation), e.g. because there is application layer semantics that can be used for mitigation or because lower layers provide security associations that can prevent attacks. However, it is less common to apply timeouts and keepalive mechanisms at the application layer than at lower layers. Servers MAY want to clean up accumulated state by timing it out (cf. response code 4.08), and clients SHOULD be prepared to run blockwise transfers in an expedient way to minimize the likelihood of running into such a timeout.

7.2. Mitigating Amplification Attacks

[I-D.ietf-core-coap] discusses the susceptibility of CoAP end-points for use in amplification attacks.

A CoAP server can reduce the amount of amplification it provides to an attacker by offering large resource representations only in relatively small blocks. With this, e.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

8. Acknowledgements

Much of the content of this draft is the result of discussions with the [I-D.ietf-core-coap] authors, and via many CoRE WG discussions.

Charles Palmer provided extensive editorial comments to a previous version of this draft, some of which the authors hope to have covered in this version. Esko Dijk reviewed a more recent version, leading to a number of further editorial improvements, a solution to the 4.13 ambiguity problem, and the section about combining Block and multicast. Markus Becker proposed getting rid of an ill-conceived default value for the Block2 and Block1 options. Peter Bigot insisted on a more systematic coverage of the options and response code.

Kepeng Li, Linyi Tian, and Barry Leiba wrote up an early version of the Size Option, which has informed this draft. Klaus Hartke wrote some of the text describing the interaction of Block2 with Observe.

Matthias Kovatsch provided a number of significant simplifications of the protocol.

9. References

9.1. Normative References

- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-core-observe] Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-11 (work in progress), October 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

9.2. Informative References

- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Zach Shelby (editor)
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2013

Z. Shelby
Sensinode
K. Hartke
C. Bormann
Universitaet Bremen TZI
June 28, 2013

Constrained Application Protocol (CoAP)
draft-ietf-core-coap-18

Abstract

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as 6LoWPAN often have high packet error rates and a typical throughput of 10s of kbit/s. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

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 <http://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 December 30, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	5
1.1. Features	5
1.2. Terminology	6
2. Constrained Application Protocol	9
2.1. Messaging Model	10
2.2. Request/Response Model	12
2.3. Intermediaries and Caching	14
2.4. Resource Discovery	15
3. Message Format	15
3.1. Option Format	17
3.2. Option Value Formats	19
4. Message Transmission	20
4.1. Messages and Endpoints	20
4.2. Messages Transmitted Reliably	20
4.3. Messages Transmitted Without Reliability	22
4.4. Message Correlation	23
4.5. Message Deduplication	24
4.6. Message Size	24
4.7. Congestion Control	25
4.8. Transmission Parameters	26
4.8.1. Changing The Parameters	27
4.8.2. Time Values derived from Transmission Parameters	28
5. Request/Response Semantics	30
5.1. Requests	30
5.2. Responses	30
5.2.1. Piggy-backed	32
5.2.2. Separate	32
5.2.3. Non-confirmable	33
5.3. Request/Response Matching	33
5.3.1. Token	34
5.3.2. Request/Response Matching Rules	35

5.4. Options	35
5.4.1. Critical/Elective	36
5.4.2. Proxy Unsafe/Safe-to-Forward and NoCacheKey	37
5.4.3. Length	38
5.4.4. Default Values	38
5.4.5. Repeatable Options	38
5.4.6. Option Numbers	38
5.5. Payloads and Representations	39
5.5.1. Representation	39
5.5.2. Diagnostic Payload	40
5.5.3. Selected Representation	40
5.5.4. Content Negotiation	40
5.6. Caching	41
5.6.1. Freshness Model	42
5.6.2. Validation Model	42
5.7. Proxying	43
5.7.1. Proxy Operation	43
5.7.2. Forward-Proxies	45
5.7.3. Reverse-Proxies	45
5.8. Method Definitions	46
5.8.1. GET	46
5.8.2. POST	46
5.8.3. PUT	46
5.8.4. DELETE	47
5.9. Response Code Definitions	47
5.9.1. Success 2.xx	47
5.9.2. Client Error 4.xx	49
5.9.3. Server Error 5.xx	50
5.10. Option Definitions	51
5.10.1. Uri-Host, Uri-Port, Uri-Path and Uri-Query	52
5.10.2. Proxy-Uri and Proxy-Scheme	53
5.10.3. Content-Format	53
5.10.4. Accept	54
5.10.5. Max-Age	54
5.10.6. ETag	54
5.10.7. Location-Path and Location-Query	55
5.10.8. Conditional Request Options	56
5.10.9. Size1 Option	57
6. CoAP URIs	57
6.1. coap URI Scheme	58
6.2. coaps URI Scheme	59
6.3. Normalization and Comparison Rules	59
6.4. Decomposing URIs into Options	60
6.5. Composing URIs from Options	61
7. Discovery	62
7.1. Service Discovery	62
7.2. Resource Discovery	63
7.2.1. 'ct' Attribute	63

8. Multicast CoAP	64
8.1. Messaging Layer	64
8.2. Request/Response Layer	65
8.2.1. Caching	66
8.2.2. Proxying	66
9. Securing CoAP	66
9.1. DTLS-secured CoAP	68
9.1.1. Messaging Layer	69
9.1.2. Request/Response Layer	69
9.1.3. Endpoint Identity	70
10. Cross-Protocol Proxying between CoAP and HTTP	73
10.1. CoAP-HTTP Proxying	74
10.1.1. GET	74
10.1.2. PUT	75
10.1.3. DELETE	75
10.1.4. POST	75
10.2. HTTP-CoAP Proxying	76
10.2.1. OPTIONS and TRACE	76
10.2.2. GET	76
10.2.3. HEAD	77
10.2.4. POST	77
10.2.5. PUT	78
10.2.6. DELETE	78
10.2.7. CONNECT	78
11. Security Considerations	78
11.1. Protocol Parsing, Processing URIs	78
11.2. Proxying and Caching	79
11.3. Risk of amplification	80
11.4. IP Address Spoofing Attacks	81
11.5. Cross-Protocol Attacks	82
11.6. Constrained node considerations	84
12. IANA Considerations	84
12.1. CoAP Code Registries	84
12.1.1. Method Codes	85
12.1.2. Response Codes	85
12.2. Option Number Registry	87
12.3. Content-Format Registry	89
12.4. URI Scheme Registration	90
12.5. Secure URI Scheme Registration	91
12.6. Service Name and Port Number Registration	92
12.7. Secure Service Name and Port Number Registration	93
12.8. Multicast Address Registration	94
13. Acknowledgements	94
14. References	95
14.1. Normative References	95
14.2. Informative References	97
Appendix A. Examples	100
Appendix B. URI Examples	105

Appendix C. Changelog	107
Authors' Addresses	117

1. Introduction

The use of web services (web APIs) on the Internet has become ubiquitous in most applications, and depends on the fundamental Representational State Transfer [REST] architecture of the web.

The Constrained RESTful Environments (CoRE) work aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN, [RFC4944]). Constrained networks such as 6LoWPAN support the fragmentation of IPv6 packets into small link-layer frames, however incurring significant reduction in packet delivery probability. One design goal of CoAP has been to keep message overhead small, thus limiting the need for fragmentation.

One of the main goals of CoAP is to design a generic web protocol for the special requirements of this constrained environment, especially considering energy, building automation and other machine-to-machine (M2M) applications. The goal of CoAP is not to blindly compress HTTP [RFC2616], but rather to realize a subset of REST common with HTTP but optimized for M2M applications. Although CoAP could be used for refashioning simple HTTP interfaces into a more compact protocol, it more importantly also offers features for M2M such as built-in discovery, multicast support and asynchronous message exchanges.

This document specifies the Constrained Application Protocol (CoAP), which easily translates to HTTP for integration with the existing web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments and M2M applications.

1.1. Features

CoAP has the following main features:

- o Constrained web protocol fulfilling M2M requirements.
- o UDP [RFC0768] binding with optional reliability supporting unicast and multicast requests.
- o Asynchronous message exchanges.
- o Low header overhead and parsing complexity.
- o URI and Content-type support.

- o Simple proxy and caching capabilities.
- o A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.
- o Security binding to Datagram Transport Layer Security (DTLS) [RFC6347].

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 [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC2616], including "resource", "representation", "cache", and "fresh". In addition, this specification defines the following terminology:

Endpoint

An entity participating in the CoAP protocol. Colloquially, an endpoint lives on a "Node", although "Host" would be more consistent with Internet standards usage, and is further identified by transport layer multiplexing information that can include a UDP port number and a security association (Section 4.1).

Sender

The originating endpoint of a message. When the aspect of identification of the specific sender is in focus, also "source endpoint".

Recipient

The destination endpoint of a message. When the aspect of identification of the specific recipient is in focus, also "destination endpoint".

Client

The originating endpoint of a request; the destination endpoint of a response.

Server

The destination endpoint of a request; the originating endpoint of a response.

Origin Server

The server on which a given resource resides or is to be created.

Intermediary

A CoAP endpoint that acts both as a server and as a client towards (possibly via further intermediaries) an origin server. A common form of an intermediary is a proxy; several classes of such proxies are discussed in this specification.

Proxy

An intermediary that mainly is concerned with forwarding requests and relaying back responses, possibly performing caching, namespace translation, or protocol translation in the process. As opposed to intermediaries in the general sense, proxies generally do not implement specific application semantics. Based on the position in the overall structure of the request forwarding, there are two common forms of proxy: forward-proxy and reverse-proxy. In some cases, a single endpoint might act as an origin server, forward-proxy, or reverse-proxy, switching behavior based on the nature of each request.

Forward-Proxy

A "forward-proxy" is an endpoint selected by a client, usually via local configuration rules, to perform requests on behalf of the client, doing any necessary translations. Some translations are minimal, such as for proxy requests for "coap" URIs, whereas other requests might require translation to and from entirely different application-layer protocols.

Reverse-Proxy

A "reverse-proxy" is an endpoint that stands in for one or more other server(s) and satisfies requests on behalf of these, doing any necessary translations. Unlike a forward-proxy, the client may not be aware that it is communicating with a reverse-proxy; a reverse-proxy receives requests as if it was the origin server for the target resource.

CoAP-to-CoAP Proxy

A proxy that maps from a CoAP request to a CoAP request, i.e. uses the CoAP protocol both on the server and the client side. Contrast to cross-proxy.

Cross-Proxy

A cross-protocol proxy, or "cross-proxy" for short, is a proxy that translates between different protocols, such as a CoAP-to-HTTP proxy or an HTTP-to-CoAP proxy. While this specification makes very specific demands of CoAP-to-CoAP proxies, there is more variation possible in cross-proxies.

Confirmable Message

Some messages require an acknowledgement. These messages are called "Confirmable". When no packets are lost, each Confirmable message elicits exactly one return message of type Acknowledgement or type Reset.

Non-confirmable Message

Some other messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor.

Acknowledgement Message

An Acknowledgement message acknowledges that a specific Confirmable message arrived. By itself, an Acknowledgement message does not indicate success or failure of any request encapsulated in the Confirmable message, but the Acknowledgement message may also carry a Piggy-Backed Response (q.v.).

Reset Message

A Reset message indicates that a specific message (Confirmable or Non-confirmable) was received, but some context is missing to properly process it. This condition is usually caused when the receiving node has rebooted and has forgotten some state that would be required to interpret the message. Provoking a Reset message (e.g., by sending an Empty Confirmable message) is also useful as an inexpensive check of the liveness of an endpoint ("CoAP ping").

Piggy-backed Response

A Piggy-backed Response is included right in a CoAP Acknowledgement (ACK) message that is sent to acknowledge receipt of the Request for this Response (Section 5.2.1).

Separate Response

When a Confirmable message carrying a Request is acknowledged with an Empty message (e.g., because the server doesn't have the answer right away), a Separate Response is sent in a separate message exchange (Section 5.2.2).

Empty Message

A message with a Code of 0.00; neither a request nor a response. An Empty message only contains the four-byte header.

Critical Option

An option that would need to be understood by the endpoint ultimately receiving the message in order to properly process the message (Section 5.4.1). Note that the implementation of critical options is, as the name "Option" implies, generally optional:

unsupported critical options lead to an error response or summary rejection of the message.

Elective Option

An option that is intended to be ignored by an endpoint that does not understand it. Processing the message even without understanding the option is acceptable (Section 5.4.1).

Unsafe Option

An option that would need to be understood by a proxy receiving the message in order to safely forward the message (Section 5.4.2). Not every critical option is an unsafe option.

Safe-to-Forward Option

An option that is intended to be safe for forwarding by a proxy that does not understand it. Forwarding the message even without understanding the option is acceptable (Section 5.4.2).

Resource Discovery

The process where a CoAP client queries a server for its list of hosted resources (i.e., links, Section 7).

Content-Format

The combination of an Internet media type, potentially with specific parameters given, and a content-coding (which is often the identity content-coding), identified by a numeric identifier defined by the CoAP Content-Format Registry. When the focus is less on the numeric identifier than on the combination of these characteristics of a resource representation, this is also called "representation format".

Additional terminology for constrained nodes and constrained node networks can be found in [I-D.ietf-lwig-terminology].

In this specification, the term "byte" is used in its now customary sense as a synonym for "octet".

All multi-byte integers in this protocol are interpreted in network byte order.

Where arithmetic is used, this specification uses the notation familiar from the programming language C, except that the operator "***" stands for exponentiation.

2. Constrained Application Protocol

The interaction model of CoAP is similar to the client/server model of HTTP. However, machine-to-machine interactions typically result

in a CoAP implementation acting in both client and server roles. A CoAP request is equivalent to that of HTTP, and is sent by a client to request an action (using a method code) on a resource (identified by a URI) on a server. The server then sends a response with a response code; this response may include a resource representation.

Unlike HTTP, CoAP deals with these interchanges asynchronously over a datagram-oriented transport such as UDP. This is done logically using a layer of messages that supports optional reliability (with exponential back-off). CoAP defines four types of messages: Confirmable, Non-confirmable, Acknowledgement, Reset; method codes and response codes included in some of these messages make them carry requests or responses. The basic exchanges of the four types of messages are somewhat orthogonal to the request/response interactions; requests can be carried in Confirmable and Non-confirmable messages, and responses can be carried in these as well as piggy-backed in Acknowledgement messages.

One could think of CoAP logically as using a two-layer approach, a CoAP messaging layer used to deal with UDP and the asynchronous nature of the interactions, and the request/response interactions using Method and Response codes (see Figure 1). CoAP is however a single protocol, with messaging and request/response just features of the CoAP header.

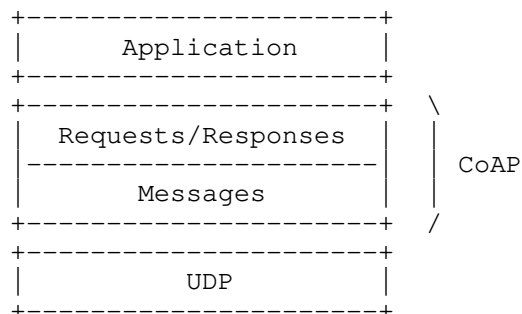


Figure 1: Abstract layering of CoAP

2.1. Messaging Model

The CoAP messaging model is based on the exchange of messages over UDP between endpoints.

CoAP uses a short fixed-length binary header (4 bytes) that may be followed by compact binary options and a payload. This message format is shared by requests and responses. The CoAP message format is specified in Section 3. Each message contains a Message ID used

to detect duplicates and for optional reliability. (The Message ID is compact; its 16-bit size enables up to about 250 messages per second from one endpoint to another with default protocol parameters.)

Reliability is provided by marking a message as Confirmable (CON). A Confirmable message is retransmitted using a default timeout and exponential back-off between retransmissions, until the recipient sends an Acknowledgement message (ACK) with the same Message ID (in this example, 0x7d34) from the corresponding endpoint; see Figure 2. When a recipient is not at all able to process a Confirmable message (i.e., not even able to provide a suitable error response), it replies with a Reset message (RST) instead of an Acknowledgement (ACK).

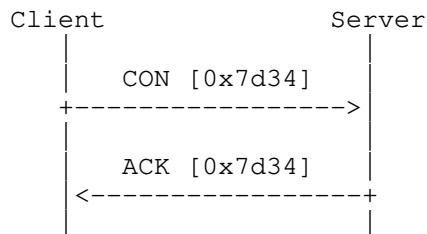


Figure 2: Reliable message transmission

A message that does not require reliable transmission, for example each single measurement out of a stream of sensor data, can be sent as a Non-confirmable message (NON). These are not acknowledged, but still have a Message ID for duplicate detection (in this example, 0x01a0); see Figure 3. When a recipient is not able to process a Non-confirmable message, it may reply with a Reset message (RST).

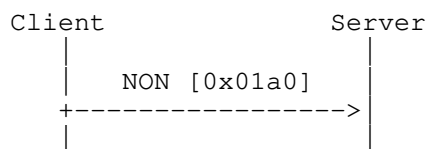


Figure 3: Unreliable message transmission

See Section 4 for details of CoAP messages.

As CoAP runs over UDP, it also supports the use of multicast IP destination addresses, enabling multicast CoAP requests. Section 8 discusses the proper use of CoAP messages with multicast addresses and precautions for avoiding response congestion.

Several security modes are defined for CoAP in Section 9 ranging from no security to certificate-based security. This document specifies a binding to DTLS for securing the protocol; the use of IPsec with CoAP is discussed in [I-D.bormann-core-ipsec-for-coap].

2.2. Request/Response Model

CoAP request and response semantics are carried in CoAP messages, which include either a Method code or Response code, respectively. Optional (or default) request and response information, such as the URI and payload media type are carried as CoAP options. A Token is used to match responses to requests independently from the underlying messages (Section 5.3). (Note that the Token is a concept separate from the Message ID.)

A request is carried in a Confirmable (CON) or Non-confirmable (NON) message, and if immediately available, the response to a request carried in a Confirmable message is carried in the resulting Acknowledgement (ACK) message. This is called a piggy-backed response, detailed in Section 5.2.1. (There is no need for separately acknowledging a piggy-backed response, as the client will retransmit the request if the Acknowledgement message carrying the piggy-backed response is lost.) Two examples for a basic GET request with piggy-backed response are shown in Figure 4, one successful, one resulting in a 4.04 (Not Found) response.

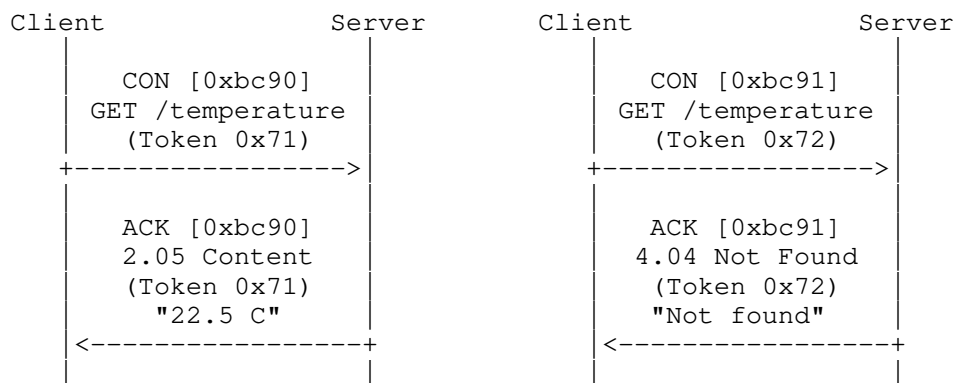


Figure 4: Two GET requests with piggy-backed responses

If the server is not able to respond immediately to a request carried in a Confirmable message, it simply responds with an Empty Acknowledgement message so that the client can stop retransmitting the request. When the response is ready, the server sends it in a new Confirmable message (which then in turn needs to be acknowledged by the client). This is called a separate response, as illustrated in Figure 5 and described in more detail in Section 5.2.2.

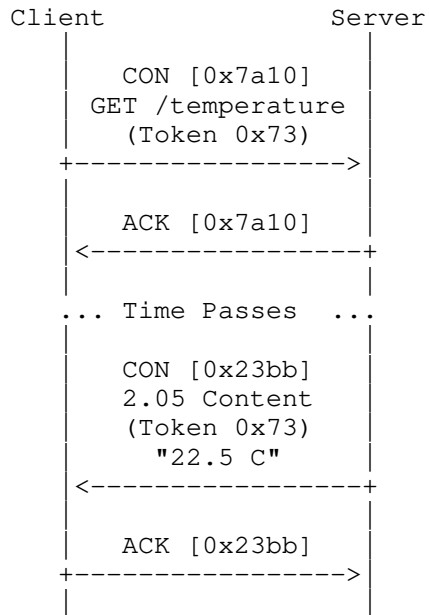
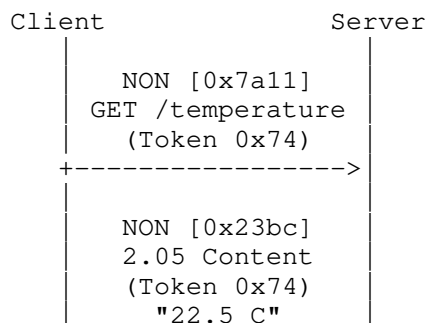


Figure 5: A GET request with a separate response

If a request is sent in a Non-confirmable message, then the response is sent using a new Non-confirmable message, although the server may instead send a Confirmable message. This type of exchange is illustrated in Figure 6.



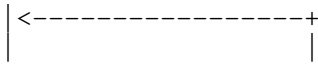


Figure 6: A NON request and response

CoAP makes use of GET, PUT, POST and DELETE methods in a similar manner to HTTP, with the semantics specified in Section 5.8. (Note that the detailed semantics of CoAP methods are "almost, but not entirely unlike" [HHGTTG] those of HTTP methods: Intuition taken from HTTP experience generally does apply well, but there are enough differences that make it worthwhile to actually read the present specification.)

Methods beyond the basic four can be added to CoAP in separate specifications. New methods do not necessarily have to use requests and responses in pairs. Even for existing methods, a single request may yield multiple responses, e.g. for a multicast request (Section 8) or with the Observe option [I-D.ietf-core-observe].

URI support in a server is simplified as the client already parses the URI and splits it into host, port, path and query components, making use of default values for efficiency. Response codes relate to a small subset of HTTP response codes with a few CoAP specific codes added, as defined in Section 5.9.

2.3. Intermediaries and Caching

The protocol supports the caching of responses in order to efficiently fulfill requests. Simple caching is enabled using freshness and validity information carried with CoAP responses. A cache could be located in an endpoint or an intermediary. Caching functionality is specified in Section 5.6.

Proxying is useful in constrained networks for several reasons, including network traffic limiting, to improve performance, to access resources of sleeping devices or for security reasons. The proxying of requests on behalf of another CoAP endpoint is supported in the protocol. When using a proxy, the URI of the resource to request is included in the request, while the destination IP address is set to the address of the proxy. See Section 5.7 for more information on proxy functionality.

As CoAP was designed according to the REST architecture [REST] and thus exhibits functionality similar to that of the HTTP protocol, it is quite straightforward to map from CoAP to HTTP and from HTTP to CoAP. Such a mapping may be used to realize an HTTP REST interface using CoAP, or for converting between HTTP and CoAP. This conversion can be carried out by a cross-protocol proxy ("cross-proxy"), which

converts the method or response code, media type, and options to the corresponding HTTP feature. Section 10 provides more detail about HTTP mapping.

2.4. Resource Discovery

Resource discovery is important for machine-to-machine interactions, and is supported using the CoRE Link Format [RFC6690] as discussed in Section 7.

3. Message Format

CoAP is based on the exchange of compact messages which, by default, are transported over UDP (i.e. each CoAP message occupies the data section of one UDP datagram). CoAP may also be used over Datagram Transport Layer Security (DTLS) (see Section 9.1). It could also be used over other transports such as SMS, TCP or SCTP, the specification of which is out of this document's scope. (UDP-lite [RFC3828] and UDP zero checksum [RFC6936] are not supported by CoAP.)

CoAP messages are encoded in a simple binary format. The message format starts with a fixed-size 4-byte header. This is followed by a variable-length Token value which can be between 0 and 8 bytes long. Following the Token value comes a sequence of zero or more CoAP Options in Type-Length-Value (TLV) format, optionally followed by a payload which takes up the rest of the datagram.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Ver| T |  TKL  |      Code      |      Message ID      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  Token (if any, TKL bytes) ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  Options (if any) ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|1 1 1 1 1 1 1 1|      Payload (if any) ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Figure 7: Message Format

The fields in the header are defined as follows:

Version (Ver): 2-bit unsigned integer. Indicates the CoAP version number. Implementations of this specification MUST set this field to 1 (01 binary). Other values are reserved for future versions. Messages with unknown version numbers MUST be silently ignored.

Type (T): 2-bit unsigned integer. Indicates if this message is of type Confirmable (0), Non-confirmable (1), Acknowledgement (2) or Reset (3). The semantics of these message types are defined in Section 4.

Token Length (TKL): 4-bit unsigned integer. Indicates the length of the variable-length Token field (0-8 bytes). Lengths 9-15 are reserved, MUST NOT be sent, and MUST be processed as a message format error.

Code: 8-bit unsigned integer, split into a 3-bit class (most significant bits) and a 5-bit detail (least significant bits), documented as c.dd where c is a digit from 0 to 7 for the 3-bit subfield and dd are two digits from 00 to 31 for the 5-bit subfield. The class can indicate a request (0), a success response (2), a client error response (4), or a server error response (5). (All other class values are reserved.) As a special case, Code 0.00 indicates an Empty message. In case of a request, the Code field indicates the Request Method; in case of a response a Response Code. Possible values are maintained in the CoAP Code Registries (Section 12.1). The semantics of requests and responses are defined in Section 5.

Message ID: 16-bit unsigned integer in network byte order. Used for the detection of message duplication, and to match messages of type Acknowledgement/Reset to messages of type Confirmable/Non-confirmable. The rules for generating a Message ID and matching messages are defined in Section 4.

The header is followed by the Token value, which may be 0 to 8 bytes, as given by the Token Length field. The Token value is used to correlate requests and responses. The rules for generating a Token and correlating requests and responses are defined in Section 5.3.1.

Header and Token are followed by zero or more Options (Section 3.1). An Option can be followed by the end of the message, by another Option, or by the Payload Marker and the payload.

Following the header, token, and options, if any, comes the optional payload. If present and of non-zero length, it is prefixed by a fixed, one-byte Payload Marker (0xFF) which indicates the end of options and the start of the payload. The payload data extends from after the marker to the end of the UDP datagram, i.e., the Payload Length is calculated from the datagram size. The absence of the Payload Marker denotes a zero-length payload. The presence of a marker followed by a zero-length payload MUST be processed as a message format error.

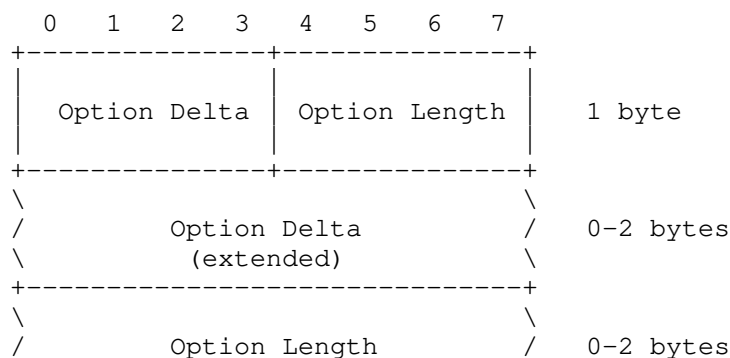
Implementation Note: The byte value 0xFF may also occur within an option length or value, so simple byte-wise scanning for 0xFF is not a viable technique for finding the payload marker. The byte 0xFF has the meaning of a payload marker only where the beginning of another option could occur.

3.1. Option Format

CoAP defines a number of options which can be included in a message. Each option instance in a message specifies the Option Number of the defined CoAP option, the length of the Option Value and the Option Value itself.

Instead of specifying the Option Number directly, the instances MUST appear in order of their Option Numbers and a delta encoding is used between them: The Option Number for each instance is calculated as the sum of its delta and the Option Number of the preceding instance in the message. For the first instance in a message, a preceding option instance with Option Number zero is assumed. Multiple instances of the same option can be included by using a delta of zero.

Option Numbers are maintained in the CoAP Option Number Registry (Section 12.2). See Section 5.4 for the semantics of the options defined in this document.



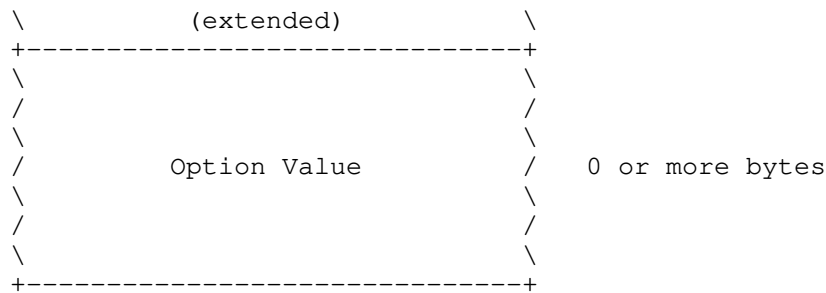


Figure 8: Option Format

The fields in an option are defined as follows:

Option Delta: 4-bit unsigned integer. A value between 0 and 12 indicates the Option Delta. Three values are reserved for special constructs:

- 13: An 8-bit unsigned integer follows the initial byte and indicates the Option Delta minus 13.
- 14: A 16-bit unsigned integer in network byte order follows the initial byte and indicates the Option Delta minus 269.
- 15: Reserved for the Payload Marker. If the field is set to this value but the entire byte is not the payload marker, this MUST be processed as a message format error.

The resulting Option Delta is used as the difference between the Option Number of this option and that of the previous option (or zero for the first option). In other words, the Option Number is calculated by simply summing the Option Delta values of this and all previous options before it.

Option Length: 4-bit unsigned integer. A value between 0 and 12 indicates the length of the Option Value, in bytes. Three values are reserved for special constructs:

- 13: An 8-bit unsigned integer precedes the Option Value and indicates the Option Length minus 13.
- 14: A 16-bit unsigned integer in network byte order precedes the Option Value and indicates the Option Length minus 269.
- 15: Reserved for future use. If the field is set to this value, it MUST be processed as a message format error.

Value: A sequence of exactly Option Length bytes. The length and format of the Option Value depend on the respective option, which MAY define variable length values. See Section 3.2 for the formats used in this document; options defined in other documents MAY make use of other option value formats.

3.2. Option Value Formats

The options defined in this document make use of the following option value formats.

empty: A zero-length sequence of bytes.

opaque: An opaque sequence of bytes.

uint: A non-negative integer which is represented in network byte order using the number of bytes given by the Option Length field.

An option definition may specify a range of permissible numbers of bytes; if it has a choice, a sender SHOULD represent the integer with as few bytes as possible, i.e., without leading zero bytes. For example, the number 0 is represented with an empty option value (a zero-length sequence of bytes), and the number 1 by a single byte with the numerical value of 1 (bit combination 00000001 in most significant bit first notation). A recipient MUST be prepared to process values with leading zero bytes.

Implementation Note: The exceptional behavior permitted for the sender is intended for highly constrained, templated implementations (e.g., hardware implementations) that use fixed size options in the templates.

string: A Unicode string which is encoded using UTF-8 [RFC3629] in Net-Unicode form [RFC5198].

Note that here and in all other places where UTF-8 encoding is used in the CoAP protocol, the intention is that the encoded strings can be directly used and compared as opaque byte strings by CoAP protocol implementations. There is no expectation and no need to perform normalization within a CoAP implementation (except where Unicode strings that are not known to be normalized are imported from sources outside the CoAP protocol). Note also that ASCII strings (that do not make use of special control characters) are always valid UTF-8 Net-Unicode strings.

4. Message Transmission

CoAP messages are exchanged asynchronously between CoAP endpoints. They are used to transport CoAP requests and responses, the semantics of which are defined in Section 5.

As CoAP is bound to non-reliable transports such as UDP, CoAP messages may arrive out of order, appear duplicated, or go missing without notice. For this reason, CoAP implements a lightweight reliability mechanism, without trying to re-create the full feature set of a transport like TCP. It has the following features:

- o Simple stop-and-wait retransmission reliability with exponential back-off for Confirmable messages.
- o Duplicate detection for both Confirmable and Non-confirmable messages.

4.1. Messages and Endpoints

A CoAP endpoint is the source or destination of a CoAP message. The specific definition of an endpoint depends on the transport being used for CoAP. For the transports defined in this specification, the endpoint is identified depending on the security mode used (see Section 9): With no security, the endpoint is solely identified by an IP address and a UDP port number. With other security modes, the endpoint is identified as defined by the security mode.

There are different types of messages. The type of a message is specified by the Type field of the CoAP Header.

Separate from the message type, a message may carry a request, a response, or be Empty. This is signaled by the Request/Response Code field in the CoAP Header and is relevant to the request/response model. Possible values for the field are maintained in the CoAP Code Registries (Section 12.1).

An Empty message has the Code field set to 0.00. The Token Length field MUST be set to 0 and bytes of data MUST NOT be present after the Message ID field. If there are any bytes, they MUST be processed as a message format error.

4.2. Messages Transmitted Reliably

The reliable transmission of a message is initiated by marking the message as Confirmable in the CoAP header. A Confirmable message always carries either a request or response, unless it is used only to elicit a Reset message in which case it is Empty. A recipient

MUST acknowledge a Confirmable message with an Acknowledgement message or, if it lacks context to process the message properly (including the case where the message is Empty, uses a code with a reserved class (1, 6 or 7), or has a message format error), MUST reject it; rejecting a Confirmable message is effected by sending a matching Reset message and otherwise ignoring it. The Acknowledgement message MUST echo the Message ID of the Confirmable message, and MUST carry a response or be Empty (see Section 5.2.1 and Section 5.2.2). The Reset message MUST echo the Message ID of the Confirmable message, and MUST be Empty. Rejecting an Acknowledgement or Reset message (including the case where the Acknowledgement carries a request or a code with a reserved class, or the Reset message is not Empty) is effected by silently ignoring it. More generally, recipients of Acknowledgement and Reset messages MUST NOT respond with either Acknowledgement or Reset messages.

The sender retransmits the Confirmable message at exponentially increasing intervals, until it receives an acknowledgement (or Reset message), or runs out of attempts.

Retransmission is controlled by two things that a CoAP endpoint MUST keep track of for each Confirmable message it sends while waiting for an acknowledgement (or reset): a timeout and a retransmission counter. For a new Confirmable message, the initial timeout is set to a random duration (often not an integral number of seconds) between `ACK_TIMEOUT` and $(\text{ACK_TIMEOUT} * \text{ACK_RANDOM_FACTOR})$ (see Section 4.8), and the retransmission counter is set to 0. When the timeout is triggered and the retransmission counter is less than `MAX_RETRANSMIT`, the message is retransmitted, the retransmission counter is incremented, and the timeout is doubled. If the retransmission counter reaches `MAX_RETRANSMIT` on a timeout, or if the endpoint receives a Reset message, then the attempt to transmit the message is canceled and the application process informed of failure. On the other hand, if the endpoint receives an acknowledgement in time, transmission is considered successful.

This specification makes no strong requirements on the accuracy of the clocks used to implement the above binary exponential backoff algorithm. In particular, an endpoint may be late for a specific retransmission due to its sleep schedule, and maybe catch up on the next one. However, the minimum spacing before another retransmission is `ACK_TIMEOUT`, and the entire sequence of (re-)transmissions MUST stay in the envelope of `MAX_TRANSMIT_SPAN` (see Section 4.8.2), even if that means a sender may miss an opportunity to transmit.

A CoAP endpoint that sent a Confirmable message MAY give up in attempting to obtain an ACK even before the `MAX_RETRANSMIT` counter value is reached: E.g., the application has canceled the request as

it no longer needs a response, or there is some other indication that the CON message did arrive. In particular, a CoAP request message may have elicited a separate response, in which case it is clear to the requester that only the ACK was lost and a retransmission of the request would serve no purpose. However, a responder MUST NOT in turn rely on this cross-layer behavior from a requester, i.e. it MUST retain the state to create the ACK for the request, if needed, even if a Confirmable response was already acknowledged by the requester.

Another reason for giving up retransmission MAY be the receipt of ICMP errors. If it is desired to take account of ICMP errors, to mitigate potential spoofing attacks, implementations SHOULD take care to check the information about the original datagram in the ICMP message, including port numbers and CoAP header information such as message type and code, Message ID, and Token; if this is not possible due to limitations of the UDP service API, ICMP errors SHOULD be ignored. Packet Too Big errors [RFC4443] ("fragmentation needed and DF set" for IPv4 [RFC0792]) cannot properly occur and SHOULD be ignored if the implementation note in Section 4.6 is followed; otherwise, they SHOULD feed into a path MTU discovery algorithm [RFC4821]. Source Quench and Time Exceeded ICMP messages SHOULD be ignored. Host, network, port or protocol unreachable errors, or parameter problem errors MAY, after appropriate vetting, be used to inform the application of a failure in sending.

4.3. Messages Transmitted Without Reliability

Some messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual success is sufficient.

As a more lightweight alternative, a message can be transmitted less reliably by marking the message as Non-confirmable. A Non-confirmable message always carries either a request or response and MUST NOT be Empty. A Non-confirmable message MUST NOT be acknowledged by the recipient. If a recipient lacks context to process the message properly (including the case where the message is Empty, uses a code with a reserved class (1, 6 or 7), or has a message format error), it MUST reject the message; rejecting a Non-confirmable message MAY involve sending a matching Reset message, and apart from the Reset message the rejected message MUST be silently ignored.

At the CoAP level, there is no way for the sender to detect if a Non-confirmable message was received or not. A sender MAY choose to transmit multiple copies of a Non-confirmable message within

MAX_TRANSMIT_SPAN (limited by the provisions of Section 4.7, in particular by PROBING_RATE if no response is received), or the network may duplicate the message in transit. To enable the receiver to act only once on the message, Non-confirmable messages specify a Message ID as well. (This Message ID is drawn from the same number space as the Message IDs for Confirmable messages.)

Summarizing Section 4.2 and Section 4.3, the four message types can be used as in Table 1. "*" means that the combination is not used in normal operation, but only to elicit a Reset message ("CoAP ping").

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

Table 1: Usage of message types

4.4. Message Correlation

An Acknowledgement or Reset message is related to a Confirmable message or Non-confirmable message by means of a Message ID along with additional address information of the corresponding endpoint. The Message ID is a 16-bit unsigned integer that is generated by the sender of a Confirmable or Non-confirmable message and included in the CoAP header. The Message ID MUST be echoed in the Acknowledgement or Reset message by the recipient.

The same Message ID MUST NOT be re-used (in communicating with the same endpoint) within the EXCHANGE_LIFETIME (Section 4.8.2).

Implementation Note: Several implementation strategies can be employed for generating Message IDs. In the simplest case a CoAP endpoint generates Message IDs by keeping a single Message ID variable, which is changed each time a new Confirmable or Non-confirmable message is sent regardless of the destination address or port. Endpoints dealing with large numbers of transactions could keep multiple Message ID variables, for example per prefix or destination address (note that some receiving endpoints may not be able to distinguish unicast and multicast packets addressed to it, so endpoints generating Message IDs need to make sure these do not overlap). It is strongly recommended that the initial value of the variable (e.g., on startup) be randomized, in order to make successful off-path attacks on the protocol less likely.

For an Acknowledgement or Reset message to match a Confirmable or Non-confirmable message, the Message ID and source endpoint of the Acknowledgement or Reset message MUST match the Message ID and destination endpoint of the Confirmable or Non-confirmable message.

4.5. Message Deduplication

A recipient might receive the same Confirmable message (as indicated by the Message ID and source endpoint) multiple times within the EXCHANGE_LIFETIME (Section 4.8.2), for example, when its Acknowledgement went missing or didn't reach the original sender before the first timeout. The recipient SHOULD acknowledge each duplicate copy of a Confirmable message using the same Acknowledgement or Reset message, but SHOULD process any request or response in the message only once. This rule MAY be relaxed in case the Confirmable message transports a request that is idempotent (see Section 5.1) or can be handled in an idempotent fashion. Examples for relaxed message deduplication:

- o A server might relax the requirement to answer all retransmissions of an idempotent request with the same response (Section 4.2), so that it does not have to maintain state for Message IDs. For example, an implementation might want to process duplicate transmissions of a GET, PUT or DELETE request as separate requests if the effort incurred by duplicate processing is less expensive than keeping track of previous responses would be.
- o A constrained server might even want to relax this requirement for certain non-idempotent requests if the application semantics make this trade-off favorable. For example, if the result of a POST request is just the creation of some short-lived state at the server, it may be less expensive to incur this effort multiple times for a request than keeping track of whether a previous transmission of the same request already was processed.

A recipient might receive the same Non-confirmable message (as indicated by the Message ID and source endpoint) multiple times within NON_LIFETIME (Section 4.8.2). As a general rule that MAY be relaxed based on the specific semantics of a message, the recipient SHOULD silently ignore any duplicated Non-confirmable message, and SHOULD process any request or response in the message only once.

4.6. Message Size

While specific link layers make it beneficial to keep CoAP messages small enough to fit into their link layer packets (see Section 1), this is a matter of implementation quality. The CoAP specification itself provides only an upper bound to the message size. Messages

larger than an IP packet result in undesirable packet fragmentation. A CoAP message, appropriately encapsulated, SHOULD fit within a single IP packet (i.e., avoid IP fragmentation) and (by fitting into one UDP payload) obviously needs to fit within a single IP datagram. If the Path MTU is not known for a destination, an IP MTU of 1280 bytes SHOULD be assumed; if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size.

Implementation Note: CoAP's choice of message size parameters works well with IPv6 and with most of today's IPv4 paths. (However, with IPv4, it is harder to absolutely ensure that there is no IP fragmentation. If IPv4 support on unusual networks is a consideration, implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes; worse, the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload. Implementations extremely focused on this problem set might also set the IPv4 DF bit and perform some form of path MTU discovery [RFC4821]; this should generally be unnecessary in most realistic use cases for CoAP, however.) A more important kind of fragmentation in many constrained networks is that on the adaptation layer (e.g., 6LoWPAN L2 packets are limited to 127 bytes including various overheads); this may motivate implementations to be frugal in their packet sizes and to move to block-wise transfers [I-D.ietf-core-block] when approaching three-digit message sizes.

Message sizes are also of considerable importance to implementations on constrained nodes. Many implementations will need to allocate a buffer for incoming messages. If an implementation is too constrained to allow for allocating the above-mentioned upper bound, it could apply the following implementation strategy for messages not using DTLS security: Implementations receiving a datagram into a buffer that is too small are usually able to determine if the trailing portion of a datagram was discarded and to retrieve the initial portion. So, if not all of the payload, at least the CoAP header and options are likely to fit within the buffer. A server can thus fully interpret a request and return a 4.13 (Request Entity Too Large, see Section 5.9.2.9) response code if the payload was truncated. A client sending an idempotent request and receiving a response larger than would fit in the buffer can repeat the request with a suitable value for the Block Option [I-D.ietf-core-block].

4.7. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.2.

In order not to cause congestion, Clients (including proxies) MUST strictly limit the number of simultaneous outstanding interactions that they maintain to a given server (including proxies) to NSTART. An outstanding interaction is either a CON for which an ACK has not yet been received but is still expected (message layer) or a request for which neither a response nor an Acknowledgment message has yet been received but is still expected (which may both occur at the same time, counting as one outstanding interaction). The default value of NSTART for this specification is 1.

Further congestion control optimizations and considerations are expected in the future, which may for example provide automatic initialization of the CoAP transmission parameters defined in Section 4.8, and thus may allow a value for NSTART greater than one.

A client stops expecting a response to a Confirmable request for which no acknowledgment message was received, after EXCHANGE_LIFETIME. The specific algorithm by which a client stops to "expect" a response to a Confirmable request that was acknowledged, or to a Non-confirmable request, is not defined. Unless this is modified by additional congestion control optimizations, it MUST be chosen in such a way that an endpoint does not exceed an average data rate of PROBING_RATE in sending to another endpoint that does not respond.

Note: CoAP places the onus of congestion control mostly on the clients. However, clients may malfunction or actually be attackers, e.g. to perform amplification attacks (Section 11.3). To limit the damage (to the network and to its own energy resources), a server SHOULD implement some rate limiting for its response transmission based on reasonable assumptions about application requirements. This is most helpful if the rate limit can be made effective for the misbehaving endpoints, only.

4.8. Transmission Parameters

Message transmission is controlled by the following parameters:

name	default value
ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1

DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 Byte/second

Table 2: CoAP Protocol Parameters

4.8.1. Changing The Parameters

The values for `ACK_TIMEOUT`, `ACK_RANDOM_FACTOR`, `MAX_RETRANSMIT`, `NSTART`, `DEFAULT_LEISURE` (Section 8.2), and `PROBING_RATE` may be configured to values specific to the application environment (including dynamically adjusted values), however the configuration method is out of scope of this document. It is RECOMMENDED that an application environment use consistent values for these parameters; the specific effects of operating with inconsistent values in an application environment are outside the scope of the present specification.

The transmission parameters have been chosen to achieve a behavior in the presence of congestion that is safe in the Internet. If a configuration desires to use different values, the onus is on the configuration to ensure these congestion control properties are not violated. In particular, a decrease of `ACK_TIMEOUT` below 1 second would violate the guidelines of [RFC5405].

([I-D.allman-tcpm-rto-consider] provides some additional background.) CoAP was designed to enable implementations that do not maintain round-trip-time (RTT) measurements. However, where it is desired to decrease the `ACK_TIMEOUT` significantly or increase `NSTART`, this can only be done safely when maintaining such measurements. Configurations MUST NOT decrease `ACK_TIMEOUT` or increase `NSTART` without using mechanisms that ensure congestion control safety, either defined in the configuration or in future standards documents.

`ACK_RANDOM_FACTOR` MUST NOT be decreased below 1.0, and it SHOULD have a value that is sufficiently different from 1.0 to provide some protection from synchronization effects.

`MAX_RETRANSMIT` can be freely adjusted, but a too small value will reduce the probability that a Confirmable message is actually received, while a larger value than given here will require further adjustments in the time values (see Section 4.8.2).

If the choice of transmission parameters leads to an increase of derived time values (see Section 4.8.2), the configuration mechanism MUST ensure the adjusted value is also available to all the endpoints that these adjusted values are to be used to communicate with.

4.8.2. Time Values derived from Transmission Parameters

The combination of `ACK_TIMEOUT`, `ACK_RANDOM_FACTOR` and `MAX_RETRANSMIT` influences the timing of retransmissions, which in turn influences how long certain information items need to be kept by an implementation. To be able to unambiguously reference these derived time values, we give them names as follows:

- o `MAX_TRANSMIT_SPAN` is the maximum time from the first transmission of a Confirmable message to its last retransmission. For the default transmission parameters, the value is $(2+4+8+16)*1.5 = 45$ seconds, or more generally:

$$\text{ACK_TIMEOUT} * ((2 ** \text{MAX_RETRANSMIT}) - 1) * \text{ACK_RANDOM_FACTOR}$$

- o `MAX_TRANSMIT_WAIT` is the maximum time from the first transmission of a Confirmable message to the time when the sender gives up on receiving an acknowledgement or reset. For the default transmission parameters, the value is $(2+4+8+16+32)*1.5 = 93$ seconds, or more generally:

$$\text{ACK_TIMEOUT} * ((2 ** (\text{MAX_RETRANSMIT} + 1)) - 1) * \text{ACK_RANDOM_FACTOR}$$

In addition, some assumptions need to be made on the characteristics of the network and the nodes.

- o `MAX_LATENCY` is the maximum time a datagram is expected to take from the start of its transmission to the completion of its reception. This constant is related to the MSL (Maximum Segment Lifetime) of [RFC0793], which is "arbitrarily defined to be 2 minutes" ([RFC0793] glossary, page 81). Note that this is not necessarily smaller than `MAX_TRANSMIT_WAIT`, as `MAX_LATENCY` is not intended to describe a situation when the protocol works well, but the worst case situation against which the protocol has to guard. We, also arbitrarily, define `MAX_LATENCY` to be 100 seconds. Apart from being reasonably realistic for the bulk of configurations as well as close to the historic choice for TCP, this value also allows Message ID lifetime timers to be represented in 8 bits (when measured in seconds). In these calculations, there is no assumption that the direction of the transmission is irrelevant (i.e. that the network is symmetric), just that the same value can reasonably be used as a maximum value for both directions. If that is not the case, the following calculations become only slightly more complex.
- o `PROCESSING_DELAY` is the time a node takes to turn around a Confirmable message into an acknowledgement. We assume the node

will attempt to send an ACK before having the sender time out, so as a conservative assumption we set it equal to ACK_TIMEOUT.

- o MAX_RTT is the maximum round-trip time, or:

$$(2 * MAX_LATENCY) + PROCESSING_DELAY$$

From these values, we can derive the following values relevant to the protocol operation:

- o EXCHANGE_LIFETIME is the time from starting to send a Confirmable message to the time when an acknowledgement is no longer expected, i.e. message layer information about the message exchange can be purged. EXCHANGE_LIFETIME includes a MAX_TRANSMIT_SPAN, a MAX_LATENCY forward, PROCESSING_DELAY, and a MAX_LATENCY for the way back. Note that there is no need to consider MAX_TRANSMIT_WAIT if the configuration is chosen such that the last waiting period ($ACK_TIMEOUT * (2 ** MAX_RETRANSMIT)$) or the difference between MAX_TRANSMIT_SPAN and MAX_TRANSMIT_WAIT is less than MAX_LATENCY -- which is a likely choice, as MAX_LATENCY is a worst case value unlikely to be met in the real world. In this case, EXCHANGE_LIFETIME simplifies to:

$$MAX_TRANSMIT_SPAN + (2 * MAX_LATENCY) + PROCESSING_DELAY$$

or 247 seconds with the default transmission parameters.

- o NON_LIFETIME is the time from sending a Non-confirmable message to the time its Message ID can be safely reused. If multiple transmission of a NON message is not used, its value is MAX_LATENCY, or 100 seconds. However, a CoAP sender might send a NON message multiple times, in particular for multicast applications. While the period of re-use is not bounded by the specification, an expectation of reliable detection of duplication at the receiver is in the timescales of MAX_TRANSMIT_SPAN. Therefore, for this purpose, it is safer to use the value:

$$MAX_TRANSMIT_SPAN + MAX_LATENCY$$

or 145 seconds with the default transmission parameters; however, an implementation that just wants to use a single timeout value for retiring Message IDs can safely use the larger value for EXCHANGE_LIFETIME.

Table 3 summarizes the derived parameters introduced in this subsection with their default values.

name	default value
MAX_TRANSMIT_SPAN	45 s
MAX_TRANSMIT_WAIT	93 s
MAX_LATENCY	100 s
PROCESSING_DELAY	2 s
MAX_RTT	202 s
EXCHANGE_LIFETIME	247 s
NON_LIFETIME	145 s

Table 3: Derived Protocol Parameters

5. Request/Response Semantics

CoAP operates under a similar request/response model as HTTP: a CoAP endpoint in the role of a "client" sends one or more CoAP requests to a "server", which services the requests by sending CoAP responses. Unlike HTTP, requests and responses are not sent over a previously established connection, but exchanged asynchronously over CoAP messages.

5.1. Requests

A CoAP request consists of the method to be applied to the resource, the identifier of the resource, a payload and Internet media type (if any), and optional meta-data about the request.

CoAP supports the basic methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP. They have the same properties of safe (only retrieval) and idempotent (you can invoke it multiple times with the same effects) as HTTP (see Section 9.1 of [RFC2616]). The GET method is safe, therefore it MUST NOT take any other action on a resource other than retrieval. The GET, PUT and DELETE methods MUST be performed in such a way that they are idempotent. POST is not idempotent, because its effect is determined by the origin server and dependent on the target resource; it usually results in a new resource being created or the target resource being updated.

A request is initiated by setting the Code field in the CoAP header of a Confirmable or a Non-confirmable message to a Method Code and including request information.

The methods used in requests are described in detail in Section 5.8.

5.2. Responses

After receiving and interpreting a request, a server responds with a CoAP response, which is matched to the request by means of a client-generated token (Section 5.3, note that this is different from the Message ID that matches a Confirmable message to its Acknowledgement).

A response is identified by the Code field in the CoAP header being set to a Response Code. Similar to the HTTP Status Code, the CoAP Response Code indicates the result of the attempt to understand and satisfy the request. These codes are fully defined in Section 5.9. The Response Code numbers to be set in the Code field of the CoAP header are maintained in the CoAP Response Code Registry (Section 12.1.2).

```

0
0 1 2 3 4 5 6 7
+---+---+---+---+
|class|  detail  |
+---+---+---+---+

```

Figure 9: Structure of a Response Code

The upper three bits of the 8-bit Response Code number define the class of response. The lower five bits do not have any categorization role; they give additional detail to the overall class (Figure 9).

As a human readable notation for specifications and protocol diagnostics, CoAP code numbers including the response code are documented in the format "c.dd", where "c" is the class in decimal, and "dd" is the detail as a two-digit decimal. For example, "Forbidden" is written as 4.03 -- indicating an 8-bit code value of hexadecimal 0x83 (4*0x20+3) or decimal 131 (4*32+3).

There are 3 classes of response codes:

- 2 - Success: The request was successfully received, understood, and accepted.
- 4 - Client Error: The request contains bad syntax or cannot be fulfilled.
- 5 - Server Error: The server failed to fulfill an apparently valid request.

The response codes are designed to be extensible: Response Codes in the Client Error and Server Error class that are unrecognized by an endpoint are treated as being equivalent to the generic Response Code

of that class (4.00 and 5.00, respectively). However, there is no generic Response Code indicating success, so a Response Code in the Success class that is unrecognized by an endpoint can only be used to determine that the request was successful without any further details.

The possible response codes are described in detail in Section 5.9.

Responses can be sent in multiple ways, which are defined in the following subsections.

5.2.1. Piggy-backed

In the most basic case, the response is carried directly in the Acknowledgement message that acknowledges the request (which requires that the request was carried in a Confirmable message). This is called a "Piggy-backed" Response.

The response is returned in the Acknowledgement message independent of whether the response indicates success or failure. In effect, the response is piggy-backed on the Acknowledgement message, and no separate message is required to return the response.

Implementation Note: The protocol leaves the decision whether to piggy-back a response or not (i.e., send a separate response) to the server. The client MUST be prepared to receive either. On the quality of implementation level, there is a strong expectation that servers will implement code to piggy-back whenever possible -- saving resources in the network and both at the client and at the server.

5.2.2. Separate

It may not be possible to return a piggy-backed response in all cases. For example, a server might need longer to obtain the representation of the resource requested than it can wait sending back the Acknowledgement message, without risking the client to repeatedly retransmit the request message (see also the discussion of `PROCESSING_DELAY` in Section 4.8.2). The Response to a request carried in a Non-confirmable message is always sent separately (as there is no Acknowledgement message).

One way to implement this in a server is to initiate the attempt to obtain the resource representation and, while that is in progress, time out an acknowledgement timer. A server may also immediately send an acknowledgement knowing in advance that there will be no piggy-backed response. In both cases, the acknowledgement effectively is a promise that the request will be acted upon later.

When the server finally has obtained the resource representation, it sends the response. When it is desired that this message is not lost, it is sent as a Confirmable message from the server to the client and answered by the client with an Acknowledgement, echoing the new Message ID chosen by the server. (It may also be sent as a Non-confirmable message; see Section 5.2.3.)

When the server chooses to use a separate response, it sends the Acknowledgement to the Confirmable request as an Empty message. Once the server sends back an Empty Acknowledgement, it **MUST NOT** send back the response in another Acknowledgement, even if the client retransmits another identical request. If a retransmitted request is received (perhaps because the original Acknowledgement was delayed), another Empty Acknowledgement is sent and any response **MUST** be sent as a separate response.

If the server then sends a Confirmable response, the client's Acknowledgement to that response **MUST** also be an Empty message (one that carries neither a request nor a response). The server **MUST** stop retransmitting its response on any matching Acknowledgement (silently ignoring any response code or payload) or Reset message.

Implementation Notes: Note that, as the underlying datagram transport may not be sequence-preserving, the Confirmable message carrying the response may actually arrive before or after the Acknowledgement message for the request; for the purposes of terminating the retransmission sequence, this also serves as an acknowledgement. Note also that, while the CoAP protocol itself does not make any specific demands here, there is an expectation that the response will come within a time frame that is reasonable from an application point of view; as there is no underlying transport protocol that could be instructed to run a keep-alive mechanism, the requester may want to set up a timeout that is unrelated to CoAP's retransmission timers in case the server is destroyed or otherwise unable to send the response.)

5.2.3. Non-confirmable

If the request message is Non-confirmable, then the response **SHOULD** be returned in a Non-confirmable message as well. However, an endpoint **MUST** be prepared to receive a Non-confirmable response (preceded or followed by an Empty Acknowledgement message) in reply to a Confirmable request, or a Confirmable response in reply to a Non-confirmable request.

5.3. Request/Response Matching

Regardless of how a response is sent, it is matched to the request by means of a token that is included by the client in the request, along with additional address information of the corresponding endpoint.

5.3.1. Token

The Token is used to match a response with a request. The token value is a sequence of 0 to 8 bytes. (Note that every message carries a token, even if it is of zero length.) Every request carries a client-generated token, which the server **MUST** echo in any resulting response without modification.

A token is intended for use as a client-local identifier for differentiating between concurrent requests (see Section 5.3); it could have been called a "request ID".

The client **SHOULD** generate tokens in such a way that tokens currently in use for a given source/destination endpoint pair are unique. (Note that a client implementation can use the same token for any request if it uses a different endpoint each time, e.g. a different source port number.) An empty token value is appropriate e.g. when no other tokens are in use to a destination, or when requests are made serially per destination and receive piggy-backed responses. There are however multiple possible implementation strategies to fulfill this.

A client sending a request without using transport layer security (Section 9) **SHOULD** use a non-trivial, randomized token to guard against spoofing of responses (Section 11.4). This protective use of tokens is the reason they are allowed to be up to 8 bytes in size. The actual size of the random component to be used for the Token depends on the security requirements of the client and the level of threat posed by spoofing of responses. A client that is connected to the general Internet **SHOULD** use at least 32 bits of randomness; keeping in mind that not being directly connected to the Internet is not necessarily sufficient protection against spoofing. (Note that the Message ID adds little in protection as it is usually sequentially assigned, i.e. guessable, and can be circumvented by spoofing a separate response.) Clients that want to optimize the Token length may further want to detect the level of ongoing attacks (e.g., by tallying recent Token mismatches in incoming messages) and adjust the Token length upwards appropriately. [RFC4086] discusses randomness requirements for security.

An endpoint receiving a token it did not generate **MUST** treat it as opaque and make no assumptions about its content or structure.

5.3.2. Request/Response Matching Rules

The exact rules for matching a response to a request are as follows:

1. The source endpoint of the response MUST be the same as the destination endpoint of the original request.
2. In a piggy-backed response, both the Message ID of the Confirmable request and the Acknowledgement, and the token of the response and original request MUST match. In a separate response, just the token of the response and original request MUST match.

In case a message carrying a response is unexpected (the client is not waiting for a response from the identified endpoint, at the endpoint addressed, and/or with the given token), the response is rejected (Section 4.2, Section 4.3).

Implementation Note: A client that receives a response in a CON message may want to clean up the message state right after sending the ACK. If that ACK is lost and the server retransmits the CON, the client may no longer have any state to correlate this response to, making the retransmission an unexpected message; the client will likely send a Reset message so it does not receive any more retransmissions. This behavior is normal and not an indication of an error. (Clients that are not aggressively optimized in their state memory usage will still have message state that will identify the second CON as a retransmission. Clients that actually expect more messages from the server [I-D.ietf-core-observe] will have to keep state in any case.)

5.4. Options

Both requests and responses may include a list of one or more options. For example, the URI in a request is transported in several options, and meta-data that would be carried in an HTTP header in HTTP is supplied as options as well.

CoAP defines a single set of options that are used in both requests and responses:

- o Content-Format
- o ETag
- o Location-Path
- o Location-Query

- o Max-Age
- o Proxy-Uri
- o Proxy-Scheme
- o Uri-Host
- o Uri-Path
- o Uri-Port
- o Uri-Query
- o Accept
- o If-Match
- o If-None-Match
- o Size1

The semantics of these options along with their properties are defined in detail in Section 5.10.

Not all options are defined for use with all methods and response codes. The possible options for methods and response codes are defined in Section 5.8 and Section 5.9 respectively. In case an option is not defined for a method or response code, it MUST NOT be included by a sender and MUST be treated like an unrecognized option by a recipient.

5.4.1. Critical/Elective

Options fall into one of two classes: "critical" or "elective". The difference between these is how an option unrecognized by an endpoint is handled:

- o Upon reception, unrecognized options of class "elective" MUST be silently ignored.
- o Unrecognized options of class "critical" that occur in a Confirmable request MUST cause the return of a 4.02 (Bad Option) response. This response SHOULD include a diagnostic payload describing the unrecognized option(s) (see Section 5.5.2).

- o Unrecognized options of class "critical" that occur in a Confirmable response, or piggy-backed in an Acknowledgement, MUST cause the response to be rejected (Section 4.2).
- o Unrecognized options of class "critical" that occur in a Non-confirmable message MUST cause the message to be rejected (Section 4.3).

Note that, whether critical or elective, an option is never "mandatory" (it is always optional): These rules are defined in order to enable implementations to stop processing options they do not understand or implement.

Critical/Elective rules apply to non-proxying endpoints. A proxy processes options based on Unsafe/Safe-to-Forward classes as defined in Section 5.7.

5.4.2. Proxy Unsafe/Safe-to-Forward and NoCacheKey

In addition to an option being marked as Critical or Elective, options are also classified based on how a proxy is to deal with the option if it does not recognize it. For this purpose, an option can either be considered Unsafe to Forward (Unsafe is set) or Safe-to-Forward (Unsafe is clear).

In addition, for an option that is marked Safe-to-Forward, the option number indicates whether it is intended to be part of the Cache-Key (Section 5.6) in a request or not; if some of the NoCacheKey bits are 0, it is, if all NoCacheKey bits are 1, it is not (see Section 5.4.6).

Note: The Cache-Key indication is relevant only for proxies that do not implement the given option as a request option and instead rely on the Unsafe/Safe-to-Forward indication only. E.g., for ETag, actually using the request option as a part of the Cache-Key is grossly inefficient, but it is the best thing one can do if ETag is not implemented by a proxy, as the response is going to differ based on the presence of the request option. A more useful proxy that does implement the ETag request option is not using ETag as a part of the Cache-Key.

NoCacheKey is indicated in three bits so that only one out of eight codepoints is qualified as NoCacheKey, assuming this is the less likely case.

Proxy behavior with regard to these classes is defined in Section 5.7.

5.4.3. Length

Option values are defined to have a specific length, often in the form of an upper and lower bound. If the length of an option value in a request is outside the defined range, that option **MUST** be treated like an unrecognized option (see Section 5.4.1).

5.4.4. Default Values

Options may be defined to have a default value. If the value of option is intended to be this default value, the option **SHOULD NOT** be included in the message. If the option is not present, the default value **MUST** be assumed.

Where a critical option has a default value, this is chosen in such a way that the absence of the option in a message can be processed properly both by implementations unaware of the critical option and by implementations that interpret this absence as the presence of the default value for the option.

5.4.5. Repeatable Options

The definition of some options specifies that those options are repeatable. An option that is repeatable **MAY** be included one or more times in a message. An option that is not repeatable **MUST NOT** be included more than once in a message.

If a message includes an option with more occurrences than the option is defined for, each supernumerary option occurrence that appears subsequently in the message **MUST** be treated like an unrecognized option (see Section 5.4.1).

5.4.6. Option Numbers

An Option is identified by an option number, which also provides some additional semantics information: e.g., odd numbers indicate a critical option, while even numbers indicate an elective option. Note that this is not just a convention, it is a feature of the protocol: Whether an option is elective or critical is entirely determined by whether its option number is even or odd.

More generally speaking, an Option number is constructed with a bit mask to indicate if an option is Critical/Elective, Unsafe/Safe-to-Forward and in the case of Safe-to-Forward, also a Cache-Key indication as shown by the following figure. In the following text, the bit mask is expressed as a single byte that is applied to the least significant byte of the option number in unsigned integer representation. When bit 7 (the least significant bit) is 1, an

option is Critical (and likewise Elective when 0). When bit 6 is 1, an option is Unsafe (and likewise Safe-to-Forward when 0). When bit 6 is 0, i.e., the option is not Unsafe, it is not a Cache-Key (NoCacheKey) if and only if bits 3-5 are all set to 1; all other bit combinations mean that it indeed is a Cache-Key. These classes of options are explained in the next sections.

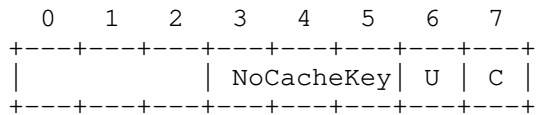


Figure 10: Option Number Mask (Least Significant Byte)

An endpoint may use an equivalent of the C code in Figure 11 to derive the characteristics of an option number "onum".

```

Critical = (onum & 1);
UnSafe = (onum & 2);
NoCacheKey = ((onum & 0x1e) == 0x1c);

```

Figure 11: Determining Characteristics from an Option Number

The option numbers for the options defined in this document are listed in the CoAP Option Number Registry (Section 12.2).

5.5. Payloads and Representations

Both requests and responses may include a payload, depending on the method or response code respectively. If a method or response code is not defined to have a payload, then a sender MUST NOT include one, and a recipient MUST ignore it.

5.5.1. Representation

The payload of requests or of responses indicating success is typically a representation of a resource ("resource representation") or the result of the requested action ("action result"). Its format is specified by the Internet media type and content coding given by the Content-Format Option. In the absence of this option, no default value is assumed and the format will need to be inferred by the application (e.g., from the application context). Payload "sniffing" SHOULD only be attempted if no content type is given.

Implementation Note: On a quality of implementation level, there is a strong expectation that a Content-Format indication will be provided with resource representations whenever possible. This is not a "SHOULD"-level requirement solely because it is not a

protocol requirement, and it also would be difficult to outline exactly in what cases this expectation can be violated.

For responses indicating a client or server error, the payload is considered a representation of the result of the requested action only if a Content-Format Option is given. In the absence of this option, the payload is a Diagnostic Payload (Section 5.5.2).

5.5.2. Diagnostic Payload

If no Content-Format option is given, the payload of responses indicating a client or server error is a brief human-readable diagnostic message, explaining the error situation. This diagnostic message **MUST** be encoded using UTF-8 [RFC3629], more specifically using Net-Unicode form [RFC5198].

The message is similar to the Reason-Phrase on an HTTP status line. It is not intended for end-users but for software engineers that during debugging need to interpret it in the context of the present, English-language specification; therefore no mechanism for language tagging is needed or provided. In contrast to what is usual in HTTP, the payload **SHOULD** be empty if there is no additional information beyond the response code.

5.5.3. Selected Representation

Not all responses carry a payload that provides a representation of the resource addressed by the request. It is, however, sometimes useful to be able to refer to such a representation in relation to a response, independent of whether it actually was enclosed.

We use the term "selected representation" to refer to the current representation of a target resource that would have been selected in a successful response if the corresponding request had used the method GET and excluded any conditional request options (Section 5.10.8).

Certain response options provide metadata about the selected representation, which might differ from the representation included in the message for responses to some state-changing methods. Of the response options defined in this specification, only the ETag response option (Section 5.10.6) is defined as selected representation metadata.

5.5.4. Content Negotiation

A server may be able to supply a representation for a resource in one of multiple representation formats. Without further information from

the client, it will provide the representation in the format it prefers.

By using the Accept Option (Section 5.10.4) in a request, the client can indicate which content-format it prefers to receive.

5.6. Caching

CoAP endpoints MAY cache responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests.

The goal of caching in CoAP is to reuse a prior response message to satisfy a current request. In some cases, a stored response can be reused without the need for a network request, reducing latency and network round-trips; a "freshness" mechanism is used for this purpose (see Section 5.6.1). Even when a new request is required, it is often possible to reuse the payload of a prior response to satisfy the request, thereby reducing network bandwidth usage; a "validation" mechanism is used for this purpose (see Section 5.6.2).

Unlike HTTP, the cacheability of CoAP responses does not depend on the request method, but the Response Code. The cacheability of each Response Code is defined along the Response Code definitions in Section 5.9. Response Codes that indicate success and are unrecognized by an endpoint MUST NOT be cached.

For a presented request, a CoAP endpoint MUST NOT use a stored response, unless:

- o the presented request method and that used to obtain the stored response match,
- o all options match between those in the presented request and those of the request used to obtain the stored response (which includes the request URI), except that there is no need for a match of any request options marked as NoCacheKey (Section 5.4) or recognized by the Cache and fully interpreted with respect to its specified cache behavior (such as the ETag request option, Section 5.10.6, see also Section 5.4.2), and
- o the stored response is either fresh or successfully validated as defined below.

The set of request options that is used for matching the cache entry is also collectively referred to as the "Cache-Key". For URI schemes other than coap and coaps, matching of those options that constitute the request URI may be performed under rules specific to the URI scheme.

5.6.1. Freshness Model

When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.

The mechanism for determining freshness is for an origin server to provide an explicit expiration time in the future, using the Max-Age Option (see Section 5.10.5). The Max-Age Option indicates that the response is to be considered not fresh after its age is greater than the specified number of seconds.

The Max-Age Option defaults to a value of 60. Thus, if it is not present in a cacheable response, then the response is considered not fresh after its age is greater than 60 seconds. If an origin server wishes to prevent caching, it MUST explicitly include a Max-Age Option with a value of zero seconds.

If a client has a fresh stored response and makes a new request matching the request for that stored response, the new response invalidates the old response.

5.6.2. Validation Model

When an endpoint has one or more stored responses for a GET request, but cannot use any of them (e.g., because they are not fresh), it can use the ETag Option (Section 5.10.6) in the GET request to give the origin server an opportunity to both select a stored response to be used, and to update its freshness. This process is known as "validating" or "revalidating" the stored response.

When sending such a request, the endpoint SHOULD add an ETag Option specifying the entity-tag of each stored response that is applicable.

A 2.03 (Valid) response indicates the stored response identified by the entity-tag given in the response's ETag Option can be reused, after updating it as described in Section 5.9.1.3.

Any other response code indicates that none of the stored responses nominated in the request is suitable. Instead, the response SHOULD be used to satisfy the request and MAY replace the stored response.

5.7. Proxying

A proxy is a CoAP endpoint that can be tasked by CoAP clients to perform requests on their behalf. This may be useful, for example, when the request could otherwise not be made, or to service the response from a cache in order to reduce response time and network bandwidth or energy consumption.

In an overall architecture for a Constrained RESTful Environment, proxies can serve quite different purposes. Proxies can be explicitly selected by clients, a role that we term "forward-proxy". Proxies can also be inserted to stand in for origin servers, a role that we term "reverse-proxy". Orthogonal to this distinction, a proxy can map from a CoAP request to a CoAP request (CoAP-to-CoAP proxy) or translate from or to a different protocol ("cross-proxy"). Full definitions of these terms are provided in Section 1.2.

Notes: The terminology in this specification has been selected to be culturally compatible with the terminology used in the wider Web application environments, without necessarily matching it in every detail (which may not even be relevant to Constrained RESTful Environments). Not too much semantics should be ascribed to the components of the terms (such as "forward", "reverse", or "cross").

HTTP proxies, besides acting as HTTP proxies, often offer a transport protocol proxying function ("CONNECT") to enable end-to-end transport layer security through the proxy. No such function is defined for CoAP-to-CoAP proxies in this specification, as forwarding of UDP packets is unlikely to be of much value in Constrained RESTful environments. See also Section 10.2.7 for the cross-proxy case.

When a client uses a proxy to make a request that will use a secure URI scheme (e.g., coaps or https), the request towards the proxy SHOULD be sent using DTLS security except where equivalent lower layer security is used for the leg between the client and the proxy.

5.7.1. Proxy Operation

A proxy generally needs a way to determine potential request parameters for a request to a destination based on the request it received. This way is fully specified for a forward-proxy, but may depend on the specific configuration for a reverse-proxy. In particular, the client of a reverse-proxy generally does not indicate a locator for the destination, necessitating some form of namespace translation in the reverse-proxy. However, some aspects of the operation of proxies are common to all its forms.

If a proxy does not employ a cache, then it simply forwards the translated request to the determined destination. Otherwise, if it does employ a cache but does not have a stored response that matches the translated request and is considered fresh, then it needs to refresh its cache according to Section 5.6. For options in the request that the proxy recognizes, it knows whether the option is intended to act as part of the key used in looking up the cached value or not. E.g., since requests for different Uri-Path values address different resources, Uri-Path values are always part of the Cache-Key, while, e.g., Token values are never part of the Cache-Key. For options that the proxy does not recognize but that are marked Safe-to-Forward in the option number, the option also indicates whether it is to be included in the Cache-Key (NoCacheKey is not all set) or not (NoCacheKey is all set). (Options that are unrecognized and marked Unsafe lead to 4.02 Bad Option.)

If the request to the destination times out, then a 5.04 (Gateway Timeout) response MUST be returned. If the request to the destination returns a response that cannot be processed by the proxy (e.g, due to unrecognized critical options, message format errors), then a 5.02 (Bad Gateway) response MUST be returned. Otherwise, the proxy returns the response to the client.

If a response is generated out of a cache, the generated (or implied) Max-Age Option MUST NOT extend the max-age originally set by the server, considering the time the resource representation spent in the cache. E.g., the Max-Age Option could be adjusted by the proxy for each response using the formula:

$$\text{proxy-max-age} = \text{original-max-age} - \text{cache-age}$$

For example if a request is made to a proxied resource that was refreshed 20 seconds ago and had an original Max-Age of 60 seconds, then that resource's proxied max-age is now 40 seconds. Considering potential network delays on the way from the origin server, a proxy should be conservative in the max-age values offered.

All options present in a proxy request MUST be processed at the proxy. Unsafe options in a request that are not recognized by the proxy MUST lead to a 4.02 (Bad Option) response being returned by the proxy. A CoAP-to-CoAP proxy MUST forward to the origin server all Safe-to-Forward options that it does not recognize. Similarly, Unsafe options in a response that are not recognized by the CoAP-to-CoAP proxy server MUST lead to a 5.02 (Bad Gateway) response. Again, Safe-to-Forward options that are not recognized MUST be forwarded.

Additional considerations for cross-protocol proxying between CoAP and HTTP are discussed in Section 10.

5.7.2. Forward-Proxies

CoAP distinguishes between requests made (as if) to an origin server and a request made through a forward-proxy. CoAP requests to a forward-proxy are made as normal Confirmable or Non-confirmable requests to the forward-proxy endpoint, but specify the request URI in a different way: The request URI in a proxy request is specified as a string in the Proxy-Uri Option (see Section 5.10.2), while the request URI in a request to an origin server is split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options (see Section 5.10.1); alternatively the URI in a proxy request can be assembled from a Proxy-Scheme option and the split options mentioned.

When a proxy request is made to an endpoint and the endpoint is unwilling or unable to act as proxy for the request URI, it MUST return a 5.05 (Proxying Not Supported) response. If the authority (host and port) is recognized as identifying the proxy endpoint itself (see Section 5.10.2), then the request MUST be treated as a local (non-proxied) request.

Unless a proxy is configured to forward the proxy request to another proxy, it MUST translate the request as follows: The scheme of the request URI defines the outgoing protocol and its details (e.g., CoAP is used over UDP for the "coap" scheme and over DTLS for the "coaps" scheme.) For a CoAP-to-CoAP proxy, the origin server's IP address and port are determined by the authority component of the request URI, and the request URI is decoded and split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options. This consumes the Proxy-Uri or Proxy-Scheme option, which is therefore not forwarded to the origin server.

5.7.3. Reverse-Proxies

Reverse-proxies do not make use of the Proxy-Uri or Proxy-Scheme options, but need to determine the destination (next hop) of a request from information in the request and information in their configuration. E.g., a reverse-proxy might offer various resources the existence of which it has learned through resource discovery as if they were its own resources. The reverse-proxy is free to build a namespace for the URIs that identify these resources. A reverse-proxy may also build a namespace that gives the client more control over where the request goes, e.g. by embedding host identifiers and port numbers into the URI path of the resources offered.

In processing the response, a reverse-proxy has to be careful that ETag option values from different sources are not mixed up on one resource offered to its clients. In many cases, the ETag can be forwarded unchanged. If the mapping from a resource offered by the

reverse-proxy to resources offered by its various origin servers is not unique, the reverse-proxy may need to generate a new ETag, making sure the semantics of this option are properly preserved.

5.8. Method Definitions

In this section each method is defined along with its behavior. A request with an unrecognized or unsupported Method Code MUST generate a 4.05 (Method Not Allowed) piggy-backed response.

5.8.1. GET

The GET method retrieves a representation for the information that currently corresponds to the resource identified by the request URI. If the request includes an Accept Option, that indicates the preferred content-format of a response. If the request includes an ETag Option, the GET method requests that ETag be validated and that the representation be transferred only if validation failed. Upon success a 2.05 (Content) or 2.03 (Valid) response code SHOULD be present in the response.

The GET method is safe and idempotent.

5.8.2. POST

The POST method requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and dependent on the target resource. It usually results in a new resource being created or the target resource being updated.

If a resource has been created on the server, the response returned by the server SHOULD have a 2.01 (Created) response code and SHOULD include the URI of the new resource in a sequence of one or more Location-Path and/or Location-Query Options (Section 5.10.7). If the POST succeeds but does not result in a new resource being created on the server, the response SHOULD have a 2.04 (Changed) response code. If the POST succeeds and results in the target resource being deleted, the response SHOULD have a 2.02 (Deleted) response code.

POST is neither safe nor idempotent.

5.8.3. PUT

The PUT method requests that the resource identified by the request URI be updated or created with the enclosed representation. The representation format is specified by the media type and content coding given in the Content-Format Option, if provided.

If a resource exists at the request URI the enclosed representation SHOULD be considered a modified version of that resource, and a 2.04 (Changed) response code SHOULD be returned. If no resource exists then the server MAY create a new resource with that URI, resulting in a 2.01 (Created) response code. If the resource could not be created or modified, then an appropriate error response code SHOULD be sent.

Further restrictions to a PUT can be made by including the If-Match (see Section 5.10.8.1) or If-None-Match (see Section 5.10.8.2) options in the request.

PUT is not safe, but is idempotent.

5.8.4. DELETE

The DELETE method requests that the resource identified by the request URI be deleted. A 2.02 (Deleted) response code SHOULD be used on success or in case the resource did not exist before the request.

DELETE is not safe, but is idempotent.

5.9. Response Code Definitions

Each response code is described below, including any options required in the response. Where appropriate, some of the codes will be specified in regards to related response codes in HTTP [RFC2616]; this does not mean that any such relationship modifies the HTTP mapping specified in Section 10.

5.9.1. Success 2.xx

This class of status code indicates that the clients request was successfully received, understood, and accepted.

5.9.1.1. 2.01 Created

Like HTTP 201 "Created", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

If the response includes one or more Location-Path and/or Location-Query Options, the values of these options specify the location at which the resource was created. Otherwise, the resource was created at the request URI. A cache receiving this response MUST mark any stored response for the created resource as not fresh.

This response is not cacheable.

5.9.1.2. 2.02 Deleted

Like HTTP 204 "No Content", but only used in response to requests that cause the resource to cease being available, such as DELETE and in certain circumstances POST. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache MUST mark any stored response for the deleted resource as not fresh.

5.9.1.3. 2.03 Valid

Related to HTTP 304 "Not Modified", but only used to indicate that the response identified by the entity-tag identified by the included ETag Option is valid. Accordingly, the response MUST include an ETag Option, and MUST NOT include a payload.

When a cache that recognizes and processes the ETag response option receives a 2.03 (Valid) response, it MUST update the stored response with the value of the Max-Age Option included in the response (explicitly, or implicitly as a default value; see also Section 5.6.2). For each type of Safe-to-Forward option present in the response, the (possibly empty) set of options of this type that are present in the stored response MUST be replaced with the set of options of this type in the response received. (Unsafe options may trigger similar option specific processing as defined by the option.)

5.9.1.4. 2.04 Changed

Like HTTP 204 "No Content", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache MUST mark any stored response for the changed resource as not fresh.

5.9.1.5. 2.05 Content

Like HTTP 200 "OK", but only used in response to GET requests.

The payload returned with the response is a representation of the target resource.

This response is cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1) and (if present) the ETag Option for validation (see Section 5.6.2).

5.9.2. Client Error 4.xx

This class of response code is intended for cases in which the client seems to have erred. These response codes are applicable to any request method.

The server SHOULD include a diagnostic payload under the conditions detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

5.9.2.1. 4.00 Bad Request

Like HTTP 400 "Bad Request".

5.9.2.2. 4.01 Unauthorized

The client is not authorized to perform the requested action. The client SHOULD NOT repeat the request without first improving its authentication status to the server. Which specific mechanism can be used for this is outside this document's scope; see also Section 9.

5.9.2.3. 4.02 Bad Option

The request could not be understood by the server due to one or more unrecognized or malformed options. The client SHOULD NOT repeat the request without modification.

5.9.2.4. 4.03 Forbidden

Like HTTP 403 "Forbidden".

5.9.2.5. 4.04 Not Found

Like HTTP 404 "Not Found".

5.9.2.6. 4.05 Method Not Allowed

Like HTTP 405 "Method Not Allowed", but with no parallel to the "Allow" header field.

5.9.2.7. 4.06 Not Acceptable

Like HTTP 406 "Not Acceptable", but with no response entity.

5.9.2.8. 4.12 Precondition Failed

Like HTTP 412 "Precondition Failed".

5.9.2.9. 4.13 Request Entity Too Large

Like HTTP 413 "Request Entity Too Large".

The response SHOULD include a Size1 Option (Section 5.10.9) to indicate the maximum size of request entity the server is able and willing to handle, unless the server is not in a position to make this information available.

5.9.2.10. 4.15 Unsupported Content-Format

Like HTTP 415 "Unsupported Media Type".

5.9.3. Server Error 5.xx

This class of response code indicates cases in which the server is aware that it has erred or is incapable of performing the request. These response codes are applicable to any request method.

The server SHOULD include a diagnostic payload under the conditions detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

5.9.3.1. 5.00 Internal Server Error

Like HTTP 500 "Internal Server Error".

5.9.3.2. 5.01 Not Implemented

Like HTTP 501 "Not Implemented".

5.9.3.3. 5.02 Bad Gateway

Like HTTP 502 "Bad Gateway".

5.9.3.4. 5.03 Service Unavailable

Like HTTP 503 "Service Unavailable", but using the Max-Age Option in place of the "Retry-After" header field to indicate the number of seconds after which to retry.

5.9.3.5. 5.04 Gateway Timeout

Like HTTP 504 "Gateway Timeout".

5.9.3.6. 5.05 Proxying Not Supported

The server is unable or unwilling to act as a forward-proxy for the URI specified in the Proxy-Uri Option or using Proxy-Scheme (see Section 5.10.2).

5.10. Option Definitions

The individual CoAP options are summarized in Table 4 and explained in the subsections of this section.

In this table, the C, U, and N columns indicate the properties, Critical, UnSafe, and NoCacheKey, respectively. Since NoCacheKey only has a meaning for options that are Safe-to-Forward (not marked UnSafe), the column is filled with a dash for UnSafe options. (The present specification does not define any NoCacheKey options, but the format of the table is intended to be useful for additional specifications.)

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

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

Table 4: Options

5.10.1. Uri-Host, Uri-Port, Uri-Path and Uri-Query

The Uri-Host, Uri-Port, Uri-Path and Uri-Query Options are used to specify the target resource of a request to a CoAP origin server. The options encode the different components of the request URI in a way that no percent-encoding is visible in the option values and that the full URI can be reconstructed at any involved endpoint. The syntax of CoAP URIs is defined in Section 6.

The steps for parsing URIs into options is defined in Section 6.4. These steps result in zero or more Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in a request, where each option holds the following values:

- o the Uri-Host Option specifies the Internet host of the resource being requested,
- o the Uri-Port Option specifies the transport layer port number of the resource,
- o each Uri-Path Option specifies one segment of the absolute path to the resource, and
- o each Uri-Query Option specifies one argument parameterizing the resource.

Note: Fragments ([RFC3986], Section 3.5) are not part of the request URI and thus will not be transmitted in a CoAP request.

The default value of the Uri-Host Option is the IP literal representing the destination IP address of the request message. Likewise, the default value of the Uri-Port Option is the destination UDP port. The default values for the Uri-Host and Uri-Port Options are sufficient for requests to most servers. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers.

The Uri-Path and Uri-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Uri-Path Option MUST NOT be "." or ".." (as the request URI must be resolved before parsing it into options).

The steps for constructing the request URI from the options are defined in Section 6.5. Note that an implementation does not

necessarily have to construct the URI; it can simply look up the target resource by looking at the individual options.

Examples can be found in Appendix B.

5.10.2. Proxy-Uri and Proxy-Scheme

The Proxy-Uri Option is used to make a request to a forward-proxy (see Section 5.7). The forward-proxy is requested to forward the request or service it from a valid cache, and return the response.

The option value is an absolute-URI ([RFC3986], Section 4.3).

Note that the forward-proxy MAY forward the request on to another proxy or directly to the server specified by the absolute-URI. In order to avoid request loops, a proxy MUST be able to recognize all of its server names, including any aliases, local variations, and the numeric IP addresses.

An endpoint receiving a request with a Proxy-Uri Option that is unable or unwilling to act as a forward-proxy for the request MUST cause the return of a 5.05 (Proxying Not Supported) response.

The Proxy-Uri Option MUST take precedence over any of the Uri-Host, Uri-Port, Uri-Path or Uri-Query options (which MUST NOT be included at the same time in a request containing the Proxy-Uri Option).

As a special case to simplify many proxy clients, the absolute-URI can be constructed from the Uri-* options. When a Proxy-Scheme Option is present, the absolute-URI is constructed as follows: A CoAP URI is constructed from the Uri-* options as defined in Section 6.5. In the resulting URI, the initial scheme up to, but not including the following colon is then replaced by the content of the Proxy-Scheme Option. Note that this case is only applicable if the components of the desired URI other than the scheme component actually can be expressed using Uri-* options; e.g., to represent a URI with a userinfo component in the authority, only Proxy-Uri can be used.

5.10.3. Content-Format

The Content-Format Option indicates the representation format of the message payload. The representation format is given as a numeric content format identifier that is defined in the CoAP Content Format Registry (Section 12.3). In the absence of the option, no default value is assumed, i.e. the representation format of any representation message payload is indeterminate (Section 5.5).

5.10.4. Accept

The CoAP Accept option can be used to indicate which Content-Format is acceptable to the client. The representation format is given as a numeric Content-Format identifier that is defined in the CoAP Content-Format Registry (Section 12.3). If no Accept option is given, the client does not express a preference (thus no default value is assumed). The client prefers the representation returned by the server to be in the Content-Format indicated. The server returns the preferred Content-Format if available. If the preferred Content-Format cannot be returned, then a 4.06 "Not Acceptable" MUST be sent as a response, unless another error code takes precedence for this response.

5.10.5. Max-Age

The Max-Age Option indicates the maximum time a response may be cached before it is considered not fresh (see Section 5.6.1).

The option value is an integer number of seconds between 0 and $2^{32}-1$ inclusive (about 136.1 years). A default value of 60 seconds is assumed in the absence of the option in a response.

The value is intended to be current at the time of transmission. Servers that provide resources with strict tolerances on the value of Max-Age SHOULD update the value before each retransmission. (See also Section 5.7.1.)

5.10.6. ETag

An entity-tag is intended for use as a resource-local identifier for differentiating between representations of the same resource that vary over time. It is generated by the server providing the resource, which may generate it in any number of ways including a version, checksum, hash or time. An endpoint receiving an entity-tag MUST treat it as opaque and make no assumptions about its content or structure. (Endpoints that generate an entity-tag are encouraged to use the most compact representation possible, in particular in regards to clients and intermediaries that may want to store multiple ETag values.)

5.10.6.1. ETag as a Response Option

The ETag Option in a response provides the current value (i.e., after the request was processed) of the entity-tag for the "tagged representation". If no Location-* options are present, the tagged representation is the selected representation (Section 5.5.3) of the target resource. If one or more Location-* options are present and

thus a location URI is indicated (Section 5.10.7), the tagged representation is the representation that would be retrieved by a GET request to the location URI.

An ETag response option can be included with any response for which there is a tagged representation (e.g., it would not be meaningful in a 4.04 or 4.00 response). The ETag Option MUST NOT occur more than once in a response.

There is no default value for the ETag Option; if it is not present in a response, the server makes no statement about the entity-tag for the tagged representation.

5.10.6.2. ETag as a Request Option

In a GET request, an endpoint that has one or more representations previously obtained from the resource, and has obtained ETag response options with these, can specify an instance of the ETag Option for one or more of these stored responses.

A server can issue a 2.03 Valid response (Section 5.9.1.3) in place of a 2.05 Content response if one of the ETags given is the entity-tag for the current representation, i.e. is valid; the 2.03 Valid response then echoes this specific ETag in a response option.

In effect, a client can determine if any of the stored representations is current (see Section 5.6.2) without needing to transfer them again.

The ETag Option MAY occur zero, one or more times in a request.

5.10.7. Location-Path and Location-Query

The Location-Path and Location-Query Options together indicate a relative URI that consists either of an absolute path, a query string or both. A combination of these options is included in a 2.01 (Created) response to indicate the location of the resource created as the result of a POST request (see Section 5.8.2). The location is resolved relative to the request URI.

If a response with one or more Location-Path and/or Location-Query Options passes through a cache that interprets these options and the implied URI identifies one or more currently stored responses, those entries MUST be marked as not fresh.

Each Location-Path Option specifies one segment of the absolute path to the resource, and each Location-Query Option specifies one argument parameterizing the resource. The Location-Path and

Location-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Location-Path Option MUST NOT be "." or "..".

The steps for constructing the location URI from the options are analogous to Section 6.5, except that the first five steps are skipped and the result is a relative URI-reference, which is then interpreted relative to the request URI. Note that the relative URI-reference constructed this way always includes an absolute-path (e.g., leaving out Location-Path but supplying Location-Query means the path component in the URI is "/").

The options that are used to compute the relative URI-reference are collectively called Location-* options. Beyond Location-Path and Location-Query, more Location-* options may be defined in the future, and have been reserved option numbers 128, 132, 136, and 140. If any of these reserved option numbers occurs in addition to Location-Path and/or Location-Query and are not supported, then a 4.02 (Bad Option) error MUST be returned.

5.10.8. Conditional Request Options

Conditional request options enable a client to ask the server to perform the request only if certain conditions specified by the option are fulfilled.

For each of these options, if the condition given is not fulfilled, then the server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

If the condition is fulfilled, the server performs the request method as if the conditional request options were not present.

If the request would, without the conditional request options, result in anything other than a 2.xx or 4.12 response code, then any conditional request options MAY be ignored.

5.10.8.1. If-Match

The If-Match Option MAY be used to make a request conditional on the current existence or value of an ETag for one or more representations of the target resource. If-Match is generally useful for resource update requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource (i.e., the "lost update" problem).

The value of an If-Match option is either an ETag or the empty string. An If-Match option with an ETag matches a representation with that exact ETag. An If-Match option with an empty value matches any existing representation (i.e., it places the precondition on the existence of any current representation for the target resource).

The If-Match Option can occur multiple times. If any of the options match, then the condition is fulfilled.

If there is one or more If-Match Option, but none of the options match, then the condition is not fulfilled.

5.10.8.2. If-None-Match

The If-None-Match Option MAY be used to make a request conditional on the non-existence of the target resource. If-None-Match is useful for resource creation requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource. The If-None-Match Option carries no value.

If the target resource does exist, then the condition is not fulfilled.

(It is not very useful to combine If-Match and If-None-Match options in one request, because the condition will then never be fulfilled.)

5.10.9. Size1 Option

The Size1 option provides size information about the resource representation in a request. The option value is an integer number of bytes. Its main use is with block-wise transfers [I-D.ietf-core-block]. In the present specification, it is used in 4.13 responses (Section 5.9.2.9) to indicate the maximum size of request entity that the server is able and willing to handle.

6. CoAP URIs

CoAP uses the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. Resources are organized hierarchically and governed by a potential CoAP origin server listening for CoAP requests ("coap") or DTLS-secured CoAP requests ("coaps") on a given UDP port. The CoAP server is identified via the generic syntax's authority component, which includes a host component and optional UDP port number. The remainder of the URI is considered to be identifying a resource which can be operated on by the methods defined by the CoAP protocol. The "coap" and "coaps" URI schemes can thus be compared to the "http" and "https" URI schemes respectively.

The syntax of the "coap" and "coaps" URI schemes is specified in this section in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", "query", "segment", "IP-literal", "IPv4address" and "reg-name" are adopted from [RFC3986].

Implementation Note: Unfortunately, over time the URI format has acquired significant complexity. Implementers are encouraged to examine [RFC3986] closely. E.g., the ABNF for IPv6 addresses is more complicated than maybe expected. Also, implementers should take care to perform the processing of percent decoding/encoding exactly once on the way from a URI to its decoded components or back. Percent encoding is crucial for data transparency, but may lead to unusual results such as a slash in a path component.

6.1. coap URI Scheme

coap-URI = "coap:" "/" host [":" port] path-abempty ["?" query]

If the host component is provided as an IP-literal or IPv4address, then the CoAP server can be reached at that IP address. If host is a registered name, then that name is considered an indirect identifier and the endpoint might use a name resolution service, such as DNS, to find the address of that host. The host MUST NOT be empty; if a URI is received with a missing authority or an empty host, then it MUST be considered invalid. The port subcomponent indicates the UDP port at which the CoAP server is located. If it is empty or not given, then the default port 5683 is assumed.

The path identifies a resource within the scope of the host and port. It consists of a sequence of path segments separated by a slash character (U+002F SOLIDUS "/").

The query serves to further parameterize the resource. It consists of a sequence of arguments separated by an ampersand character

(U+0026 AMPERSAND "&"). An argument is often in the form of a "key=value" pair.

The "coap" URI scheme supports the path prefix `"/.well-known/"` defined by [RFC5785] for "well-known locations" in the name-space of a host. This enables discovery of policy or other information about a host ("site-wide metadata"), such as hosted resources (see Section 7).

Application designers are encouraged to make use of short, but descriptive URIs. As the environments that CoAP is used in are usually constrained for bandwidth and energy, the trade-off between these two qualities should lean towards the shortness, without ignoring descriptiveness.

6.2. coaps URI Scheme

```
coaps-URI = "coaps:" "/" host [ ":" port ] path-abempty
           [ "?" query ]
```

All of the requirements listed above for the "coap" scheme are also requirements for the "coaps" scheme, except that a default UDP port of [IANA_TBD_PORT] is assumed if the port subcomponent is empty or not given, and the UDP datagrams MUST be secured through the use of DTLS as described in Section 9.1.

Considerations for caching of responses to "coaps" identified requests are discussed in Section 11.2.

Resources made available via the "coaps" scheme have no shared identity with the "coap" scheme even if their resource identifiers indicate the same authority (the same host listening to the same UDP port). They are distinct name spaces and are considered to be distinct origin servers.

6.3. Normalization and Comparison Rules

Since the "coap" and "coaps" schemes conform to the URI generic syntax, such URIs are normalized and compared according to the algorithm defined in [RFC3986], Section 6, using the defaults described above for each scheme.

If the port is equal to the default port for a scheme, the normal form is to elide the port subcomponent. Likewise, an empty path component is equivalent to an absolute path of `"/"`, so the normal form is to provide a path of `"/"` instead. The scheme and host are case-insensitive and normally provided in lowercase; IP-literals are

in recommended form [RFC5952]; all other components are compared in a case-sensitive manner. Characters other than those in the "reserved" set are equivalent to their percent-encoded bytes (see [RFC3986], Section 2.1): the normal form is to not encode them.

For example, the following three URIs are equivalent, and cause the same options and option values to appear in the CoAP messages:

```
coap://example.com:5683/~sensors/temp.xml
coap://EXAMPLE.com/%7Eensors/temp.xml
coap://EXAMPLE.com:/%7esensors/temp.xml
```

6.4. Decomposing URIs into Options

The steps to parse a request's options from a string `|url|` are as follows. These steps either result in zero or more of the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in the request, or they fail.

1. If the `|url|` string is not an absolute URI ([RFC3986]), then fail this algorithm.
2. Resolve the `|url|` string using the process of reference resolution defined by [RFC3986]. At this stage the URL is in ASCII encoding [RFC0020], even though the decoded components will be interpreted in UTF-8 [RFC3629] after step 5, 8 and 9.

NOTE: It doesn't matter what it is resolved relative to, since we already know it is an absolute URL at this point.

3. If `|url|` does not have a `<scheme>` component whose value, when converted to ASCII lowercase, is "coap" or "coaps", then fail this algorithm.
4. If `|url|` has a `<fragment>` component, then fail this algorithm.
5. If the `<host>` component of `|url|` does not represent the request's destination IP address as an IP-literal or IPv4address, include a Uri-Host Option and let that option's value be the value of the `<host>` component of `|url|`, converted to ASCII lowercase, and then converting all percent-encodings ("% followed by two hexadecimal digits) to the corresponding characters.

NOTE: In the usual case where the request's destination IP address is derived from the host part, this ensures that a Uri-Host Option is only used for a `<host>` component of the form reg-name.

6. If `|url|` has a `<port>` component, then let `|port|` be that component's value interpreted as a decimal integer; otherwise, let `|port|` be the default port for the scheme.
7. If `|port|` does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be `|port|`.
8. If the value of the `<path>` component of `|url|` is empty or consists of a single slash character (U+002F SOLIDUS "/"), then move to the next step.

Otherwise, for each segment in the `<path>` component, include a Uri-Path Option and let that option's value be the segment (not including the delimiting slash characters) after converting each percent-encoding ("% followed by two hexadecimal digits) to the corresponding byte.

9. If `|url|` has a `<query>` component, then, for each argument in the `<query>` component, include a Uri-Query Option and let that option's value be the argument (not including the question mark and the delimiting ampersand characters) after converting each percent-encoding to the corresponding byte.

Note that these rules completely resolve any percent-encoding.

6.5. Composing URIs from Options

The steps to construct a URI from a request's options are as follows. These steps either result in a URI, or they fail. In these steps, percent-encoding a character means replacing each of its (UTF-8 encoded) bytes by a "%" character followed by two hexadecimal digits representing the byte, where the digits A-F are in upper case (as defined in [RFC3986] Section 2.1; to reduce variability, the hexadecimal notation for percent-encoding in CoAP URIs MUST use uppercase letters). The definitions of "unreserved" and "sub-delims" are adopted from [RFC3986].

1. If the request is secured using DTLS, let `|url|` be the string "coaps://". Otherwise, let `|url|` be the string "coap://".
2. If the request includes a Uri-Host Option, let `|host|` be that option's value, where any non-ASCII characters are replaced by their corresponding percent-encoding. If `|host|` is not a valid reg-name or IP-literal or IPv4address, fail the algorithm. If the request does not include a Uri-Host Option, let `|host|` be the IP-literal (making use of the conventions of [RFC5952]) or IPv4address representing the request's destination IP address.

3. Append `|host|` to `|url|`.
4. If the request includes a Uri-Port Option, let `|port|` be that option's value. Otherwise, let `|port|` be the request's destination UDP port.
5. If `|port|` is not the default port for the scheme, then append a single U+003A COLON character (:) followed by the decimal representation of `|port|` to `|url|`.
6. Let `|resource name|` be the empty string. For each Uri-Path Option in the request, append a single character U+002F SOLIDUS (/) followed by the option's value to `|resource name|`, after converting any character that is not either in the "unreserved" set, "sub-delims" set, a U+003A COLON (:) or U+0040 COMMERCIAL AT (@) character, to its percent-encoded form.
7. If `|resource name|` is the empty string, set it to a single character U+002F SOLIDUS (/).
8. For each Uri-Query Option in the request, append a single character U+003F QUESTION MARK (?) (first option) or U+0026 AMPERSAND (&) (subsequent options) followed by the option's value to `|resource name|`, after converting any character that is not either in the "unreserved" set, "sub-delims" set (except U+0026 AMPERSAND (&)), a U+003A COLON (:), U+0040 COMMERCIAL AT (@), U+002F SOLIDUS (/) or U+003F QUESTION MARK (?) character, to its percent-encoded form.
9. Append `|resource name|` to `|url|`.
10. Return `|url|`.

Note that these steps have been designed to lead to a URI in normal form (see Section 6.3).

7. Discovery

7.1. Service Discovery

As a part of discovering the services offered by a CoAP server, a client has to learn about the endpoint used by a server.

A server is discovered by a client by the client (knowing or) learning a URI that references a resource in the namespace of the server. Alternatively, clients can use Multicast CoAP (see Section 8) and the "All CoAP Nodes" multicast address to find CoAP servers.

Unless the port subcomponent in a "coap" or "coaps" URI indicates the UDP port at which the CoAP server is located, the server is assumed to be reachable at the default port.

The CoAP default port number 5683 MUST be supported by a server that offers resources for resource discovery (see Section 7.2 below) and SHOULD be supported for providing access to other resources. The default port number [IANA_TBD_PORT] for DTLS-secured CoAP MAY be supported by a server for resource discovery and for providing access to other resources. In addition other endpoints may be hosted at other ports, e.g. in the dynamic port space.

Implementation Note: When a CoAP server is hosted by a 6LoWPAN node, header compression efficiency is improved when it also supports a port number in the 61616-61631 compressed UDP port space defined in [RFC4944] (note that, as its UDP port differs from the default port, it is a different endpoint from the server at the default port).

7.2. Resource Discovery

The discovery of resources offered by a CoAP endpoint is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. To maximize interoperability in a CoRE environment, a CoAP endpoint SHOULD support the CoRE Link Format of discoverable resources as described in [RFC6690], except where fully manual configuration is desired. It is up to the server which resources are made discoverable (if any).

7.2.1. 'ct' Attribute

This section defines a new Web Linking [RFC5988] attribute for use with [RFC6690]. The Content-Format code "ct" attribute provides a hint about the Content-Formats this resource returns. Note that this is only a hint, and does not override the Content-Format Option of a CoAP response obtained by actually requesting the representation of the resource. The value is in the CoAP identifier code format as a decimal ASCII integer and MUST be in the range of 0-65535 (16-bit unsigned integer). For example application/xml would be indicated as "ct=41". If no Content-Format code attribute is present then nothing about the type can be assumed. The Content-Format code attribute MAY include a space-separated sequence of Content-Format codes, indicating that multiple content-formats are available. The syntax of the attribute value is summarized in the production ct-value in Figure 12, where cardinal, SP and DQUOTE are defined as in [RFC6690].

```
ct-value = cardinal
          / DQUOTE cardinal *( 1*SP cardinal ) DQUOTE
```

Figure 12

8. Multicast CoAP

CoAP supports making requests to a IP multicast group. This is defined by a series of deltas to Unicast CoAP. A more general discussion of group communication with CoAP is in [I-D.ietf-core-groupcomm].

CoAP endpoints that offer services that they want other endpoints to be able to find using multicast service discovery, join one or more of the appropriate all-CoAP-nodes multicast addresses (Section 12.8) and listen on the default CoAP port. Note that an endpoint might receive multicast requests on other multicast addresses, including the all-nodes IPv6 address (or via broadcast on IPv4); an endpoint MUST therefore be prepared to receive such messages but MAY ignore them if multicast service discovery is not desired.

8.1. Messaging Layer

A multicast request is characterized by being transported in a CoAP message that is addressed to an IP multicast address instead of a CoAP endpoint. Such multicast requests MUST be Non-confirmable.

A server SHOULD be aware that a request arrived via multicast, e.g. by making use of modern APIs such as IPV6_RECVPKTINFO [RFC3542], if available.

To avoid an implosion of error responses, when a server is aware that a request arrived via multicast, it MUST NOT return a RST in reply to NON. If it is not aware, it MAY return a RST in reply to NON as usual. Because such a Reset message will look identical to an RST for a unicast message from the sender, the sender MUST avoid using a Message ID that is also still active from this endpoint with any unicast endpoint that might receive the multicast message.

At the time of writing, multicast messages can only be carried in UDP, not in DTLS. This means that the security modes defined for CoAP in this document are not applicable to multicast.

8.2. Request/Response Layer

When a server is aware that a request arrived via multicast, the server MAY always ignore the request, in particular if it doesn't have anything useful to respond (e.g., if it only has an empty payload or an error response). The decision for this may depend on the application. (For example, in [RFC6690] query filtering, a server should not respond to a multicast request if the filter does not match. More examples are in [I-D.ietf-core-groupcomm].)

If a server does decide to respond to a multicast request, it should not respond immediately. Instead, it should pick a duration for the period of time during which it intends to respond. For purposes of this exposition, we call the length of this period the *Leisure*. The specific value of this *Leisure* may depend on the application, or MAY be derived as described below. The server SHOULD then pick a random point of time within the chosen *Leisure* period to send back the unicast response to the multicast request. If further responses need to be sent based on the same multicast address membership, a new *leisure* period starts at the earliest after the previous one finishes.

To compute a value for *Leisure*, the server should have a group size estimate *G*, a target data transfer rate *R* (which both should be chosen conservatively) and an estimated response size *S*; a rough lower bound for *Leisure* can then be computed as

$$lb_Leisure = S * G / R$$

E.g., for a multicast request with link-local scope on an 2.4 GHz IEEE 802.15.4 (6LoWPAN) network, *G* could be (relatively conservatively) set to 100, *S* to 100 bytes, and the target rate to 8 kbit/s = 1 kB/s. The resulting lower bound for the *Leisure* is 10 seconds.

If a CoAP endpoint does not have suitable data to compute a value for *Leisure*, it MAY resort to `DEFAULT_LEISURE`.

When matching a response to a multicast request, only the token MUST match; the source endpoint of the response does not need to (and will not) be the same as the destination endpoint of the original request.

For the purposes of interpreting the `Location-*` options and any links embedded in the representation and, the request URI (base URI) relative to which the response is interpreted, is formed by replacing the multicast address in the Host component of the original request URI by the literal IP address of the endpoint actually responding.

8.2.1. Caching

When a client makes a multicast request, it always makes a new request to the multicast group (since there may be new group members that joined meanwhile or ones that did not get the previous request). It MAY update a cache with the received responses. Then it uses both cached-still-fresh and 'new' responses as the result of the request.

A response received in reply to a GET request to a multicast group MAY be used to satisfy a subsequent request on the related unicast request URI. The unicast request URI is obtained by replacing the authority part of the request URI with the transport layer source address of the response message.

A cache MAY revalidate a response by making a GET request on the related unicast request URI.

A GET request to a multicast group MUST NOT contain an ETag option. A mechanism to suppress responses the client already has is left for further study.

8.2.2. Proxying

When a forward-proxy receives a request with a Proxy-Uri or URI constructed from Proxy-Scheme that indicates a multicast address, the proxy obtains a set of responses as described above and sends all responses (both cached-still-fresh and new) back to the original client.

This specification does not provide a way to indicate the unicast-modified request URI (base URI) in responses thus forwarded. Proxying multicast requests is discussed in more detail in [I-D.ietf-core-groupcomm]; one proposal to address the base URI issue can be found in section 3 of [I-D.bormann-coap-misc].

9. Securing CoAP

This section defines the DTLS binding for CoAP.

During the provisioning phase, a CoAP device is provided with the security information that it needs, including keying materials and access control lists. This specification defines provisioning for the RawPublicKey mode in Section 9.1.3.2.1. At the end of the provisioning phase, the device will be in one of four security modes with the following information for the given mode. The NoSec and RawPublicKey modes are mandatory to implement for this specification.

NoSec: There is no protocol level security (DTLS is disabled). Alternative techniques to provide lower layer security SHOULD be used when appropriate. The use of IPsec is discussed in [I-D.bormann-core-ipsec-for-coap]. Certain link layers in use with constrained nodes also provide link layer security, which may be appropriate with proper key management.

PreSharedKey: DTLS is enabled and there is a list of pre-shared keys [RFC4279] and each key includes a list of which nodes it can be used to communicate with as described in Section 9.1.3.1. At the extreme there may be one key for each node this CoAP node needs to communicate with (1:1 node/key ratio). Conversely, if more than two entities share a specific pre-shared key, this key only enables the entities to authenticate as a member of that group and not as a specific peer.

RawPublicKey: DTLS is enabled and the device has an asymmetric key pair without a certificate (a raw public key) that is validated using an out-of-band mechanism [I-D.ietf-tls-oob-pubkey] as described in Section 9.1.3.2. The device also has an identity calculated from the public key and a list of identities of the nodes it can communicate with.

Certificate: DTLS is enabled and the device has an asymmetric key pair with an X.509 certificate [RFC5280] that binds it to its Authority Name and is signed by some common trust root as described in Section 9.1.3.3. The device also has a list of root trust anchors that can be used for validating a certificate.

In the "NoSec" mode, the system simply sends the packets over normal UDP over IP and is indicated by the "coap" scheme and the CoAP default port. The system is secured only by keeping attackers from being able to send or receive packets from the network with the CoAP nodes; see Section 11.5 for an additional complication with this approach.

The other three security modes are achieved using DTLS and are indicated by the "coaps" scheme and DTLS-secured CoAP default port. The result is a security association that can be used to authenticate (within the limits of the security model) and, based on this authentication, authorize the communication partner. CoAP itself does not provide protocol primitives for authentication or authorization; where this is required, it can either be provided by communication security (i.e., IPsec or DTLS) or by object security (within the payload). Devices that require authorization for certain operations are expected to require one of these two forms of security. Necessarily, where an intermediary is involved, communication security only works when that intermediary is part of

the trust relationships; CoAP does not provide a way to forward different levels of authorization that clients may have with an intermediary to further intermediaries or origin servers -- it therefore may be required to perform all authorization at the first intermediary.

9.1. DTLS-secured CoAP

Just as HTTP is secured using Transport Layer Security (TLS) over TCP, CoAP is secured using Datagram TLS (DTLS) [RFC6347] over UDP (see Figure 13). This section defines the CoAP binding to DTLS, along with the minimal mandatory-to-implement configurations appropriate for constrained environments. The binding is defined by a series of deltas to Unicast CoAP. DTLS is in practice TLS with added features to deal with the unreliable nature of the UDP transport.

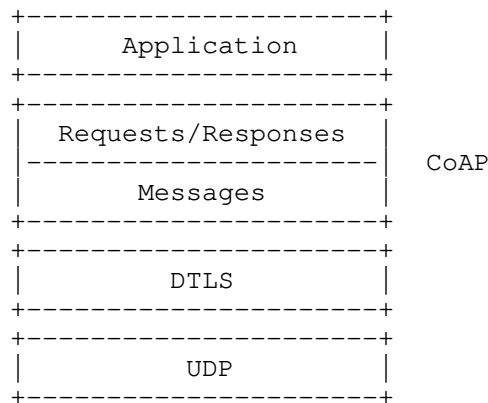


Figure 13: Abstract layering of DTLS-secured CoAP

In some constrained nodes (limited flash and/or RAM) and networks (limited bandwidth or high scalability requirements), and depending on the specific cipher suites in use, all modes of DTLS may not be applicable. Some DTLS cipher suites can add significant implementation complexity as well as some initial handshake overhead needed when setting up the security association. Once the initial handshake is completed, DTLS adds a limited per-datagram overhead of approximately 13 bytes, not including any initialization vectors/nonces (e.g., 8 bytes with TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]), integrity check values (e.g., 8 bytes with TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]) and padding required by the cipher suite. Whether and which mode of using DTLS is applicable for a CoAP-based application should be carefully weighed considering the specific cipher suites that may be applicable, and whether the session maintenance makes it

compatible with application flows and sufficient resources are available on the constrained nodes and for the added network overhead. (For some modes of using DTLS, this specification identifies a mandatory to implement cipher suite. This is an implementation requirement to maximize interoperability in those cases where these cipher suites are indeed appropriate. The specific security policies of an application may determine the actual (set of) cipher suites that can be used.) DTLS is not applicable to group keying (multicast communication); however, it may be a component in a future group key management protocol.

9.1.1. Messaging Layer

The endpoint acting as the CoAP client should also act as the DTLS client. It should initiate a session to the server on the appropriate port. When the DTLS handshake has finished, the client may initiate the first CoAP request. All CoAP messages MUST be sent as DTLS "application data".

The following rules are added for matching an ACK or RST to a CON message or a RST to a NON message: The DTLS session MUST be the same and the epoch MUST be the same.

A message is the same when it is sent within the same DTLS session and same epoch and has the same Message ID.

Note: When a Confirmable message is retransmitted, a new DTLS sequence_number is used for each attempt, even though the CoAP Message ID stays the same. So a recipient still has to perform deduplication as described in Section 4.5. Retransmissions MUST NOT be performed across epochs.

DTLS connections in RawPublicKey and Certificate mode are set up using mutual authentication so they can remain up and be reused for future message exchanges in either direction. Devices can close a DTLS connection when they need to recover resources but in general they should keep the connection up for as long as possible. Closing the DTLS connection after every CoAP message exchange is very inefficient.

9.1.2. Request/Response Layer

The following rules are added for matching a response to a request: The DTLS session MUST be the same and the epoch MUST be the same.

This means the response to a DTLS secured request MUST always be DTLS secured using the same security session and epoch. Any attempt to supply a NoSec response to a DTLS request simply does not match the

request and (unless it does match an unrelated NoSec request) therefore MUST be rejected.

9.1.3. Endpoint Identity

Devices SHOULD support the Server Name Indication (SNI) to indicate their Authority Name in the SNI HostName field as defined in Section 3 of [RFC6066]. This is needed so that when a host that acts as a virtual server for multiple Authorities receives a new DTLS connection, it knows which keys to use for the DTLS session.

9.1.3.1. Pre-Shared Keys

When forming a connection to a new node, the system selects an appropriate key based on which nodes it is trying to reach and then forms a DTLS session using a PSK (Pre-Shared Key) mode of DTLS. Implementations in these modes MUST support the mandatory to implement cipher suite TLS_PSK_WITH_AES_128_CCM_8 as specified in [RFC6655].

Depending on the commissioning model, applications may need to define an application profile for identity hints as required and detailed in [RFC4279] (Section 5.2) to enable the use of PSK identity hints.

The security considerations of [RFC4279] (Section 7) apply. In particular, applications should carefully weigh whether they need Perfect Forward Secrecy (PFS) or not and select an appropriate cipher suite (7.1). The entropy of the PSK must be sufficient to mitigate against brute-force and (where the PSK is not chosen randomly but by a human) dictionary attacks (7.2). The cleartext communication of client identities may leak data or compromise privacy (7.3).

9.1.3.2. Raw Public Key Certificates

In this mode the device has an asymmetric key pair but without an X.509 certificate (called a raw public key); e.g., the asymmetric key pair is generated by the manufacturer and installed on the device (see also Section 11.6). A device MAY be configured with multiple raw public keys. The type and length of the raw public key depends on the cipher suite used. Implementations in RawPublicKey mode MUST support the mandatory to implement cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as specified in [I-D.mcgrew-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. The key used MUST be ECDSA-capable. The curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. Implementations MUST use the Supported Elliptic Curves Extension and Supported Point Format extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090]

can be used as an implementation method. Some guidance relevant to the implementation of this cipher suite can be found in [W3CXMLSEC]. The mechanism for using raw public keys with TLS is specified in [I-D.ietf-tls-oob-pubkey].

Implementation Note: Specifically, this means the extensions listed in Figure 14 with at least the values listed will be present in the DTLS handshake.

Extension: elliptic_curves
Type: elliptic_curves (0x000a)
Length: 4
Elliptic Curves Length: 2
Elliptic curves (1 curve)
Elliptic curve: secp256r1 (0x0017)

Extension: ec_point_formats
Type: ec_point_formats (0x000b)
Length: 2
EC point formats Length: 1
Elliptic curves point formats (1)
EC point format: uncompressed (0)

Extension: signature_algorithms
Type: signature_algorithms (0x000d)
Length: 4
Data (4 bytes): 00 02 04 03
HashAlgorithm: sha256 (4)
SignatureAlgorithm: ecdsa (3)

Figure 14: DTLS extensions present for
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

9.1.3.2.1. Provisioning

The RawPublicKey mode was designed to be easily provisioned in M2M deployments. It is assumed that each device has an appropriate asymmetric public key pair installed. An identifier is calculated by the endpoint from the public key as described in Section 2 of [RFC6920]. All implementations that support checking RawPublicKey identities MUST support at least the sha-256-120 mode (SHA-256 truncated to 120 bits). Implementations SHOULD support also longer length identifiers and MAY support shorter lengths. Note that the shorter lengths provide less security against attacks and their use is NOT RECOMMENDED.

Depending on how identifiers are given to the system that verifies them, support for URI, binary, and/or human-speakable format

[RFC6920] needs to be implemented. All implementations SHOULD support the binary mode and implementations that have a user interface SHOULD also support the human-speakable format.

During provisioning, the identifier of each node is collected, for example by reading a barcode on the outside of the device or by obtaining a pre-compiled list of the identifiers. These identifiers are then installed in the corresponding endpoint, for example an M2M data collection server. The identifier is used for two purposes, to associate the endpoint with further device information and to perform access control. During (initial and ongoing) provisioning, an access control list of identifiers the device may start DTLS sessions with SHOULD also be installed and maintained.

9.1.3.3. X.509 Certificates

Implementations in Certificate Mode MUST support the mandatory to implement cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as specified in [I-D.mcgregor-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. Namely, the certificate includes a SubjectPublicKeyInfo that indicates an algorithm of id-ecPublicKey with namedCurves secp256r1 [RFC5480]; the public key format is uncompressed [RFC5480]; the hash algorithm is SHA-256; if included the key usage extension indicates digitalSignature. Certificates MUST be signed with ECDSA using secp256r1, and the signature MUST use SHA-256. The key used MUST be ECDSA-capable. The curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. Implementations MUST use the Supported Elliptic Curves Extension and Supported Point Format extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090] can be used as an implementation method.

The Authority Name in the certificate would be built out of a long term unique identifier for the device such as the EUI-64 [EUI64]. The Authority Name could also be based on the FQDN that was used as the Host part of the CoAP URI. However, the device's IP address should not typically be used as the Authority name as it would change over time. The discovery process used in the system would build up the mapping between IP addresses of the given devices and the Authority Name for each device. Some devices could have more than one Authority and would need more than a single certificate.

When a new connection is formed, the certificate from the remote device needs to be verified. If the CoAP node has a source of absolute time, then the node SHOULD check that the validity dates of the certificate are within range. The certificate MUST be validated as appropriate for the security requirements, using functionality equivalent to the algorithm specified in [RFC5280] section 6. If the

certificate contains a SubjectAltName, then the Authority Name MUST match at least one of the authority names of any CoAP URI found in a field of URI type in the SubjectAltName set. If there is no SubjectAltName in the certificate, then the Authoritative Name MUST match the CN found in the certificate using the matching rules defined in [RFC2818] with the exception that certificates with wildcards are not allowed.

CoRE support for certificate status checking requires further study. As a mapping of OCSP [RFC2560] onto CoAP is not currently defined and OCSP may also not be easily applicable in all environments, an alternative approach may be using the TLS Certificate Status Request extension ([RFC6066] section 8, also known as "OCSP stapling") or preferably the Multiple Certificate Status Extension ([I-D.ietf-tls-multiple-cert-status-extension]), if available.

If the system has a shared key in addition to the certificate, then a cipher suite that includes the shared key such as TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA [RFC5489] SHOULD be used.

10. Cross-Protocol Proxying between CoAP and HTTP

CoAP supports a limited subset of HTTP functionality, and thus cross-protocol proxying to HTTP is straightforward. There might be several reasons for proxying between CoAP and HTTP, for example when designing a web interface for use over either protocol or when realizing a CoAP-HTTP proxy. Likewise, CoAP could equally be proxied to other protocols such as XMPP [RFC6120] or SIP [RFC3264]; the definition of these mechanisms is out of scope of this specification.

There are two possible directions to access a resource via a forward-proxy:

CoAP-HTTP Proxying: Enables CoAP clients to access resources on HTTP servers through an intermediary. This is initiated by including the Proxy-Uri or Proxy-Scheme Option with an "http" or "https" URI in a CoAP request to a CoAP-HTTP proxy.

HTTP-CoAP Proxying: Enables HTTP clients to access resources on CoAP servers through an intermediary. This is initiated by specifying a "coap" or "coaps" URI in the Request-Line of an HTTP request to an HTTP-CoAP proxy.

Either way, only the Request/Response model of CoAP is mapped to HTTP. The underlying model of Confirmable or Non-confirmable messages, etc., is invisible and MUST have no effect on a proxy function. The following sections describe the handling of requests to a forward-proxy. Reverse proxies are not specified as the proxy

function is transparent to the client with the proxy acting as if it was the origin server. However, similar considerations apply to reverse-proxies as to forward-proxies, and there generally will be an expectation that reverse-proxies operate in a similar way forward-proxies would. As an implementation note, HTTP client libraries may make it hard to operate an HTTP-CoAP forward proxy by not providing a way to put a CoAP URI on the HTTP Request-Line; reverse-proxying may therefore lead to wider applicability of a proxy. A separate specification may define a convention for URIs operating such a HTTP-CoAP reverse proxy [I-D.castellani-core-http-mapping].

10.1. CoAP-HTTP Proxying

If a request contains a Proxy-Uri or Proxy-Scheme Option with an 'http' or 'https' URI [RFC2616], then the receiving CoAP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated HTTP resource and return the result to the client. (See also Section 5.7 for how the request to the proxy is formulated, including security requirements.)

This section specifies for any CoAP request the CoAP response that the proxy should return to the client. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to an HTTP origin server.

Since HTTP and CoAP share the basic set of request methods, performing a CoAP request on an HTTP resource is not so different from performing it on a CoAP resource. The meanings of the individual CoAP methods when performed on HTTP resources are explained in the subsections of this section.

If the proxy is unable or unwilling to service a request with an HTTP URI, a 5.05 (Proxying Not Supported) response is returned to the client. If the proxy services the request by interacting with a third party (such as the HTTP origin server) and is unable to obtain a result within a reasonable time frame, a 5.04 (Gateway Timeout) response is returned; if a result can be obtained but is not understood, a 5.02 (Bad Gateway) response is returned.

10.1.1. GET

The GET method requests the proxy to return a representation of the HTTP resource identified by the request URI.

Upon success, a 2.05 (Content) response code SHOULD be returned. The payload of the response MUST be a representation of the target HTTP resource, and the Content-Format Option be set accordingly. The

response MUST indicate a Max-Age value that is no greater than the remaining time the representation can be considered fresh. If the HTTP entity has an entity tag, the proxy SHOULD include an ETag Option in the response and process ETag Options in requests as described below.

A client can influence the processing of a GET request by including the following option:

Accept: The request MAY include an Accept Option, identifying the preferred response content-format.

ETag: The request MAY include one or more ETag Options, identifying responses that the client has stored. This requests the proxy to send a 2.03 (Valid) response whenever it would send a 2.05 (Content) response with an entity tag in the requested set otherwise. Note that CoAP ETags are always strong ETags in the HTTP sense; CoAP does not have the equivalent of HTTP weak ETags, and there is no good way to make use of these in a cross-proxy.

10.1.2. PUT

The PUT method requests the proxy to update or create the HTTP resource identified by the request URI with the enclosed representation.

If a new resource is created at the request URI, a 2.01 (Created) response MUST be returned to the client. If an existing resource is modified, a 2.04 (Changed) response MUST be returned to indicate successful completion of the request.

10.1.3. DELETE

The DELETE method requests the proxy to delete the HTTP resource identified by the request URI at the HTTP origin server.

A 2.02 (Deleted) response MUST be returned to client upon success or if the resource does not exist at the time of the request.

10.1.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the HTTP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 2.04 (Changed) response MUST be returned to the client. If a resource has been created on the origin server, a 2.01 (Created) response MUST be returned.

10.2. HTTP-CoAP Proxying

If an HTTP request contains a Request-URI with a 'coap' or 'coaps' URI, then the receiving HTTP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated CoAP resource and return the result to the client.

This section specifies for any HTTP request the HTTP response that the proxy should return to the client. Unless otherwise specified all the statements made are RECOMMENDED behavior; some highly constrained implementations may need to resort to shortcuts. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to a CoAP origin server. The meanings of the individual HTTP methods when performed on CoAP resources are explained in the subsections of this section.

If the proxy is unable or unwilling to service a request with a CoAP URI, a 501 (Not Implemented) response is returned to the client. If the proxy services the request by interacting with a third party (such as the CoAP origin server) and is unable to obtain a result within a reasonable time frame, a 504 (Gateway Timeout) response is returned; if a result can be obtained but is not understood, a 502 (Bad Gateway) response is returned.

10.2.1. OPTIONS and TRACE

As the OPTIONS and TRACE methods are not supported in CoAP a 501 (Not Implemented) error MUST be returned to the client.

10.2.2. GET

The GET method requests the proxy to return a representation of the CoAP resource identified by the Request-URI.

Upon success, a 200 (OK) response is returned. The payload of the response MUST be a representation of the target CoAP resource, and the Content-Type and Content-Encoding header fields be set accordingly. The response MUST indicate a max-age directive that indicates a value no greater than the remaining time the representation can be considered fresh. If the CoAP response has an ETag option, the proxy should include an ETag header field in the response.

A client can influence the processing of a GET request by including the following options:

Accept: The most preferred Media-type of the HTTP Accept header field in a request is mapped to a CoAP Accept option. HTTP Accept Media-type ranges, parameters and extensions are not supported by the CoAP Accept option. If the proxy cannot send a response which is acceptable according to the combined Accept field value, then the proxy sends a 406 (not acceptable) response. The proxy MAY then retry the request with further Media-types from the HTTP Accept header field.

Conditional GETs: Conditional HTTP GET requests that include an "If-Match" or "If-None-Match" request-header field can be mapped to a corresponding CoAP request. The "If-Modified-Since" and "If-Unmodified-Since" request-header fields are not directly supported by CoAP, but are implemented locally by a caching proxy.

10.2.3. HEAD

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response.

Although there is no direct equivalent of HTTP's HEAD method in CoAP, an HTTP-CoAP proxy responds to HEAD requests for CoAP resources, and the HTTP headers are returned without a message-body.

Implementation Note: An HTTP-CoAP proxy may want to try using a block-wise transfer [I-D.ietf-core-block] option to minimize the amount of data actually transferred, but needs to be prepared for the case that the origin server does not support block-wise transfers.

10.2.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the CoAP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 200 (OK) or 204 (No Content) response MUST be returned to the client. If a resource has been created on the origin server, a 201 (Created) response MUST be returned.

If any of the Location-* Options are present in the CoAP response, a Location header field constructed from the values of these options is returned.

10.2.5. PUT

The PUT method requests the proxy to update or create the CoAP resource identified by the Request-URI with the enclosed representation.

If a new resource is created at the Request-URI, a 201 (Created) response is returned to the client. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes is sent to indicate successful completion of the request.

10.2.6. DELETE

The DELETE method requests the proxy to delete the CoAP resource identified by the Request-URI at the CoAP origin server.

A successful response is 200 (OK) if the response includes an entity describing the status or 204 (No Content) if the action has been enacted but the response does not include an entity.

10.2.7. CONNECT

This method can not currently be satisfied by an HTTP-CoAP proxy function as TLS to DTLS tunneling has not yet been specified. For now, a 501 (Not Implemented) error is returned to the client.

11. Security Considerations

This section analyzes the possible threats to the protocol. It is meant to inform protocol and application developers about the security limitations of CoAP as described in this document. As CoAP realizes a subset of the features in HTTP/1.1, the security considerations in Section 15 of [RFC2616] are also pertinent to CoAP. This section concentrates on describing limitations specific to CoAP.

11.1. Protocol Parsing, Processing URIs

A network-facing application can exhibit vulnerabilities in its processing logic for incoming packets. Complex parsers are well-known as a likely source of such vulnerabilities, such as the ability to remotely crash a node, or even remotely execute arbitrary code on it. CoAP attempts to narrow the opportunities for introducing such vulnerabilities by reducing parser complexity, by giving the entire range of encodable values a meaning where possible, and by

aggressively reducing complexity that is often caused by unnecessary choice between multiple representations that mean the same thing. Much of the URI processing has been moved to the clients, further reducing the opportunities for introducing vulnerabilities into the servers. Even so, the URI processing code in CoAP implementations is likely to be a large source of remaining vulnerabilities and should be implemented with special care. CoAP access control implementations need to ensure they don't introduce vulnerabilities through discrepancies between the code deriving access control decisions from a URI and the code finally serving up the resource addressed by the URI. The most complex parser remaining could be the one for the CoRE Link Format, although this also has been designed with a goal of reduced implementation complexity [RFC6690]. (See also section 15.2 of [RFC2616].)

11.2. Proxying and Caching

As mentioned in 15.7 of [RFC2616], proxies are by their very nature men-in-the-middle, breaking any IPsec or DTLS protection that a direct CoAP message exchange might have. They are therefore interesting targets for breaking confidentiality or integrity of CoAP message exchanges. As noted in [RFC2616], they are also interesting targets for breaking availability.

The threat to confidentiality and integrity of request/response data is amplified where proxies also cache. Note that CoAP does not define any of the cache-suppressing Cache-Control options that HTTP/1.1 provides to better protect sensitive data.

For a caching implementation, any access control considerations that would apply to making the request that generated the cache entry also need to be applied to the value in the cache. This is relevant for clients that implement multiple security domains, as well as for proxies that may serve multiple clients. Also, a caching proxy **MUST NOT** make cached values available to requests that have lesser transport security properties than to which it would make available the process of forwarding the request in the first place.

Unlike the "coap" scheme, responses to "coaps" identified requests are never "public" and thus **MUST NOT** be reused for shared caching unless the cache is able to make equivalent access control decisions to the ones that led to the cached entry. They can, however, be reused in a private cache if the message is cacheable by default in CoAP.

Finally, a proxy that fans out Separate Responses (as opposed to Piggy-backed Responses) to multiple original requesters may provide additional amplification (see Section 11.3).

11.3. Risk of amplification

CoAP servers generally reply to a request packet with a response packet. This response packet may be significantly larger than the request packet. An attacker might use CoAP nodes to turn a small attack packet into a larger attack packet, an approach known as amplification. There is therefore a danger that CoAP nodes could become implicated in denial of service (DoS) attacks by using the amplifying properties of the protocol: An attacker that is attempting to overload a victim but is limited in the amount of traffic it can generate, can use amplification to generate a larger amount of traffic.

This is particularly a problem in nodes that enable NoSec access, that are accessible from an attacker and can access potential victims (e.g. on the general Internet), as the UDP protocol provides no way to verify the source address given in the request packet. An attacker need only place the IP address of the victim in the source address of a suitable request packet to generate a larger packet directed at the victim.

As a mitigating factor, many constrained networks will only be able to generate a small amount of traffic, which may make CoAP nodes less attractive for this attack. However, the limited capacity of the constrained network makes the network itself a likely victim of an amplification attack.

Therefore, large amplification factors SHOULD NOT be provided in the response if the request is not authenticated. A CoAP server can reduce the amount of amplification it provides to an attacker by using slicing/blocking modes of CoAP [I-D.ietf-core-block] and offering large resource representations only in relatively small slices. E.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

CoAP also supports the use of multicast IP addresses in requests, an important requirement for M2M. Multicast CoAP requests may be the source of accidental or deliberate denial of service attacks, especially over constrained networks. This specification attempts to reduce the amplification effects of multicast requests by limiting when a response is returned. To limit the possibility of malicious use, CoAP servers SHOULD NOT accept multicast requests that can not be authenticated in some way, cryptographically or by some multicast boundary limiting the potential sources. If possible a CoAP server SHOULD limit the support for multicast requests to the specific resources where the feature is required.

On some general purpose operating systems providing a Posix-style API, it is not straightforward to find out whether a packet received was addressed to a multicast address. While many implementations will know whether they have joined a multicast group, this creates a problem for packets addressed to multicast addresses of the form FF0x::1, which are received by every IPv6 node. Implementations SHOULD make use of modern APIs such as IPV6_RECVPKTINFO [RFC3542], if available, to make this determination.

11.4. IP Address Spoofing Attacks

Due to the lack of a handshake in UDP, a rogue endpoint which is free to read and write messages carried by the constrained network (i.e. NoSec or PreSharedKey deployments with nodes/key ratio > 1:1), may easily attack a single endpoint, a group of endpoints, as well as the whole network e.g. by:

1. spoofing RST in response to a CON or NON message, thus making an endpoint "deaf"; or
2. spoofing an ACK in response to a CON message, thus potentially preventing the sender of the CON message from retransmitting, and drowning out the actual response; or
3. spoofing the entire response with forged payload/options (this has different levels of impact: from single response disruption, to much bolder attacks on the supporting infrastructure, e.g. poisoning proxy caches, or tricking validation / lookup interfaces in resource directories and, more generally, any component that stores global network state and uses CoAP as the messaging facility to handle state set/update's is a potential target.); or
4. spoofing a multicast request for a target node which may result in both network congestion/collapse and victim DoS'ing / forced wakeup from sleeping; or
5. spoofing observe messages, etc.

Response spoofing by off-path attackers can be detected and mitigated even without transport layer security by choosing a non-trivial, randomized token in the request (Section 5.3.1). [RFC4086] discusses randomness requirements for security.

In principle, other kinds of spoofing can be detected by CoAP only in case CON semantics is used, because of unexpected ACK/RSTs coming from the deceived endpoint. But this imposes keeping track of the used Message IDs which is not always possible, and moreover detection

becomes available usually after the damage is already done. This kind of attack can be prevented using security modes other than NoSec.

With or without source address spoofing, a client can attempt to overload a server by sending requests, preferably complex ones, to a server; address spoofing makes tracing back, and blocking, this attack harder. Given that the cost of a CON request is small, this attack can easily be executed. Under this attack, a constrained node with limited total energy available may exhaust that energy much more quickly than planned (battery depletion attack). Also, if the client uses a Confirmable message and the server responds with a Confirmable separate response to a (possibly spoofed) address that does not respond, the server will have to allocate buffer and retransmission logic for each response up to the exhaustion of MAX_TRANSMIT_SPAN, making it more likely that it runs out of resources for processing legitimate traffic. The latter problem can be mitigated somewhat by limiting the rate of responses as discussed in Section 4.7. An attacker could also spoof the address of a legitimate client, which, if the server uses separate responses, might block legitimate responses to that client because of NSTART=1. All these attacks can be prevented using a security mode other than NoSec, leaving only attacks on the security protocol.

11.5. Cross-Protocol Attacks

The ability to incite a CoAP endpoint to send packets to a fake source address can be used not only for amplification, but also for cross-protocol attacks against a victim listening to UDP packets at a given address (IP address and port):

- o the attacker sends a message to a CoAP endpoint with the given address as the fake source address,
- o the CoAP endpoint replies with a message to the given source address,
- o the victim at the given address receives a UDP packet that it interprets according to the rules of a different protocol.

This may be used to circumvent firewall rules that prevent direct communication from the attacker to the victim, but happen to allow communication from the CoAP endpoint (which may also host a valid role in the other protocol) to the victim.

Also, CoAP endpoints may be the victim of a cross-protocol attack generated through an endpoint of another UDP-based protocol such as DNS. In both cases, attacks are possible if the security properties

of the endpoints rely on checking IP addresses (and firewalling off direct attacks sent from outside using fake IP addresses). In general, because of their lack of context, UDP-based protocols are relatively easy targets for cross-protocol attacks.

Finally, CoAP URIs transported by other means could be used to incite clients to send messages to endpoints of other protocols.

One mitigation against cross-protocol attacks is strict checking of the syntax of packets received, combined with sufficient difference in syntax. As an example, it might help if it were difficult to incite a DNS server to send a DNS response that would pass the checks of a CoAP endpoint. Unfortunately, the first two bytes of a DNS reply are an ID that can be chosen by the attacker, which map into the interesting part of the CoAP header, and the next two bytes are then interpreted as CoAP's Message ID (i.e., any value is acceptable). The DNS count words may be interpreted as multiple instances of a (non-existent, but elective) CoAP option 0, or possibly as a Token. The echoed query finally may be manufactured by the attacker to achieve a desired effect on the CoAP endpoint; the response added by the server (if any) might then just be interpreted as added payload.

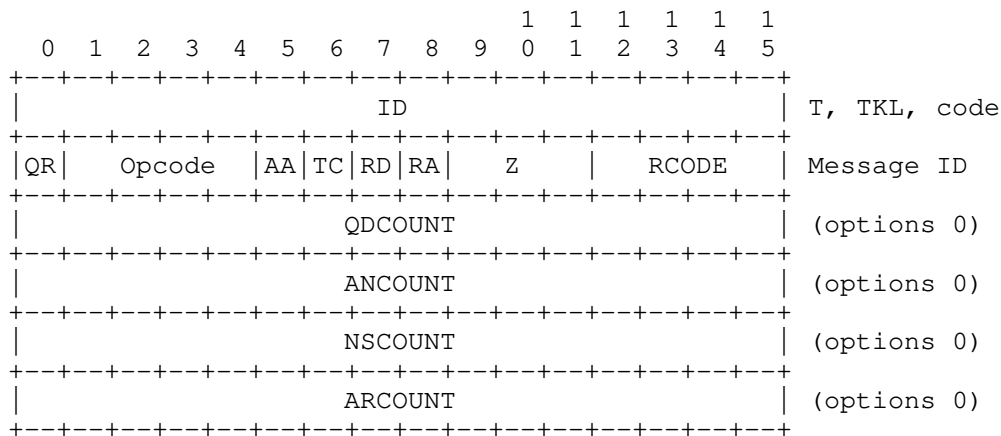


Figure 15: DNS Header vs. CoAP Message

In general, for any pair of protocols, one of the protocols can very well have been designed in a way that enables an attacker to cause the generation of replies that look like messages of the other protocol. It is often much harder to ensure or prove the absence of viable attacks than to generate examples that may not yet completely enable an attack but might be further developed by more creative minds. Cross-protocol attacks can therefore only be completely

mitigated if endpoints don't authorize actions desired by an attacker just based on trusting the source IP address of a packet. Conversely, a NoSec environment that completely relies on a firewall for CoAP security not only needs to firewall off the CoAP endpoints but also all other endpoints that might be incited to send UDP messages to CoAP endpoints using some other UDP-based protocol.

In addition to the considerations above, the security considerations for DTLS with respect to cross-protocol attacks apply. E.g., if the same DTLS security association ("connection") is used to carry data of multiple protocols, DTLS no longer provides protection against cross-protocol attacks between these protocols.

11.6. Constrained node considerations

Implementers on constrained nodes often find themselves without a good source of entropy [RFC4086]. If that is the case, the node **MUST** NOT be used for processes that require good entropy, such as key generation. Instead, keys should be generated externally and added to the device during manufacturing or commissioning.

Due to their low processing power, constrained nodes are particularly susceptible to timing attacks. Special care must be taken in implementation of cryptographic primitives.

Large numbers of constrained nodes will be installed in exposed environments and will have little resistance to tampering, including recovery of keying materials. This needs to be considered when defining the scope of credentials assigned to them. In particular, assigning a shared key to a group of nodes may make any single constrained node a target for subverting the entire group.

12. IANA Considerations

12.1. CoAP Code Registries

This document defines two sub-registries for the values of the Code field in the CoAP header within the Constrained RESTful Environments (CoRE) Parameters ("CoRE Parameters") registry.

Values in the two sub-registries are eight-bit values notated as three decimal digits c.dd separated by a period between the first and the second digit; the first digit c is between 0 and 7 and denotes the code class; the second and third digit dd denote a decimal number between 00 and 31 for the detail.

All Code values are assigned by sub-registries according to the following ranges:

0.00 Indicates an Empty message (see Section 4.1).

0.01-0.31 Indicates a request. Values in this range are assigned by the "CoAP Method Codes" sub-registry (see Section 12.1.1).

1.00-1.31 Reserved

2.00-5.31 Indicates a response. Values in this range are assigned by the "CoAP Response Codes" sub-registry (see Section 12.1.2).

6.00-7.31 Reserved

12.1.1. Method Codes

The name of the sub-registry is "CoAP Method Codes".

Each entry in the sub-registry must include the Method Code in the range 0.01-0.31, the name of the method, and a reference to the method's documentation.

Initial entries in this sub-registry are as follows:

Code	Name	Reference
0.01	GET	[RFCXXXX]
0.02	POST	[RFCXXXX]
0.03	PUT	[RFCXXXX]
0.04	DELETE	[RFCXXXX]

Table 5: CoAP Method Codes

All other Method Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG approval" as described in [RFC5226].

The documentation of a method code should specify the semantics of a request with that code, including the following properties:

- o The response codes the method returns in the success case.
- o Whether the method is idempotent, safe, or both.

12.1.2. Response Codes

The name of the sub-registry is "CoAP Response Codes".

Each entry in the sub-registry must include the Response Code in the range 2.00–5.31, a description of the Response Code, and a reference to the Response Code's documentation.

Initial entries in this sub-registry are as follows:

Code	Description	Reference
2.01	Created	[RFCXXXX]
2.02	Deleted	[RFCXXXX]
2.03	Valid	[RFCXXXX]
2.04	Changed	[RFCXXXX]
2.05	Content	[RFCXXXX]
4.00	Bad Request	[RFCXXXX]
4.01	Unauthorized	[RFCXXXX]
4.02	Bad Option	[RFCXXXX]
4.03	Forbidden	[RFCXXXX]
4.04	Not Found	[RFCXXXX]
4.05	Method Not Allowed	[RFCXXXX]
4.06	Not Acceptable	[RFCXXXX]
4.12	Precondition Failed	[RFCXXXX]
4.13	Request Entity Too Large	[RFCXXXX]
4.15	Unsupported Content-Format	[RFCXXXX]
5.00	Internal Server Error	[RFCXXXX]
5.01	Not Implemented	[RFCXXXX]
5.02	Bad Gateway	[RFCXXXX]
5.03	Service Unavailable	[RFCXXXX]
5.04	Gateway Timeout	[RFCXXXX]
5.05	Proxying Not Supported	[RFCXXXX]

Table 6: CoAP Response Codes

The Response Codes 3.00–3.31 are Reserved for future use. All other Response Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG approval" as described in [RFC5226].

The documentation of a response code should specify the semantics of a response with that code, including the following properties:

- o The methods the response code applies to.
- o Whether payload is required, optional or not allowed.

- o The semantics of the payload. For example, the payload of a 2.05 (Content) response is a representation of the target resource; the payload in an error response is a human-readable diagnostic payload.
- o The format of the payload. For example, the format in a 2.05 (Content) response is indicated by the Content-Format Option; the format of the payload in an error response is always Net-Unicode text.
- o Whether the response is cacheable according to the freshness model.
- o Whether the response is validatable according to the validation model.
- o Whether the response causes a cache to mark responses stored for the request URI as not fresh.

12.2. Option Number Registry

This document defines a sub-registry for the Option Numbers used in CoAP options within the "CoRE Parameters" registry. The name of the sub-registry is "CoAP Option Numbers".

Each entry in the sub-registry must include the Option Number, the name of the option and a reference to the option's documentation.

Initial entries in this sub-registry are as follows:

Number	Name	Reference
0	(Reserved)	[RFCXXXX]
1	If-Match	[RFCXXXX]
3	Uri-Host	[RFCXXXX]
4	ETag	[RFCXXXX]
5	If-None-Match	[RFCXXXX]
7	Uri-Port	[RFCXXXX]
8	Location-Path	[RFCXXXX]
11	Uri-Path	[RFCXXXX]
12	Content-Format	[RFCXXXX]
14	Max-Age	[RFCXXXX]
15	Uri-Query	[RFCXXXX]
17	Accept	[RFCXXXX]
20	Location-Query	[RFCXXXX]
35	Proxy-Uri	[RFCXXXX]
39	Proxy-Scheme	[RFCXXXX]

60	Size1	[RFCXXXX]
128	(Reserved)	[RFCXXXX]
132	(Reserved)	[RFCXXXX]
136	(Reserved)	[RFCXXXX]
140	(Reserved)	[RFCXXXX]

Table 7: CoAP Option Numbers

The IANA policy for future additions to this sub-registry is split into three tiers as follows. The range of 0..255 is reserved for options defined by the IETF (IETF Review or IESG approval). The range of 256..2047 is reserved for commonly used options with public specifications (Specification Required). The range of 2048..64999 is for all other options including private or vendor specific ones, which undergo a Designated Expert review to help ensure that the option semantics are defined correctly. The option numbers between 65000 and 65535 inclusive are reserved for experiments. They are not meant for vendor specific use of any kind and MUST NOT be used in operational deployments.

Option Number	Policy [RFC5226]
0..255	IETF Review or IESG approval
256..2047	Specification Required
2048..64999	Designated Expert
65000..65535	Reserved for experiments

Table 8: CoAP Option Number Registry Policy

The documentation of an Option Number should specify the semantics of an option with that number, including the following properties:

- o The meaning of the option in a request.
- o The meaning of the option in a response.
- o Whether the option is critical or elective, as determined by the Option Number.
- o Whether the option is Safe-to-Forward, and, if yes, whether it is part of the Cache-Key, as determined by the Option Number (see Section 5.4.2).
- o The format and length of the option's value.

- o Whether the option must occur at most once or whether it can occur multiple times.
- o The default value, if any. For a critical option with a default value, a discussion on how the default value enables processing by implementations not implementing the critical option (Section 5.4.4).

12.3. Content-Format Registry

Internet media types are identified by a string, such as "application/xml" [RFC2046]. In order to minimize the overhead of using these media types to indicate the format of payloads, this document defines a sub-registry for a subset of Internet media types to be used in CoAP and assigns each, in combination with a content-coding, a numeric identifier. The name of the sub-registry is "CoAP Content-Formats", within the "CoRE Parameters" registry.

Each entry in the sub-registry must include the media type registered with IANA, the numeric identifier in the range 0-65535 to be used for that media type in CoAP, the content-coding associated with this identifier, and a reference to a document describing what a payload with that media type means semantically.

CoAP does not include a separate way to convey content-encoding information with a request or response, and for that reason the content-encoding is also specified for each identifier (if any). If multiple content-encodings will be used with a media type, then a separate Content-Format identifier for each is to be registered. Similarly, other parameters related to an Internet media type, such as level, can be defined for a CoAP Content-Format entry.

Initial entries in this sub-registry are as follows:

Media type	Encoding	Id.	Reference
text/plain; charset=utf-8	-	0	[RFC2046] [RFC3676] [RFC5147]
application/ link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/ octet-stream	-	42	[RFC2045] [RFC2046]
application/exi	-	47	[EXIMIME]
application/json	-	50	[RFC4627]

Table 9: CoAP Content-Formats

The identifiers between 65000 and 65535 inclusive are reserved for experiments. They are not meant for vendor specific use of any kind and MUST NOT be used in operational deployments. The identifiers between 256 and 9999 are reserved for future use in IETF specifications (IETF review or IESG approval). All other identifiers are Unassigned.

Because the name space of single-byte identifiers is so small, the IANA policy for future additions in the range 0-255 inclusive to the sub-registry is "Expert Review" as described in [RFC5226]. The IANA policy for additions in the range 10000-64999 inclusive is "First Come First Served" as described in [RFC5226].

In machine to machine applications, it is not expected that generic Internet media types such as text/plain, application/xml or application/octet-stream are useful for real applications in the long term. It is recommended that M2M applications making use of CoAP will request new Internet media types from IANA indicating semantic information about how to create or parse a payload. For example, a Smart Energy application payload carried as XML might request a more specific type like application/se+xml or application/se-exi.

12.4. URI Scheme Registration

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap". The registration request complies with [RFC4395].

URI scheme name.
coap

Status.

Permanent.

URI scheme syntax.

Defined in Section 6.1 of [RFCXXXX].

URI scheme semantics.

The "coap" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP). The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "http" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.

The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.

The scheme is used by CoAP endpoints to access CoAP resources.

Interoperability considerations.

None.

Security considerations.

See Section 11.1 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

References.

[RFCXXXX]

12.5. Secure URI Scheme Registration

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coaps". The registration request complies with [RFC4395].

URI scheme name.

coaps

Status.

Permanent.

URI scheme syntax.

Defined in Section 6.2 of [RFCXXXX].

URI scheme semantics.

The "coaps" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using Datagram Transport Layer Security (DTLS) for transport security. The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "https" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.

The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.

The scheme is used by CoAP endpoints to access CoAP resources using DTLS.

Interoperability considerations.

None.

Security considerations.

See Section 11.1 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

References.

[RFCXXXX]

12.6. Service Name and Port Number Registration

One of the functions of CoAP is resource discovery: a CoAP client can ask a CoAP server about the resources offered by it (see Section 7). To enable resource discovery just based on the knowledge of an IP address, the CoAP port for resource discovery needs to be standardized.

IANA has assigned the port number 5683 and the service name "coap", in accordance with [RFC6335].

Besides unicast, CoAP can be used with both multicast and anycast.

Service Name.

coap

Transport Protocol.

UDP

Assignee.

IESG <iesg@ietf.org>

Contact.

IETF Chair <chair@ietf.org>

Description.

Constrained Application Protocol (CoAP)

Reference.

[RFCXXXX]

Port Number.

5683

12.7. Secure Service Name and Port Number Registration

CoAP resource discovery may also be provided using the DTLS-secured CoAP "coaps" scheme. Thus the CoAP port for secure resource discovery needs to be standardized.

This document requests the assignment of the port number [IANA_TBD_PORT] and the service name "coaps", in accordance with [RFC6335].

Besides unicast, DTLS-secured CoAP can be used with anycast.

Service Name.

coaps

Transport Protocol.

UDP

Assignee.

IESG <iesg@ietf.org>

Contact.

IETF Chair <chair@ietf.org>

Description.

DTLS-secured CoAP

Reference.

[RFCXXXX]

Port Number.

[IANA_TBD_PORT]

12.8. Multicast Address Registration

Section 8, "Multicast CoAP", defines the use of multicast. This document requests the assignment of the following multicast addresses for use by CoAP nodes:

IPv4 -- "All CoAP Nodes" address [TBD1], from the IPv4 Multicast Address Space Registry. As the address is used for discovery that may span beyond a single network, it should come from the Internetwork Control Block (224.0.1.x, RFC 5771).

IPv6 -- "All CoAP Nodes" address [TBD2], from the IPv6 Multicast Address Space Registry, in the Variable Scope Multicast Addresses space (RFC3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only. The address should be of the form FF0x::nn, where nn is a single byte, to ensure good compression of the local-scope address with [RFC6282].

[The explanatory text to be removed upon allocation of the addresses, except for the note about the distinct multicast addresses.]

13. Acknowledgements

Brian Frank was a contributor to and co-author of previous drafts of this specification.

Special thanks to Peter Bigot, Esko Dijk and Cullen Jennings for substantial contributions to the ideas and text in the document, along with countless detailed reviews and discussions.

Thanks to Ed Beroaset, Angelo P. Castellani, Gilbert Clark, Robert Cragie, Esko Dijk, Lisa Dusseault, Mehmet Ersue, Thomas Fossati, Tom Herbst, Richard Kelsey, Ari Keranen, Matthias Kovatsch, Salvatore Loreto, Kerry Lynn, Alexey Melnikov, Guido Moritz, Petri Mutka, Colin O'Flynn, Charles Palmer, Adriano Pezzuto, Robert Quattlebaum, Akbar Rahman, Eric Rescorla, Dan Romascanu, David Ryan, Szymon Sasin, Michael Scharf, Dale Seed, Robby Simpson, Peter van der Stok, Michael Stuber, Linyi Tian, Gilman Tolle, Matthieu Vial and Alper Yegin for helpful comments and discussions that have shaped the document.

Special thanks also to the IESG reviewers, Adrian Farrel, Martin Stiernerling, Pete Resnick, Richard Barnes, Sean Turner, Spencer Dawkins, Stephen Farrell, and Ted Lemon, who contributed in-depth reviews.

Some of the text has been borrowed from the working documents of the IETF httpbis working group.

14. References

14.1. Normative References

- [I-D.ietf-tls-oob-pubkey]
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-07 (work in progress), February 2013.
- [I-D.mcgreg-tls-aes-ccm-ecc]
McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM ECC Cipher Suites for TLS", draft-mcgreg-tls-aes-ccm-ecc-06 (work in progress), February 2013.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.
- [RFC2045] Freed, N. and N.S. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

- [RFC3676] Gellens, R., "The Text/Plain Format and DelSp Parameters", RFC 3676, February 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.
- [RFC5147] Wilde, E. and M. Duerst, "URI Fragment Identifiers for the text/plain Media Type", RFC 5147, April 2008.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, April 2013.

14.2. Informative References

- [EUI64] , "GUIDELINES FOR 64-BIT GLOBAL IDENTIFIER (EUI-64) REGISTRATION AUTHORITY", April 2010, <<http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>>.
- [EXIMIME] , "Efficient XML Interchange (EXI) Format 1.0", December 2009, <<http://www.w3.org/TR/2009/CR-exi-20091208/#mediaTypeRegistration>>.
- [HHGTTG] Adams, D., "The Hitchhiker's Guide to the Galaxy", October 1979.
- [I-D.allman-tcpm-rto-consider]
Allman, M., "Retransmission Timeout Considerations", draft-allman-tcpm-rto-consider-01 (work in progress), May 2012.
- [I-D.bormann-coap-misc]
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-22 (work in progress), December 2012.
- [I-D.bormann-core-ipsec-for-coap]
Bormann, C., "Using CoAP with IPsec", draft-bormann-core-ipsec-for-coap-00 (work in progress), December 2012.
- [I-D.castellani-core-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-http-mapping-07 (work in progress), February 2013.
- [I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-10 (work in progress), October 2012.

[I-D.ietf-core-groupcomm]

Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-06 (work in progress), April 2013.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and A. Keraenen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-04 (work in progress), April 2013.

[I-D.ietf-tls-multiple-cert-status-extension]

Pettersen, Y., "The TLS Multiple Certificate Status Request Extension", draft-ietf-tls-multiple-cert-status-extension-08 (work in progress), April 2013.

[REST]

Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

- [RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.
- [RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, July 2004.
- [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.
- [RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.
- [RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.
- [RFC5489] Badra, M. and I. Hajjeh, "ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)", RFC 5489, March 2009.
- [RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, September 2011.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA)

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July 2012.

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, April 2013.

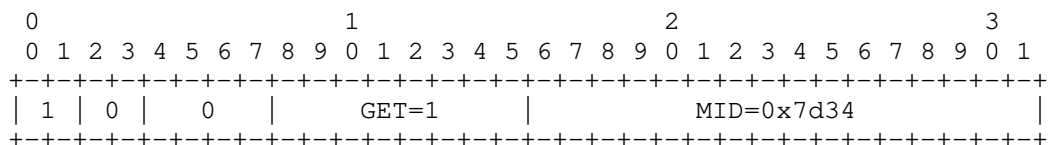
[W3CXMLSEC] Wenning, R., "Report of the XML Security PAG", October 2012, <<http://www.w3.org/2011/xmlsec-pag/pagreport.html>>.

This section gives a number of short examples with message flows for GET requests. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and multicast.

```

Client  Server
|       |
|       |
+----->   Header: GET (T=CON, Code=0.01, MID=0x7d34)
GET         Uri-Path: "temperature"
|       |
|       |
<-----+   Header: 2.05 Content (T=ACK, Code=2.05, MID=0x7d34)
2.05       Payload: "22.3 C"
|       |
|       |

```



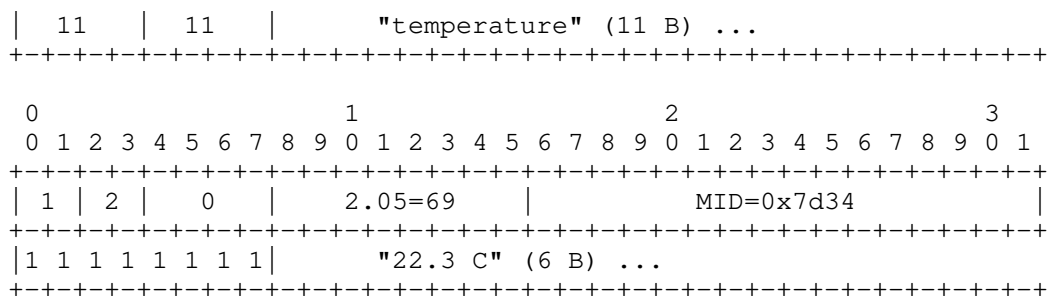
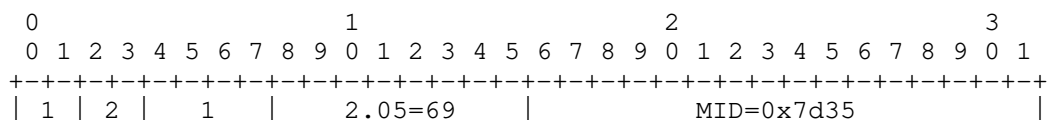
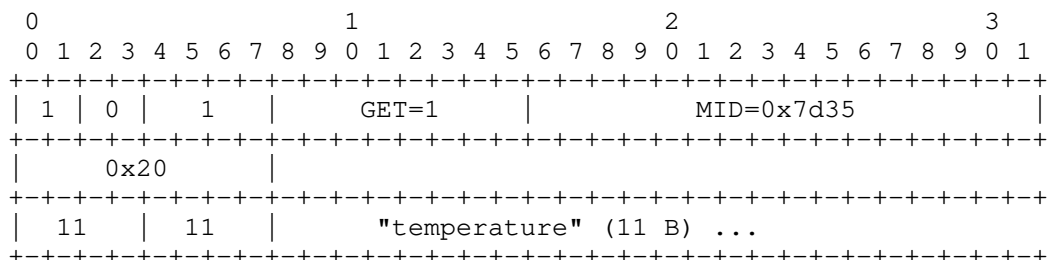
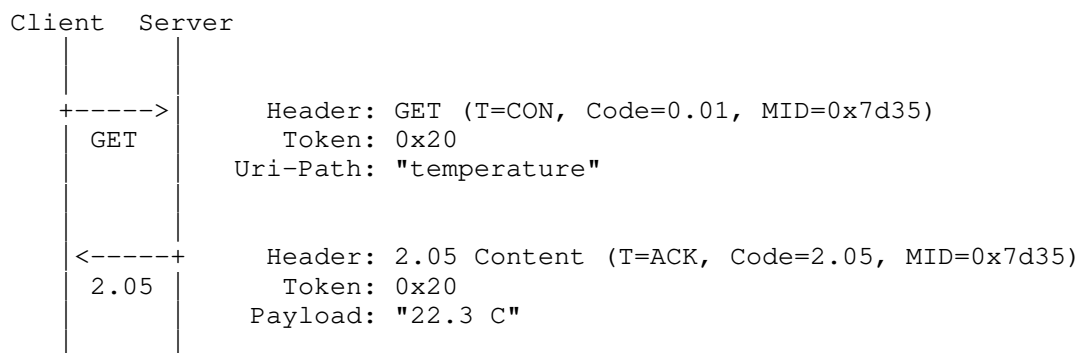


Figure 16: Confirmable request; piggy-backed response

Figure 17 shows a similar example, but with the inclusion of a non-empty Token (Value 0x20) in the request and the response, increasing the sizes to 17 and 12 bytes, respectively.



```

+-----+
| 0x20 |
+-----+
| 1 1 1 1 1 1 1 | "22.3 C" (6 B) ...
+-----+

```

Figure 17: Confirmable request; piggy-backed response

In Figure 18, the Confirmable GET request is lost. After ACK_TIMEOUT seconds, the client retransmits the request, resulting in a piggy-backed response as in the previous example.

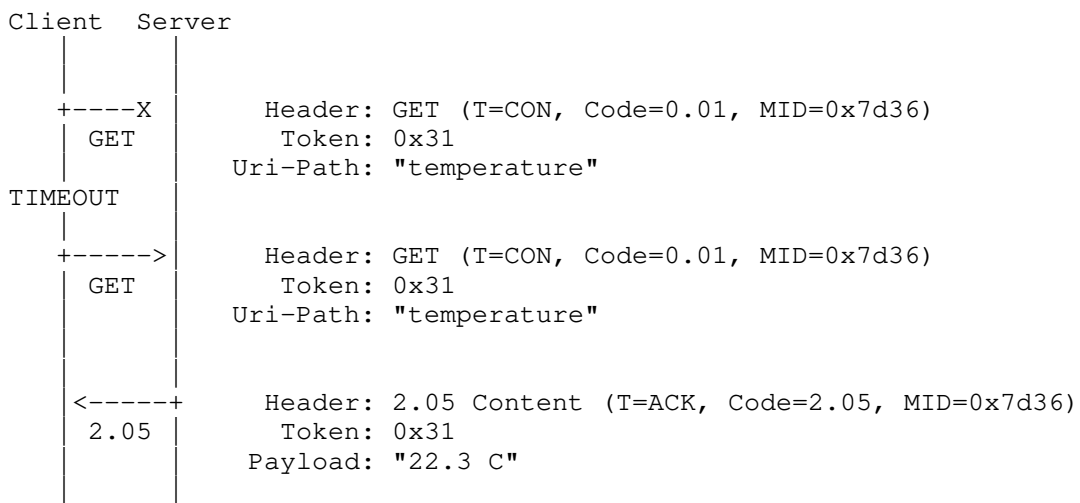
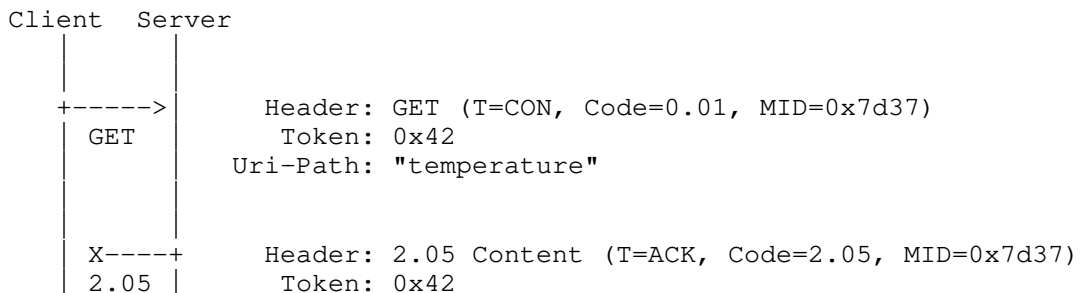


Figure 18: Confirmable request (retransmitted); piggy-backed response

In Figure 19, the first Acknowledgement message from the server to the client is lost. After ACK_TIMEOUT seconds, the client retransmits the request.



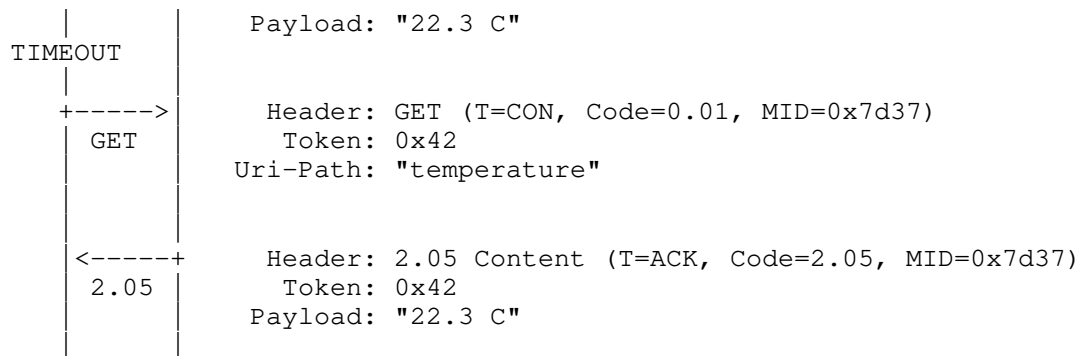


Figure 19: Confirmable request; piggy-backed response (retransmitted)

In Figure 20, the server acknowledges the Confirmable request and sends a 2.05 (Content) response separately in a Confirmable message. Note that the Acknowledgement message and the Confirmable response do not necessarily arrive in the same order as they were sent. The client acknowledges the Confirmable response.

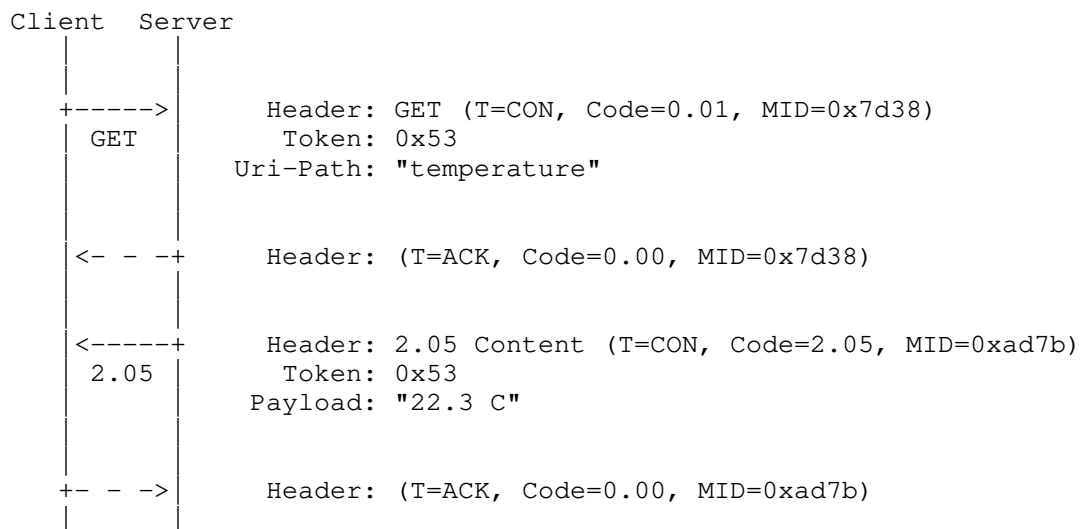


Figure 20: Confirmable request; separate response

Figure 21 shows an example where the client loses its state (e.g., crashes and is rebooted) right after sending a Confirmable request, so the separate response arriving some time later comes unexpected. In this case, the client rejects the Confirmable response with a Reset message. Note that the unexpected ACK is silently ignored.

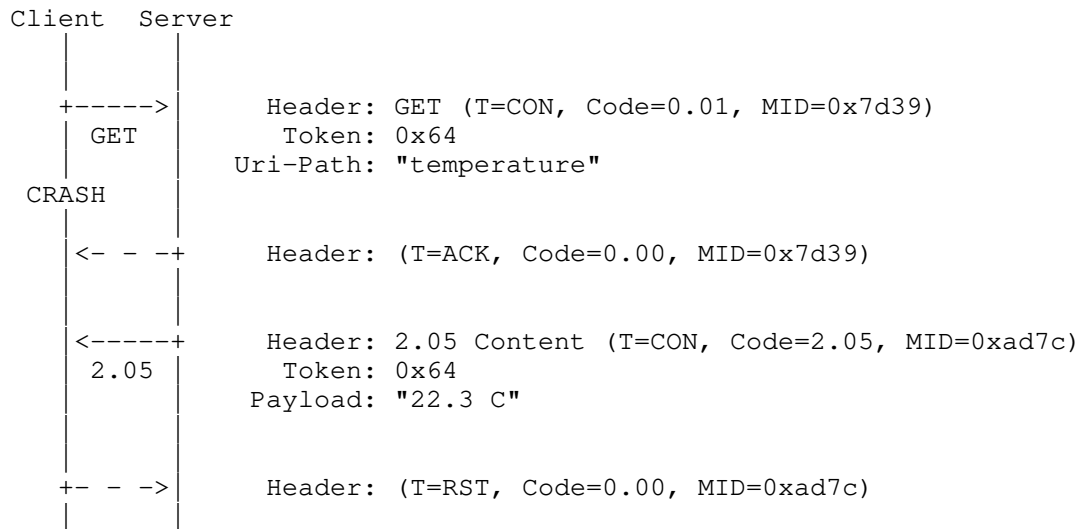


Figure 21: Confirmable request; separate response (unexpected)

Figure 22 shows a basic GET request where the request and the response are Non-confirmable, so both may be lost without notice.

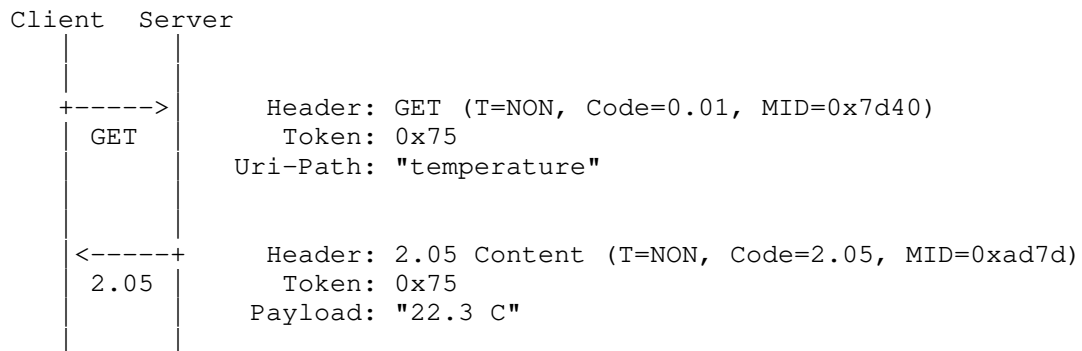


Figure 22: Non-confirmable request; Non-confirmable response

In Figure 23, the client sends a Non-confirmable GET request to a multicast address: all nodes in link-local scope. There are 3 servers on the link: A, B and C. Servers A and B have a matching resource, therefore they send back a Non-confirmable 2.05 (Content) response. The response sent by B is lost. C does not have matching response, therefore it sends a Non-confirmable 4.04 (Not Found) response.

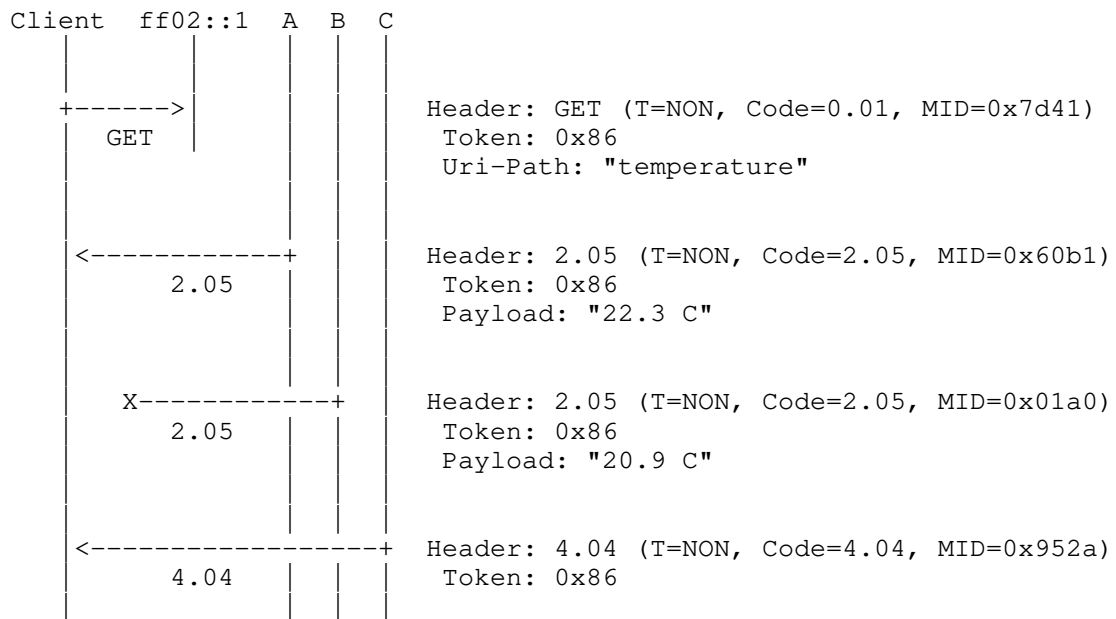


Figure 23: Non-confirmable request (multicast); Non-confirmable response

Appendix B. URI Examples

The following examples demonstrate different sets of Uri options, and the result after constructing an URI from them. In addition to the options, Section 6.5 refers to the destination IP address and port, but not all paths of the algorithm cause the destination IP address and port to be included in the URI.

o Input:

Destination IP Address = [2001:db8::2:1]
Destination UDP Port = 5683

Output:

coap://[2001:db8::2:1]/

o Input:

Destination IP Address = [2001:db8::2:1]
Destination UDP Port = 5683
Uri-Host = "example.net"

Output:

```
coap://example.net/
```

o Input:

```
Destination IP Address = [2001:db8::2:1]
Destination UDP Port = 5683
Uri-Host = "example.net"
Uri-Path = ".well-known"
Uri-Path = "core"
```

Output:

```
coap://example.net/.well-known/core
```

o Input:

```
Destination IP Address = [2001:db8::2:1]
Destination UDP Port = 5683
Uri-Host = "xn--18j4d.example"
Uri-Path = the string composed of the Unicode characters U+3053
U+3093 U+306b U+3061 U+306f, usually represented in UTF-8 as
E38193E38293E381ABE381A1E381AF hexadecimal
```

Output:

```
coap://xn--18j4d.example/
%E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF
```

(The line break has been inserted for readability; it is not part of the URI.)

o Input:

```
Destination IP Address = 198.51.100.1
Destination UDP Port = 61616
Uri-Path = ""
Uri-Path = "/"
Uri-Path = ""
Uri-Path = ""
Uri-Query = "/"
Uri-Query = "?&"
```

Output:

```
coap://198.51.100.1:61616//%2F//?%2F%2F&?%26
```

Appendix C. Changelog

(To be removed by RFC editor before publication.)

Changes from ietf-17 to ietf-18: Address comments from the IESG reviews.

- o Accept is now critical.
- o Add Size1 option for 4.13 responses.

Changes from ietf-15 to ietf-16: Address comments from the IESG reviews. These should not impact interoperability.

- o Clarify that once there has been an empty ACK, all further ACKs to the same message also must be empty (#301).
- o Define Cache-key properly (#302).
- o Clarify that ACKs don't get retransmitted, the CONs do (#303).
- o Clarify: NON is like separate for CON (#304).
- o Don't use decimal response codes, keep the 3+5 structure throughout (#305).
- o RFC 2119 usage in 4.5 (#306) and 8.2 (#307).
- o Ensure all protocol reactions to reserved or prohibited values are defined (#308).
- o URI matching rules may be scheme specific (#309).
- o Don't dally beyond MAX_TRANSMIT_SPAN during retransmission (#310).
- o More about selecting a token length for anti-spoofing (#311).
- o Discuss spoofing ACKs (#312).
- o Qualify partial discard strategy implementation note as UDP only (#313).
- o Explicitly point out that UDP and DTLS don't mix (#314).
- o Point out security consideration re URIs and access control (#315).
- o Point to RFC5280 section 6 (#316).

- o Add a paragraph about cert status checking (#317).
- o RSA is out, ECDHE is in for cert-with-PSK, too (#318).
- o Point out that requests and responses don't always come in pairs (#319).
- o Clarify when there is a need for Unicode normalization (#320).
- o Point out that Uri-Host doesn't handle user-part (#321).
- o Clarify the use of non-FQDN Authority Names in certificates.
- o Numerous editorial improvements and clarifications.

Changes from ietf-14 to ietf-15: Address comments from IETF last-call, mostly implementation notes and editorial improvements. These should not impact interoperability.

- o Clarify bytes/characters and UTF-8/ASCII in "Decomposing URIs into Options" (#282).
- o Make reference to ECC/CCM DTLS ciphersuite normative (#286).
- o Add a quick warning that bitwise scanning for a payload marker is not a good idea (#287).
- o Make reference to PROBING_RATE explicit for saturation discussion (#288).
- o Mention PROCESSING_DELAY when discussion piggy-backing (#290).
- o Various editorial nits: Clarify use of noun "service" (#283), Reference terminology from lwig-terminology (#284), make reference to HTTP terms more explicit (#285), add a forward reference to 5.9.2.9 (#289), 8 kbit/s is not "conservative" (#291).
- o Add description of resource depletion attack (#292).
- o Add description of DoS attack on congestion control (#293).
- o Add discussion of using non-trivial token for protecting against hijacking (#294).
- o Clarify implementation note about per-destination Message ID generation.

Changed from ietf-13 to ietf-14:

- o Made Accept option non-repeatable.
- o Clarified that Safe options in a 2.03 Valid response update the cache.
- o Clarified that payload sniffing is acceptable only if no Content-Format was supplied.
- o Clarified URI examples (Appendix B).
- o Numerous editorial improvements and clarifications.

Changed from ietf-12 to ietf-13:

- o Simplified message format.
 - * Removed the OC (Option Count) field in the CoAP Header.
 - * Changed the End-of-Options Marker into the Payload Marker.
 - * Changed the format of Options: use 4 bits for option length and delta; insert one or two additional bytes after the option header if necessary.
 - * Promoted the Token Option to a field following the CoAP Header.
- o Clarified when a payload is a diagnostic payload (#264).
- o Moved IPsec discussion to separate draft (#262).
- o Added a reference to a separate draft on reverse-proxy URI embedding (#259).
- o Clarified the use of ETags and of 2.03 responses (#265, #254, #256).
- o Added reserved Location-* numbers and clarified Location-*.
- o Added Proxy-Scheme proposal.
- o Clarified terms such as content negotiation, selected representation, representation-format, message format error.
- o Numerous clarifications and a few bugfixes.

Changed from ietf-11 to ietf-12:

- o Extended options to support lengths of up to 1034 bytes (#202).

- o Added new Jump mechanism for options and removed Fenceposting (#214).
- o Added new IANA option number registration policy (#214).
- o Added Proxy Unsafe/Safe and Cache-Key masking to option numbers (#241).
- o Re-numbered option numbers to use Unsafe/Safe and Cache-Key compliant numbers (#241).
- o Defined NSTART and restricted the value to 1 with a MUST (#215).
- o Defined PROBING_RATE and set it to 1 Byte/second (#215).
- o Defined DEFAULT_LEISURE (#246).
- o Renamed Content-Type into Content-Format, and Media Type registry into Content-Format registry.
- o A large number of small editorial changes, clarifications and improvements have been made.

Changed from ietf-10 to ietf-11:

- o Expanded section 4.8 on Transmission Parameters, and used the derived values defined there (#201). Changed parameter names to be shorter and more to the point.
- o Several more small editorial changes, clarifications and improvements have been made.

Changed from ietf-09 to ietf-10:

- o Option deltas are restricted to 0 to 14; the option delta 15 is used exclusively for the end-of-options marker (#239).
- o Option numbers that are a multiple of 14 are not reserved, but are required to have an empty default value (#212).
- o Fixed misleading language that was introduced in 5.10.2 in coap-07 re Uri-Host and Uri-Port (#208).
- o Segments and arguments can have a length of zero characters (#213).
- o The Location-* options describe together describe one location. The location is a relative URI, not an "absolute path URI" (#218).

- o The value of the Location-Path Option must not be '.' or '..' (#218).
- o Added a sentence on constructing URIs from Location-* options (#231).
- o Reserved option numbers for future Location-* options (#230).
- o Fixed response codes with payload inconsistency (#233).
- o Added advice on default values for critical options (#207).
- o Clarified use of identifiers in RawPublicKey Mode Provisioning (#222).
- o Moved "Securing CoAP" out of the "Security Considerations" (#229).
- o Added "All CoAP Nodes" multicast addresses to "IANA Considerations" (#216).
- o Over 100 small editorial changes, clarifications and improvements have been made.

Changed from ietf-08 to ietf-09:

- o Improved consistency of statements about RST on NON: RST is a valid response to a NON message (#183).
- o Clarified that the protocol constants can be configured for specific application environments.
- o Added implementation note recommending piggy-backing whenever possible (#182).
- o Added a content-encoding column to the media type registry (#181).
- o Minor improvements to Appendix D.
- o Added text about multicast response suppression (#177).
- o Included the new End-of-options Marker (#176).
- o Added a reference to draft-ietf-tls-oob-pubkey and updated the RPK text accordingly.

Changed from ietf-07 to ietf-08:

- o Clarified matching rules for messages (#175)

- o Fixed a bug in Section 8.2.2 on Etags (#168)
- o Added an IP address spoofing threat analysis contribution (#167)
- o Re-focused the security section on raw public keys (#166)
- o Added an 4.06 error to Accept (#165)

Changed from ietf-06 to ietf-07:

- o application/link-format added to Media types registration (#160)
- o Moved content-type attribute to the document from link-format.
- o Added coaps scheme and DTLS-secured CoAP default port (#154)
- o Allowed 0-length Content-type options (#150)
- o Added congestion control recommendations (#153)
- o Improved text on PUT/POST response payloads (#149)
- o Added an Accept option for content-negotiation (#163)
- o Added If-Match and If-None-Match options (#155)
- o Improved Token Option explanation (#147)
- o Clarified mandatory to implement security (#156)
- o Added first come first server policy for 2-byte Media type codes (#161)
- o Clarify matching rules for messages and tokens (#151)
- o Changed OPTIONS and TRACE to always return 501 in HTTP-CoAP mapping (#164)

Changed from ietf-05 to ietf-06:

- o HTTP mapping section improved with the minimal protocol standard text for CoAP-HTTP and HTTP-CoAP forward proxying (#137).
- o Eradicated percent-encoding by including one Uri-Query Option per &-delimited argument in a query.
- o Allowed RST message in reply to a NON message with unexpected token (#135).

- o Cache Invalidation only happens upon successful responses (#134).
- o 50% jitter added to the initial retransmit timer (#142).
- o DTLS cipher suites aligned with ZigBee IP, DTLS clarified as default CoAP security mechanism (#138, #139)
- o Added a minimal reference to draft-kivinen-ipsecme-ikev2-minimal (#140).
- o Clarified the comparison of UTF-8s (#136).
- o Minimized the initial media type registry (#101).

Changed from ietf-04 to ietf-05:

- o Renamed Immediate into Piggy-backed and Deferred into Separate -- should finally end the confusion on what this is about.
- o GET requests now return a 2.05 (Content) response instead of 2.00 (OK) response (#104).
- o Added text to allow 2.02 (Deleted) responses in reply to POST requests (#105).
- o Improved message deduplication rules (#106).
- o Section added on message size implementation considerations (#103).
- o Clarification made on human readable error payloads (#109).
- o Definition of CoAP methods improved (#108).
- o Max-Age removed from requests (#107).
- o Clarified uniqueness of tokens (#112).
- o Location-Query Option added (#113).
- o ETag length set to 1-8 bytes (#123).
- o Clarified relation between elective/critical and option numbers (#110).
- o Defined when to update Version header field (#111).
- o URI scheme registration improved (#102).

- o Added review guidelines for new CoAP codes and numbers.

Changes from ietf-03 to ietf-04:

- o Major document reorganization (#51, #63, #71, #81).
- o Max-age length set to 0-4 bytes (#30).
- o Added variable unsigned integer definition (#31).
- o Clarification made on human readable error payloads (#50).
- o Definition of POST improved (#52).
- o Token length changed to 0-8 bytes (#53).
- o Section added on multiplexing CoAP, DTLS and STUN (#56).
- o Added cross-protocol attack considerations (#61).
- o Used new Immediate/Deferred response definitions (#73).
- o Improved request/response matching rules (#74).
- o Removed unnecessary media types and added recommendations for their use in M2M (#76).
- o Response codes changed to base 32 coding, new Y.XX naming (#77).
- o References updated as per AD review (#79).
- o IANA section completed (#80).
- o Proxy-Uri Option added to disambiguate between proxy and non-proxy requests (#82).
- o Added text on critical options in cached states (#83).
- o HTTP mapping sections improved (#88).
- o Added text on reverse proxies (#72).
- o Some security text on multicast added (#54).
- o Trust model text added to introduction (#58, #60).
- o AES-CCM vs. AES-CCB text added (#55).

- o Text added about device capabilities (#59).
- o DTLS section improvements (#87).
- o Caching semantics aligned with RFC2616 (#78).
- o Uri-Path Option split into multiple path segments.
- o MAX_RETRANSMIT changed to 4 to adjust for RESPONSE_TIME = 2.

Changes from ietf-02 to ietf-03:

- o Token Option and related use in asynchronous requests added (#25).
- o CoAP specific error codes added (#26).
- o Erroring out on unknown critical options changed to a MUST (#27).
- o Uri-Query Option added.
- o Terminology and definitions of URIs improved.
- o Security section completed (#22).

Changes from ietf-01 to ietf-02:

- o Sending an error on a critical option clarified (#18).
- o Clarification on behavior of PUT and idempotent operations (#19).
- o Use of Uri-Authority clarified along with server processing rules; Uri-Scheme Option removed (#20, #23).
- o Resource discovery section removed to a separate CoRE Link Format draft (#21).
- o Initial security section outline added.

Changes from ietf-00 to ietf-01:

- o New cleaner transaction message model and header (#5).
- o Removed subscription while being designed (#1).
- o Section 2 re-written (#3).
- o Text added about use of short URIs (#4).

- o Improved header option scheme (#5, #14).
- o Date option removed while being designed (#6).
- o New text for CoAP default port (#7).
- o Completed proxying section (#8).
- o Completed resource discovery section (#9).
- o Completed HTTP mapping section (#10).
- o Several new examples added (#11).
- o URI split into 3 options (#12).
- o MIME type defined for link-format (#13, #16).
- o New text on maximum message size (#15).
- o Location Option added.

Changes from shelby-01 to ietf-00:

- o Removed the TCP binding section, left open for the future.
- o Fixed a bug in the example.
- o Marked current Sub/Notify as (Experimental) while under WG discussion.
- o Fixed maximum datagram size to 1280 for both IPv4 and IPv6 (for CoAP-CoAP proxying to work).
- o Temporarily removed the Magic Byte header as TCP is no longer included as a binding.
- o Removed the Uri-code Option as different URI encoding schemes are being discussed.
- o Changed the rel= field to desc= for resource discovery.
- o Changed the maximum message size to 1024 bytes to allow for IP/UDP headers.
- o Made the URI slash optimization and method idempotence MUSTs
- o Minor editing and bug fixing.

Changes from shelby-00 to shelby-01:

- o Unified the message header and added a notify message type.
- o Renamed methods with HTTP names and removed the NOTIFY method.
- o Added a number of options field to the header.
- o Combines the Option Type and Length into an 8-bit field.
- o Added the magic byte header.
- o Added new ETag Option.
- o Added new Date Option.
- o Added new Subscription Option.
- o Completed the HTTP Code - CoAP Code mapping table appendix.
- o Completed the Content-type Identifier appendix and tables.
- o Added more simplifications for URI support.
- o Initial subscription and discovery sections.
- o A Flag requirements simplified.

Authors' Addresses

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: April 05, 2014

A. Rahman, Ed.
InterDigital Communications, LLC
E. Dijk, Ed.
Philips Research
October 02, 2013

Group Communication for CoAP
draft-ietf-core-groupcomm-16

Abstract

CoAP is a specialized web transfer protocol for constrained devices and constrained networks. It is anticipated that constrained devices will 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 provides guidance for how the CoAP protocol should be used in a group communication context. An approach for using CoAP on top of IP multicast is detailed. Also, various use cases and corresponding protocol flows are provided to illustrate important concepts. Finally, guidance is provided for deployment in various network topologies.

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 <http://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 April 05, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Background	3
1.2. Scope	3
1.3. Conventions and Terminology	4
2. Protocol Considerations	4
2.1. IP Multicast Background	4
2.2. Group Definition and Naming	5
2.3. Port and URI Configuration	6
2.4. Group Methods	7
2.5. Group Member Discovery	8
2.6. Configuring Group Membership in Endpoints	8
2.6.1. Background	8
2.6.2. RESTful Interface	9
2.7. Multicast Request Acceptance and Response Suppression . .	11
2.8. Congestion Control	13
2.9. Proxy Operation	14
2.10. Exceptions	15
3. Use Cases and Corresponding Protocol Flows	16
3.1. Introduction	16
3.2. Network Configuration	16
3.3. Discovery of Resource Directory	18
3.4. Lighting Control	20
3.5. Lighting Control in MLD Enabled Network	23
3.6. Commissioning the Network Based On Resource Directory . .	24
4. Deployment Guidelines	25
4.1. Target Network Topologies	25
4.2. Networks Using the MLD Protocol	25
4.3. Networks Using RPL Multicast Without MLD	26
4.4. Networks Using MPL Forwarding Without MLD	26
4.5. 6LoWPAN Specific Guidelines for the 6LBR	27
5. Security Considerations	28
5.1. Security Configuration	28
5.2. Threats	28
5.3. Threat Mitigation	28
5.3.1. WiFi Scenario	28
5.3.2. 6LoWPAN Scenario	29
5.3.3. Future Evolution	29
6. IANA Considerations	29
6.1. New 'core.gp' Resource Type	29

6.2. New 'coap-group+json' Internet Media Type	29
7. Acknowledgements	31
8. References	31
8.1. Normative References	31
8.2. Informative References	32
Appendix A. Multicast Listener Discovery (MLD)	33
Appendix B. Change Log	33
Authors' Addresses	41

1. Introduction

1.1. Background

Constrained Application Protocol (CoAP) is a Representational State Transfer (REST) based web transfer protocol for resource constrained devices operating in an IP network [I-D.ietf-core-coap]. CoAP has many similarities to HTTP [RFC2616] but also has some key differences. Constrained devices can be large in numbers, but are often related to each other in function or location. For example, all the light switches in a building may belong to one group and all the thermostats may belong to another group. Groups may be pre-configured 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. HTTP does not support any equivalent functionality to CoAP group communication.

1.2. Scope

Group communication involves a one-to-many relationship between CoAP endpoints. Specifically, a single CoAP client will simultaneously get (or set) resource representations from multiple CoAP servers using CoAP over IP multicast. An example would be a CoAP light switch turning on/off multiple lights in a room with a single CoAP group communication PUT request, and handling the potential multitude of (unicast) responses.

The normative protocol aspects of running CoAP on top of IP Multicast and processing the responses are given in [I-D.ietf-core-coap]. The main contribution of this document lies in providing additional guidance for several important group communication features. Among the topics covered are group definition, group resource manipulation, and group configuration. Also, proxy operation and minimizing network congestion for group communication is discussed. Finally, specific use cases and deployment guidelines for CoAP group communication are outlined.

1.3. Conventions and 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 [RFC2119].

The above key words are used to establish a set of guidelines for CoAP group communication. An implementation of CoAP group communication MAY implement these guidelines; an implementation claiming compliance to this document MUST implement the set of guidelines.

This document assumes readers are familiar with the terms and concepts that are used in [I-D.ietf-core-coap]. In addition, this document defines the following terminology:

Group Communication

A source node sends a single message which is delivered to multiple destination nodes, where all destinations are identified to belong to a specific group. The source node itself may be part of the group. The underlying mechanism for group communication is assumed to be multicast based. The network involved may be a constrained network such as a low-power, lossy network.

Multicast

Sending a message to multiple destination nodes with one network invocation. There are various options to implement multicast including layer 2 (Media Access Control) and layer 3 (IP) mechanisms.

IP Multicast

A specific multicast solution based on the use of IP multicast addresses as defined in "IANA Guidelines for IPv4 Multicast Address Assignments" [RFC5771] and "IP Version 6 Addressing Architecture" [RFC4291].

Low power and Lossy Network (LLN)

A type of constrained IP network where devices are interconnected by low-power and lossy links. The links may be composed of one or more technologies such as IEEE 802.15.4, Bluetooth Low Energy (BLE), Digital Enhanced Cordless Telecommunication (DECT), and IEEE P1901.2 power-line communication.

2. Protocol Considerations

2.1. IP Multicast Background

IP Multicast protocols have been evolving for decades, resulting in standards such as Protocol Independent Multicast - Sparse Mode (PIM-SM) [RFC4601]. IP Multicast is very popular in specific deployments such as in enterprise networks (e.g., for video conferencing), smart home networks (e.g., Universal Plug and Play (UPnP)) and carrier IPTV deployments. The packet economy and minimal host complexity of IP multicast make it attractive for group communication in constrained environments.

To achieve IP multicast beyond link-local scope, an IP multicast routing or forwarding protocol needs to be active on IP routers. An example of a routing protocol specifically for LLNs is the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) (Section 12 of [RFC6550]) and an example of a forwarding protocol for LLNs is Multicast Protocol for Low power and Lossy Networks (MPL) [I-D.ietf-roll-trickle-mcast]. RPL and MPL do not depend on each other; each can be used in isolation and both can be used in combination in a network. Finally, PIM-SM [RFC4601] is often used for multicast routing in traditional IP networks (i.e. networks that are not constrained).

IP multicast can also be run in a Link-Local (LL) scope. This means that there is no routing involved and an IP multicast message is only received over the link on which it was sent.

For a complete IP multicast solution, in addition to a routing/forwarding protocol, a "listener" protocol may be needed for the devices to subscribe to groups (see Section 4.2).

2.2. Group Definition and Naming

A group is defined as a set of CoAP endpoints, where each endpoint is configured to receive multicast CoAP requests that are sent to the group's associated IP multicast address. An endpoint MAY be a member of multiple groups. Group membership of an endpoint MAY dynamically change over time.

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

Note: A Group URI is needed to initiate CoAP group communications. If a CoAP implementation accepts a CoAP URI as input in the group communications request API, then the parsing method of Section 6.4 of [I-D.ietf-core-coap] should be used in such way that a CoAP multicast request is generated.

It is recommended, for sending nodes, to use the IP multicast address literal in a Group URI. In case a Group hostname is used, it can be uniquely mapped to a site-local or global IP multicast address via DNS resolution (if supported). Some examples of hierarchical Group FQDN naming (and scoping) for a building control application are shown below (and are derived from [I-D.vanderstok-core-dna]):

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

Similarly, if supported, reverse mapping (from IP multicast address to Group FQDN) is possible using the reverse DNS resolution technique ([I-D.vanderstok-core-dna]).

2.3. Port and URI Configuration

A CoAP server that is a member of a group listens for CoAP messages on the group's IP multicast address, on a specified UDP port. The default UDP port is the CoAP default port 5683 but a non-default UDP port MAY be specified for the group; in which case implementers MUST ensure that all group members are configured to use this same port.

Multicast based group communication will not work if there is diversity in the authority port (e.g., different dynamic port addresses across the group) or if other parts of the URI such as the path, or the query, differ on different endpoints. Therefore, some measures must be present to ensure uniformity in port number and resource names/locations within a group. All CoAP multicast requests MUST be sent using a port number according to one of below options:

1. A pre-configured port number. The pre-configuration mechanism MUST ensure that the same port number is pre-configured across all endpoints in a group and across all CoAP clients performing the group requests.
2. If the client is configured to use service discovery including port discovery, it uses a port number obtained via a service discovery lookup operation for the targeted CoAP multicast group.
3. Use the default CoAP UDP port (5683).

All CoAP multicast requests SHOULD operate on URI paths in one of the following ways:

1. Pre-configured URI paths, if available. The pre-configuration mechanism SHOULD ensure that these paths are pre-configured across all CoAP servers in a group and all CoAP clients performing the group requests. Note that [I-D.ietf-appsawg-uri-get-off-my-lawn] prescribes that any specification must not constrain, define structure or semantics for any path component.
2. If the client is configured to use default CoRE resource discovery, it uses URI paths retrieved from a `"/.well-known/core"` lookup on a group member. The URI paths the client will use MUST be known to be available also in all other endpoints in the group. The URI path configuration mechanism on servers MUST ensure that these URIs (identified as being supported by the group) are configured on all group endpoints.
3. If the client is configured to use another form of service discovery, it uses URI paths from an equivalent service discovery lookup which returns the resources supported by all group members.
4. If the client has received a Group URI through a previous RESTful interaction with a trusted server, for the purpose of the client using this URI in a request, it can use this URI in a multicast request. For example, a commissioning tool may instruct a sensor device in this way to which target (multicast URI) it should report sensor events.

2.4. Group Methods

Idempotent methods (i.e., CoAP GET, PUT, and DELETE) SHOULD be used for group communication, with one exception as follows. A non-idempotent method (i.e., CoAP POST) MAY be used for group communication if the resource being POSTed to has been designed to

cope with the lossy nature of multicast. Note that not all group members are guaranteed to receive the multicast request, and the sender cannot readily find out which group members did not receive it.

All CoAP messages that are sent via multicast MUST be Non-confirmable. A unicast response per server MAY be sent back to answer the group communication request (e.g., response "2.05 Content" to a group GET request) taking into account the congestion control rules defined in Section 2.8. The unicast responses received 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 (see section 8 of [I-D.ietf-core-coap]).

2.5. Group Member Discovery

CoAP Groups, and the membership of these groups, can be discovered via the lookup interfaces defined in [I-D.ietf-core-resource-directory]. An example of doing some of these lookups is given in Section 3.6.

2.6. Configuring Group Membership in Endpoints

2.6.1. Background

The group membership of a CoAP endpoint may be configured in one of the following ways. First, the group membership may be pre-configured before node deployment. Second, a node may be programmed to discover (query) its group membership during operation using a specific service discovery means. Third, it may be configured during operation by another node (e.g., a commissioning device).

In the first case, the pre-configured group information may be either directly a IP multicast address, or a hostname (FQDN) which is during operation resolved to a IP multicast address by the endpoint using DNS (if supported).

For the second case, a CoAP endpoint may look up its group membership using techniques such as DNS-SD and Resource Directory [I-D.ietf-core-resource-directory]. The latter is detailed more in Section 3.6.

In the third case, typical in scenarios such as building control, a commissioning tool determines to which group a sensor or actuator node belongs, and writes this information to the node, which can subsequently join the correct IP multicast group on its network interface. The information written may again be an IP multicast address or a hostname.

2.6.2. RESTful Interface

To achieve better interoperability between endpoints from different manufacturers, an OPTIONAL RESTful interface for configuring CoAP endpoints with relevant group information is described here. This interface provides a solution for the third case mentioned above. To access this interface a client MUST use unicast methods (GET/PUT/POST) only as it is a method of configuring group information in individual endpoints. Using multicast operations in this situation may lead to unexpected (possibly circular) behavior in the network. Also, the (unicast) methods to configure group membership SHOULD use DTLS-secured CoAP [I-D.ietf-core-coap]. Thus only authorized controllers should be allowed by an endpoint to configure its group membership.

CoAP endpoints implementing this optional mechanism MUST support the group configuration Internet Media Type "application/coap-group+json" (Section 6.2). A resource offering this representation can be annotated for direct discovery [RFC6690] using the resource type (rt) "core.gp" where "gp" is shorthand for "group" (Section 6.1). An authorized controller uses this media type to query/manage group membership of a CoAP endpoint as defined below.

2.6.2.1. GET Interface

The group configuration resource has a JSON-based content format (as indicated by the media type). A (unicast) GET on a CoAP endpoint with a resource with this format returns a JSON array of group objects, each group object being a JSON object that indicates one multicast group membership. The example below illustrates a request to an endpoint querying its group membership information, where the response is in the "application/coap-group+json" content format containing a single group object:

```
Req: GET /gp
Res: 2.05 Content (Content-Format: application/coap-group+json)
[ { "n": "Room-A-Lights.floor1.west.bldg6.example.com",
    "a": "ff15::4200:f7fe:ed37:14ca" }
]
```

In a response, the OPTIONAL "n" key/value pair stands for "name" and identifies the group with a hostname, for example a FQDN.

The "a" key/value pair specifies the IP multicast address (and optionally the port number) of the group. It contains an IPv4 address or an IPv6 address in dotted decimal notation. The following ABNF rule can be used for parsing the address, referring to the definitions in Section 6 of [I-D.ietf-core-coap] and [RFC3986].

```
group-address = IPv4address [ ":" port ]  
               / "[" IPv6address "]" [ ":" port ]
```

If the port number is not provided then it is assumed to be the default CoAP port (5683). The "a" key/value pair **MUST** be included if the IP address is known at the time of generating the response, and **SHOULD NOT** be included if unknown.

Note that each group object in the JSON array represents a single IP multicast group for the endpoint. If there are multiple elements in the array then the endpoint is a member of multiple IP multicast groups.

2.6.2.2. POST Interface

A (unicast) POST with a group configuration media type as payload instructs the CoAP endpoint to join the defined group(s). The endpoint adds the specified IP multicast address(es) to its network interface configuration. The endpoint also updates the resource by adding the specified group object(s) to the existing ones:

```
Req: POST /gp (Content-Format: application/coap-group+json)  
[ { "n": "All-Devices.floor1.west.bldg6.example.com",  
    "a": "ff15::4200:f7fe:ed37:abcd" } ]  
Res: 2.04 Changed
```

In a request, the "a" key/value pair is **OPTIONAL** to define the group's associated IP multicast address. The "n" pair is also **OPTIONAL**. If the "a" pair is given, this takes priority and the "n" pair becomes informational. If only the "n" pair is given, the CoAP endpoint may perform DNS resolution (if supported) to obtain the IP multicast address from the hostname. At least one of the "n"/"a" pairs **MUST** be given per group object.

After any change on a Group configuration resource, the endpoint **MUST** effect registration/de-registration from the corresponding IP multicast group(s) as soon as possible. When a POST payload contains in "a" a multicast address to which the endpoint is already subscribed, the endpoint **MUST** re-register to that multicast address.

2.6.2.3. PUT Interface

A (unicast) PUT with a group configuration media type as payload will replace all current group memberships in the endpoint with the new ones defined in the PUT request. The format of the group configuration payload, and the requirements for it, are the same as in Section 2.6.2.2.

A (unicast) PUT with an empty array [] will delete all group memberships from the endpoint. Note that it is not possible to delete individual group memberships on an endpoint. (A DELETE interface is not defined as the underlying resource should not be removed even if it is empty).

2.7. Multicast Request Acceptance and Response Suppression

CoAP [I-D.ietf-core-coap] and CoRE Link Format [RFC6690] define normative behaviors for:

1. Multicast request acceptance - in which cases a CoAP request is accepted and executed, and when not.
2. Multicast response suppression - in which cases the CoAP response to an already-executed request is returned to the requesting endpoint, and when not.

A CoAP response differs from a CoAP ACK; ACKs are never sent by servers in response to a multicast CoAP request. This section first summarizes these normative behaviors and then presents additional guidelines for response suppression. Also a number of multicast example applications are given to illustrate the overall approach.

To apply any rules for request and/or response suppression, a CoAP server must be aware that an incoming request arrived via multicast by making use of APIs such as IPV6_RECVPKTINFO [RFC3542].

For multicast request acceptance, the REQUIRED behaviors are:

- o A server SHOULD NOT accept a multicast request that cannot be "authenticated" in some way (cryptographically or by some multicast boundary limiting the potential sources) [I-D.ietf-core-coap]. See Section 5.3 for examples of multicast boundary limiting methods.
- o A server SHOULD NOT accept a multicast discovery request with a query string (as defined in CoRE Link Format [RFC6690]) if filtering ([RFC6690]) is not supported by the server.

- o A server SHOULD NOT accept a multicast request that acts on a specific resource for which multicast support is not required. (Note that for the resource `"/.well-known/core"`, multicast support is required if "multicast resource discovery" is supported as specified in section 1.2.1 of [RFC6690]). Implementers are advised to disable multicast support by default on any other resource, until explicitly enabled by an application or by configuration.)
- o Otherwise accept the multicast request.

For multicast response suppression, the REQUIRED behaviors are:

- o A server SHOULD NOT respond to a multicast discovery request if the filter specified by the request's query string does not match.
- o A server MAY choose not to respond to a multicast request, if there's nothing useful to respond (e.g., error or empty response).
- o Otherwise respond to the multicast request.

The above response suppression behaviors are complemented by the following guidelines. CoAP servers SHOULD implement configurable response suppression, enabling at least the following options per resource that supports multicast requests:

- o Suppression of all 2.xx success responses;
- o Suppression of all 4.xx client errors;
- o Suppression of all 5.xx server errors;
- o Suppression of all 2.05 responses with empty payload.

A number of group communication example applications are given below to illustrate how to make use of response suppression:

- o CoAP resource discovery: Suppress 2.05 responses with empty payload and all 4.xx and 5.xx errors.
- o Lighting control: Suppress all 2.xx responses after a lighting change command.
- o Update configuration data in a group of devices using multicast PUT: No suppression at all. The client uses collected responses to identify which group members did not receive the new configuration; then attempts using CoAP CON unicast to update those specific group members.

- o Multicast firmware update by sending blocks of data: Suppress all 2.xx and 5.xx responses. After having sent all multicast blocks, the client checks each endpoint by unicast to identify which data blocks are still missing in each endpoint.
- o Conditional reporting for a group (e.g., sensors) based on a URI query: Suppress all 2.05 responses with empty payload (i.e., if a query produces no matching results).

2.8. Congestion Control

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

CoAP [I-D.ietf-core-coap] reduces multicast-specific congestion risks through the following measures:

- o A server MAY choose not to respond to a multicast request if there's nothing useful to respond (e.g., error or empty response). See Section 2.7 for more detailed guidelines on response suppression.
- o A server SHOULD limit the support for multicast requests to specific resources where multicast operation is required.
- o A multicast request MUST be Non-confirmable.
- o A response to a multicast request SHOULD be Non-confirmable (Section 5.2.3 of [I-D.ietf-core-coap]).
- o A server does not respond immediately to a multicast request, but SHOULD first wait for a time that is randomly picked within a predetermined time interval called the Leisure.

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

- o A server in an LLN should only support multicast GET for resources that are small. For example, the payload of the response is 5% of the IP Maximum Transmit Unit (MTU) size (e.g. so it fits into a single link-layer frame).

- o A server can 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].
- o A server can also minimize the payload length of a response to a multicast GET (e.g., on `"/.well-known/core"`) using CoAP blockwise transfers [I-D.ietf-core-block], returning only a first block of the CoRE Link Format description. For this reason, a CoAP client sending a multicast CoAP request to `"/.well-known/core"` SHOULD support core-block.
- o A client should use CoAP multicast with the smallest possible 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.

More guidelines specific to use of CoAP in 6LoWPAN networks [RFC4944] are given in Section 4.5.

2.9. Proxy Operation

CoAP [I-D.ietf-core-coap] allows a client to request a forward-proxy to process its CoAP request. For this purpose the client either specifies the request URI as a string in the Proxy-URI option, or it specifies the Proxy-Scheme option with the 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 group communication request made in this manner through a proxy.

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

- o Aggregation of (unicast) responses to a 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.
- o There is no default format defined in CoAP for aggregation of multiple responses into a single response.

Alternatively, if a proxy follows directly the specification for a CoAP Proxy [I-D.ietf-core-coap], the proxy would simply forward all

the individual (unicast) responses to a group communication request to the client (i.e., no aggregation). There are also issues with this approach:

- o The client may be confused as it may not have known that the Proxy-URI contained a multicast target. That is, the client may be expecting only one (unicast) response but instead receives multiple (unicast) responses potentially leading to fault conditions in the application.
- o Each individual CoAP response will appear to originate (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.

Due to above issues, a guideline is defined here that a CoAP Proxy SHOULD NOT support processing a multicast CoAP request but rather return a 501 (Not Implemented) response in such case. The exception case here (i.e., to process it) is allowed under following conditions:

- o The CoAP Proxy MUST be explicitly configured (whitelist) to allow proxied multicast requests by specific client(s).
- o The proxy SHOULD return individual (unicast) CoAP responses to the client (i.e., not aggregated). The exception case here occurs when a (future) standardized aggregation format is being used.
- o 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.

2.10. Exceptions

Group communication using IP multicast offers improved network efficiency and latency amongst other benefits. However, group communication may not always be possible to implement in a given network. The primary reason for this will be if IP multicast is not (fully) supported in the network.

For example, in an LLN where the RPL protocol is used for routing in "Non-storing mode" (Mode Of Operation=1) or "Storing mode with no multicast support" (Mode Of Operation=2) [RFC6550] and no other routing/forwarding protocol is defined, there will be no IP multicast routing beyond link-local scope. This means that any CoAP group communication above link-local scope will not be supported in this network.

3. Use Cases and Corresponding Protocol Flows

3.1. Introduction

The use of CoAP group communication is shown in the context of the following two use cases and corresponding protocol flows:

- o Discovery of Resource Directory (RD, [I-D.ietf-core-resource-directory]): discovering the local CoAP RD which contains links to resources stored on other CoAP servers [RFC6690].
- o Lighting Control: synchronous operation of a group of IPv6-connected lights (e.g., 6LoWPAN [RFC4944] lights).

3.2. Network Configuration

To illustrate the use cases we define two network configurations. Both are based on the topology as shown in Figure 1. The two configurations using this topology are:

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:

- o A large room (Room-A) with three lights (Light-1, Light-2, Light-3) controlled by a Light Switch. The devices are organized into two subnets. In reality, there could be more lights (up to several hundreds) but these are not shown for clarity.
- o Light-1 and the Light Switch are connected to a router (Rtr-1).
- o Light-2 and the Light-3 are connected to another router (Rtr-2).
- o The routers are connected to an IPv6 network backbone which 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 Multicast Listener Discovery (MLD) for forming groups.
- o A CoAP RD is connected to the network backbone.
- o The 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 hostname to 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).

- o A Controller (CoAP client) is connected to the backbone, which is able to control various building functions including lighting.

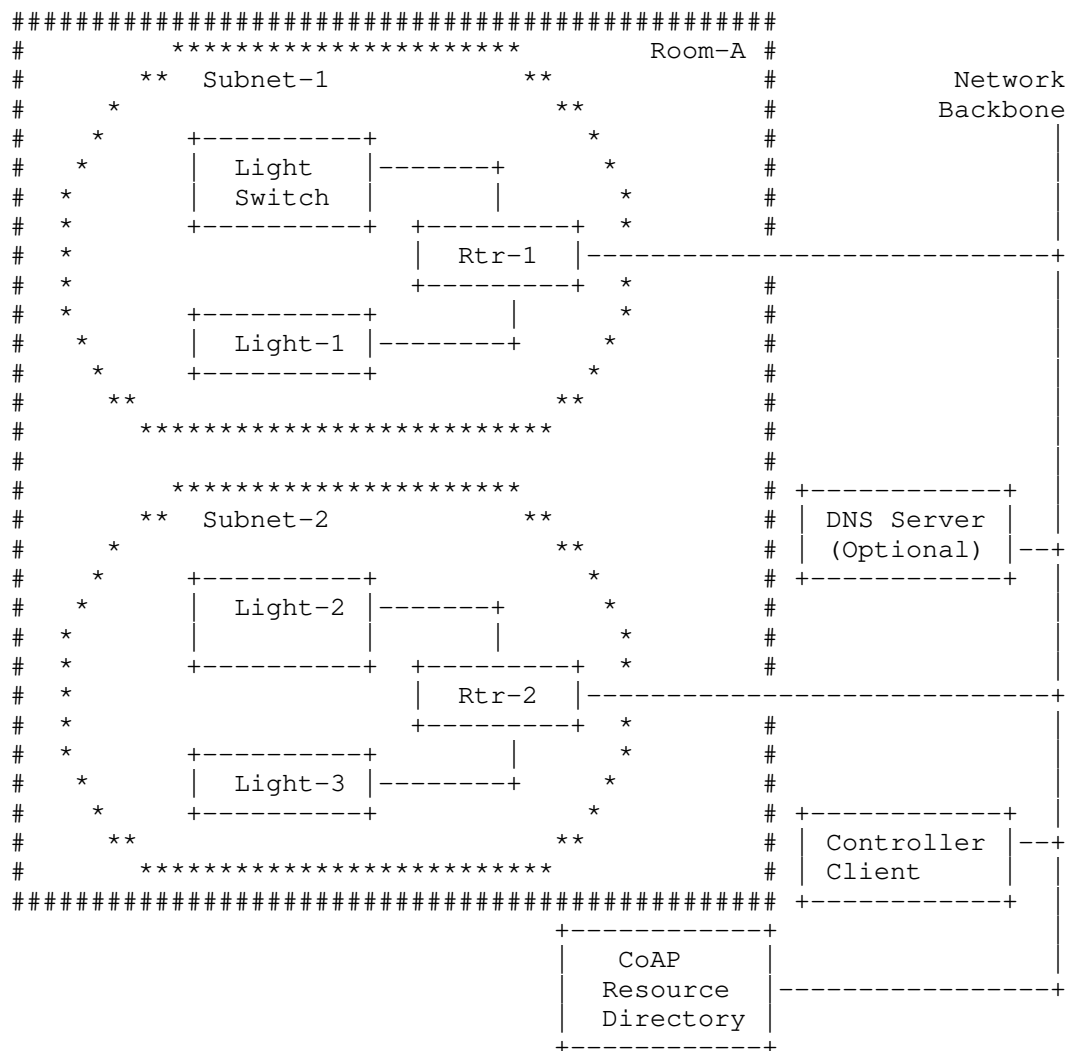


Figure 1: Network Topology of a Large Room (Room-A)

3.3. Discovery of Resource Directory

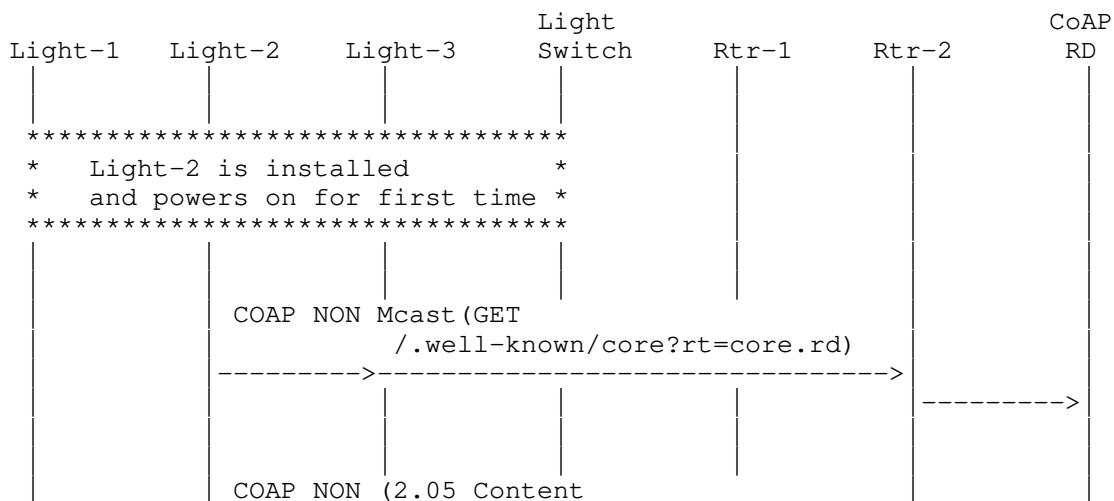
The protocol flow for discovery of the CoAP RD for the given network (of Figure 1) is shown in Figure 2:

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

- o Light-2 will then search for the local CoAP RD by sending out a GET request (with the `"/.well-known/core?rt=core.rd"` request URI) to the site-local "All CoAP Nodes" multicast address.
- o This multicast message will then go to each node in subnet-2. Rtr-2 will then forward into to the Network Backbone where it will be received by the CoAP RD. All other nodes in subnet-2 will ignore the multicast GET 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`).
- o The CoAP 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`.
- o Note that the flow is shown only for Light-2 for clarity. Similar flows will happen for Light-1, Light-3 and the Light Switch when they are first installed.

The CoAP 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. (See [I-D.ietf-core-resource-directory] for more details).

For other discovery use cases such as discovering local CoAP servers, services or resources group communication can be used in a similar fashion as in the above use case. Both Link-Local (LL) and site-local discovery are possible this way.



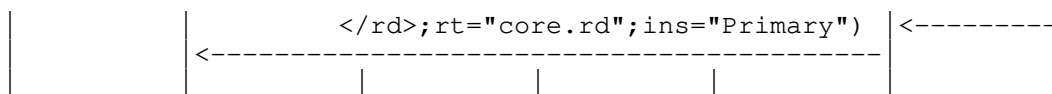


Figure 2: Resource Directory Discovery via Multicast Request

3.4. Lighting Control

The protocol flow for a building automation lighting control scenario for the network (Figure 1) is shown 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 multicast CoAP requests related to lighting control. (See Section 2.7 for more details on response suppression by a server.)

In addition, Figure 4 shows a protocol flow example for the case that servers do respond to a lighting control 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 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 CoAP 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 multicast forwarding/routing will be of sufficient quality to provide on average a very high probability of packet delivery to all CoAP endpoints in a multicast group. An example protocol to accomplish this using randomized retransmission is the MPL forwarding protocol for LLNs [I-D.ietf-roll-trickle-mcast].

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

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

2. Network configuration (application-independent): 6LBRs are configured with multicast addresses, or address blocks, to filter out or to pass through to/from the 6LoWPAN.
3. 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.
4. 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 which each nodes 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 a multicast CoAP request.

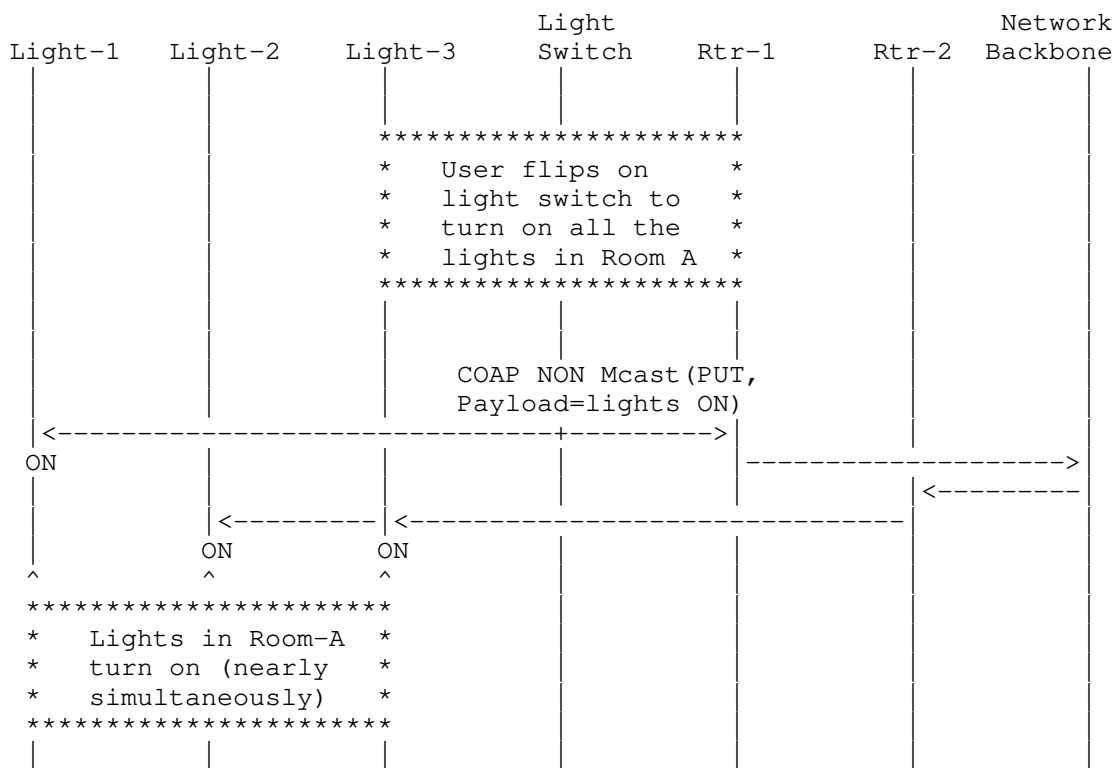


Figure 3: Light Switch Sends Multicast Control Message

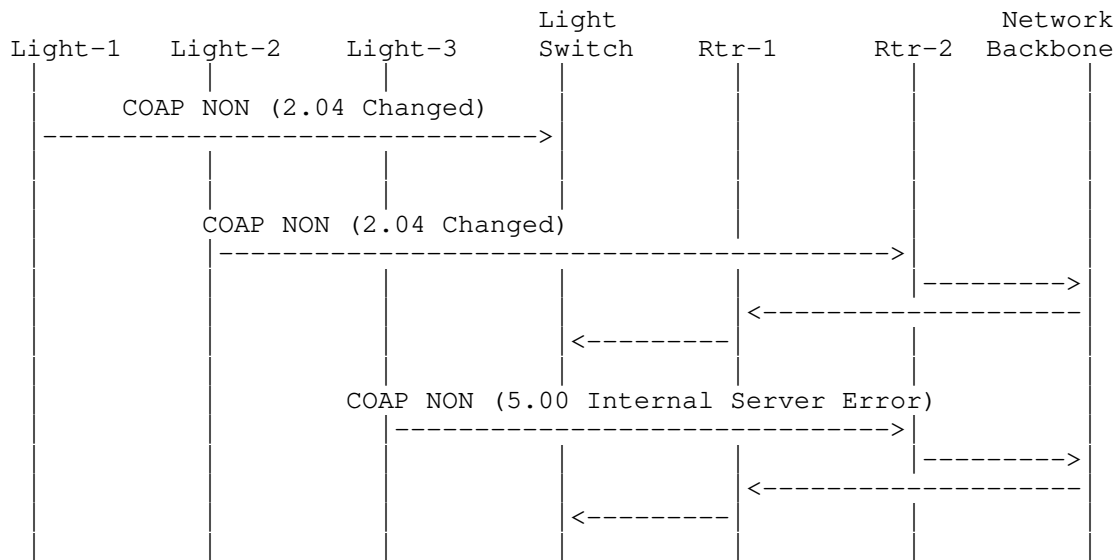
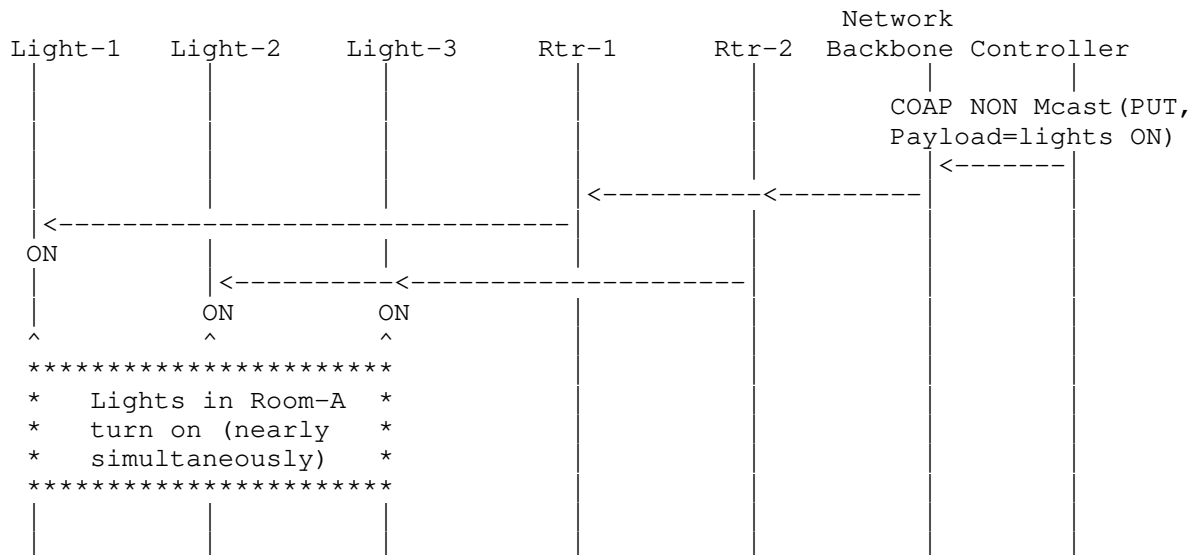


Figure 4: Lights (Optionally) Respond to Multicast CoAP Request

Another, but similar, lighting control use case is shown in Figure 5. In this case a controller connected to the Network Backbone sends a CoAP multicast request to turn on all lights in Room-A. Every Light sends back a CoAP response to the Controller after being turned on.



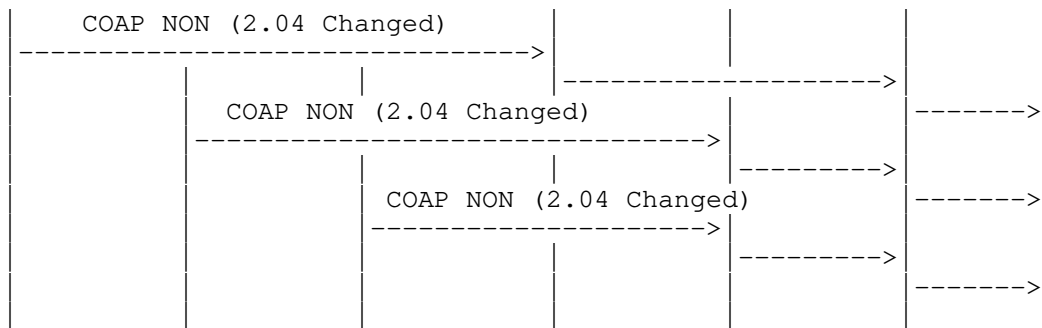


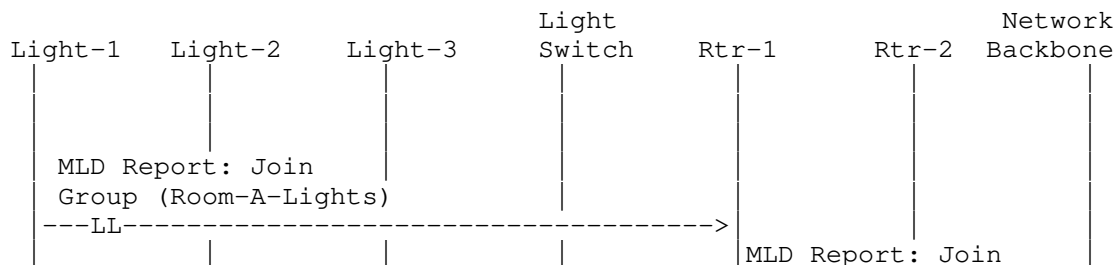
Figure 5: Controller On Backbone Sends Multicast Control Message

3.5. Lighting Control in MLD Enabled Network

The use case of previous section can also apply in networks where nodes support the MLD protocol [RFC3810]. The Lights then take on the role of MLDv2 listener and the routers (Rtr-1, Rtr-2) are MLDv2 Routers. In the Ethernet based network configuration, MLD may be available on all involved network interfaces. Use of MLD in the 6LoWPAN based configuration is also possible, but requires MLD support in all nodes in the 6LoWPAN. In current 6LoWPAN implementations, MLD is however not supported.

The resulting protocol flow is shown in Figure 6. This flow is executed after the commissioning phase, as soon as Lights are configured with a group address to listen to. The (unicast) MLD Reports may require periodic refresh activity as specified by the MLD protocol. In the figure, LL denotes Link Local communication.

After the shown sequence of MLD Report messages has been executed, both Rtr-1 and Rtr-2 are automatically configured to forward multicast traffic destined to Room-A-Lights onto their connected subnet. Hence, no manual Network Configuration of routers, as previously indicated in Section 3.4, is needed anymore.



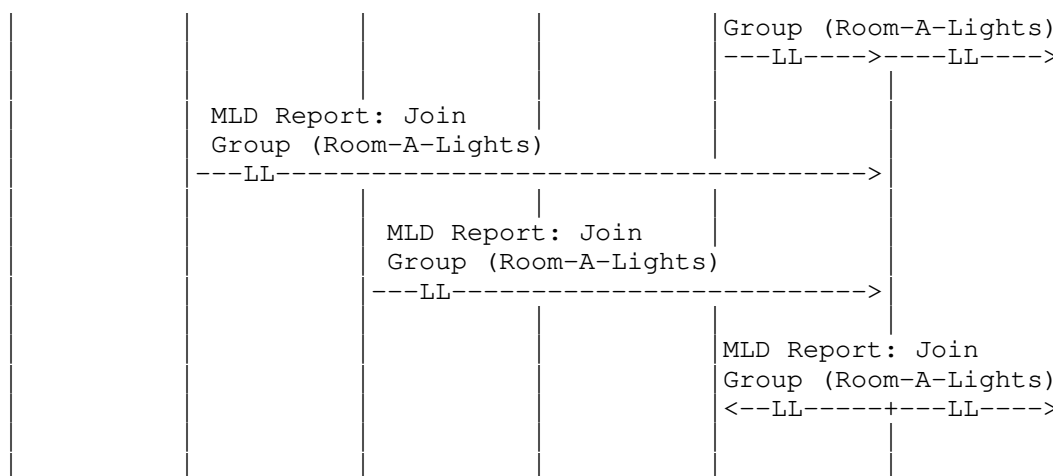


Figure 6: Joining Lighting Groups Using MLD

3.6. Commissioning the Network Based On Resource Directory

This section outlines how devices in the lighting use case (both Switches and Lights) can be commissioned, making use of Resource Directory [I-D.ietf-core-resource-directory] and its group configuration feature.

Once the Resource Directory (RD) is discovered, the Switches and Lights need to be discovered and their groups need to be defined. For the commissioning of these devices, a commissioning tool can be used that defines the entries in the RD. The commissioning tool has the authority to change the contents of the RD and the Light/Switch nodes. DTLS based security is used by the commissioning tool to modify operational data in RD, Switches and Lights.

In our particular use case, a group of three lights is defined with one multicast address and hostname "Room-A-Lights.floor1.west.bldg6.example.com". The commissioning tool has a list of the three lights and the associated multicast address. For each light in the list the tool learns the IP address of the light and instructs the RD with three POST commands to store the endpoints associated with the three lights as prescribed by the RD specification [I-D.ietf-core-resource-directory]. Finally the commissioning tool defines the group in the RD to contain these three endpoints. Also the commissioning tool writes the multicast address in the Light endpoints with, for example, the POST /gp command discussed in Section 2.6.2.2.

The light switch can discover the group in RD and thus learn the multicast address of the group. The light switch will use this address to send multicast commands to the members of the group. When the message arrives the Lights should recognize the multicast address and accept the message.

4. Deployment Guidelines

This section provides guidelines how IP Multicast based CoAP group communication can be deployed in various network configurations.

4.1. Target Network Topologies

CoAP group communication can be deployed in various network topologies. First, the target network may be a traditional IP network, or a LLN such as a 6LoWPAN network, or consist of mixed traditional/constrained network segments. Second, it may be a single subnet only or multi-subnet; e.g., multiple 6LoWPAN networks joined by a single backbone LAN. Third, a wireless network segment may have all its nodes reachable in a single IP hop (fully connected), or it may require multiple IP hops for some pairs of nodes to reach each other.

Each topology may pose different requirements on the configuration of routers and protocol(s), in order to enable efficient CoAP group communication. To enable all the above target network topologies, an implementation of CoAP group communication needs to allow:

1. Routing/forwarding of IP multicast packets over multiple hops
2. Routing/forwarding of IP multicast packets over subnet boundaries between traditional and constrained (e.g. LLN) networks.

The remainder of this section discusses solutions to enable this.

4.2. Networks Using the MLD Protocol

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

The Multicast Listener Discovery (MLD) protocol [RFC3810] (see Appendix A) is the standard IPv6 method to achieve this; therefore this approach should be used on traditional IP networks. CoAP server nodes would then act in the role of MLD Multicast Address Listener.

The guidelines from [RFC6636] on tuning of MLD for mobile and wireless networks may be useful when implementing MLD in LLNs. However, on LLNs and 6LoWPAN networks the use of MLD may not be feasible at all due to constraints on code size, memory, or network capacity.

4.3. Networks Using RPL Multicast Without MLD

It is assumed in this section that the MLD protocol is not implemented in a network, for example due to resource constraints. The RPL routing protocol (see Section 12 of [RFC6550]) defines the advertisement of IP multicast destinations using DAO messages and routing of multicast IP packets based on this. It requires the RPL Mode of Operation (MOP) to be 3 (Storing Mode with multicast support).

Hence, RPL DAO can be used by CoAP nodes (that are also RPL routers) to advertise IP multicast group membership to parent routers. Then, the RPL protocol is used to route multicast CoAP requests over multiple hops to the correct CoAP servers.

The same mechanism can be used to convey IP multicast group membership information to an edge router (e.g., 6LBR), in case the edge router is also the root of the RPL DODAG. This is useful because the edge router then learns which IP multicast traffic it needs to pass through from the backbone network into the LLN subnet. In 6LoWPAN networks, such selective "filtering" may avoid congestion of a 6LoWPAN subnet by IP multicast traffic from the traditional backbone network.

4.4. Networks Using MPL Forwarding Without MLD

The MPL forwarding protocol [I-D.ietf-roll-trickle-mcast] can be used for propagation of IP multicast packets to all MPL Forwarders within a predefined network domain, over multiple hops. MPL is designed to work in LLNs. In this section it is again assumed that the MLD protocol is not implemented in the network, for example due to resource limitations in an LLN.

In a typical use of MPL, all nodes in an LLN are MPL Forwarders. So it would appear there is no need for CoAP servers to advertise their multicast group membership, since any IP multicast packet that enters the MPL Domain is distributed to all MPL Forwarders without regard to what multicast addresses the individual nodes are listening to.

However, if an IP multicast request originates outside the MPL Domain, this request will by default not be propagated by MPL to the CoAP server(s) within the MPL Domain that need to receive it. This

situation can become a problem in building control use cases. For example, in a network topology of Figure 1 where the Subnets are 6LoWPAN subnets and per 6LoWPAN subnet one Realm-Local MPL Domain is defined. Suppose that the Controller Client needs to send a single CoAP multicast request to group Room-A-Lights. By default, the request would be blocked by Rtr-1 and by Rtr-2, and not enter the Realm-Local MPL Domains associated to Subnet-1 and Subnet-2. The reason is that Rtr-1 and Rtr-2 do not have the knowledge that devices in Subnet-1/2 want to listen for IP packets destined to multicast group Room-A-Lights.

To solve the above issue, the following solutions could be applied:

1. Extend the MPL Domain. E.g. in above example, include the Network Backbone to be part of the MPL Domain.
2. Manual configuration of edge router(s) as MPL Seed(s) for specific multicast traffic. E.g. in above example, configure Rtr-1 and Rtr-2 to act as MLD Address Listeners for the Room-A-Lights multicast group. Then any received traffic from the backbone, destined to that group, would be injected into the MPL Domain that is associated to that router. Also a whitelist filtering should be enabled in these routers, to only pass through multicast traffic to configured groups such as Room-A-Lights. This would prevent unnecessarily flooding of the MPL Domain with multicast packets, which is especially important if the subnets are LLNs.
3. Use an additional protocol for injection of multicast traffic from outside an MPL Domain into the MPL Domain, based on multicast group subscriptions of Forwarders within the MPL Domain. Such protocol is currently not defined in IETF and outside scope of [I-D.ietf-roll-trickle-mcast].

Concluding, MPL can be used directly in case all sources of multicast CoAP requests (CoAP clients) and also all the destinations (CoAP servers) are inside a single MPL Domain. Then, each source node acts as an MPL Seed. In all other cases, MPL can only be used with additional protocols and/or configuration.

4.5. 6LoWPAN Specific Guidelines for the 6LBR

To support multi-subnet scenarios for CoAP group communication, it is recommended that a 6LoWPAN Border Router (6LBR) will act in an MLD Router role on the backbone link. If this is not possible then the 6LBR should be configured to act as an MLD Multicast Address Listener (see Appendix A) on the backbone link.

5. Security Considerations

This section describes the relevant security configuration for CoAP group communication using IP multicast. The threats to CoAP group communication are also identified and various approaches to mitigate these threats are summarized.

5.1. Security Configuration

As defined in [I-D.ietf-core-coap], CoAP group communication based on IP multicast:

- o Will operate in CoAP NoSec (No Security) mode, until a future group security solution is developed (see also Section 5.3.3).
- o MUST NOT use "coaps" scheme. That is, all group communication MUST use only "coap" scheme.

5.2. Threats

Essentially the above configuration means that there is no security at the CoAP layer for group communication. This is due to the fact that the current DTLS based approach for CoAP is exclusively unicast oriented and does not support group security features such as group key exchange and group authentication. As a direct consequence of this, CoAP group communication is vulnerable to all attacks mentioned in [I-D.ietf-core-coap] for IP multicast.

5.3. Threat Mitigation

The [I-D.ietf-core-coap] identifies various threat mitigation techniques for CoAP multicast. In addition to those guidelines, it is recommended that for sensitive data or safety-critical control, a combination of appropriate link-layer security and administrative control of IP multicast boundaries should be used. Some examples are given below.

5.3.1. WiFi Scenario

In a home automation scenario (using WiFi), the WiFi encryption should be enabled to prevent rogue nodes from joining. The Customer Premise Equipment (CPE) that enables access to the Internet should also have its 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 domain of the IP multicast should be set to be either link-local scope or site-local scope. For site-local scope, the CPE will be an appropriate multicast scope boundary point.

5.3.2. 6LoWPAN Scenario

In a building automation scenario, a particular room may have a single 6LoWPAN network with a single Edge Router (6LBR). Nodes on the subnet can use link-layer encryption to prevent rogue nodes from joining. The 6LBR can be configured so that it blocks any incoming (6LoWPAN-bound) IP multicast traffic. Another example topology could be a multi-subnet 6LoWPAN in a large conference room. In this case, the backbone can implement port authentication (IEEE 802.1X) to ensure only authorized devices can join the Ethernet backbone. The access router to this secured network segment can also be configured to block incoming IP multicast traffic.

5.3.3. Future Evolution

In the future, to further mitigate the threats, the developing approach for DTLS-based IP multicast security for CoAP networks (see [I-D.keoh-tls-multicast-security]) or similar approaches should be considered once they mature.

6. IANA Considerations

6.1. New 'core.gp' Resource Type

This memo registers a new resource type (rt) from the CoRE Parameters Registry called 'core.gp'.

(Note to IANA/RFC Editor: This registration follows the process described in section 7.4 of [RFC6690]).

Attribute Value: core.gp

Description: Group Configuration resource. This resource is used to query/manage the group membership of a CoAP server.

Reference: See Section 2.6.2.

6.2. New 'coap-group+json' Internet Media Type

This memo registers a new Internet Media Type for CoAP group configuration resource called 'application/coap-group+json'.

(Note to IANA/RFC Editor: This registration follows the guidance from [RFC6839], and (last paragraph) of section 12.3 of [I-D.ietf-core-coap].

Type name: application

Subtype name: coap-group+json

Required parameters: None

Optional parameters: None

Encoding considerations: 8bit UTF-8.

JSON to be represented using UTF-8 which is 8bit compatible (and most efficient for resource constrained implementations).

Security considerations:

Denial of Service attacks could be performed by constantly setting the group configuration resource of a CoAP endpoint to different values. This will cause the endpoint to register (or de-register) from the related IP multicast group. To prevent this it is recommended that DTLS-secured (unicast) CoAP communication be used for setting the group configuration resource. Thus only authorized clients will be allowed by a server to configure its group membership.

Interoperability considerations: None

Published specification: (This I-D when it becomes an RFC)

Applications that use this media type:

CoAP client and server implementations that wish to set/read the group configuration resource via 'application/coap-group+json' payload as described in Section 2.6.2.

Additional Information:

Magic number(s): None

File extension(s): *.json

Macintosh file type code(s): TEXT

Intended usage: COMMON

Restrictions on usage: None

Author: CoRE WG

Change controller: IETF

7. Acknowledgements

Thanks to Peter Bigot, Carsten Bormann, Anders Brandt, Angelo Castellani, Bjoern Hoehrmann, Matthias Kovatsch, Guang Lu, Salvatore Loreto, Kerry Lynn, Andrew McGregor, Dale Seed, Zach Shelby, Peter van der Stok, and Juan Carlos Zuniga for their helpful comments and discussions that have helped shape this document.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, October 2002.
- [RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.
- [RFC3810] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.
- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5771] Cotton, M., Vegoda, L., and D. Meyer, "IANA Guidelines for IPv4 Multicast Address Assignments", BCP 51, RFC 5771, March 2010.

- [RFC6092] Woodyatt, J., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, January 2011.
- [RFC6550] Winter, T., Thubert, P., 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, March 2012.
- [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, May 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.
- [RFC6839] Hansen, T. and A. Melnikov, "Additional Media Type Structured Syntax Suffixes", RFC 6839, January 2013.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

8.2. Informative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-12 (work in progress), June 2013.
- [I-D.vanderstok-core-dna]
Stok, P., Lynn, K., and A. Brandt, "CoRE Discovery, Naming, and Addressing", draft-vanderstok-core-dna-02 (work in progress), July 2012.
- [I-D.ietf-roll-trickle-mcast]
Hui, J. and R. Kelsey, "Multicast Protocol for Low power and Lossy Networks (MPL)", draft-ietf-roll-trickle-mcast-05 (work in progress), August 2013.
- [I-D.keoh-tls-multicast-security]

Keoh, S., Kumar, S., and E. Dijk, "DTLS-based Multicast Security for Low-Power and Lossy Networks (LLNs)", draft-keoh-tls-multicast-security-00 (work in progress), October 2012.

[I-D.ietf-core-resource-directory]

Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-ietf-core-resource-directory-00 (work in progress), June 2013.

[I-D.ietf-appsawg-uri-get-off-my-lawn]

Nottingham, M., "Standardising Structure in URIs", draft-ietf-appsawg-uri-get-off-my-lawn-00 (work in progress), September 2013.

Appendix A. Multicast Listener Discovery (MLD)

In order to extend the scope of IP multicast beyond link-local scope, an IP multicast routing or forwarding protocol has to be active in routers on an LLN. To achieve efficient multicast routing (i.e., avoid always flooding IP multicast packets), routers have to learn which hosts need to receive packets addressed to specific IP multicast destinations.

The Multicast Listener Discovery (MLD) protocol [RFC3810] (or its IPv4 equivalent IGMP [RFC3376]) is today the method of choice used by an (IP multicast enabled) router to discover the presence of multicast listeners on directly attached links, and to discover which multicast addresses are of interest to those listening nodes. MLD was specifically designed to cope with fairly dynamic situations in which multicast listeners may join and leave at any time.

[RFC6636] discusses optimal tuning of the parameters of MLD/IGMP for routers for mobile and wireless networks. These guidelines may be useful when implementing MLD in LLNs.

Appendix B. Change Log

[Note to RFC Editor: Please remove this section before publication.]

Changes from ietf-15 to ietf-16:

- o In section 2.6.2, changed DELETE in group management interface to a PUT with empty JSON array to clear the list (#345).
- o In section 2.6.2, aligned the syntax for IP addresses to follow RFC 3986 URI syntax, which is also used by coap-18. This allows re-use of the parsing code for CoAP URIs for this purpose (#342).

- o Addressed some more editorial comments provided by Carsten Bormann in preparation for WGLC.
- o Various editorial updates for improved readability.

Changes from ietf-14 to ietf-15:

- o In section 2.2, provided guidance on how implementers should parse URIs for group communication (#339).
- o In section 2.6.2.1, specified that for group membership configuration interface the "ip" (i.e. "a" parameter) key/value is not required when it is unknown (#338).
- o In section 2.6.2.1, specified that for group membership configuration interface the port configuration be defaulted to standard CoAP port 5683, and if not default then should follow standard notation (#340).
- o In section 2.6.2.1, specified that notation of IP address in group membership configuration interface should follow standard notation (#342).
- o In section 6.2, "coap-group+json" Media Type encoding simplified to just support UTF-8 (and not UTF-16 and UTF-32) (#344).
- o Various editorial updates for improved readability.

Changes from ietf-13 to ietf-14:

- o Update to address final editorial comments from the Chair's review (by Carsten Bormann) of the draft. This included restructuring of Section 2.6 (Configuring Group Memberships) and Section 4 (Deployment Guidelines) to make it easier to read. Also various other editorial changes.
- o Changed "ip" field to "a" in Section 2.6 (#337)

Changes from ietf-12 to ietf-13:

- o Extensive editorial updates due to comments from the Chair's review (by Carsten Bormann) of the draft. The best way to see the changes will be to do a -Diff with Rev. 12.
- o The technical comments from the Chair's review will be addressed in a future revision after tickets are generated and the solutions are agreed to on the WG E-mail list.

Changes from ietf-11 to ietf-12:

- o Removed reference to "CoAP Ping" in Section 3.5 (Group Member Discovery) and replaced it with the more efficient support of discovery of groups and group members via the CORE RD as suggested by Zach Shelby.
- o Various editorial updates for improved readability.

Changes from ietf-10 to ietf-11:

- o Added text to section 3.8 (Congestion Control) to clarify that a "CoAP client sending a multicast CoAP request to /.well-known/core SHOULD support core-block" (#332).
- o Various editorial updates for improved readability.

Changes from ietf-09 to ietf-10:

- o Various editorial updates including:
- o Added a fourth option in section 3.3 on ways to obtain the URI path for a group request.
- o Clarified use of content format in GET/PUT requests for Configuring Group Membership in Endpoints (in section 3.6).
- o Changed reference "draft-shelby-core-resource-directory" to "draft-ietf-core-resource-directory".
- o Clarified (in section 3.7) that ACKs are never used for a multicast request (from #296).
- o Clarified (in section 5.2/5.2.3) that MPL does not support group membership advertisement.
- o Adding introductory paragraph to Scope (section 2.2).
- o Wrote out fully the URIs in table section 3.2.
- o Reworded security text in section 7.2 (New Internet Media Type) to make it consistent with section 3.6 (Configuring Group Membership).
- o Fixed formatting of hyperlinks in sections 6.3 and 7.2.

Changes from ietf-08 to ietf-09:

- o Cleaned up requirements language in general. Also, requirements language are now only used in section 3 (Protocol Considerations) and section 6 (Security Considerations). Requirements language has been removed from other sections to keep them to a minimum (#271).
- o Addressed final comment from Peter van der Stok to define what "IP stack" meant (#296). Following the lead of CoAP-17, we now refer instead to "APIs such as IPV6_RECVPKTINFO [RFC3542]".
- o Changed text in section 3.4 (Group Methods) to allow multicast POST under specific conditions and highlighting the risks with using it (#328).
- o Various editorial updates for improved readability.

Changes from ietf-07 to ietf-08:

- o Updated text in section 3.6 (Configuring Group Membership in Endpoints) to make it more explicit that the Internet Media Type is used in the processing rules (#299).
- o Addressed various comments from Peter van der Stok (#296).
- o Various editorial updates for improved readability including defining all acronyms.

Changes from ietf-06 to ietf-07:

- o Added an IANA request (in section 7.2) for a dedicated content-format (Internet Media type) for the group management JSON format called 'application/coap-group+json' (#299).
- o Clarified semantics (in section 3.6) of group management JSON format (#300).
- o Added details of IANA request (in section 7.1) for a new CORE Resource Type called 'core.gp'.
- o Clarified that DELETE method (in section 3.6) is also a valid group management operation.
- o Various editorial updates for improved readability.

Changes from ietf-05 to ietf-06:

- o Added a new section on commissioning flow when using discovery services when end devices discover in which multicast group they are allocated (#295).
- o Added a new section on CoAP Proxy Operation (section 3.9) that outlines the potential issues and limitations of doing CoAP multicast requests via a CoAP Proxy (#274).
- o Added use case of multicasting controller on the backbone (#279).
- o Use cases were updated to show only a single CoAP RD (to replace the previous multiple RDs with one in each subnet). This is a more efficient deployment and also avoids RD specific issues such as synchronization of RD information between serves (#280).
- o Added text to section 3.6 (Configuring Group Membership in Endpoints) that clarified that any (unicast) operation to change an endpoint's group membership must use DTLS-secured CoAP.
- o Clarified relationship of this document to [I-D.ietf-core-coap] in section 2.2 (Scope).
- o Removed IPSec related requirement, as IPSec is not part of [I-D.ietf-core-coap] anymore.
- o Editorial reordering of subsections in section 3 to have a better flow of topics. Also renamed some of the (sub)sections to better reflect their content. Finally, moved the URI Configuration text to the same section as the Port Configuration section as it was a more natural grouping (now in section 3.3) .
- o Editorial rewording of section 3.7 (Multicast Request Acceptance and Response Suppression) to make the logic easier to comprehend (parse).
- o Various editorial updates for improved readability.

Changes from ietf-04 to ietf-05:

- o Added a new section 3.9 (Exceptions) that highlights that IP multicast (and hence group communication) is not always available (#187).
- o Updated text on the use of [RFC2119] language (#271) in Section 1.
- o Included guidelines on when (not) to use CoAP responses to multicast requests and when (not) to accept multicast requests (#273).

- o Added guideline on use of core-block for minimizing response size (#275).
- o Restructured section 6 (Security Considerations) to more fully describe threats and threat mitigation (#277).
- o Clearly indicated that DNS resolution and reverse DNS lookup are optional.
- o Removed confusing text about a single group having multiple IP addresses. If multiple IP addresses are required then multiple groups (with the same members) should be created.
- o Removed repetitive text about the fact that group communication is not guaranteed.
- o Merged previous section 5.2 (Multicast Routing) into 3.1 (IP Multicast Routing Background) and added link to section 5.2 (Advertising Membership of Multicast Groups).
- o Clarified text in section 3.8 (Congestion Control) regarding precedence of use of IP multicast domains (i.e. first try to use link-local scope, then site-local scope, and only use global IP multicast as a last resort).
- o Extended group resource manipulation guidelines with use of pre-configured ports/paths for the multicast group.
- o Consolidated all text relating to ports in a new section 3.3 (Port Configuration).
- o Clarified that all methods (GET/PUT/POST) for configuring group membership in endpoints should be unicast (and not multicast) in section 3.7 (Configuring Group Membership In Endpoints).
- o Various editorial updates for improved readability, including editorial comments by Peter van der Stok to WG list of December 18th, 2012.

Changes from ietf-03 to ietf-04:

- o Removed section 2.3 (Potential Solutions for Group Communication) as it is purely background information and moved section to draft-dijk-core-groupcomm-misc (#266).
- o Added reference to draft-keoh-tls-multicast-security to section 6 (Security Considerations).

- o Removed Appendix B (CoAP-Observe Alternative to Group Communications) as it is as an alternative to IP Multicast that the WG has not adopted and moved section to draft-dijk-core-groupcomm-misc (#267).
- o Deleted section 8 (Conclusions) as it is redundant (#268).
- o Simplified light switch use case (#269) by splitting into basic operations and additional functions (#269).
- o Moved section 3.7 (CoAP Multicast and HTTP Unicast Interworking) to draft-dijk-core-groupcomm-misc (#270).
- o Moved section 3.3.1 (DNS-SD) and 3.3.2 (CoRE Resource Directory) to draft-dijk-core-groupcomm-misc as these sections essentially just repeated text from other drafts regarding DNS based features. Clarified remaining text in this draft relating to DNS based features to clearly indicate that these features are optional (#272).
- o Focus section 3.5 (Configuring Group Membership) on a single proposed solution.
- o Scope of section 5.3 (Use of MLD) widened to multicast destination advertisement methods in general.
- o Rewrote section 2.2 (Scope) for improved readability.
- o Moved use cases that are not addressed to draft-dijk-core-groupcomm-misc.
- o Various editorial updates for improved readability.

Changes from ietf-02 to ietf-03:

- o Clarified that a group resource manipulation may return back a mixture of successful and unsuccessful responses (section 3.4 and Figure 6) (#251).
- o Clarified that security option for group communication must be NoSec mode (section 6) (#250).
- o Added mechanism for group membership configuration (#249).
- o Removed IANA request for multicast addresses (section 7) and replaced with a note indicating that the request is being made in [I-D.ietf-core-coap] (#248).

- o Made the definition of 'group' more specific to group of CoAP endpoints and included text on UDP port selection (#186).
- o Added explanatory text in section 3.4 regarding why not to use group communication for non-idempotent messages (i.e. CoAP POST) (#186).
- o Changed link-local RD discovery to site-local in RD discovery use case to make it more realistic.
- o Fixed lighting control use case CoAP proxying; now returns individual CoAP responses as in coap-12.
- o Replaced link format I-D with RFC6690 reference.
- o Various editorial updates for improved readability

Changes from ietf-01 to ietf-02:

- o Rewrote congestion control section based on latest CoAP text including Leisure concept (#188)
- o Updated the CoAP/HTTP interworking section and example use case with more details and use of MLD for multicast group joining
- o Key use cases added (#185)
- o References to [I-D.vanderstok-core-dna] and draft-castellani-core-advanced-http-mapping added
- o Moved background sections on "MLD" and "CoAP-Observe" to Appendices
- o Removed requirements section (and moved it to draft-dijk-core-groupcomm-misc)
- o Added details for IANA request for group communication multicast addresses
- o Clarified text to distinguish between "link local" and general multicast cases
- o Moved lengthy background section 5 to draft-dijk-core-groupcomm-misc and replaced with a summary
- o Various editorial updates for improved readability
- o Change log added

Changes from ietf-00 to ietf-01:

- o Moved CoAP-observe solution section to section 2
- o Editorial changes
- o Moved security requirements into requirements section
- o Changed multicast POST to PUT in example use case
- o Added CoAP responses in example use case

Changes from rahman-07 to ietf-00:

- o Editorial changes
- o Use cases section added
- o CoRE Resource Directory section added
- o Removed section 3.3.5. IP Multicast Transmission Methods
- o Removed section 3.4 Overlay Multicast
- o Removed section 3.5 CoAP Application Layer Group Management
- o Clarified section 4.3.1.3 RPL Routers with Non-RPL Hosts case
- o References added and some normative/informative status changes

Authors' Addresses

Akbar Rahman (editor)
InterDigital Communications, LLC
Email: Akbar.Rahman@InterDigital.com

Esko Dijk (editor)
Philips Research
Email: esko.dijk@philips.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: April 15, 2014

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
KoanLogic
E. Dijk
Philips Research
October 12, 2013

Guidelines for HTTP-CoAP Mapping Implementations
draft-ietf-core-http-mapping-02

Abstract

This draft provides reference information for HTTP-CoAP protocol translation proxy implementors, focusing primarily on the reverse proxy case. It details deployment options, defines a safe method for URI mapping, and provides a set of guidelines and considerations related to protocol translation.

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 <http://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 April 15, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Cross-Protocol Usage of URIs	4
4. Use Cases	5
5. Default HTTP to CoAP URI Mapping	5
5.1. Introduction	5
5.2. URI Mapping Template	6
5.3. Examples	6
6. HTTP-CoAP Reverse Proxy	7
6.1. Proxy Placement	8
6.2. Response Code Translations	9
6.3. Media Type Translations	11
6.4. Caching and Congestion Control	12
6.5. Cache Refresh via Observe	12
6.6. Use of CoAP Blockwise Transfer	13
6.7. Security Translation	14
6.8. Other guidelines	14
7. IANA Considerations	15
8. Security Considerations	15
8.1. Traffic overflow	15
8.2. Handling Secured Exchanges	16
9. Acknowledgements	16
10. References	17
10.1. Normative References	17
10.2. Informative References	18
Appendix A. Change Log	18
Authors' Addresses	18

1. Introduction

CoAP [I-D.ietf-core-coap] has been designed with the twofold aim to be an application protocol specialized for constrained environments and to be easily used in REST architectures such as the Web. The latter goal has led to define CoAP to easily interoperate with HTTP [RFC2616] through an intermediary proxy which performs cross-protocol conversion.

Section 10 of [I-D.ietf-core-coap] describes the fundamentals of the CoAP-to-HTTP and the HTTP-to-CoAP cross-protocol mapping process. However, implementing such a cross-protocol proxy can be complex, and many details regarding its internal procedures and design choices require further elaboration. Therefore a first goal of this document is to provide more detailed information to proxy designers and implementers, to help implement proxies that correctly inter-work with other CoAP and HTTP client/server implementations that adhere to the HTTP and CoAP specifications.

The second goal of this informational document is to define a consistent set of guidelines that a HTTP-to-CoAP proxy implementation MAY adhere to. The main reason of adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability. (As an example use case, a proxy conforming to these guidelines made by vendor A can be easily replaced by a proxy from vendor B that also conforms to the guidelines.)

This draft is organized as follows:

- o Section 2 describes terminology to identify proxy types, mapping approaches and proxy deployments;
- o Section 3 discusses how URIs refer to resources independent of access protocols;
- o Section 4 briefly lists use cases in which HTTP clients need to contact CoAP servers;
- o Section 5 introduces a default HTTP-to-CoAP URI mapping syntax;
- o Section 6 analyzes the mapping that allows HTTP clients to contact CoAP servers;
- o Section 8 discusses possible security impact related to HTTP-CoAP cross-protocol mapping.

2. Terminology

This document assumes readers are familiar with the terms Reverse Proxy as defined in [I-D.ietf-httpbis-pl-messaging] and Interception Proxy as defined in [RFC3040]. In addition, the following terms are defined:

Cross-Protocol Proxy (or Cross Proxy): is a proxy performing a cross-protocol mapping, in the context of this document a HTTP-CoAP (HC) mapping. A Cross-Protocol Proxy can behave as a Forward Proxy, Reverse Proxy or Interception Proxy. Note: In this document we focus on the Reverse Proxy mode of the Cross-Protocol Proxy.

Forward Proxy: a message forwarding agent that is selected by the client, usually via local configuration rules, to receive requests for some type(s) of absolute URI and to attempt to satisfy those requests via translation to the protocol indicated by the absolute URI. The user decides (is willing to) use the proxy as the forwarding/dereferencing agent for a predefined subset of the URI space.

Reverse Proxy: a receiving agent that acts as a layer above some other server(s) and translates the received requests to the underlying server's protocol. It behaves as an origin (HTTP) server on its connection towards the (HTTP) client and as a (CoAP) client on its connection towards the (CoAP) origin server. The (HTTP) client uses the "origin-form" [I-D.ietf-httpbis-pl-messaging] as a request-target URI.

Reverse and Forward proxies are technically very similar, with main differences being that the former appears to a client as an origin server while the latter does not, and that clients may be unaware they are communicating with a proxy.

Placement terms: a server-side (SS) proxy is placed in the same network domain as the server; conversely a client-side (CS) proxy is in the same network domain as the client. In any other case than SS or CS, the proxy is said to be External (E).

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 [RFC2119].

3. Cross-Protocol Usage of URIs

A Uniform Resource Identifier (URI) provides a simple and extensible method for identifying a resource. It enables uniform identification of resources via a separately defined extensible set of naming schemes [RFC3986].

URIs are formed of at least three components: scheme, authority and path. The scheme often corresponds to the protocol used to access the resource. However, as noted in Section 1.2.2 of [RFC3986] the scheme does not imply that a particular protocol is used to access

the resource. So, we can define the same resource to be accessible by different protocols i.e. the resource can have cross-protocol URIs referring to it.

HTTP clients typically only support 'http' and 'https' schemes. Therefore, they cannot directly access CoAP servers (which support 'coap' and/or 'coaps'). In this situation, communication is enabled by a Cross-Protocol Proxy, as shown in Figure 1, supporting URI mapping features. Such features are discussed in Section 5.

4. Use Cases

To illustrate in which situations HTTP to CoAP request mapping may be used, three use cases are briefly described.

1. Smartphone and home sensor: A smartphone, when at home, can perform 'coap' requests directly to a home sensor via WiFi. When the smartphone is away from home, the same request is done by an authenticated 'https' request over an external IP network to the home router. The home router contains a HTTP-CoAP proxy.

2. Legacy building control application without CoAP: A building control application can use HTTP, but not CoAP. It checks the status of sensors or actuators via a HTTP-CoAP proxy.

3. Making sensor data available to 3rd parties: For demonstration or public interest purposes, a HTTP-CoAP proxy may be configured to expose the contents of a sensor to the world via the web (HTTP and/or HTTPS). The sensor can only handle secure 'coaps' requests, therefore the proxy is configured to translate any request to a 'coaps' secured request. The proxy is furthermore configured to only pass through GET requests.

5. Default HTTP to CoAP URI Mapping

5.1. Introduction

This section defines a default URI mapping function that is RECOMMENDED to be implemented and activated by default in a HTTP-to-CoAP proxy, unless there are reasons (e.g. application-specific) to use other mapping function(s).

From a high level viewpoint, the URI mapping is implemented in a HTTP client by appending the CoAP URI to a HTTP proxy base URI. For example: a base URI may be `http://p.example.com/.well-known/core/` and the CoAP URI may be `coap://s.coap.example.com/foo`. The mapping then embeds the COAP URI into a HTTP URI by appending it, as follows:

`http://p.example.com/.well-known/core/coap://s.coap.example.com/foo`

A formal definition of this function and more examples are provided below.

5.2. URI Mapping Template

The URI template is one of the following two expressions, using the notation of [RFC6570].

- o `{+proxy-origin}/.well-known/core/{scheme}://{+authority}{+path-absolute}`
- o `{+proxy-origin}/.well-known/core/{scheme}://{+authority}{+path-absolute}{+?query}`

Where:

- o `proxy-origin` identifies the proxy HTTP side as usual scheme `://"` authority;
- o `scheme` is the scheme of the embedded CoAP URI, either `'coap'` or `'coaps'`;
- o `authority` is the CoAP URI authority. If the host is in IPv6address format, then the `'['` and `']'` characters MUST be percent-encoded, in order to comply with the syntax defined in Section 3.3. of [RFC3986] for a URI path segment;
- o `path-absolute` (defined by [RFC3986]) is the absolute CoAP URI path;
- o `query` (defined by [RFC3986]) is the optional query component of the CoAP URI.

5.3. Examples

Each example below shows a HTTP URI that is used by a HTTP client to make a request to a HTTP-CoAP proxy `http://p.example.com`. The CoAP URI shown below is the URI that is extracted from the HTTP URI by the proxy, using the default URI mapping scheme introduced earlier.

Example 1:

- o `http://p.example.com/.well-known/core/coap://s.coap.example.com:4567/foo/bar?a=3`
- o `coap://s.coap.example.com:4567/foo/bar?a=3`

Example 2:

- o `http://p.example.com/.well-known/core/coap://%5B2001:db8::1%5D:4567/foo/bar?a=3`
- o `coap://[2001:db8::1]:4567/foo/bar?a=3`

Example 3:

- o `http://p.example.com/.well-known/core/coaps://s.coap.example.com/foo(bar)?a=3`
- o `coaps://s.coap.example.com/foo(bar)?a=3`

Example 4:

- o `http://p.example.com/.well-known/core/coap://1.2.3.4:4567/foo/bar?a=3&key=value`
- o `coap://1.2.3.4:4567/foo/bar?a=3&key=value`

Example 5:

- o `http://p.example.com/.well-known/core/coap://%5B2001:db8::1%5D/foo%5Bbar%5D?a=3/5/7`
- o `coap://[2001:db8::1]/foo%5Bbar%5D?a=3/5/7`

6. HTTP-CoAP Reverse Proxy

A HTTP-CoAP Reverse Cross-Protocol Proxy is accessed by web clients only supporting HTTP, and handles their requests by mapping these to CoAP requests, which are forwarded to CoAP servers; and mapping back the received CoAP responses to HTTP. This mechanism is transparent to the client, which may assume that it is communicating with the intended target HTTP server. In other words, the client accesses the proxy as an origin server using the "origin-form" [I-D.ietf-httpbis-pl-messaging] as a Request Target.

Normative requirements on the translation of HTTP requests to CoAP and of the CoAP responses back to HTTP responses are defined in Section 10.2 of [I-D.ietf-core-coap]. However, that section only considers the case of a HTTP-CoAP Forward Cross-Protocol Proxy in which a client explicitly indicates it targets a request to a CoAP server, and does not cover all aspects of proxy implementation in detail. The present section provides guidelines and more details for the implementation of a Reverse Cross-Protocol Proxy, which MAY be followed in addition to the normative requirements.

Translation of unicast HTTP requests into multicast CoAP requests is currently out of scope since in a reverse proxy scenario a HTTP client typically expects to receive a single response, not multiple. However a Cross-Protocol Proxy MAY include custom application-specific functions to generate a multicast CoAP request based on a unicast HTTP request and aggregate multiple CoAP responses into a single HTTP response.

Note that the guidelines in this section also apply to an HTTP-CoAP Intercepting Cross-Protocol Proxy.

6.1. Proxy Placement

Typically, a Cross-Protocol Proxy is located at the edge of the constrained network. See Figure 1. The arguments supporting server-side (SS) placement are the following:

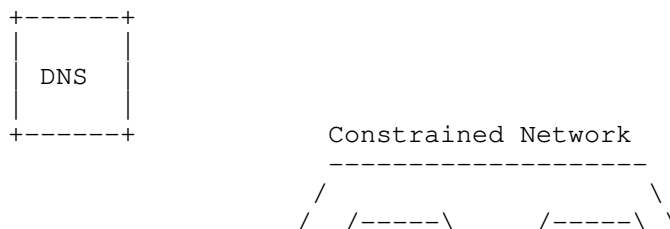
Caching: Efficient caching requires that all request traffic to a CoAP server is handled by the same proxy which receives HTTP requests from multiple source locations. This maximally reduces the load on (constrained) CoAP servers.

Multicast: To support CoAPs use of local-multicast functionality available in a constrained network, the Cross-Protocol Proxy requires a network interface directly attached to the constrained network.

TCP/UDP: Translation between HTTP and CoAP requires also TCP/UDP translation; TCP may be the preferred way for communicating with the constrained network due to its reliability or due to intermediate gateways configured to block UDP traffic.

Arguments against SS placement, in favor of client-side (CS), are:

Scalability: A solution where a single SS proxy has to manage numerous open TCP/IP connections to a large number of HTTP clients is not scalable. (Unless multiple SS proxies are employed with a load-balancing mechanism, which adds complexity.)



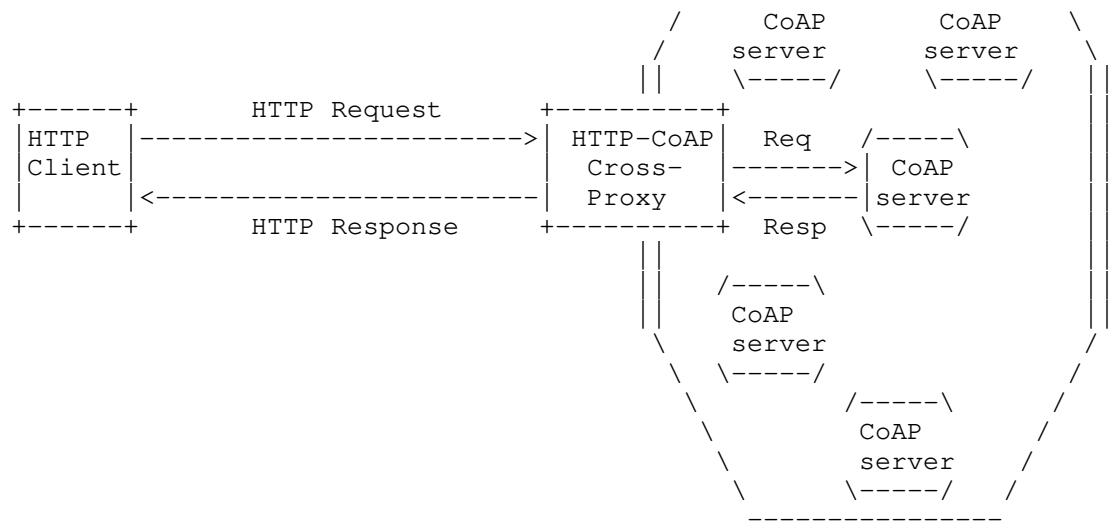


Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

6.2. Response Code Translations

Table 1 defines all possible CoAP responses along with the HTTP response to which each CoAP response SHOULD be translated. This table complies with the Section 10.2 requirements of [I-D.ietf-core-coap] and is intended to cover all possible cases. Multiple appearances of a HTTP status code in the second column indicates multiple equivalent HTTP responses are possible, depending on the conditions cited in the Notes (third column).

CoAP Response Code	HTTP Status Code	Notes
2.01 Created	201 Created	1
2.02 Deleted	200 OK	2
	204 No Content	2
2.03 Valid	304 Not Modified	3
	200 OK	4
2.04 Changed	200 OK	2
	204 No Content	2
2.05 Content	200 OK	
4.00 Bad Request	400 Bad Request	
4.01 Unauthorized	400 Bad Request	5
4.02 Bad Option	400 Bad Request	6
4.03 Forbidden	403 Forbidden	
4.04 Not Found	404 Not Found	

4.05 Method Not Allowed	400 Bad Request	7
4.06 Not Acceptable	406 Not Acceptable	
4.12 Precondition Failed	412 Precondition Failed	
4.13 Request Entity Too Large	413 Request Repr. Too Large	
4.15 Unsupported Media Type	415 Unsupported Media Type	
5.00 Internal Server Error	500 Internal Server Error	
5.01 Not Implemented	501 Not Implemented	
5.02 Bad Gateway	502 Bad Gateway	
5.03 Service Unavailable	503 Service Unavailable	8
5.04 Gateway Timeout	504 Gateway Timeout	
5.05 Proxying Not Supported	502 Bad Gateway	9

Table 1: HTTP-CoAP Response Mapping

Notes:

1. A CoAP server may return an arbitrary format payload along with this response. This payload SHOULD be returned as entity in the HTTP 201 response. Section 7.3.2 of [I-D.ietf-httpbis-p2-semantics] does not put any requirement on the format of the payload. (In the past, [RFC2616] did.)
2. The HTTP code is 200 or 204 respectively for the case that a CoAP server returns a payload or not. [I-D.ietf-httpbis-p2-semantics] Section 5.3 requires code 200 in case a representation of the action result is returned for DELETE, POST and PUT and code 204 if not. Hence, a proxy SHOULD transfer any CoAP payload contained in a 2.02 response to the HTTP client in a 200 OK response.
3. A CoAP 2.03 (Valid) response only (1) confirms that the request ETag is valid and (2) provides a new Max-Age value. HTTP 304 (Not Modified) also updates some header fields of a stored response. A non-caching proxy may not have enough information to fill in the required values in the HTTP 304 (Not Modified) response, so it may not be advisable for a non-caching proxy to provoke the 2.03 (Valid) response by forwarding an ETag. A caching proxy will fill the information out of the cache.

4. A 200 response to a CoAP 2.03 occurs only when the proxy is caching and translated a HTTP request (without validation request) to a CoAP request that includes validation, for efficiency. The proxy receiving 2.03 updates the freshness of the cached representation and returns the entire representation to the HTTP client.
5. The HTTP code 401 Unauthorized MUST NOT be used, as long as in CoAP there is no equivalent defined of the required WWW-Authenticate header (Section 3.1 of [I-D.ietf-httpbis-p7-auth]).
6. In some cases a proxy receiving 4.02 may retry the request with less CoAP Options in the hope that the server will understand the newly formulated request. For example, if the proxy tried using a Block Option which was not recognized by the CoAP server it may retry without that Block Option.
7. The HTTP code "405 Method Not Allowed" MUST NOT be used since CoAP does not provide enough information to determine a value for the required "Allow" response-header field.
8. The value of the HTTP "Retry-After" response-header field is taken from the value of the CoAP Max-Age Option, if present.
9. This CoAP response can only happen if the proxy itself is configured to use a CoAP Forward Proxy to execute some, or all, of its CoAP requests.

6.3. Media Type Translations

A Cross-Protocol Proxy translates a media type string, carried in a HTTP Content-Type header in a request, to a CoAP Content-Format Option with the equivalent numeric value. The media types supported by CoAP are defined in the CoAP Content-Format Registry. Any HTTP request with a Content-Type for which the proxy does not know an equivalent CoAP Content-Format number, MUST lead to HTTP response 415 (Unsupported Media Type).

Also, a CoAP Content-Format value in a response is translated back to the equivalent HTTP Content-Type. If a proxy receives a CoAP Content-Format value that it does not recognize (e.g. because the value is IANA-registered after the proxy software was deployed), and is unable to look up the equivalent HTTP Content-Type on the fly, the proxy SHOULD return an HTTP entity (payload) without Content-Type header (complying to Section 3.1.1.5 of [I-D.ietf-httpbis-p2-semantics]).

6.4. Caching and Congestion Control

A Cross-Protocol Proxy SHOULD limit the number of requests to CoAP servers by responding, where applicable, with a cached representation of the resource.

Duplicate idempotent pending requests by a Cross-Protocol Proxy to the same CoAP resource SHOULD in general be avoided, by duplexing the response to the requesting HTTP clients without duplicating the CoAP request.

If the HTTP client times out and drops the HTTP session to the Cross-Protocol Proxy (closing the TCP connection) after the HTTP request was made, a Cross-Protocol Proxy SHOULD wait for the associated CoAP response and cache it if possible. Further requests to the Cross-Protocol Proxy for the same resource can use the result present in cache, or, if a response has still to come, the HTTP requests will wait on the open CoAP session.

According to [I-D.ietf-core-coap], a proxy MUST limit the number of outstanding interactions to a given CoAP server to NSTART. To limit the amount of aggregate traffic to a constrained network, the Cross-Protocol Proxy SHOULD also pose a limit to the number of concurrent CoAP requests pending on the same constrained network; further incoming requests MAY either be queued or dropped (returning 503 Service Unavailable). This limit and the proxy queueing/dropping behavior SHOULD be configurable. In order to efficiently apply this congestion control, the Cross-Protocol Proxy SHOULD be SS placed.

Resources experiencing a high access rate coupled with high volatility MAY be observed [I-D.ietf-core-observe] by the Cross-Protocol Proxy to keep their cached representation fresh while minimizing the number CoAP messages. See Section 6.5.

6.5. Cache Refresh via Observe

There are cases where using the CoAP observe protocol [I-D.ietf-core-observe] to handle proxy cache refresh is preferable to the validation mechanism based on ETag as defined in [I-D.ietf-core-coap]. Such scenarios include, but are not limited to, sleepy nodes -- with possibly high variance in requests' distribution -- which would greatly benefit from a server driven cache update mechanism. Ideal candidates would also be crowded or very low throughput networks, where reduction of the total number of exchanged messages is an important requirement.

This subsection aims at providing a practical evaluation method to decide whether the refresh of a cached resource R is more efficiently handled via ETag validation or by establishing an observation on R.

Let T_R be the mean time between two client requests to resource R, let F_R be the freshness lifetime of R representation, and let M_R be the total number of messages exchanged towards resource R. If we assume that the initial cost for establishing the observation is negligible, an observation on R reduces M_R iff $T_R < 2 * F_R$ with respect to using ETag validation, that is iff the mean arrival time of requests for resource R is greater than half the refresh rate of R.

When using observations M_R is always upper bounded by $2 * F_R$: in the constrained network no more than $2 * F_R$ messages will be generated towards resource R.

6.6. Use of CoAP Blockwise Transfer

A Cross-Protocol Proxy SHOULD support CoAP blockwise transfers [I-D.ietf-core-block] to allow transport of large CoAP payloads while avoiding excessive link-layer fragmentation in LLNs, and to cope with small datagram buffers in CoAP end-points as described in [I-D.ietf-core-coap] Section 4.6.

A Cross-Protocol Proxy SHOULD attempt to retry a payload-carrying CoAP PUT or POST request with blockwise transfer if the destination CoAP server responded with 4.13 (Request Entity Too Large) to the original request. A Cross-Protocol Proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than a value `BLOCKWISE_THRESHOLD`. The value of `BLOCKWISE_THRESHOLD` MAY be implementation-specific, for example calculated based on a known or typical UDP datagram buffer size for CoAP end-points, or set to N times the size of a link-layer frame where e.g. $N=5$, or preset to a known IP MTU value, or set to a known Path MTU value. The value `BLOCKWISE_THRESHOLD` or parameters from which it is calculated SHOULD be configurable in a proxy implementation.

The Cross-Protocol Proxy SHOULD detect CoAP end-points not supporting blockwise transfers by checking for a 4.02 (Bad Option) response returned by an end-point in response to a CoAP request with a Block* Option. This allows the Cross-Protocol Proxy to be more efficient, not attempting repeated blockwise transfers to CoAP servers that do not support it. However if a request payload is too large to be sent as a single CoAP request and blockwise transfer would be unavoidable, the proxy still SHOULD attempt blockwise transfer on such an end-point before returning 413 (Request Entity Too Large) to the HTTP client.

For improved latency a cross proxy MAY initiate a blockwise CoAP request triggered by an incoming HTTP request even when the HTTP request message has not yet been fully received, but enough data has been received to send one or more data blocks to a CoAP server already. This is particularly useful on slow client-to-proxy connections.

6.7. Security Translation

A HC proxy SHOULD implement explicit rules for security context translations. A translation may involve e.g. applying a rule that any "https" request is translated to a "coaps" request, or e.g. applying a rule that a "https" request is translated to an unsecured "coap" request. Another rule could specify the security policy and parameters used for DTLS connections. Such rules will largely depend on the application and network context in which a proxy is applied. To enable widest possible use of a proxy implementation, these rules SHOULD be configurable in a HC proxy.

If a policy for access to 'coaps' URIs is configurable in a HC proxy, it is RECOMMENDED that the policy is by default configured to disallow access to any 'coaps' URI by a HTTP client using an unsecured (non-TLS) connection. Naturally, a user MAY reconfigure the policy to allow such access in specific cases.

6.8. Other guidelines

For long delays of a CoAP server, the HTTP client or any other proxy in between MAY timeout. Further discussion of timeouts in HTTP is available in Section 6.2.4 of [I-D.ietf-httpbis-pl-messaging].

A cross proxy MUST define an internal timeout for each pending CoAP request, because the CoAP server may silently die before completing the request. The timeout value SHOULD be approximately less than or equal to MAX_RTT defined in [I-D.ietf-core-coap].

When the DNS protocol is not used between CoAP nodes in a constrained network, defining valid FQDN (i.e., DNS entries) for constrained CoAP servers, where possible, MAY help HTTP clients to access the resources offered by these servers via a HC proxy.

HTTP connection pipelining (section 6.2.2.1 of [I-D.ietf-httpbis-pl-messaging]) MAY be supported by the proxy and is transparent to the CoAP network: the HC cross proxy will sequentially serve the pipelined requests by issuing different CoAP requests.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

The security concerns raised in Section 15.7 of [RFC2616] also apply to the cross proxy scenario. In fact, the cross proxy is a trusted (not rarely a transparently trusted) component in the network path.

The trustworthiness assumption on the cross proxy cannot be dropped. Even if we had a blind, bi-directional, end-to-end, tunneling facility like the one provided by the CONNECT method in HTTP, and also assuming the existence of a DTLS-TLS transparent mapping, the two tunneled ends should be speaking the same application protocol, which is not the case. Basically, the protocol translation function is a core duty of the cross proxy that can't be removed, and makes it a necessarily trusted, impossible to bypass, component in the communication path.

A reverse proxy deployed at the boundary of a constrained network is an easy single point of failure for reducing availability. As such, a special care should be taken in designing, developing and operating it, keeping in mind that, in most cases, it could have fewer limitations than the constrained devices it is serving.

The following sub paragraphs categorize and argue about a set of specific security issues related to the translation, caching and forwarding functionality exposed by a cross proxy module.

8.1. Traffic overflow

Due to the typically constrained nature of CoAP nodes, particular attention SHOULD be posed in the implementation of traffic reduction mechanisms (see Section 6.4), because inefficient implementations can be targeted by unconstrained Internet attackers. Bandwidth or complexity involved in such attacks is very low.

An amplification attack to the constrained network may be triggered by a multicast request generated by a single HTTP request mapped to a CoAP multicast resource, as considered in Section TBD of [I-D.ietf-core-coap].

The impact of this amplification technique is higher than an amplification attack carried out by a malicious constrained device (e.g. ICMPv6 flooding, like Packet Too Big, or Parameter Problem on a multicast destination [RFC4732]), since it does not require direct access to the constrained network.

The feasibility of this attack, disruptive in terms of CoAP server availability, can be limited by access controlling the exposed HTTP multicast resource, so that only known/authorized users access such URIs.

8.2. Handling Secured Exchanges

It is possible that the request from the client to the cross proxy is sent over a secured connection. However, there may or may not exist a secure connection mapping to the other protocol. For example, a secure distribution method for multicast traffic is complex and MAY not be implemented (see [I-D.ietf-core-groupcomm]).

By default, a cross proxy SHOULD reject any secured client request if there is no configured security policy mapping. This recommendation MAY be relaxed in case the destination network is believed to be secured by other, complementary, means. E.g.: assumed that CoAP nodes are isolated behind a firewall (e.g. as the SS cross proxy deployment shown in Figure 1), the cross proxy may be configured to translate the incoming HTTPS request using plain CoAP (i.e. NoSec mode.)

The HC URI mapping MUST NOT map to HTTP (see Section 5) a CoAP resource intended to be accessed only using HTTPS.

A secured connection that is terminated at the cross proxy, i.e. the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The cross proxy SHOULD NOT cache any secured content to avoid any leak of secured information. However in some specific scenario, a security/efficiency trade-off could motivate caching secured information; in that case the caching behavior MAY be tuned to some extent on a per-resource basis.

9. Acknowledgements

An initial version of the table found in Section 6.2 has been provided in revision -05 of [I-D.ietf-core-coap]. Special thanks to

Peter van der Stok for countless comments and discussions on this document, that contributed to its current structure and text.

Thanks to Carsten Bormann, Zach Shelby, Michele Rossi, Nicola Bui, Michele Zorzi, Klaus Hartke, Cullen Jennings, Kepeng Li, Brian Frank, Peter Saint-Andre, Kerry Lynn, Linyi Tian, Dorothy Gellert, Francesco Corazza for helpful comments and discussions that have shaped the document.

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n. [251557].

10. References

10.1. Normative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-12 (work in progress), June 2013.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-10 (work in progress), September 2013.
- [I-D.ietf-httpbis-p1-messaging]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", draft-ietf-httpbis-p1-messaging-24 (work in progress), September 2013.
- [I-D.ietf-httpbis-p2-semantics]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", draft-ietf-httpbis-p2-semantics-24 (work in progress), September 2013.
- [I-D.ietf-httpbis-p7-auth]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", draft-ietf-httpbis-p7-auth-24 (work in progress), September 2013.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

10.2. Informative References

- [I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-16 (work in progress), October 2013.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, January 2001.
- [RFC4732] Handley, M., Rescorla, E., IAB, "Internet Denial-of-Service Considerations", RFC 4732, December 2006.

Appendix A. Change Log

Changes from ietf-01 to ietf-02:

- o Selection of single default URI mapping proposal as proposed to WG mailing list 2013-10-09.

Changes from ietf-00 to ietf-01:

- o Added URI mapping proposals to Section 4 as per the Email proposals to WG mailing list from Esko.

Authors' Addresses

Angelo P. Castellani
University of Padova
Via Gradenigo 6/B
Padova 35131
Italy

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal H3A 3G4
Canada

Phone: +1 514 585 0761
Email: Akbar.Rahman@InterDigital.com

Thomas Fossati
KoanLogic

Email: tho@koanlogic.com

Esko Dijk
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
The Netherlands

Email: esko.dijk@philips.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 18, 2014

K. Hartke
Universitaet Bremen TZI
October 15, 2013

Observing Resources in CoAP
draft-ietf-core-observe-11

Abstract

CoAP is a RESTful application protocol for constrained nodes and networks. The state of a resource on a CoAP server can change over time. This document specifies a simple protocol extension for CoAP that enables CoAP clients to "observe" resources, i.e., to retrieve a representation of a resource and keep this representation updated by the server over a period of time. The protocol follows a best-effort approach for sending new representations to clients and provides eventual consistency between the state observed by each client and the actual resource state at the server.

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 <http://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 April 18, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Background	3
1.2. Protocol Overview	3
1.3. Observable Resources	5
1.4. Consistency	6
1.5. Requirements Notation	7
2. The Observe Option	7
3. Client-side Requirements	8
3.1. Request	8
3.2. Notifications	8
3.3. Caching	9
3.4. Reordering	10
3.5. Transmission	11
3.6. Cancellation	11
4. Server-side Requirements	11
4.1. Request	11
4.2. Notifications	12
4.3. Caching	12
4.4. Reordering	13
4.5. Transmission	14
5. Intermediaries	16
6. Web Linking	17
7. Security Considerations	17
8. IANA Considerations	18
9. Acknowledgements	18
10. References	18
10.1. Normative References	18
10.2. Informative References	19
Appendix A. Examples	19
A.1. Client/Server Examples	20
A.2. Proxy Examples	24
Appendix B. Changelog	26

1. Introduction

1.1. Background

CoAP [I-D.ietf-core-coap] is an application protocol for constrained nodes and networks. It is intended to provide RESTful services [REST] not unlike HTTP [RFC2616] while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

The model of REST is that of a client exchanging representations of resources with a server, where a representation captures the current or intended state of a resource and the server is the definitive source for representations of the resources in its namespace. A client interested in the state of a resource initiates a request to the server; the server then returns a response with a representation of the resource that is current at the time of the request.

This model does not work well when a client is interested in having a current representation of a resource over a period of time. Existing approaches from HTTP, such as repeated polling or HTTP long polling [RFC6202], generate significant complexity and/or overhead and thus are less applicable in a constrained environment.

The protocol specified in this document extends the CoAP core protocol with a mechanism for a CoAP client to "observe" a resource on a CoAP server: the client can retrieve a representation of the resource and request this representation be updated by the server over a period of time.

The protocol keeps the architectural properties of REST. It enables high scalability and efficiency through the support of caches and proxies. There is no intention for it, though, to solve the full set of problems that the existing HTTP solutions solve, or to replace publish/subscribe networks that solve a much more general problem [RFC5989].

1.2. Protocol Overview

The protocol is based on the well-known observer design pattern [GOF]. In this design pattern, components called "observers" register at a specific, known provider called the "subject" that they are interested in being notified whenever the subject undergoes a change in state. The subject is responsible for administering its list of registered observers. If multiple subjects are of interest to an observer, the observer must register separately for all of them.

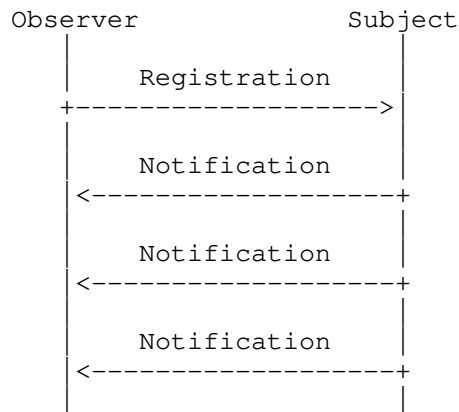


Figure 1: The Observer Design Pattern

The observer design pattern is realized in CoAP as follows:

Subject: In the context of CoAP, the subject is a resource in the namespace of a CoAP server. The state of the resource can change over time, ranging from infrequent updates to continuous state transformations.

Observer: An observer is a CoAP client that is interested in having a current representation of the resource at any given time.

Registration: A client registers its interest in a resource by initiating an extended GET request to the server. In addition to returning a representation of the target resource, this request causes the server to add the client to the list of observers of the resource.

Notification: Whenever the state of a resource changes, the server notifies each client in the list of observers of the resource. Each notification is an additional CoAP response sent by the server in reply to the GET request and includes a complete, updated representation of the new resource state.

Figure 2 below shows an example of a CoAP client registering its interest in a resource and receiving three notifications: the first upon registration with the current state, and then two upon changes to the resource state. Both the registration request and the notifications are identified as such by the presence of the Observe Option defined in this document. In notifications, the Observe Option additionally provides a sequence number for reordering detection. All notifications carry the token specified by the client, so the client can easily correlate them to the request.

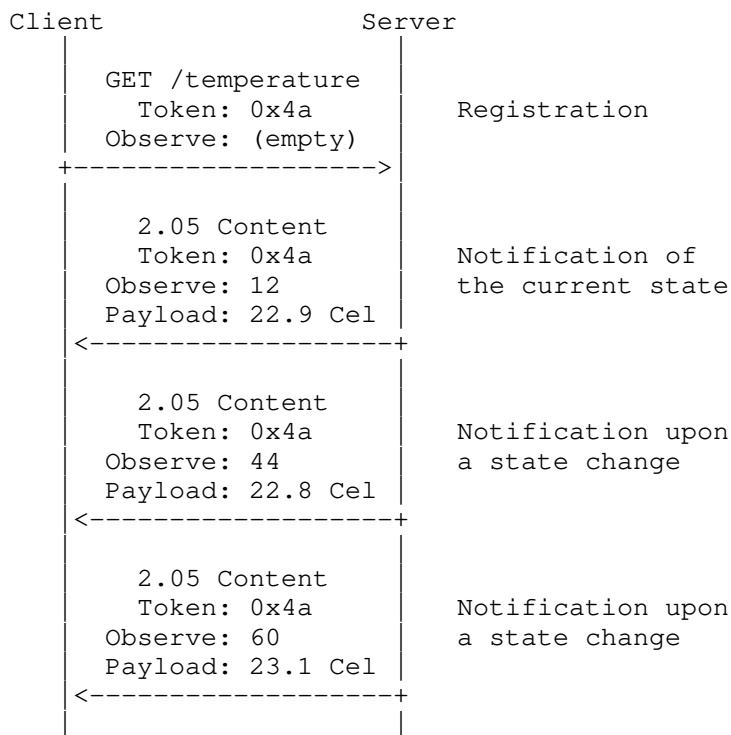


Figure 2: Observing a Resource in CoAP

A client remains on the list of observers as long as the server can determine the client's continued interest in the resource. The interest is determined from the client's acknowledgement of notifications sent in confirmable CoAP messages by the server: If the client actively rejects a notification or if the transmission of a notification times out after several transmission attempts, then the client is assumed to be no longer interested and it is removed from the list of observers.

1.3. Observable Resources

A CoAP server is the authority for determining under what conditions resources change their state and thus when observers are notified of new resource states. The protocol does not offer explicit means for setting up triggers or thresholds; it is up to the server to expose observable resources that change their state in a way that is useful in the application context.

For example, a CoAP server with an attached temperature sensor could expose one or more of the following resources:

- o `<coap://server/temperature>`, which changes its state every second to the current reading of the temperature sensor;
- o `<coap://server/temperature/felt>`, which changes its state to "cold" when the temperature reading drops below a certain pre-configured threshold, and to "warm" when the reading exceeds a second, slightly higher threshold;
- o `<coap://server/temperature/critical?above=45>`, which changes its state based on the client-specified parameter value: every second to the current temperature reading if the temperature exceeds the threshold, or to "OK" when the reading drops below; and/or
- o `<coap://server/?query=select+avg(temperature)+from+Sensor.window:time(30sec)>`, which accepts expressions of arbitrary complexity and changes its state accordingly.

So, by designing CoAP resources that change their state on certain conditions, it is possible to update the client only when these conditions occur instead of continuously supplying it with raw sensor readings. By parameterizing resources, this is not limited to conditions defined by the server, but can be extended to arbitrarily complex queries specified by the client. Thus, the application designer can choose exactly the right level of complexity for the application envisioned and devices used, and is not constrained to a "one size fits all" mechanism built into the protocol.

1.4. Consistency

While a client is in the list of observers of a resource, the goal of the protocol is to keep the resource state observed by the client as closely in sync with the actual state at the server as possible.

It cannot be avoided that the client and the server become inconsistent at times: First, there is always some latency between the change of the resource state and the receipt of the notification. Second, messages with notifications can get lost, which will cause the client to assume an old state until it receives a new notification. And third, the server may erroneously come to the conclusion that the client is no longer interested in the resource, which will cause the server to stop sending notifications and the client to assume an old state until it registers its interest eventually again.

The protocol addresses this as follows:

- o It follows a best-effort approach for sending the current representation to the client after a state change: Clients should

see the new state after a state change as soon as possible, and they should see as many states as possible. However, a client cannot rely on observing every single state that a resource might go through.

- o It labels notifications with a maximum duration up to which it is acceptable for the observed state and the actual state to be out of sync. When the age of the notification received reaches this limit, the client cannot use the enclosed representation until it receives a new notification.
- o It is designed on the principle of eventual consistency: The protocol guarantees that, if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the latest resource state.

1.5. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. The Observe Option

No.	C	U	N	R	Name	Format	Length	Default
6		x	-		Observe	empty/uint	0 B/0-3 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Table 1: The Observe Option

The Observe Option, when present in a request, extends the GET method so it does not only retrieve a current representation of the target resource, but also requests the server to add a new entry to the list of observers of the resource. The list entry consists of the client endpoint and the token specified by the client in the request.

The value of the Observe Option in a request MUST be empty on transmission and MUST be ignored on reception.

The Observe Option is not critical for processing the request. If the server is unwilling or unable to add the client to the list of observers of the target resource, then the request falls back to a normal GET request.

In a response, the Observe Option identifies the message as a notification. This implies that the server has added the client to the list of observers and that it will notify the client of changes to the resource state.

The value of the Observe Option in a response is a 24-bit sequence number for reordering detection (see Section 3.4 and Section 4.4). The sequence number is encoded in network byte order using a variable number of bytes ('uint' format; see Section 3.2 of RFC XXXX [I-D.ietf-core-coap]).

The Observe Option is not part of the cache-key: a cacheable response obtained with an Observe Option in the request can be used to satisfy a request without an Observe Option, and vice versa. When a stored response that includes an Observe Option is used to satisfy a normal GET request, the option MUST be removed before the response is returned to the client.

3. Client-side Requirements

3.1. Request

A client can register its interest in a resource by issuing a GET request that includes an empty Observe Option. If the server returns a 2.xx response that includes an Observe Option as well, the server has added the client successfully to the list of observers of the target resource and the client will be notified of changes to the resource state.

Like a fresh response can be used to satisfy a request without contacting the server, the updates resulting from one request can be used to satisfy another request if the target resource is the same. A client therefore MUST aggregate requests where possible, and MUST NOT register more than once for the same target resource. The target resource SHALL be identified for this purpose by all options in the request that are part of the cache-key, such as the full request URI and the Accept Option.

3.2. Notifications

Notifications are additional responses sent by the server in reply to the GET request. Each notification includes the token specified by the client in the GET request, an Observe Option with a sequence number for reordering detection (see Section 3.4), and a payload in the same Content-Format as the initial response.

Notifications have a 2.05 (Content) response code, or potentially a 2.03 (Valid) response code if the client included one or more ETag

Options in the request (see Section 3.3). In the event that the resource changes in a way that would cause a normal GET request at that time to return a non-2.xx response (for example, when the resource is deleted), the server sends a notification with an appropriate response code (such as 4.04 Not Found) and removes all clients from the list of observers of the resource.

3.3. Caching

As notifications are just additional responses to a GET request, notifications partake in caching as defined in Section 5.6 of RFC XXXX [I-D.ietf-core-coap]. Both the freshness model and the validation model are supported.

3.3.1. Freshness

A client MAY store a notification like a response in its cache and use a stored notification that is fresh without contacting the server. Like a response, a notification is considered fresh while its age is not greater than the value indicated by the Max-Age Option and no newer notification/response has been received.

The server will do its best to keep the resource state observed by the client as closely in sync with the actual state as possible. However, a client cannot rely on observing every single state that a resource might go through. For example, if the network is congested or the state changes more frequently than the network can handle, the server can skip notifications for any number of intermediate states.

The server uses the Max-Age Option to indicate an age up to which it is acceptable that the observed state and the actual state are inconsistent. If the age of the latest notification becomes greater than its indicated Max-Age, then the client MUST NOT assume that the enclosed representation reflects the actual resource state.

To make sure it has a current representation and/or to re-register its interest in a resource, a client MAY issue a new GET request with an Observe Option and the same token at any time. It is RECOMMENDED that the client does not issue the request while it still has a fresh notification/response for the resource in its cache. Additionally, the client SHOULD wait for a random amount of time between 5 and 15 seconds to avoid synchronicity with other clients.

3.3.2. Validation

When a client has one or more notifications stored in its cache for a resource, it can use the ETag Option in the GET request to give the server an opportunity to select a stored notification to be used.

The client MAY include an ETag Option for each stored response that is applicable in the GET request. Whenever the observed resource changes to a representation identified by one of the ETag Options, the server can select a stored response by sending a 2.03 (Valid) notification with an appropriate ETag Option instead of a 2.05 (Content) notification.

A client implementation needs to keep all candidate responses in its cache until it is no longer interested in the target resource or it issues a GET request with a new set of entity-tags.

3.4. Reordering

Messages with notifications can arrive in a different order than they were sent. Since the goal is to keep the observed state as closely in sync with the actual state as possible, a client MUST NOT update the observed state with a notification that arrives later than a newer notification.

For reordering detection, the server sets the value of the Observe Option in each notification to the 24 least-significant bits of a strictly increasing sequence number. An incoming notification is newer than the newest notification received so far when one of the following conditions is met:

$$\begin{aligned} & (V1 < V2 \text{ and } V2 - V1 < 2^{23}) \text{ or} \\ & (V1 > V2 \text{ and } V1 - V2 > 2^{23}) \text{ or} \\ & (T2 > T1 + 128 \text{ seconds}) \end{aligned}$$

where V1 is the value of the Observe Option of the newest notification received so far, V2 the value of the Observe Option of the incoming notification, T1 a client-local timestamp of the newest notification received so far, and T2 a client-local timestamp of the incoming notification.

Design Note: The first two conditions verify that V1 is less than V2 in 24-bit serial number arithmetic [RFC1982]. The third condition ensures that the time elapsed between the two incoming messages is not so large that the difference between V1 and V2 has become larger than the largest integer that it is meaningful to add to a 24-bit serial number; in other words, after 128 seconds have elapsed without any notification, a client does not need to check the sequence numbers to assume an incoming notification is new.

The duration of 128 seconds was chosen as a nice round number greater than MAX_LATENCY (see Section 4.8.2 of RFC XXXX [I-D.ietf-core-coap]).

3.5. Transmission

A notification can be confirmable or non-confirmable, i.e., be sent in a confirmable or a non-confirmable message. The message type used for a notification is independent from the type used for the request or for any previous notification.

If a client does not recognize the token in a confirmable notification, it **MUST NOT** acknowledge the message and **SHOULD** reject it with a Reset message; otherwise, the client **MUST** acknowledge the message as usual. In the case of a non-confirmable notification, rejecting the message with a Reset message is **OPTIONAL**.

An acknowledgement message signals to the server that the client is alive and interested in receiving further notifications; if the server does not receive an acknowledgement in reply to a confirmable notification, it will assume that the client is no longer interested and will eventually remove the associated entry from the list of observers.

3.6. Cancellation

A client that is no longer interested in receiving further notifications for a resource can simply "forget" the pending request. When the server then sends a notification, the client will not recognize the token in the message. If the notification was confirmable, this will cause the client to return a Reset message and thus the server to remove the associated entry from the list of observers. Entries in lists of observers are effectively "garbage collected" by the server.

When a client rejects a non-confirmable notification, the server may also (but is not required to) remove the associated entry from the list of observers. So, if the server seems to ignore the Reset messages that the client sends to reject non-confirmable notifications, the client may have to wait for a confirmable notification until the list entry is removed.

4. Server-side Requirements

4.1. Request

A GET request with an Observe Option requests the server not only to return a current representation of the target resource, but also to add the client to the list of observers of that resource. Upon success, the server **MUST** return a current representation of the resource and **MUST** notify the client of subsequent changes to the state as long as the client is on the list of observers.

The entry in the list of observers is keyed by the client endpoint and the token specified by the client in the request. If an entry with a matching endpoint/token pair is already present in the list (which, for example, happens when the client wishes to reinforce its interest in a resource), the server **MUST NOT** add a new entry but **MUST** replace or update the existing one.

A server that is unable or unwilling to add a new entry to the list of observers of a resource **MAY** silently ignore the Observe Option and process the GET request as usual. The resulting response **MUST NOT** include an Observe Option, the absence of which signals to the client that it will not be notified of changes to the resource and, e.g., needs to poll the resource for its state instead.

4.2. Notifications

A client is notified of changes to the resource state by additional responses sent by the server in reply to the GET request. Each such notification response (including the initial response) **MUST** include an Observe Option and **MUST** echo the token specified by the client in the GET request. If there are multiple entries in the list of observers, the order in which the clients are notified is not defined; the server is free to use any method to determine the order.

A notification **SHOULD** have a 2.05 (Content) or 2.03 (Valid) response code. However, in the event that the state of a resource changes in a way that would cause a normal GET request at that time to return a non-2.xx response (for example, when the resource is deleted), the server **SHOULD** notify the client by sending a notification with an appropriate response code (such as 4.04 Not Found) and **MUST** remove the client from the list of observers of the resource.

The Content-Format used in a notification **MUST** be the same as the one used in the initial response to the GET request. If the server is unable to continue sending notifications in this Content-Format, it **SHOULD** send a notification with a 4.06 (Not Acceptable) response code and **MUST** remove the client from the list of observers of the resource.

A non-2.xx notification **MUST NOT** include an Observe Option.

4.3. Caching

As notifications are just additional responses sent by the server, they are subject to caching as defined in Section 5.6 of RFC XXXX [I-D.ietf-core-coap].

4.3.1. Freshness

After returning the initial response, the server MUST try to keep the returned representation current, i.e., keep the resource state observed by the client as closely in sync with the actual resource state as possible.

Since becoming out of sync at times cannot be avoided, the server MUST indicate for each representation an age up to which it is acceptable that the observed state and the actual state are inconsistent. This age is application-dependent and MUST be specified in notifications using the Max-Age Option.

When the resource does not change and the client has a current representation, the server does not need to send a notification. However, if the client does not receive a notification, the client cannot tell if the observed state and the actual state are still in sync. Thus, when the age of the latest notification becomes greater than its indicated Max-Age, the client no longer has a usable representation of the resource state. The server MAY wish to prevent that by sending a notification with the unchanged representation and a new Max-Age just before the old Max-Age expires.

4.3.2. Validation

A client can include a set of entity-tags in its request using the ETag Option. When a observed resource changes its state and the origin server is about to send a 2.05 (Content) notification, then, whenever that notification has an entity-tag in the set of entity-tags specified by the client, the server MAY send a 2.03 (Valid) response with an appropriate ETag Option instead.

4.4. Reordering

Because messages can get reordered, the client needs a way to determine if a notification arrived later than a newer notification. For this purpose, the server MUST set the value of the Observe Option of each notification it sends to the 24 least-significant bits of a strictly increasing sequence number. The sequence number MAY start at any value and MUST NOT increase so fast that it increases by more than 2^{23} within less than 256 seconds.

The sequence number selected for a notification MUST be greater than that of any preceding notification sent to the same client with the same token for the same resource. The value of the Observe Option MUST be current at the time of transmission; if a notification is retransmitted, the server MUST update the value of the option to the sequence number that is current at that time before sending the

message.

The sequence numbers generated for a resource MUST provide an order among all notifications resulting from all requests from the same client endpoint.

Implementation Note: A simple implementation that satisfies the requirements is to obtain a timestamp from a local clock. The sequence number then is the timestamp in ticks, where 1 tick = $(256 \text{ seconds}) / (2^{23}) = 30.52 \text{ microseconds}$. It is not necessary that the clock reflects the current time/date.

Another valid implementation is to store a 24-bit unsigned integer variable per resource and increment this variable each time the resource undergoes a change of state (provided that the resource changes its state less than 2^{23} times in the next 256 seconds after every state change). This removes the need to update the value of the Observe Option on retransmission when the resource state did not change.

Design Note: The choice of a 24-bit option value and a time span of 256 seconds allows for a notification rate of up to 65536 notifications per second. Constrained nodes often have rather imprecise clocks, though, and inaccuracies of the client and server side may cancel out or add in effect. Reducing the maximum notification rate to 32768 notifications per second is still well beyond the highest known design objective of around 1 kHz (most CoAP applications will be several orders of magnitude below that), but allows total clock inaccuracies of up to -50/+100 %.

4.5. Transmission

A notification can be sent in a confirmable or a non-confirmable message. The message type used is typically application-dependent and MAY be determined by the server for each notification individually. For example, for resources that change in a somewhat predictable or regular fashion, notifications can be sent in non-confirmable messages; for resources that change infrequently, notifications can be sent in confirmable messages. The server can combine these two approaches depending on the frequency of state changes and the importance of individual notifications.

A server MAY choose to skip sending a notification if it knows that it will send another notification soon, for example, when the state is changing frequently. Similarly, it MAY choose to send a notification more than once. However, above all, the server MUST ensure that a client in the list of observers of a resource eventually observes the latest state if the resource does not undergo

a new change in state. For example, when state changes occur in bursts, the server can skip some notifications, send the notifications in non-confirmable messages, and make sure that the client observes the latest state change by repeating the last notification in a confirmable message when the burst is over.

The client's acknowledgement of a confirmable notification signals to the server that the client is interested in receiving further notifications. If a client rejects a confirmable notification with a Reset message, the client is no longer interested and the server **MUST** remove the associated entry from the list of observers. If the client rejects a non-confirmable notification, the server **MAY** remove the entry from the list of observers as well. (It is expected that the server does remove the entry if it has the information available that is needed to match the Reset message to the non-confirmable notification, but the server is not required to keep this information.)

At a minimum, the server **MUST** send a notification in a confirmable message instead of a non-confirmable message at least every 24 hours, so a client that went away or is no longer interested does not remain forever in the list of observers.

The server **MUST** limit the number of confirmable notifications for which an acknowledgement has not been received from a client yet to NSTART (1 by default; see Section 4.7 of RFC XXXX [I-D.ietf-core-coap]).

The server **SHOULD NOT** send more than one non-confirmable notification per round-trip time (RTT) to a client on average. If the server cannot maintain an RTT estimate for a client, it **SHOULD NOT** send more than one non-confirmable notification every 3 seconds, and **SHOULD** use an even less aggressive rate when possible (see also Section 3.1.2 of RFC 5405 [RFC5405]).

When the state of an observed resource changes while the number of outstanding acknowledgements is greater than or equal to NSTART, or while the waiting time for a non-confirmable notification has not elapsed yet, the server **MUST** proceed as follows:

1. Wait for the current transmission attempt to be acknowledged, rejected or to time out (confirmable transmission), or the waiting time to elapse (non-confirmable transmission).
2. If the transmission is rejected or the transmission was the last attempt to deliver a notification, remove the associated entry from the list of observers of the observed resource.

3. If the entry is still in the list of observers, start to transmit a new notification with a representation of the current resource state. Should the resource have changed its state more than once in the meantime, the notifications for the intermediate states are silently skipped.
 4. If the completed transmission attempt timed out, increment the retransmission counter and double the timeout for the new transmission; otherwise, reinitialize both the retransmission counter and timeout as described in Section 4.2 of RFC XXXX [I-D.ietf-core-coap].
5. Intermediaries

A client may be interested in a resource in the namespace of an origin server that is reached through a chain of one or more CoAP intermediaries. In this case, the client registers its interest with the first intermediary towards the origin server, acting as if it was communicating with the origin server itself as specified in Section 3. It is the task of this intermediary to provide the client with a current representation of the target resource and send notifications upon changes to the target resource state, much like an origin server as specified in Section 4.

To perform this task, the intermediary SHOULD make use of the protocol specified in this document, taking the role of the client and registering its own interest in the target resource with the next hop towards the origin server. If the next hop does not return a response with an Observe Option, the intermediary MAY resort to polling the next hop or MAY itself return a response without an Observe Option.

The communication between each pair of hops is independent; each hop in the server role MUST determine individually how many notifications to send, of which message type, and so on. Each hop MUST generate its own values for the Observe Option, and MUST set the value of the Max-Age Option according to the age of the local current representation.

If two or more clients have registered their interest in a resource with an intermediary, the intermediary MUST register itself only once with the next hop and fan out the notifications it receives to all registered clients. This relieves the next hop from sending the same notifications multiple times and thus enables scalability.

An intermediary is not required to act on behalf of a client to observe a resource; an intermediary MAY observe a resource, for example, just to keep its own cache up to date.

See Appendix A.2 for examples.

6. Web Linking

A web link [RFC5988] to a resource accessible over CoAP (for example, in a link-format document [RFC6690]) MAY include the target attribute "obs".

The "obs" attribute, when present, is a hint indicating that the destination of a link is useful for observation and thus, for example, should have a suitable graphical representation in a user interface. Note that this is only a hint; it is not a promise that the Observe Option can actually be used to perform the observation. A client may need to resort to polling the resource if the Observe Option is not returned in the response to the GET request.

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

7. Security Considerations

The security considerations of RFC XXXX [I-D.ietf-core-coap] apply.

The considerations about amplification attacks are somewhat amplified when observing resources. Without client authentication, a server MUST therefore strictly limit the number of notifications that it sends between receiving acknowledgements that confirm the actual interest of the client in the data; i.e., any notifications sent in non-confirmable messages MUST be interspersed with confirmable messages. (An attacker may still spoof the acknowledgements if the confirmable messages are sufficiently predictable.)

As with any protocol that creates state, attackers may attempt to exhaust the resources that the server has available for maintaining the list of observers for each resource. Servers may want to access-control this creation of state. As degraded behavior, the server can always fall back to processing the request as a normal GET request (without an Observe Option) if it is unwilling or unable to add a client to the list of observers of a resource, including if system resources are exhausted or nearing exhaustion.

Intermediaries must be careful to ensure that notifications cannot be employed to create a loop. A simple way to break any loops is to employ caches for forwarding notifications in intermediaries.

8. IANA Considerations

The following entry is added to the CoAP Option Numbers registry:

Number	Name	Reference
6	Observe	[RFCXXXX]

[Note to RFC Editor: Please replace XXXX with the RFC number of this specification.]

9. Acknowledgements

Carsten Bormann was an original author of this draft and is acknowledged for significant contribution to this document.

Thanks to Daniele Alessandrelli, Jari Arkko, Peter Bigot, Angelo P. Castellani, Gilbert Clark, Esko Dijk, Thomas Fossati, Brian Frank, Bert Greevenbosch, Jeroen Hoebeke, Cullen Jennings, Matthias Kovatsch, Salvatore Loreto, Charles Palmer, Zach Shelby, and Floris Van den Abeele for helpful comments and discussions that have shaped the document.

This work was supported in part by Klaus Tschira Foundation, Intel, Cisco, and Nokia.

10. References

10.1. Normative References

- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

10.2. Informative References

- [GOF] Gamma, E., Helm, R., Johnson, R., and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, USA, November 1994.
- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC5989] Roach, A., "A SIP Event Package for Subscribing to Changes to an HTTP Resource", RFC 5989, October 2010.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

Appendix A. Examples

A.1. Client/Server Examples

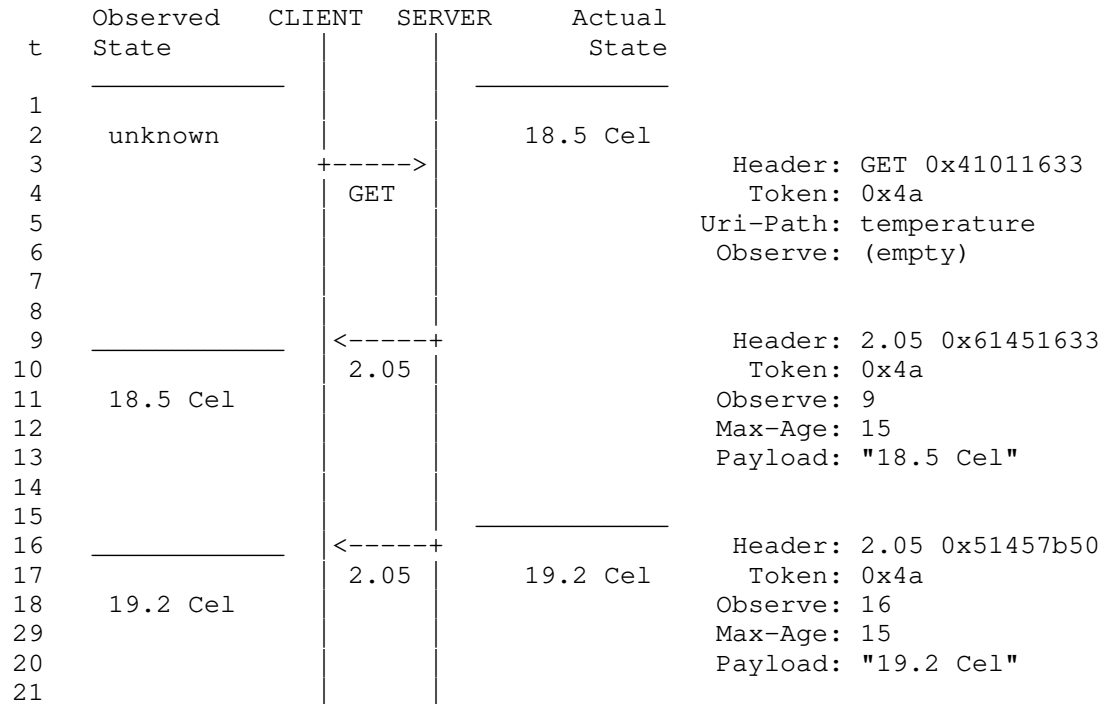


Figure 3: A client registers and receives one notification of the current state and one of a new state upon a state change

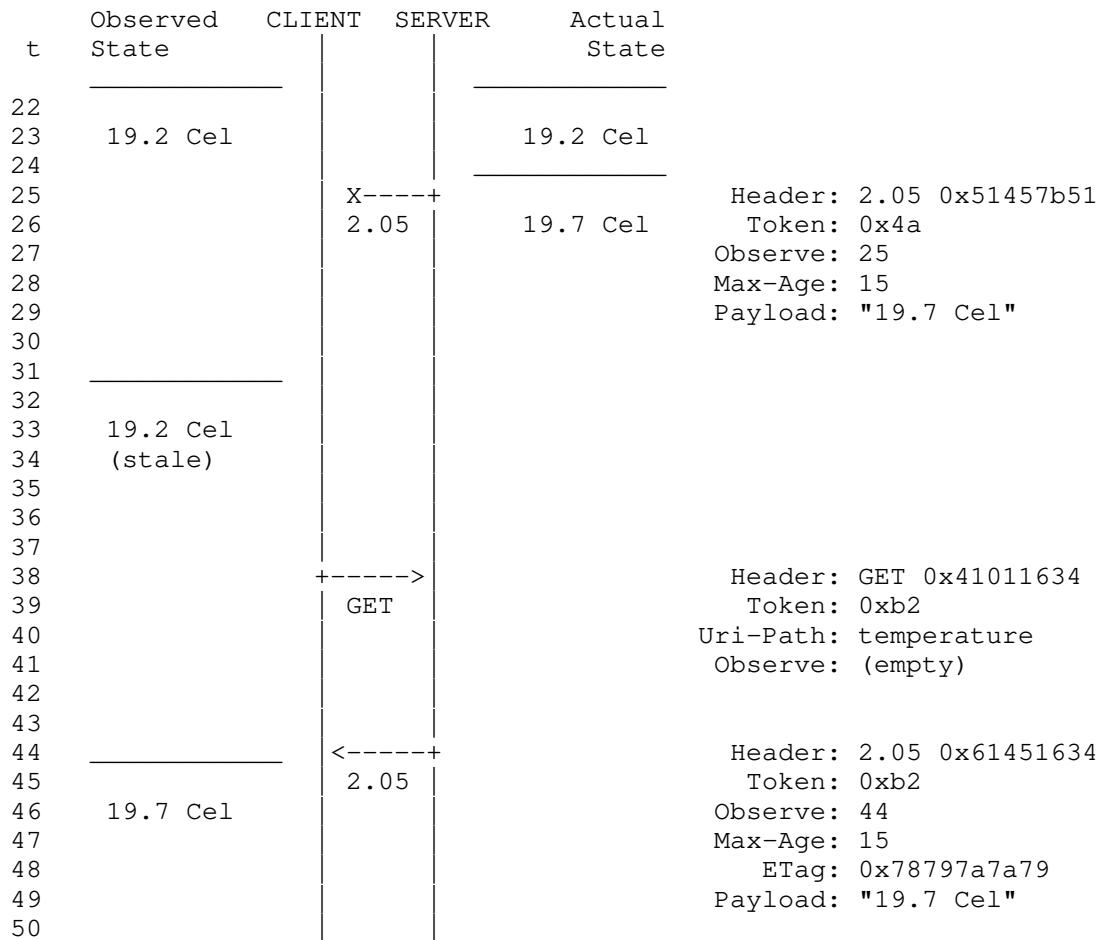


Figure 4: The client re-registers after Max-Age ends

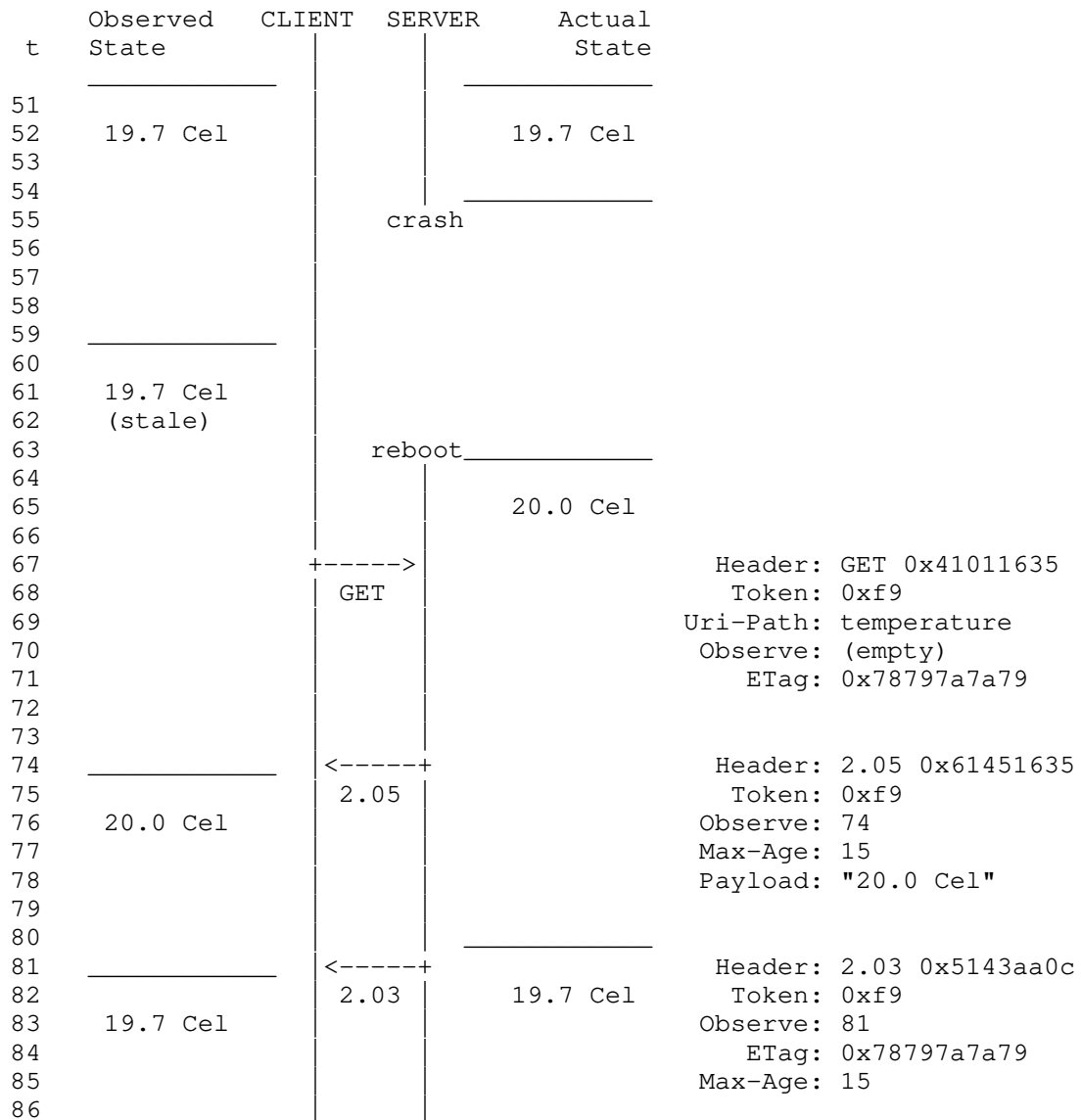


Figure 5: The client re-registers and gives the server the opportunity to select a stored response

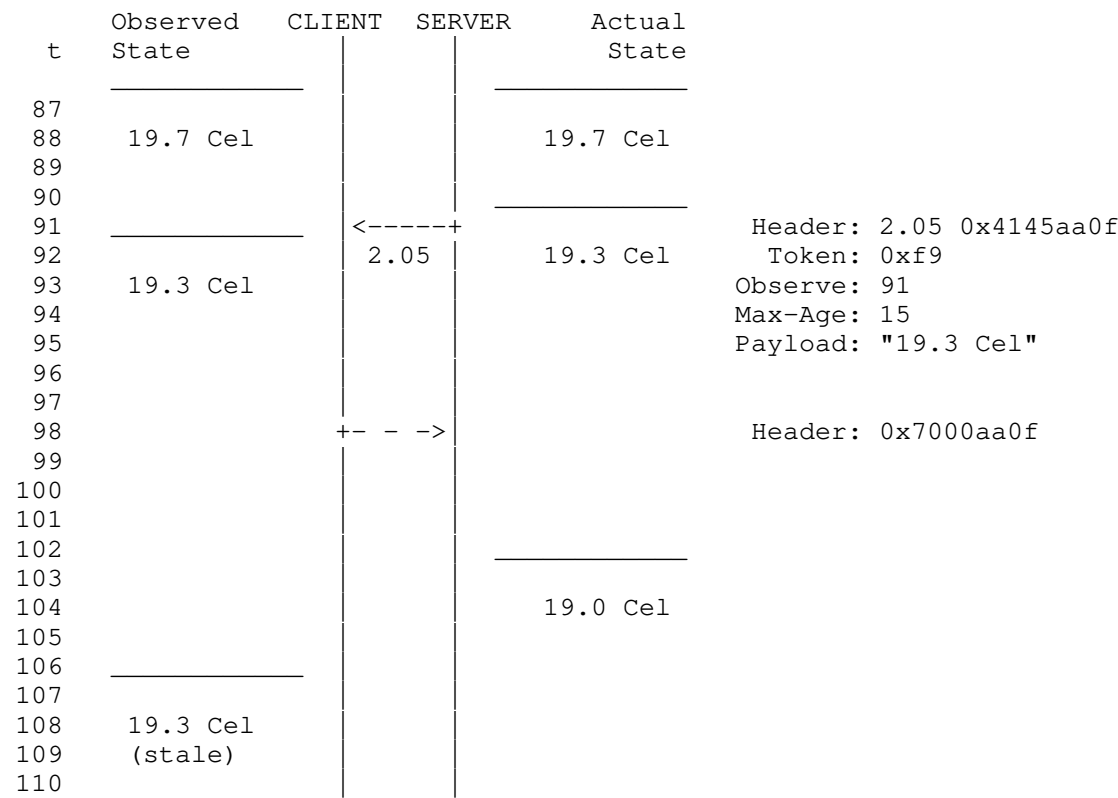


Figure 6: The client rejects a notification and thereby cancels the observation

A.2. Proxy Examples

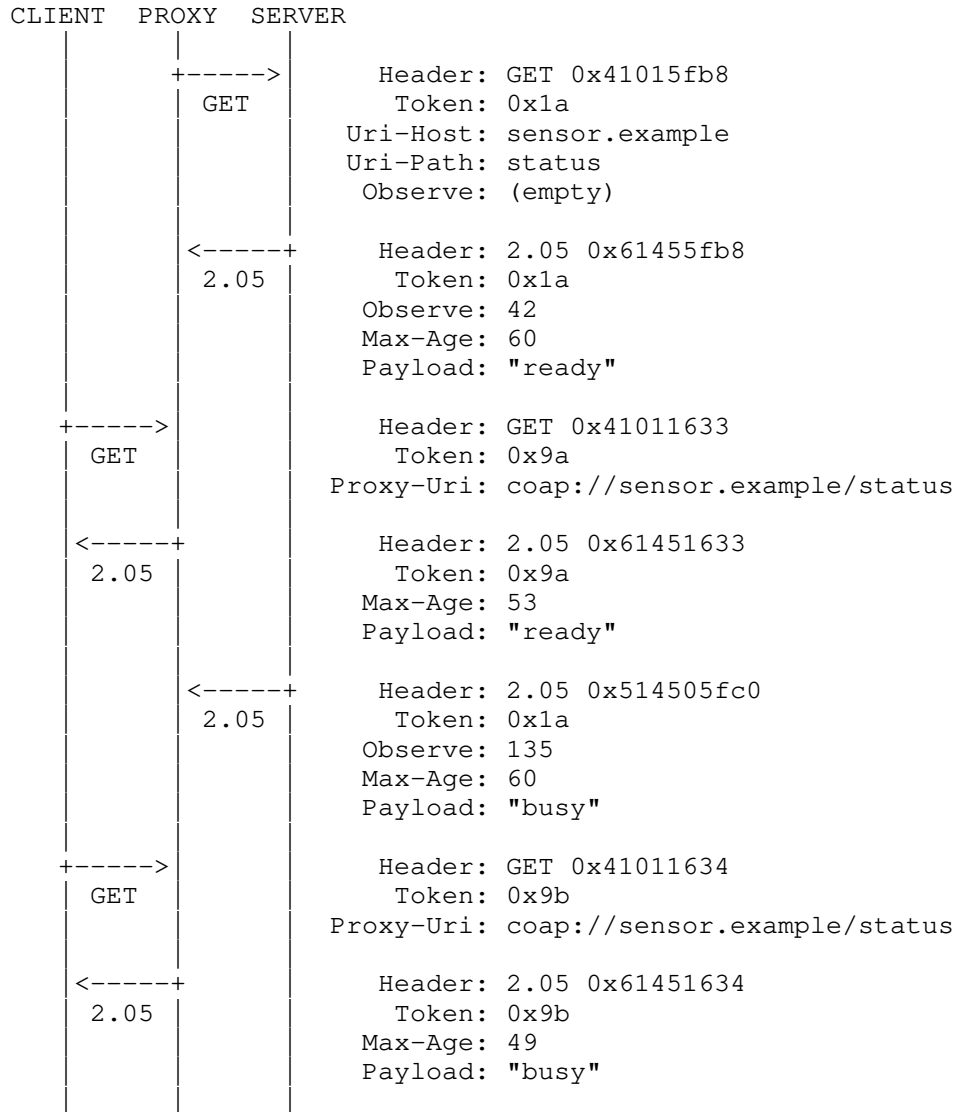


Figure 7: A proxy observes a resource to keep its cache up to date

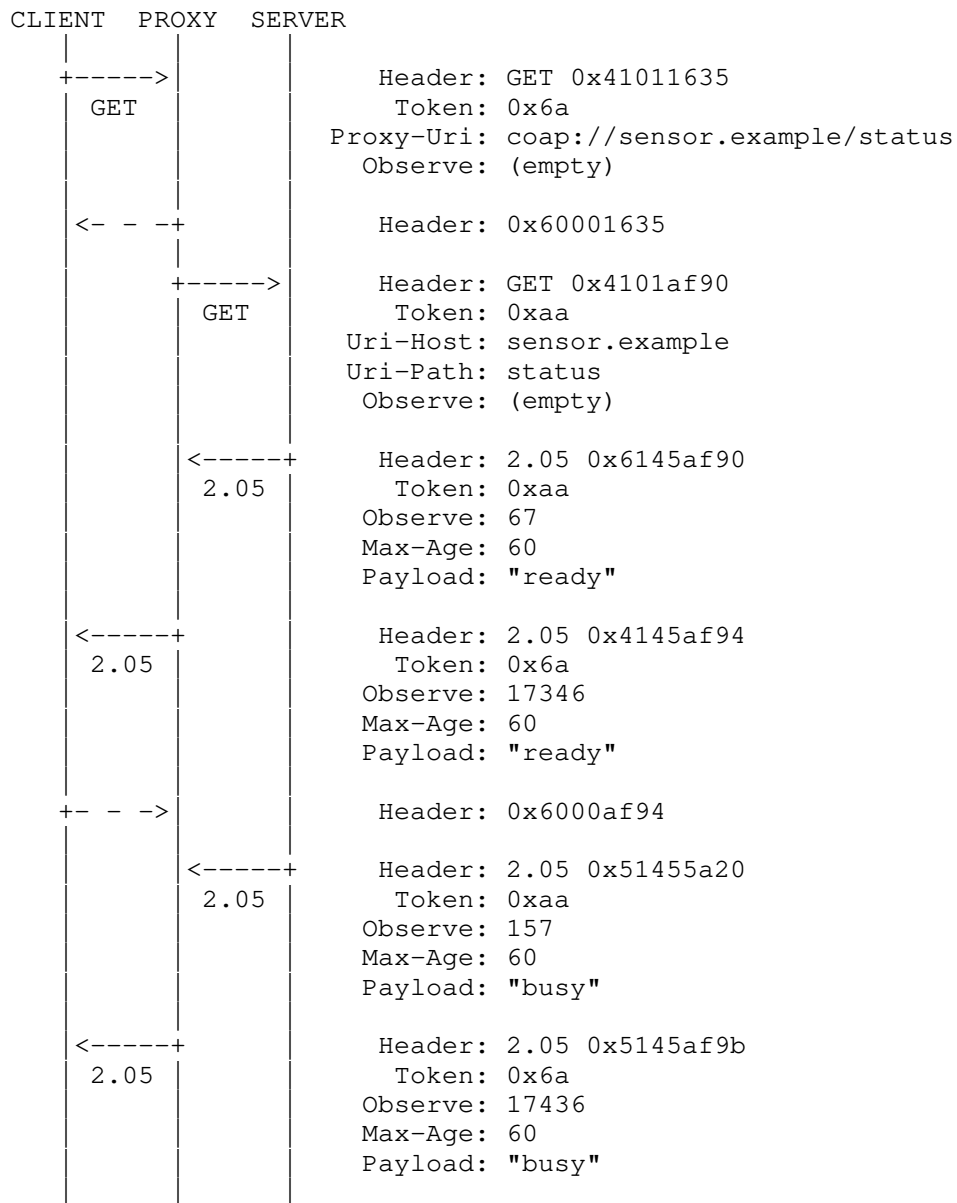


Figure 8: A client observes a resource through a proxy

Appendix B. Changelog

[Note to RFC Editor: Please remove this section before publication.]

Changes from ietf-10 to ietf-11:

- o Pointed out that client and server clocks may differ in their realization of the SI second, and added robustness to the existing reordering scheme by reducing the maximum notification rate to 32768 notifications per second (#341).

Changes from ietf-09 to ietf-10:

- o Required consistent sequence numbers across requests (#333).
- o Clarified that a server needs to update the entry in the list of observers instead of adding a new entry if the endpoint/token pair is already present.
- o Allowed that a client uses a token that is currently in use to ensure that it's still in the list of observers. This is possible because sequence numbers are now consistent across requests and servers won't add a new entry for the same token.
- o Improved text on the transmission of non-confirmable notifications to match Section 3.1.2 of RFC 5405 more closely.
- o Updated examples to use UCUM units.
- o Moved Appendix B into the introduction.

Changes from ietf-08 to ietf-09:

- o Removed the side effects of requests on existing observations. This includes removing that
 - * the client can use a GET request to cancel an observation;
 - * the server updates the entry in the list of observers instead of adding a new entry if the client is already present (#258, #281).
- o Clarified that a resource (and hence an observation relationship) is identified by the request options that are part of the Cache-Key (#258).
- o Clarified that a non-2.xx notification MUST NOT include an Observe Option.

- o Moved block-wise transfer of notifications to [I-D.ietf-core-block].

Changes from ietf-07 to ietf-08:

- o Expanded text on transmitting a notification while a previous transmission is pending (#242).
- o Changed reordering detection to use a fixed time span of 128 seconds instead of EXCHANGE_LIFETIME (#276).
- o Removed the use of the freshness model to determine if the client is still on the list of observers. This includes removing that
 - * the client assumes that it has been removed from the list of observers when Max-Age ends;
 - * the server sets the Max-Age Option of a notification to a value that indicates when the server will send the next notification;
 - * the server uses a number of retransmit attempts such that removing a client from the list of observers before Max-Age ends is avoided (#235);
 - * the server may remove the client from all lists of observers when the transmission of a confirmable notification ultimately times out.
- o Changed that an unrecognized critical option in a request must actually have no effect on the state of any observation relationship to any resource, as the option could lead to a different target resource.
- o Clarified that client implementations must be prepared to receive each notification equally as a confirmable or a non-confirmable message, regardless of the message type of the request and of any previous notification.
- o Added a requirement for sending a confirmable notification at least every 24 hours before continuing with non-confirmable notifications (#221).
- o Added congestion control considerations from [I-D.bormann-core-congestion-control-02].
- o Recommended that the client waits for a randomized time after the freshness of the latest notification expired before re-registering. This prevents that multiple clients observing a

resource perform a GET request at the same time when the need to re-register arises.

- o Changed reordering detection from 'MAY' to 'SHOULD', as the goal of the protocol (to keep the observed state as closely in sync with the actual state as possible) is not optional.
- o Fixed the length of the Observe Option (3 bytes) in the table in Section 2.
- o Replaced the 'x' in the No-Cache-Key column in the table in Section 2 with a '-', as the Observe Option doesn't have the No-Cache-Key flag set, even though it is not part of the cache key.
- o Updated examples.

Changes from ietf-06 to ietf-07:

- o Moved to 24-bit sequence numbers to allow for up to 15000 notifications per second per client and resource (#217).
- o Re-numbered option number to use Unsafe/Safe and Cache-Key compliant numbers (#241).
- o Clarified how to react to a Reset message that is sent in reply to a non-confirmable notification (#225).
- o Clarified the semantics of the "obs" link target attribute (#236).

Changes from ietf-05 to ietf-06:

- o Improved abstract and introduction to say that the protocol is about best effort and eventual consistency (#219).
- o Clarified that the value of the Observe Option in a request must have zero length.
- o Added requirement that the sequence number must be updated each time a server retransmits a notification.
- o Clarified that a server must remove a client from the list of observers when it receives a GET request with an unrecognized critical option.
- o Updated the text to use the endpoint concept from [I-D.ietf-core-coap] (#224).

- o Improved the reordering text (#223).

Changes from ietf-04 to ietf-05:

- o Recommended that a client does not re-register while a new notification from the server is still likely to arrive. This is to avoid that the request of the client and the last notification after max-age cross over each other (#174).
- o Relaxed requirements when sending a Reset message in reply to non-confirmable notifications.
- o Added an implementation note about careless GET requests (#184).
- o Updated examples.

Changes from ietf-03 to ietf-04:

- o Removed the "Max-OFE" Option.
- o Allowed a Reset message in reply to non-confirmable notifications.
- o Added a section on cancellation.
- o Updated examples.

Changes from ietf-02 to ietf-03:

- o Separated client-side and server-side requirements.
- o Fixed uncertainty if client is still on the list of observers by introducing a liveliness model based on Max-Age and a new option called "Max-OFE" (#174).
- o Simplified the text on message reordering (#129).
- o Clarified requirements for intermediaries.
- o Clarified the combination of blockwise transfers with notifications (#172).
- o Updated examples to show how the state observed by the client becomes eventually consistent with the actual state on the server.
- o Added examples for parameterization of observable resource.

Changes from ietf-01 to ietf-02:

- o Removed the requirement of periodic refreshing (#126).
- o The new "Observe" Option replaces the "Lifetime" Option.
- o Introduced a new mechanism to detect message reordering.
- o Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:

- o Changed terminology from "subscriptions" to "observation relationships" (#33).
- o Changed the name of the option to "Lifetime".
- o Clarified establishment of observation relationships.
- o Clarified that an observation is only identified by the URI of the observed resource and the identity of the client (#66).
- o Clarified rules for establishing observation relationships (#68).
- o Clarified conditions under which an observation relationship is terminated.
- o Added explanation on how clients can terminate an observation relationship before the lifetime ends (#34).
- o Clarified that the overriding objective for notifications is eventual consistency of the actual and the observed state (#67).
- o Specified how a server needs to deal with clients not acknowledging confirmable messages carrying notifications (#69).
- o Added a mechanism to detect message reordering (#35).
- o Added an explanation of how notifications can be cached, supporting both the freshness and the validation model (#39, #64).
- o Clarified that non-GET requests do not affect observation relationships, and that GET requests without "Lifetime" Option affecting relationships is by design (#65).
- o Described interaction with blockwise transfers (#36).
- o Added Resource Discovery section (#99).

- o Added IANA Considerations.
- o Added Security Considerations (#40).
- o Added examples (#38).

Author's Address

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
EMail: hartke@tzi.org

CoRE
Internet-Draft
Intended status: Standards Track
Expires: December 05, 2013

Z. Shelby
Sensinode
S. Krco
Ericsson
C. Bormann
Universitaet Bremen TZI
June 03, 2013

CoRE Resource Directory
draft-ietf-core-resource-directory-00

Abstract

In many M2M applications, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources. This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resources descriptions. Furthermore, new link attributes useful in conjunction with an RD are defined.

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 <http://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 December 05, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Architecture and Use Cases	4
3.1. Use Case: Cellular M2M	5
3.2. Use Case: Home and Building Automation	5
4. Simple Directory Discovery	6
4.1. Finding a Directory Server	7
5. Resource Directory Function Set	8
5.1. Discovery	8
5.2. Registration	10
5.3. Update	12
5.4. Validation	13
5.5. Removal	14
6. Group Function Set	15
6.1. Register a Group	15
6.2. Group Removal	17
7. RD Lookup Function Set	18
8. New Link-Format Attributes	22
8.1. Resource Instance 'ins' attribute	23
8.2. Export 'exp' attribute	23
9. Security Considerations	23
10. IANA Considerations	24
11. Acknowledgments	24
12. Changelog	24
13. References	25
13.1. Normative References	25
13.2. Informative References	26
Authors' Addresses	26

1. Introduction

The Constrained RESTful Environments (CoRE) work aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC5988]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. This specification however only describes how to discover resources from the web server that hosts them by requesting `/.well-known/core`. In many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resource descriptions. Furthermore, new link attributes useful in conjunction with a Resource Directory are defined. Although the examples in this document show the use of these interfaces with CoAP [I-D.ietf-core-coap], they may be applied in an equivalent manner to HTTP [RFC2616].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. The term "byte" is used in its now customary sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [I-D.ietf-core-coap]. The URI Template format is used to describe the REST interfaces defined in this specification [RFC6570]. This specification makes use of the following additional terminology:

Resource Directory

An web entity that stores information about web resources and implements the REST interfaces defined in this specification for registration and lookup of those resources.

Domain

In the context of a Resource Directory, a domain is a logical grouping of endpoints. This specification assumes that the list of Domains supported by an RD is pre-configured by that RD.

Group

In the context of a Resource Directory, a group is a logical grouping of endpoints for the purpose of group communications. All groups within a domain are unique.

Endpoint

An endpoint (EP) is a term used to describe a web server or client in [I-D.ietf-core-coap]. In the context of this specification an endpoint is used to describe a web server that registers resources to the Resource Directory. An endpoint is identified by its endpoint name, which is included during registration, and is unique within the associated domain of the registration.

3. Architecture and Use Cases

The resource directory architecture is shown in Figure 1. A Resource Directory (RD) is used as a repository for Web Links [RFC5988] about resources hosted on other web servers, which are called endpoints (EP). An endpoint is a web server associated with an IP address and port, thus a physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain sets of Web Links (called resource directory entries), for the RD to validate entries, and for clients to lookup resources from the RD. Endpoints themselves can also act as clients. An RD can be logically segmented by the use of Domains. The domain an endpoint is associated with can be defined by the RD or configured by an outside entity.

Endpoints are assumed to proactively register and maintain resource directory entries on the RD, which are soft state and need to be periodically refreshed. An endpoint is provided with interfaces to register, update and remove a resource directory entry. Furthermore, a mechanism to discover a RD using the CoRE Link Format is defined. It is also possible for an RD to proactively discover Web Links from endpoints and add them as resource directory entries, or to validate existing resource directory entries. A lookup interface for discovering any of the Web Links held in the RD is provided using the CoRE Link Format.

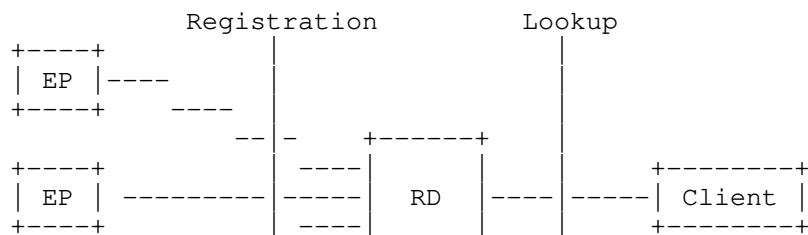




Figure 1: The resource directory architecture.

3.1. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users, i.e. machines. The machines are connected directly to a mobile network using appropriate embedded air interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. From the system design point of view, the ambition is to design horizontal solutions that can enable utilization of machines in different applications depending on their current availability and capabilities as well as application requirements, thus avoiding silo like solutions. One of the crucial enablers of such design is the ability to discover resources (machines - endpoints) capable of providing required information at a given time or acting on instructions from the end users.

In a typical scenario, during a boot-up procedure (and periodically afterwards), the machines (endpoints) register with a Resource Directory (for example EPs installed on vehicles enabling tracking of their position for the fleet management purposes and monitoring environment parameters) hosted by the mobile operator or somewhere else in the network, periodically a description of its own capabilities. Due to the usual network configuration of mobile networks, the EPs attached to the mobile network do not have routable addresses. Therefore, a remote server is usually used to provide proxy access to the EPs. The address of each (proxy) endpoint on this server is included in the resource description stored in the RD. The users, for example mobile applications for environment monitoring, contact the RD, look-up the endpoints capable of providing information about the environment using appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server) and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look-up for EPs deployed on the vehicles the application is responsible for.

3.2. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of M2M web services. The use of CoRE in home automation across multiple subnets is described in [I-D.brandt-coap-subnet-discovery] and in commercial building automation in [I-D.vanderstok-core-bc]. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices.

The exporting of resource information to other discovery systems is also important in these automation applications. In home automation there is a need to interact with other consumer electronics, which may already support DNS-SD, and in building automation larger resource directories or DNS-SD covering multiple buildings.

4. Simple Directory Discovery

Not all endpoints hosting resources are expected to know how to implement the Resource Directory Function Set and thus explicitly register with a Resource Directory (or other such directory server). Instead, simple endpoints can implement the generic Simple Directory Discovery approach described in this section. An RD implementing this specification MUST implement Simple Directory Discovery. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the endpoint makes available the hosted resources that it wants to be discovered, as links on its /.well-known/core interface as specified in [RFC6690].

The endpoint then finds one or more IP addresses of the directory server it wants to know about its resources as described in Section 4.1.

An endpoint that wants to make itself discoverable occasionally sends a POST request to the /.well-known/core URI of any candidate directory server that it finds. The body of the POST request is either

- o empty, in which case the directory server is encouraged by this POST request to perform GET requests at the requesting server's default discovery URI.

or

- o a non-empty link-format document, which indicates the specific services that the requesting server wants to make known to the directory server.

The directory server integrates the information it received this way into its resource directory. It MAY make the information available to further directories, if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

The following example shows an endpoint using simple resource discovery, by simply sending a POST with its links in the body to a directory.

```

EP                                     RD
|                                     |
| -- POST /.well-known/core "</sen/temp>..." ----> |
|                                     |
| <---- 2.01 Created -----> |
|                                     |

```

4.1. Finding a Directory Server

Endpoints that want to contact a directory server can obtain candidate IP addresses for such servers in a number of ways.

In a 6LoWPAN, good candidates can be taken from:

- o specific static configuration (e.g., anycast addresses), if any,
- o the ABRO option of 6LoWPAN-ND [RFC6775],
- o other ND options that happen to point to servers (such as RDNSS),
- o DHCPv6 options that might be defined later.

In networks with more inexpensive use of multicast, the candidate IP address may be a well-known multicast address, i.e. directory servers are found by simply sending POST requests to that well-known multicast address (details TBD).

As some of these sources are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. E.g., an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

5. Resource Directory Function Set

This section defines the REST interfaces between an RD and endpoint servers, which is called the Resource Directory Function Set. Although the examples throughout this section assume use of CoAP [I-D.ietf-core-coap], these REST interfaces can also be realized using HTTP [RFC2616]. An RD implementing this specification MUST support the discovery, registration, update, and removal interfaces defined in this section and MAY support the validation interface. For the purpose of validation, an endpoint implementing this specification SHOULD support ETag validation on /.well-known/core (which is very straightforward for static /.well-known/core link documents).

Resource directory entries are designed to be easily exported to other discovery mechanisms such as DNS-SD. For that reason, parameters that would meaningfully be mapped to DNS are limited to a maximum length of 63 bytes.

5.1. Discovery

Before an endpoint can make use of an RD, it must first know the RD's IP address, port and the path of its RD Function Set. There can be several mechanisms for discovering the RD including assuming a default location (e.g. on an Edge Router in a LoWPAN), by assigning an anycast address to the RD, using DHCP, or by discovering the RD using the CoRE Link Format (also see Section 4.1). This section defines discovery of the RD using the well-known interface of the CoRE Link Format [RFC6690] as the required mechanism. It is however expected that RDs will also be discoverable via other methods depending on the deployment.

Discovery is performed by sending either a multicast or unicast GET request to /.well-known/core and including a Resource Type (rt) parameter [RFC6690] with the value "core.rd" in the query string. Likewise, a Resource Type parameter value of "core.rd-lookup" is used to discover the RD Lookup Function Set. Upon success, the response will contain a payload with a link format entry for each RD discovered, with the URL indicating the root resource of the RD.

When performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network.

An RD implementation of this specification **MUST** support query filtering for the `rt` parameter as defined in [RFC6690].

The discovery request interface is specified as follows:

Interaction: EP -> RD

Method: GET

URI Template: `/.well-known/core{?rt}`

URI Template Variables:

`rt` := Resource Type (optional). MAY contain the value "core.rd", "core.rd-lookup", "core.rd-group" or "core.rd*"

Content-Type: application/link-format (if any)

The following response codes are defined for this interface:

Success: 2.05 "Content" with an application/link-format payload containing a matching entry for the RD resource.

Failure: 4.04 "Not Found" is returned in case no matching entry is found for a unicast request.

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

The following example shows an endpoint discovering an RD using this interface, thus learning that the base RD resource is at `/rd`. Note that it is up to the RD to choose its base RD resource, although it is recommended to use the base paths specified here where possible.

```

EP                                     RD
|                                     |
| ----- GET /.well-known/core?rt=core.rd* -----> |
|                                     |
| <----- 2.05 Content "</rd>; rt="core.rd" ----- |
|                                     |

```

Req: GET coap://[ff02::1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd",
</rd-lookup>;rt="core.rd-lookup",
</rd-group>;rt="core.rd-group"

5.2. Registration

After discovering the location of an RD Function Set, an endpoint MAY register its resources using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format along with query string parameters indicating the name of the endpoint, its domain and the lifetime of the registration. All parameters except the endpoint name are optional. It is expected that other specifications MAY define further parameters (it is to be determined if a registry of parameters is needed for this purpose). The RD then creates a new resource or updates an existing resource in the RD and returns its location. An endpoint MUST use that location when refreshing registrations using this interface. Endpoint resources in the RD are kept active for the period indicated by the lifetime parameter. The endpoint is responsible for refreshing the entry within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameter does not create multiple RD entries.

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: /{+rd}{?ep,d,et,lt,con}

URI Template Variables:

RD Function Set path (mandatory). This is the path of the RD Function Set. An RD SHOULD use the value "rd" for this variable whenever possible.

Endpoint (mandatory). The endpoint identifier or name of the registering node, unique within that domain. The maximum length of this parameter is 63 bytes.

Domain (optional). The domain to which this endpoint belongs. The maximum length of this parameter is 63 bytes. Optional. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain.

Endpoint Type (optional). The semantic type of the endpoint. The maximum length of this parameter is 63 bytes. Optional.

Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) SHOULD be assumed.

Context (optional). This parameter sets the scheme, address and port at which this server is available in the form scheme://host:port. Optional. In the absence of this parameter the scheme of the protocol, source IP address and source port of the register request are assumed.

Content-Type: application/link-format

The following response codes are defined for this interface:

Success: 2.01 "Created". The Location header MUST be included with the new resource entry for the endpoint. This Location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration (update, delete).

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows an endpoint with the name "node1" registering two resources to an RD using this interface. The resulting location /rd/4521 is just an example of an RD generated location.

```

EP                                     RD
|                                     |
| --- POST /rd?ep=node1 "</sensors..." -----> |
|                                     |

```

```
| <-- 2.01 Created Location: /rd/4521 ----- |
```

```
Req: POST coap://rd.example.com/rd?ep=node1
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor"
```

```
Res: 2.01 Created
Location: /rd/4521
```

5.3. Update

The update interface is used by an endpoint to refresh or update its registration with an RD. To use the interface, the endpoint sends a PUT request to the resource returned in the Location option in the response to the first registration. An update MAY contain registration parameters if there have been changes since the last registration or update. Parameters that have not changed SHOULD NOT be included in an update. Upon receiving an update request, the RD resets the timeout for that endpoint and stores the values of the parameters included in the update (if any).

The update request interface is specified as follows:

Interaction: EP -> RD

Method: PUT

URI Template: /{+location}{?et,lt,con}

URI Template Variables:

This is the Location path returned by the RD as a result of a successful registration.

Endpoint Type (optional). The semantic type of the endpoint. The maximum length of this parameter is 63 bytes. Optional.

Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) SHOULD be assumed.

Context (optional). This parameter sets the scheme, address and port at which this server is available in the form scheme:/host:port. Optional. In the absence of this parameter the

scheme of the protocol, source IP address and source port used to register are assumed.

Content-Type: None

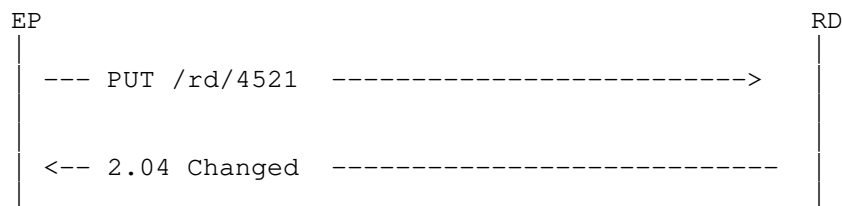
The following response codes are defined for this interface:

Success: 2.04 "Changed" in the update was successfully processed.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows an endpoint updating a new set of resources to an RD using this interface.



Req: PUT /rd/4521

Res: 2.04 Changed

5.4. Validation

In some cases, an RD may want to validate that it has the latest version of an endpoint's resources. This can be performed with a GET on the well-known interface of the CoRE Link Format including the latest ETag stored for that endpoint. For the purpose of validation, an endpoint implementing this specification SHOULD support ETag validation on /.well-known/core.

The validation request interface is specified as follows:

Interaction: RD -> EP

Method: GET

Path: /.well-known/core

Parameters: None

ETag: The ETag option MUST be included

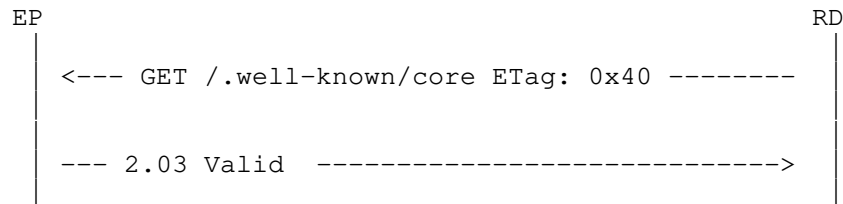
The following responses codes are defined for this interface:

Success: 2.03 "Valid" in case the ETag matches

Success: 2.05 "Content" in case the ETag does not match, the response MUST include the most recent resource representation (application/link-format) and its corresponding ETag.

Failure: 4.00 "Bad Request". Malformed request.

The following examples shows a successful validation.



Req: GET /.well-known/core

ETag: 0x40

Res: 2.03 Valid

5.5. Removal

Although RD entries have soft state and will eventually timeout after their lifetime, an endpoint SHOULD explicitly remove its entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: DELETE

URI Template: `/{"location"}`

URI Template Variables:

This is the Location path returned by the RD as a result of a successful registration.

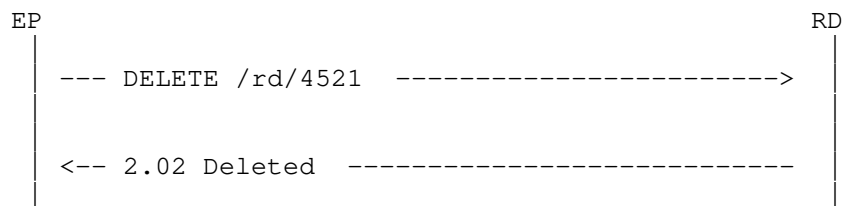
The following responses codes are defined for this interface:

Success: 2.02 "Deleted" upon successful deletion

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following examples shows successful removal of the endpoint from the RD.



Req: DELETE /rd/4521

Res: 2.02 Deleted

6. Group Function Set

This section defines a function set for the creation of groups of endpoints for the purpose of managing and looking up endpoints for group operations. The group function set is similar to the resource directory function set, in that a group may be created or removed. However unlike an endpoint entry, a group entry consists of a list of endpoints and does not have a lifetime associated with it. In order to make use of multicast requests with CoAP, a group MAY have a multicast address associated with it.

6.1. Register a Group

In order to create a group, a management entity used to configure groups, makes a request to the RD indicating the name of the group to create (or update), the optional domain the group belongs to, and the optional multicast address of the group. The registration message includes the list of endpoints that belong to that group. If an endpoint has already registered with the RD, the RD attempts to use the context of the endpoint from its RD endpoint entry. If the client registering the group knows the endpoint has already registered, then it MAY send a blank target URI for that endpoint link when registering the group.

The registration request interface is specified as follows:

Interaction: Manager -> RD

Method: POST

URI Template: `/[+rd-group]{?gp,d,con}`

URI Template Variables:

RD Group Function Set path (mandatory). This is the path of the RD Group Function Set. An RD SHOULD use the value "rd-group" for this variable whenever possible.

Group Name (mandatory). The name of the group to be created or replaced, unique within that domain. The maximum length of this parameter is 63 bytes.

Domain (optional). The domain to which this group belongs. The maximum length of this parameter is 63 bytes. Optional. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain.

Context (optional). This parameter is used to set the IP multicast address at which this server is available in the form `scheme://multicast-address:port`. Optional. In the absence of this parameter no multicast address is configured.

Content-Type: `application/link-format`

The following response codes are defined for this interface:

Success: 2.01 "Created". The Location header MUST be included with the new group entry. This Location MUST be a stable identifier generated by the RD as it is used for delete operations on this registration.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows a group with the name "lights" registering two endpoints to an RD using this interface. The resulting location /rd-group/12 is just an example of an RD generated group location.

EP	RD
<pre> - POST /rd-group?gp=lights "<>;ep=node1..." --> </pre>	<pre> <---- 2.01 Created Location: /rd-group/12 ----> </pre>

Req: POST coap://rd.example.com/rd-group?gp=lights

Payload:

<>;ep="node1",

<>;ep="node2"

Res: 2.01 Created

Location: /rd-group/12

6.2. Group Removal

A group can be removed simply by sending a removal message to the location returned when registering the group. Removing a group MUST NOT remove the endpoints of the group from the RD.

The removal request interface is specified as follows:

Interaction: Manager -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

This is the Location path returned by the RD as a result of a successful group registration.

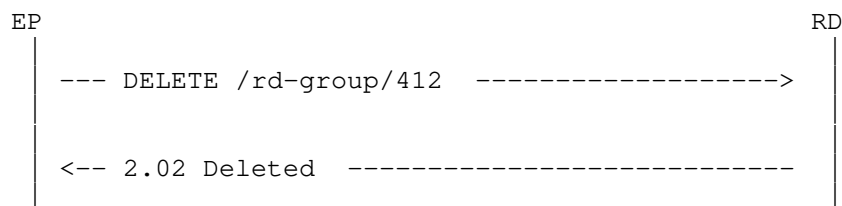
The following responses codes are defined for this interface:

Success: 2.02 "Deleted" upon successful deletion

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following examples shows successful removal of the group from the RD.



Req: DELETE /rd-group/12

Res: 2.02 Deleted

7. RD Lookup Function Set

In order for an RD to be used for discovering resources registered with it, a lookup interface can be provided using this function set. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. Atom or HTML Link) or using more advanced interfaces (e.g. supporting context or semantic based lookup).

This function set allows lookups for domains, groups, endpoints and resources using attributes defined in the RD Function Set and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) in CoRE Link Format corresponding to the type of lookup. The target of these links SHOULD be the actual location of the domain, endpoint or resource, but MAY be an intermediate proxy e.g. in the case of an HTTP lookup interface for CoAP endpoints. Multiple query parameters MAY be included in a lookup, all included parameters MUST match for a resource to be returned. The character '*' MAY be included at the end of a parameter value as a wildcard operator.

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: `/({+rd-lookup-base})/{lookup-type}{?d,ep,gp,et,rt,page,count,resource-param}`

Parameters:

`rd-lookup-base` := RD Lookup Function Set path (mandatory). This is the path of the RD Lookup Function Set. An RD SHOULD use the value "rd-lookup" for this variable whenever possible.

`lookup-type` := ("d", "ep", "res", "gp") (mandatory) This variable is used to select the kind of lookup to perform (domain, endpoint or resource).

`ep` := Endpoint (optional). Used for endpoint, group and resource lookups.

`d` := Domain (optional). Used for domain, group, endpoint and resource lookups.

`page` := Page (optional). Parameter can not be used without the count parameter. Results are returned from result set in pages that contains 'count' results starting from index (page * count).

`count` := Count (optional). Number of results is limited to this parameter value. If the parameter is not present, then an RD implementation specific default value SHOULD be used.

`rt` := Resource type (optional). Used for group, endpoint and resource lookups.

`rt` := Endpoint type (optional). Used for group, endpoint and resource lookups.

`resource-param` := Link attribute parameters (optional). Any link attribute as defined in Section 4.1 of [RFC6690], used for resource lookups.

The following responses codes are defined for this interface:

Success: 2.05 "Content" with an application/link-format payload containing a matching entries for the lookup.

Failure: 4.04 "Not Found" in case no matching entry is found for a unicast request.

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows a client performing a resource lookup:

```

Client                                                     RD
|                                                         |
| ----- GET /rd-lookup/res?rt=temperature ----->    |
|                                                         |
| <-- 2.05 Content "<coap://{node1/temp};rt="temperature" ----|
|                                                         |

```

Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content
<coap://{ip:port}/temp>

The following example shows a client performing an endpoint lookup:

```

Client                                                     RD
|                                                         |
| ----- GET /rd-lookup/ep?et=power-node ----->    |
|                                                         |
| <-- 2.05 Content "<coap://{ip:port}>;ep="node5" -----|
|                                                         |

```

Req: GET /rd-lookup/ep?et=power-node

```

Res: 2.05 Content
<coap://{ip:port}>;ep="node5",
<coap://{ip:port}>;ep="node7"

```

The following example shows a client performing a domain lookup:

```

Client                                     RD
|                                         |
|----- GET /rd-lookup/d ----->      |
|                                         |
|<-- 2.05 Content "</rd>;d=domain1,</rd>;d=domain2 -----
|                                         |

```

```

Req: GET /rd-lookup/d

```

```

Res: 2.05 Content
</rd>;d="domain1",
</rd>;d="domain2"

```

The following example shows a client performing a group lookup for all groups:

```

Client                                     RD
|                                         |
|----- GET /rd-lookup/gp ----->      |
|                                         |
|<-- 2.05 Content </rd-group/l2>;gp="lights1";d="domain1" --
|                                         |

```

```

Req: GET /rd-lookup/gp

```

```

Res: 2.05 Content
</rd-group/l2>;gp="lights1";d="domain1"

```

The following example shows a client performing a lookup for all endpoints in a particular group:

```

Client                                                     RD
|                                                         |
|----- GET GET /rd-lookup/ep?gp=lights1----->       |
|                                                         |
|<-- 2.05 Content "</rd>;d=domain1,</rd>;d=domain2 -----|
|                                                         |

```

Req: GET /rd-lookup/ep?gp=lights1

Res: 2.05 Content
 <coap://host:port>;ep="node1",
 <coap://host:port>;ep="node2",

The following example shows a client performing a lookup for all groups an endpoint belongs to:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/gp?ep=node1 ----->             |
|                                                         |
|<-- 2.05 Content "</rd>;d=domain1,</rd>;d=domain2 -----|
|                                                         |

```

Req: GET /rd-lookup/gp?ep=node1

Res: 2.05 Content
 <coap://host:port>;gp="lights1";ep="node1",

8. New Link-Format Attributes

When using the CoRE Link Format to describe resources being discovered by or posted to a resource directory service, additional information about those resources is useful. This specification defines the following new attributes for use in the CoRE Link Format [RFC6690]:

```
link-extension    = ( "ins" "=" quoted-string ) ; Max 63 bytes
link-extension    = ( "exp" )
```

8.1. Resource Instance 'ins' attribute

The Resource Instance "ins" attribute is an identifier for this resource, which makes it possible to distinguish from other similar resources. This attribute is similar in use to the "Instance" portion of a DNS-SD record, and SHOULD be unique across resources with the same Resource Type attribute in the domain it is used. A Resource Instance might be a descriptive string like "Ceiling Light, Room 3", a short ID like "AF39" or a unique UUID or iNumber. This attribute is used by a Resource Directory to distinguish between multiple instances of the same resource type within a system.

This attribute MUST be no more than 63 bytes in length. The resource identifier attribute MUST NOT appear more than once in a link description.

8.2. Export 'exp' attribute

The Export "exp" attribute is used as a flag to indicate that a link description MAY be exported by a resource directory to external directories.

The CoRE Link Format is used for many purposes between CoAP endpoints. Some are useful mainly locally, for example checking the observability of a resource before accessing it, determining the size of a resource, or traversing dynamic resource structures. However, other links are very useful to be exported to other directories, for example the entry point resource to a functional service.

9. Security Considerations

This document needs the same security considerations as described in Section 7 of [RFC5988] and Section 6 of [RFC6690]. The /.well-known/core resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [I-D.ietf-core-coap].

Access control SHOULD be performed separately for the RD Function Set and the RD Lookup Function Set, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the domain, endpoint or resource level.

10. IANA Considerations

"core.rd", "core.rd-group" and "core.rd-lookup" resource types need to be registered with the resource type registry defined by [RFC6690].

The "exp" attribute needs to be registered when a future Web Linking attribute is created.

11. Acknowledgments

Szymon Sasin, Kerry Lynn, Esko Dijk, Peter van der Stok, Anders Brandt, Matthieu Vial, Sampo Ukkola and Linyi Tian have provided helpful comments, discussions and ideas to improve and shape this document. The authors would also like to thank their colleagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

12. Changelog

Changes from -05 to WG Document -00:

- o Updated the version and date.

Changes from -04 to -05:

- o Restricted Update to parameter updates.
- o Added pagination support for the Lookup interface.
- o Minor editing, bug fixes and reference updates.
- o Added group support.
- o Changed rt to et for the registration and update interface.

Changes from -03 to -04:

- o Added the ins= parameter back for the DNS-SD mapping.

- o Integrated the Simple Directory Discovery from Carsten.
- o Editorial improvements.
- o Fixed the use of ETags.

Changes from -02 to -03:

- o Changed the endpoint name back to a single registration parameter `ep=` and removed the `h=` and `ins=` parameters.
- o Updated REST interface descriptions to use RFC6570 URI Template format.
- o Introduced an improved RD Lookup design as its own function set.
- o Improved the security considerations section.
- o Made the POST registration interface idempotent by requiring the `ep=` parameter to be present.

Changes from -01 to -02:

- o Added a terminology section.
- o Changed the including of an ETag in registration or update to a MAY.
- o Added the concept of an RD Domain and a registration parameter for it.
- o Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- o Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

13. References

13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

13.2. Informative References

- [I-D.brandt-coap-subnet-discovery]
Brandt, A., "Discovery of CoAP servers across subnets", draft-brandt-coap-subnet-discovery-00 (work in progress), March 2011.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-14 (work in progress), March 2013.
- [I-D.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-05 (work in progress), October 2011.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.

Authors' Addresses

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach@sensinode.com

Srdjan Krco
Ericsson

Email: srdjan.krco@ericsson.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Secure initial-key reconfiguration for resource constrained devices
draft-kang-core-secure-reconfiguration-00

Abstract

This document presents a secure method to configure a key for a node when it initially joins to network that is currently in operation. The method is suited for a scenario, where resource constrained objects are interconnected with each other and thus form a network called Internet of Things. It is assumed that communications for all nodes are based on TCP/IP protocols and some of the nodes use the constrained application protocol (CoAP). The method does not cover all operations of secure bootstrapping, but it is intended to securely support self-reconfiguration of the pre-installed temporary key of new node.

Status of this Memo

This Internet-Draft is submitted to IETF 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 <http://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 14, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
2. Terminology	5
3. System Architecture	6
4. Process Flow	8
5. Security Considerations	9
6. IANA Considerations	10
7. Acknowledgments	10
8. References	10
8.1. Normative References	10
8.2. Informative References	11

1. Introduction

A rapidly growing number and various types of devices including smart small things such as sensors and actuators are trying to connect with Internet as time goes by. This draft presents a simple but efficient approach to reconfigure a secure key for resource constrained small things that are often defined as network nodes having 8 bit processing microcontrollers with limited amounts of memory. The network is also constrained one (e.g. 6LoWPAN having high packet error rates and a typical throughput of 10s of kbit/s) [CoAP].

Pre-shared key (PSK) based secure schemes are well known and frequently used for various security services in Internet. All such schemes strictly assume that the PSK is only known to two entities involved in current security service. Consequently, the security of the schemes are compromised if the assumption is broken.

However, it is still not clear how PSK of resource constrained node can be initially configured in a secure manner in Internet of things (IoT). Typically, things used for IoT might be manufactured and installed by different subjects (simply person) [SecCons]. That is, in general situation, a system administrator may make orders to several different installers. After that, each of the installers purchases one or more different set of things from one or more different manufacturers. It is also unlikely that a single subject installs all nodes used for a large application domain (e.g. all nodes in huge building).

This draft considers a scenario, where nodes are initially configured by an installer during bootstrapping phase. If a PSK is also required to be configured in this phase, the trust between installer and system administrator is extremely important. This is not easy process. Even further, if the case is settled, there are several secure threats and vulnerabilities to be handled.

The basic idea of the method specified in this document is motivated from a lock of suitcase. Simple and default password such as '0000' or '1234' is initially setup on a lock of suitcase in selling. Owner can change the password after purchasing. In our method, similarly, initial key of a node is configured by installer during bootstrapping phase. When the node join to an existing network, the key (i.e. PSK) can be securely reconfigured.

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 [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

This draft uses notations and abbreviations as follows.

SBI(i)

Shorten abbreviation of a secure bootstrapping initiator i (i.e. new node required to be reconfigured)

SBR(c)

Shorten abbreviation of a secure bootstrapping respondent c (i.e. a type of controller)

SBS(s)

Shorten abbreviation of a secure bootstrapping server s (i.e. an authenticated register or authentication server)

ID_A

Denoting 32bits identifier (ID) of entity A

NID_A

Denoting network ID used for access to communication entity A (e.g. IPv4 or IPv6 address and port number).

RN_A

Denoting 128bits integer used for a secure random number generated by entity A; for example, a random number generated by SBI is referred to as RN_i.

IK_N

Denoting 128bits symmetric key pre-installed by installer or manufacturer for node N; The key is used for a partial transaction of mutual authentication and derivation of PSK (see section 4 in detail).

PSK

Shorten abbreviation of a 128bits pre-shared key derived from the IK. The PSK is a shared key between a node and authenticated register (or authentication server) in a specific service domain. A PSK can be used to derive session keys for various security protocols designed by service administrator (see [RFC4764] for example).

TS

Denoting time stamp of operation; it enables sender (TS generator) to inform timeliness and uniqueness to receiver.

SK_{cs}

Denoting a 128bits symmetric key shared between entity c and s.

||

Notation used to denote concatenation of data.

V

Notation used to denote a logical operator Exclusive OR.

E(M, SK)

Denoting a function to encrypt a plain text 'M' by using a symmetric key SK.

D(C, SK)

Denoting a function to decrypt a cipher text 'C' by using a symmetric key SK.

Other security related terminologies used in this document are based on [RFC4949].

3. System Architecture

Secure bootstrapping is regarded as a difficult problem in Internet of Things. This is mainly because lots of things connected to Internet are resource constrained. Especially, user interface they

have is not enough for doing configurations manually by person (i.e. inadequate or even no in/out equipment such as display or keyboard).

As one of solutions, this document proposes a method which allows a node to reconfigure a symmetric key automatically upon joining to existing network.

The method of this document is based on a straightforward scenario, where resource constrained things in IoT such as sensors or actuators are generally designed and manufactured according to their own specific tasks in advance. Also, a pre-defined controller covers and communicates with his associated things according to his rolls defined in a service domain. For example, a thermostat manages and communicates several temperature sensors, humidity sensors, windows, heating controller, air conditioner, and more. This document does not assume that a system administrator trusts an installer even though he makes orders for the installer. This is because trust and responsibility of installer who buys and install devices is different from those of system administrator.

In this scenario, the following transactions MUST be done prior to the key reconfiguration.

1. System administrator makes orders including setup information required to be initially configured. These are ID and NID of controller for each of nodes, temporary key used as an IK. In particular, all nodes handled by a single installer can share the same IK similarly to the default password for all suitcases manufactured by a single company.
2. System administrator also stores the same initial information for each of nodes in authentication server (or authenticated register). Note that a controller can also perform operations of an authentication server in case of a small network.
3. Installer purchases devices and then configures the information requested by the administrator in doing installation. Some of the information for a node may be pre-configured by manufacturer.
4. When a node joins to network, it knows NID of his associated controller with which he can communicate. Also, authentication server has lists of IDs for new nodes.
5. PSK reconfiguration phase is then started.

In order to make a practical and reasonable method, the proposed method requires only a single cryptographic primitive that is AES with 128bits length of key [AES]. All cryptographic primitives cannot be installed on resource restricted devices, mainly because of limited size of flash or RAM. For this reason, CoAP also do not consider all modes of cryptographic operations in DTLS which is a regarded secure protocol for CoAP applications. In case of establishing a CoAP session using a pre-shared key mod of DTLS, implementation of cipher suite TLS_PSK_WITH_AES_128_CCM_8 specified in [RFC6655] is mandatory.

4. Process Flow

There are three message exchanges between new node SBI(i) and network (SBR(c) and SBS(s)). A controller SBR(c) MAY include functions of both SBR(c) and SBS(s) depending on the size of application domain. Mutual authentication and PSK reconfiguration procedures are shown in Figure 1.

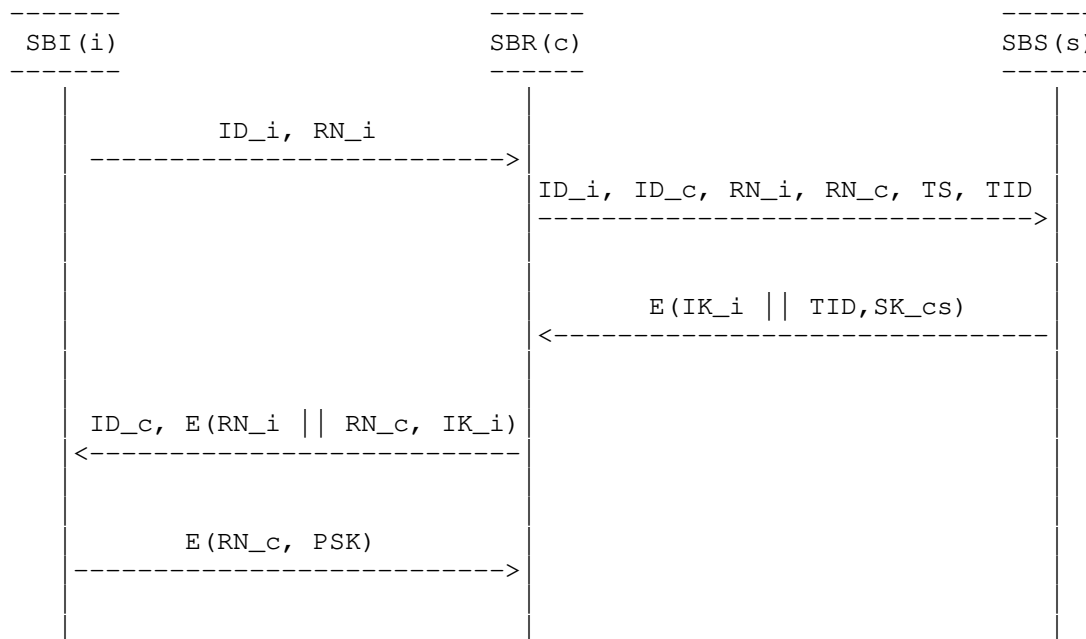


Figure 1 Message Exchange for PSK Reconfiguration

When a new node SBI(i) joins an existing network, he generates a random number RN_i and sends it with his identifier ID_i to SBR(s).

Upon receiving the message, SBR(c) generates a random number RN_c and a serial number used as a transaction ID (i.e. TID). Then he sends the two numbers with his ID_c, time stamp (TS) and the message received from SBI(i) to the authentication server SBS(s). TS allows SBR(s) to derive an expiration of key and verify the freshness of the arrived message. Specific period of the expiration of key (i.e. PSK) does not covered in this document.

The authentication server SBS(s) now can derive a new PSK for the node SBI(i) and replace the IK_i, which he initially stored, to the PSK, where the PSK for SBI(i) is derived as follows.

$$\text{PSK}_i = E(\text{RN}_i \text{ V } \text{RN}_c, \text{IK}_i)$$

At this moment, IK_i does not known to SBS(c). After reconfiguration of key for node SBI(i), SBS(s) encrypts the concatenation value of IK_i and ID_i with the symmetric key SK_cs which is a shared key between SBS(s) and SBR(c).

On receiving the encrypted value from SBS(s), SBR(c) can know the key IK_i thereby calculating PSK. SBR(c) encrypts the concatenation value of RN_i and RN_c with key IK_i. Then it sends the encryption value and his ID_r to SBI(i).

Finally, SBI(i) can reconfigure his PSK thereafter sending the encryption value of RN_c with the new key PSK to SBR(c) to verify himself.

5. Security Considerations

The method of this draft uses a single cryptographic primitive AES [AES]. Single cryptographic primitive implementation is rationally suited for the scenario where applications or services require a secure session (confidentiality of data) in IoT. Because small devices with low computing power and little storage are major entities. In this draft, a single primitive AES is used for secure bootstrapping (exactly PSK reconfiguration phase). Further, the PSK can be used for session key derivation and entity authentication.

As discussed in ESP-PSK [RFC4764], it goes without saying that a single cryptographic primitive may not support extensible security services such as identity protection, perfect forward secrecy and

others. However, small devices consisting of Internet of Things might not support all of security services inherently. Service developer should therefore define a scope of his service strictly and consider trade-off between capability and security.

Security analysis and evaluation of various aspects of the method remain to be done.

6. IANA Considerations

This memo includes no request to IANA

7. Acknowledgments

(TBD)

8. References

8.1. Normative References

- [RFC4764] F. Bersani, H. Tschofenig, "The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method", RFC 4764, January 2007.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.
- [RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July 2012.
- [CoAP] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [SecCons] O. Garcia-Morchon, S. Kumar, S. Keoh, R. Hummen, R. Struik, "Security Considerations in the IP-based Internet of Things", Internet draft (draft-garcia-core-security-06), September 2013.

[AES] National Institute of Standards and Technology, "Specification for the Advanced Encryption Standard (AES)", Federal Information Processing Standards (FIPS) 197, November 2001.

8.2. Informative References

[RFC2119] S. Brander, "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Author's Addresses

Namhi Kang
Duksung Women's University
Seoul Korea
Email: kang@duksung.ac.kr
URI: <http://www.duksung.ac.kr>

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: January 16, 2014

M. Kovatsch
ETH Zurich
O. Bergmann
Universitaet Bremen TZI
E. Dijk
Philips Research
X. He
Hitachi (China) R&D Corp.
C. Bormann, Ed.
Universitaet Bremen TZI
July 15, 2013

CoAP Implementation Guidance
draft-kovatsch-lwig-coap-01

Abstract

The Constrained Application Protocol (CoAP) is designed for resource-constrained nodes and networks, e.g., sensor nodes in a low-power lossy network (LLN). Yet to implement this Internet protocol on Class 1 devices (i.e., ~ 10 KiB of RAM and ~ 100 KiB of ROM) also lightweight implementation techniques are necessary. This document provides lessons learned from implementing CoAP for tiny, battery-operated networked embedded systems. The guidelines for transmission state management and developer APIs can also help with the implementation of CoAP for less constrained nodes.

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 <http://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 16, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Message Processing	3
2.1. Message Buffer Usage	4
2.2. Header Option Parsing and Management	5
2.2.1. On-the-fly Processing	5
2.2.2. Internal Data Structure	5
2.3. Retransmissions	6
2.4. Deduplication	7
2.4.1. Managing Peer MIDs	7
2.4.2. Resource-specific Deduplication	10
3. Transmission State Management	10
3.1. Request/Response Layer	10
3.2. Message Layer	11
4. Observing	12
5. Block-wise Transfers	13
6. Application Developer API	13
7. References	14
7.1. Normative References	14
7.2. Informative References	14
Authors' Addresses	15

1. Introduction

The Constrained Application Protocol [I-D.ietf-core-coap] has been designed specifically for machine-to-machine communication in networks with very constrained nodes. Typical application scenarios therefore include building automation and the Internet of Things. The major design objectives have been set on small protocol overhead, robustness against packet loss, and against high latency induced by small bandwidth shares or slow request processing in end nodes. To leverage integration of constrained nodes with the world-wide Internet, the protocol design was led by the REST architectural style that accounts for the scalability and robustness of the Hypertext Transfer Protocol [RFC2616].

Lightweight implementations benefit from this design in many respects: First, the use of Uniform Resource Identifiers (URIs) for naming resources and the transparent forwarding of their representations in a server-stateless request/response protocol make protocol translation to HTTP a straightforward task. Second, the set of protocol elements that are unavoidable for the core protocol and thus must be implemented on every node has been kept very small, minimizing the unnecessary accumulation of "optional" features. Options that - when present - are critical for message processing are explicitly marked as such to force immediate rejection of messages with unknown critical options. Third, the syntax of protocol data units is easy to parse and is carefully defined to avoid creation of state in servers where possible.

Although these features enable lightweight implementations of the Constrained Application Protocol, there is still a tradeoff between robustness and latency of constrained nodes on one hand and resource demands (such as battery consumption, dynamic memory needs, and static code size) on the other. The present document gives some guidance on possible strategies to solve this tradeoff for very constrained nodes (Class 1 in [I-D.ietf-lwig-terminology]). The main focus is on servers as this is deemed the predominant case where CoAP applications are faced with tight resource constraints.

Additional considerations for the implementation of CoAP on tiny sensors are given in [I-D.arkko-core-sleepy-sensors].

2. Message Processing

For constrained nodes of Class 1 or even Class 2, the most limiting factors for (wireless) network communication usually are memory size and battery lifetime. Most applications therefore try to minimize internal buffer space for both transmit and receive operations, and to maximize sleeping cycles.

In the programming styles supported by very simple operating systems, preemptive multi-threading is not an option. Instead, all operations are triggered by an event loop system, e.g., in a send-receive-dispatch cycle. It is also common practice to allocate memory statically to ensure stable behavior, as no memory management unit (MMU) or other abstractions are available. For a CoAP node, the two key parameters for memory usage are the number of (re)transmission buffers and the maximum message size that must be supported by each buffer. Often the maximum message size is set far below the 1280-byte MTU of 6LoWPAN to allow more than one open Confirmable transmission at a time (in particular for observe notifications). Note that implementations on constrained platforms often not even support the full MTU. Larger messages must then use block-wise transfers [I-D.ietf-core-block], while a good tradeoff between 6LoWPAN fragmentation and CoAP header overhead must be found. Usually the amount of available free RAM dominates this decision. For Class 1 devices, the maximum message size is typically 128 or 256 bytes plus an estimate of the maximum header size with a worst case option setting.

2.1. Message Buffer Usage

The cooperative multi-threading of an event loop system allows to optimize memory usage through in-place processing and reuse of buffers, in particular the IP buffer provided by the OS.

CoAP servers can significantly benefit from in-place processing, as they can create responses directly in the incoming IP buffer. Note that an embedded OS usually only has a single buffer for incoming and outgoing IP packets. The first few bytes of the basic header are usually parsed into an internal data structure and can be overwritten without harm. Thus, empty ACKs and RST messages can promptly be assembled and sent using the IP buffer. Also when a CoAP server only sends piggy-backed or Non-confirmable responses, no additional buffer is required at the application layer. This, however, requires careful timing so that no incoming data is overwritten before it was processed. Because of cooperative multi-threading, this requirement is relaxed, though. Once the message is sent, the IP buffer can accept new messages again. This does not work for Confirmable messages, however. They need to be stored for retransmission and would block any further IP communication.

Depending on the number of requests that can be handled in parallel, an implementation might create a stub response filled with any option that has to be copied from the original request to the separate response, especially the Token option. The drawback of this technique is that the server must be prepared to receive retransmissions of the previous (Confirmable) request to which a new

acknowledgement must be generated. If memory is an issue, a single buffer can be used for both tasks: Only the message type and code must be updated, changing the message id is optional. Once the resource representation is known, it is added as new payload at the end of the stub response. Acknowledgements still can be sent as described before as long as no additional options are required to describe the payload.

2.2. Header Option Parsing and Management

There are two alternatives to handle the header: Either process the header on the fly when an option is accessed or initially parse all values into an internal data structure.

2.2.1. On-the-fly Processing

The advantage of on-the-fly processing is that the compact encoding saves memory and fully reuses the buffer for incoming messages. The basic message header information should always be copied into an internal data structure, as Message ID and/or Token are required for request/response matching and generating the response. Once the message is accepted for further processing, the set of options contained in the received message must be decoded to check for unknown critical options. To avoid multiple passes through the option list, the option parser might maintain a bit-vector where each bit represents an option number that is present in the received request. With the wide and sparse range of option numbers, the number itself cannot be used to indicate the number of left-shift operations to mask the corresponding bit. Hence, an implementation-specific enum of supported options should be used to mask the present options of a message in the bitmap. In addition, the byte index of every option can be added to a sparse list (e.g., a one-dimensional array) for fast retrieval.

Once the option list has been processed at least up to the highest option number that is supported by the application, any known critical option and all elective options can be masked out to determine if any unknown critical option was present. If this is the case, this information can be used to create a 4.02 response accordingly. (Note that the remaining options also must be processed to add further critical options included in the original request.)

2.2.2. Internal Data Structure

Using an internal data structure for all parsed options has advantages when processing the values, as they are already in a variable of corresponding type and integers in host byte order. The incoming payload and byte strings of the header can be accessed

directly in its IP buffer using pointers. This approach also benefits from a bitmap. Otherwise special values must be reserved to encode an unset option, which might require a larger type than required for the actual value range (e.g., a 32-bit integer instead of 16-bit).

The byte strings (e.g., the URI) are usually not required when generating the response. Thus, this alternative also facilitates the usage of the IP buffer for message assembly - all important values are copied from the shared incoming/outgoing buffer.

Setting options for outgoing messages is also easier with an internal data structure. Application developers can set options independent from the option number order required for the delta encoding. The CoAP encoding is then applied in a serialization step before sending. On-the-fly processing might require extensive memmove operations to insert new header options or needs to restrict developers to set options in order.

2.3. Retransmissions

CoAP's reliable transmissions require the before-mentioned retransmission buffers. Messages, such as the requests of a client, should be stored in serialized form. For servers, retransmissions apply for Confirmable separate responses and Confirmable notifications [I-D.ietf-core-observe]. As separate responses stem from long-lasting resource handlers, the response should be stored for retransmission instead of re-dispatching a stored request (which would allow for updating the representation). For Confirmable notifications, please see Section 2.6, as simply storing the response can break the concept of eventual consistency.

String payloads such as JSON require a buffer to print to. By splitting the retransmission buffer into header and payload part, it can be reused. First to generate the payload and then storing the CoAP message by serializing into the same memory. Thus, providing a retransmission for any message type can save the need for a separate application buffer. This, however, requires an estimation about the maximum expected header size to split the buffer and a memmove to concatenate the two parts.

For platforms that disable clock tick interrupts in sleep states, the application must take into consideration the clock deviation that occurs during sleep (or ensure to remain in idle state until the message has been acknowledged or the maximum number of retransmissions is reached). Since CoAP allows up to four retransmissions with a binary exponential back-off it could take up to 45 seconds until the send operation is complete. Even in idle

state, this means substantial energy consumption for low-power nodes. Implementers therefore might choose a two-step strategy: First, do one or two retransmissions and then, in the later phases of back-off, go to sleep until the next retransmission is due. In the meantime, the node could check for new messages including the acknowledgement for any Confirmable message to send.

2.4. Deduplication

If CoAP is used directly on top of UDP (i.e., in NoSec mode), it needs to cope with the fact that the UDP datagram transport can reorder and duplicate messages. (In contrast to UDP, DTLS has its own duplicate detection.) CoAP has been designed with protocol functionality such that rejection of duplicate messages is always possible. It is at the discretion of the receiver if it actually wants to make use of this functionality. Processing of duplicate messages comes at a cost, but so does the management of the state associated with duplicate rejection. Hence, a receiver may have good reasons to decide not to do the duplicate rejection. If duplicate rejection is indeed necessary, e.g., for non-idempotent requests, it is important to control the amount of state that needs to be stored.

2.4.1. Managing Peer MIDs

CoAP's duplicate rejection functionality can be straightforwardly implemented in a CoAP end-point by storing, for each remote CoAP end-point ("peer") that it communicates with, a list of recently received CoAP Message IDs (MIDs) along with some timing information. A CoAP message from a peer with a MID that is in the list for that peer can simply be discarded.

The timing information in the list can then be used to time out entries that are older than the `_expected` extent of the re-ordering_, an upper bound for which can be estimated by adding the `_potential retransmission window_` ([I-D.ietf-core-coap] section "Reliable Messages") and the time packets can stay alive in the network.

Such a straightforward implementation is suitable in case other CoAP end-points generate random MIDs. However, this storage method may consume substantial RAM in specific cases, such as:

- o many clients are making periodic, non-idempotent requests to a single CoAP server;
- o one client makes periodic requests to a large number of CoAP servers and/or requests a large number of resources; where servers happen to mostly generate separate CoAP responses (not piggy-backed);

For example, consider the first case where the expected extent of re-ordering is 50 seconds, and N clients are sending periodic POST requests to a single CoAP server during a period of high system activity, each on average sending one client request per second. The server would need $100 * N$ bytes of RAM to store the MIDs only. This amount of RAM may be significant on a RAM-constrained platform. On a number of platforms, it may be easier to allocate some extra program memory (e.g. Flash or ROM) to the CoAP protocol handler process than to allocate extra RAM. Therefore, one may try to reduce RAM usage of a CoAP implementation at the cost of some additional program memory usage and implementation complexity.

Some CoAP clients generate MID values by using a Message ID variable [I-D.ietf-core-coap] that is incremented by one each time a new MID needs to be generated. (After the maximum value 65535 it wraps back to 0.) We call this behavior "sequential" MIDs. One approach to reduce RAM use exploits the redundancy in sequential MIDs for a more efficient MID storage in CoAP servers.

Naturally such an approach requires, in order to actually reduce RAM usage in an implementation, that a large part of the peers follow the sequential MID behavior. To realize this optimization, the authors therefore RECOMMEND that CoAP end-point implementers employ the "sequential MID" scheme if there are no reasons to prefer another scheme, such as randomly generated MID values.

Security considerations might call for a choice for (pseudo)randomized MIDs. Note however that with truly randomly generated MIDs the probability of MID collision is rather high in use cases as mentioned before, following from the Birthday Paradox. For example, in a sequence of 52 randomly drawn 16-bit values the probability of finding at least two identical values is about 2 percent.

From here on we consider efficient storage implementations for MIDs in CoAP end-points, that are optimized to store "sequential" MIDs. Because CoAP messages may be lost or arrive out-of-order, a solution has to take into account that received MIDs of CoAP messages are not actually arriving in a sequential fashion, due to lost or reordered messages. Also a peer might reset and lose its MID counter(s) state. In addition, a peer may have a single Message ID variable used in messages to many CoAP end-points it communicates with, which partly breaks sequentiality from the receiving CoAP end-point's perspective. Finally, some peers might use a randomly generated MID values approach. Due to these specific conditions, existing sliding window bitfield implementations for storing received sequence numbers are typically not directly suitable for efficiently storing MIDs.

Table 1 shows one example for a per-peer MID storage design: a table with a bitfield of a defined length `_K_` per entry to store received MIDs (one per bit) that have a value in the range `[MID_i + 1 , MID_i + K]`.

MID base	K-bit bitfield	base time value
MID_0	010010101001	t_0
MID_1	111101110111	t_1
... etc.		

Table 1: A per-peer table for storing MIDs based on `MID_i`

The presence of a table row with base `MID_i` (regardless of the bitfield values) indicates that a value `MID_i` has been received at a time `t_i`. Subsequently, each bitfield bit `k` ($0 \dots K-1$) in a row `i` corresponds to a received MID value of `MID_i + k + 1`. If a bit `k` is 0, it means a message with corresponding MID has not yet been received. A bit 1 indicates such a message has been received already at approximately time `t_i`. This storage structure allows e.g. with `k=64` to store in best case up to 130 MID values using 20 bytes, as opposed to 260 bytes that would be needed for a non-sequential storage scheme.

The time values `t_i` are used for removing rows from the table after a preset timeout period, to keep the MID store small in size and enable these MIDs to be safely re-used in future communications. (Note that the table only stores one time value per row, which therefore needs to be updated on receipt of another MID that is stored as a single bit in this row. As a consequence of only storing one time value per row, older MID entries typically time out later than with a simple per-MID time value storage scheme. The end-point therefore needs to ensure that this additional delay before MID entries are removed from the table is much smaller than the time period after which a peer starts to re-use MID values due to wrap-around of a peer's MID variable. One solution is to check that a value `t_i` in a table row is still recent enough, before using the row and updating the value `t_i` to current time. If not recent enough, e.g. older than `N` seconds, a new row with an empty bitfield is created.) [Clearly, these optimizations would benefit if the peer were much more conservative about re-using MIDs than currently required in the protocol specification.]

The optimization described is less efficient for storing randomized MIDs that a CoAP end-point may encounter from certain peers. To solve this, a storage algorithm may start in a simple MID storage mode, first assuming that the peer produces non-sequential MIDs. While storing MIDs, a heuristic is then applied based on monitoring some "hit rate", for example, the number of MIDs received that have a Most Significant Byte equal to that of the previous MID divided by the total number of MIDs received. If the hit rate tends towards 1 over a period of time, the MID store may decide that this particular CoAP end-point uses sequential MIDs and in response improve efficiency by switching its mode to the bitfield based storage.

2.4.2. Resource-specific Deduplication

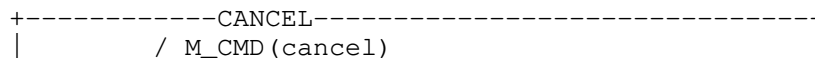
Deduplication is heavy for Class 1 devices, as the number of peer addresses can be vast. Servers should be kept stateless, i.e., the REST API should be designed idempotent whenever possible. When this is not the case, the resource handler could perform an optimized deduplication by exploiting knowledge about the application. Another, server-wide strategy is to only keep track of non-idempotent requests.

3. Transmission State Management

CoAP endpoints must keep transmission state to manage open requests, to handle the different response modes, and to implement reliable delivery at the message layer. The following finite state machines (FSMs) model the transmissions of a CoAP exchange at the request/response layer and the message layer. These layers are linked through actions. The M_CMD() action triggers a corresponding transition at the message layer and the FF_EVT() action triggers a transition at the request/response layer. The FSMs also use guard conditions to distinguish between information that is only available through the other layer (e.g., whether a request was sent using a CON or NON message).

3.1. Request/Response Layer

Figure 1 depicts the two states at the request/response layer of a CoAP client. When a request is issued, a "reliable_send" or "unreliable_send" is triggered at the message layer. The WAITING state can be left through three transitions: Either the client cancels the request and triggers cancellation of a CON transmission at the message layer, the client receives a failure event from the message layer, or a receive event containing a response.



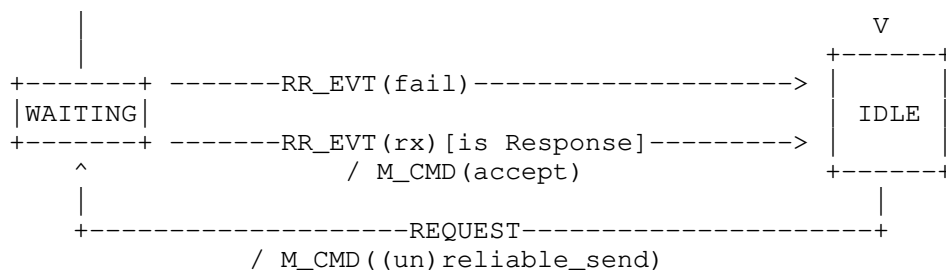


Figure 1: CoAP Client Request/Response Layer FSM

A server resource can decide at the request/response layer whether to respond with a piggy-backed or a separate response. Thus, there are two busy states in Figure 2, SERVING and SEPARATE. An incoming receive event with a NON request directly triggers the transition to the SEPARATE state.

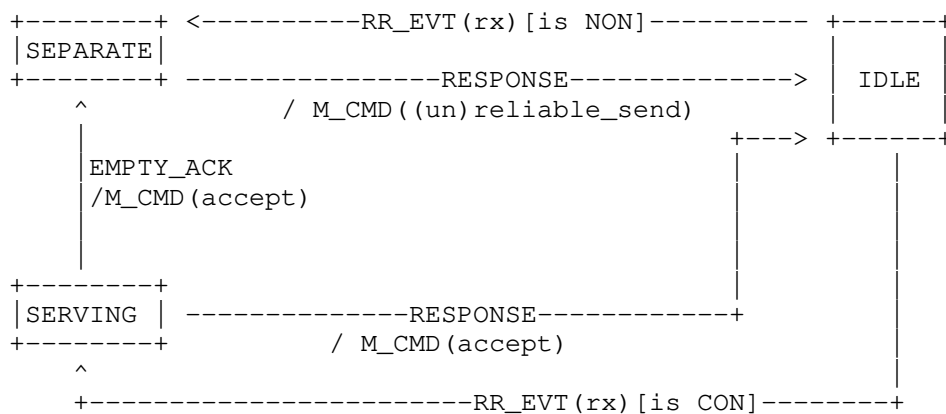
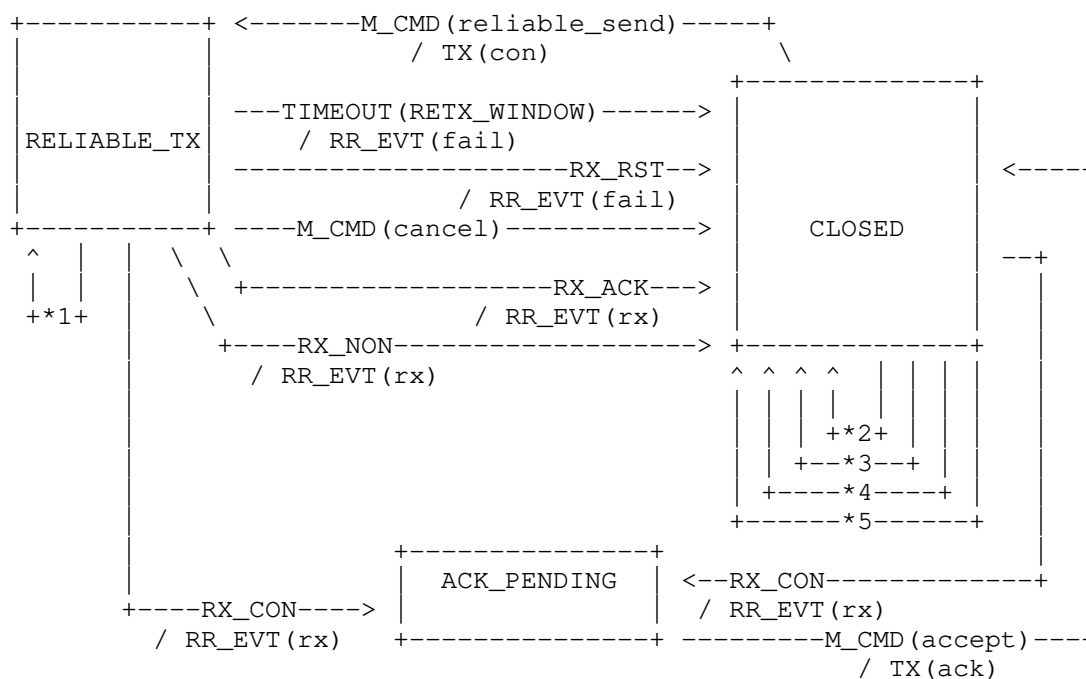


Figure 2: CoAP Server Request/Response Layer FSM

3.2. Message Layer

Figure 3 shows the different states of a CoAP endpoint per message exchange. Besides the linking action `RR_EVT()`, the message layer has a `TX` action to send a message. For sending and receiving NONs, the endpoint remains in its `CLOSED` state. When sending a CON, the endpoint remains in `RELIABLE_TX` and keeps retransmitting until the transmission times out, it receives a matching RST, the request/response layer cancels the transmission, or the endpoint receives an implicit acknowledgement through a matching NON or CON. Whenever the endpoint receives a CON, it transitions into the `ACK_PENDING` state, which can be left by sending the corresponding ACK.



```
*1: TIMEOUT(RET_X_TIMEOUT) / TX(con)
*2: M_CMD(unreliable_send) / TX(non)
*3: RX_NON / RR_EVT(rx)
*4: RX_RST / REMOVE_OBSERVER
*5: RX_ACK
```

Figure 3: CoAP Message Layer FSM

T.B.D.: (i) Rejecting messages (can be triggered at message and request/response layer). (ii) ACKs can also be triggered at both layers.

4. Observing

For each observer, the server needs to store at least address, port, token, and the last outgoing message ID. The latter is needed to match incoming RST messages and cancel the observe relationship.

It is favorable to have one retransmission buffer per observable resource that is shared among all observers. Each notification is serialized once into this buffer and only address, port, and token are changed when iterating over the observer list (note that different token lengths might require realignment). The advantage becomes clear for Confirmable notifications: Instead of one

retransmission buffer per observer, only one buffer and only individual retransmission counters and timers in the list entry need to be stored. When the notifications can be sent fast enough, even a single timer would suffice. Furthermore, per-resource buffers simplify the update with a new resource state during open deliveries.

5. Block-wise Transfers

Block-wise transfers have the main purpose of providing fragmentation at the application layer, where partial information can be processed. This is not possible at lower layers such as 6LoWPAN, as only assembled packets can be passed up the stack. While [I-D.ietf-core-block] also anticipates atomic handling of blocks, i.e., only fully received CoAP messages, this is not possible on Class 1 devices.

When receiving a block-wise transfer, each blocks is usually passed to a handler function that for instance performs stream processing or writes the blocks to external memory such as flash. Although there are no restrictions in [I-D.ietf-core-block], it is beneficial for Class 1 devices to only allow ordered transmission of blocks. Otherwise on-the-fly processing would not be possible.

When sending a block-wise transfer, Class 1 devices usually do not have sufficient memory to print the full message into a buffer, and slice and send it in a second step. When transferring the CoRE Link Format from /.well-known/core for instance, a generator function is required that generates slices of a large string with a specific offset length (a 'sonprintf()'). This functionality is required recurrently and should be included in a library.

6. Application Developer API

Bringing a Web transfer protocol to constrained environments does not only change the networking of the corresponding systems, but also the way they should be programmed. A CoAP implementation should provide a developer API similar to REST frameworks in traditional computing. A server should not be created around an event loop with several function calls, but rather by implementing handlers following the resource abstraction.

So far, the following types of RESTful resources were identified:

NORMAL A normal resource defined by a static Uri-Path that is associated with a resource handler function. Allowed methods could already be filtered by the implementation based on flags. This is the basis for all other resource types.

PARENT A parent resource manages several sub-resources by programmatically evaluating the Uri-Path, which may be longer than that of the parent resource. Defining a URI templates (see [RFC6570]) would be a convenient way to pre-parse arguments given in the Uri-Path.

PERIODIC A resource that has an additional handler function that is triggered periodically by the CoAP implementation with a resource-defined interval. It can be used to sample a sensor or perform similar periodic updates. Usually, a periodic resource is observable and sends the notifications in the periodic handler function. These periodic tasks are quite common for sensor nodes, thus it makes sense to provide this functionality in the CoAP implementation and avoid redundant code in every resource.

EVENT An event resource is similar to an periodic resource, only that the second handler is called by an irregular event such as a button.

7. References

7.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

7.2. Informative References

- [I-D.arkko-core-sleepy-sensors]
Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-01 (work in progress), July 2011.
- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-12 (work in progress), June 2013.
- [I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-05 (work in progress), July 2013.

Authors' Addresses

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
CH-8092 Zurich
Switzerland

Email: kovatsch@inf.ethz.ch

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Email: bergmann@tzi.org

Esko Dijk
Philips Research

Email: esko.dijk@philips.com

Xuan He
Hitachi (China) R&D Corp.
301, Tower C North, Raycom, 2 Kexuyuan Nanlu
Beijing, 100190
P.R.China

Email: xhe@hitachi.cn

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CORE WG
Internet-Draft
Intended status: Informational
Expires: April 14, 2014

A. Rahman
InterDigital Communications, LLC
October 11, 2013

Sleepy Devices: Do we need to Support them in CORE?
draft-rahman-core-sleepy-nodes-do-we-need-00

Abstract

This document summarizes the discussion in the CORE WG related to the question of whether support of sleepy devices is required for the CoAP protocol, CORE Link Format, CORE Resource Directory, etc. The only goal of this document is to trigger discussions in the CORE WG so that all relevant considerations for sleeping devices are taken into account.

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 <http://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 April 14, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Terminology and Conventions	2
2. Introduction	2
3. Drafts Related to Sleepy Nodes	2
4. WG Email List Poll for Sleepy Node Deliverable	3
5. Summary	4
6. Acknowledgements	4
7. IANA Considerations	4
8. Security Considerations	4
9. References	4
9.1. Normative References	4
9.2. Informative References	4
Author's Address	6

1. Terminology and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document assumes readers are familiar with the terms and concepts that are used in [I-D.ietf-core-coap] and [RFC6690].

2. Introduction

At IETF-87 (Berlin), it was suggested to review/summarize the CORE WG interest on the topic of Sleepy Node support. Specifically whether the WG feels that support of sleepy endpoints is required for the CoAP protocol, CORE Link Format, CORE Resource Directory, etc. Alternatively, whether the WG feels that Sleepy Node support can be completely done outside CORE such as in the lower Layer 2 (MAC) scheduling and/or in Layer 7 (application) logic.

3. Drafts Related to Sleepy Nodes

There have been multiple drafts in the CORE WG related to the subject of Sleepy Nodes including:

- o [I-D.rahman-core-sleepy-problem-statement] summarizes the overall problem space of Sleepy Nodes.
- o [I-D.cao-core-aol-req] defines requirements for Sleepy Nodes to behave as if they are "always on".
- o [I-D.dijk-core-sleepy-reqs] defines requirements for Sleepy Nodes based on home and building control use cases.
- o [I-D.rahman-core-sleeping] defines general requirements for Sleepy Nodes.
- o [I-D.bormann-core-roadmap] provides a classification and overview of CORE drafts (and features) including a section on Sleepy Nodes.
- o [I-D.arkko-core-sleepy-sensors] describes a sensor network implementation and shows how different communication models affect implementation complexity and energy consumption (including Sleepy Node support).
- o [I-D.giacomin-core-sleepy-option] defines a proxy that acts as a store-and-forward agent for a Sleepy Node.
- o [I-D.castellani-core-alive] defines a new CoAP message type which the Sleepy Node multicasts to all interested devices when it wakes up.
- o [I-D.fossati-core-publish-option] allows an endpoint to temporarily delegate authority of its resources (when it is sleeping) to a proxy server that is always on.
- o [I-D.fossati-core-monitor-option] extends the Observe functionality to handle the scenario when both the server and clients are Sleepy Nodes.
- o [I-D.dijk-core-sleepy-solutions] defines an architectural approach to support Sleepy Nodes.
- o [I-D.rahman-core-sleepy] defines new parameters that describe an endpoint's sleepy characteristics and stores them in the Resource Directory.
- o [I-D.vial-core-mirror-server] defines a special type of Resource Directory from which endpoints can fetch the resource regardless of the (sleep) state of the server.

4. WG Email List Poll for Sleepy Node Deliverable

A poll was taken on the WG Email list asking the following question: "Should we have a CORE deliverable for CoAP support of Sleepy Nodes?" [IETF87-Poll].

The results of the poll were as follows:

- o Votes FOR a new CORE Sleepy Node support deliverable: 9
- o Votes AGAINST a new CORE Sleepy Node support deliverable: 3

5. Summary

There have been over ten drafts related to the concept of CORE support of Sleepy Nodes. The WG Email list poll on the topic had a large majority of responders supporting creation of a CORE charter item for support of Sleepy Nodes. However there were some important and high profile dissenters that argued against such a charter item. Another point to consider is that during WG discussions, the CORE Mirror Server [I-D.vial-core-mirror-server] is sometimes referred to as the "existing" solution for CORE Sleepy Node support. However, this draft was never adopted as a WG draft.

6. Acknowledgements

Thanks to Carsten Bormann and Zach Shelby for valuable discussions and feedback on the topic of Sleepy Nodes.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

Not applicable.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2. Informative References

- [I-D.arkko-core-sleepy-sensors]
Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-01 (work in progress), July 2011.

- [I-D.bormann-core-roadmap]
Bormann, C., "CoRE Roadmap and Implementation Guide",
draft-bormann-core-roadmap-04 (work in progress), May
2013.
- [I-D.cao-core-aol-req]
Cao, Z., "Allways-online Requirement for Sleeping CoAP
Node", draft-cao-core-aol-req-00 (work in progress), July
2011.
- [I-D.castellani-core-alive]
Castellani, A. and S. Loreto, "CoAP Alive Message", draft-
castellani-core-alive-00 (work in progress), March 2012.
- [I-D.dijk-core-sleepy-reqs]
Dijk, E., "Sleepy Devices using CoAP - Requirements",
draft-dijk-core-sleepy-reqs-00 (work in progress), June
2013.
- [I-D.dijk-core-sleepy-solutions]
Dijk, E., "Sleepy Devices using CoAP - Possible
Solutions", draft-dijk-core-sleepy-solutions-01 (work in
progress), June 2013.
- [I-D.fossati-core-monitor-option]
Fossati, T., Giacomini, P., and S. Loreto, "Monitor Option
for CoAP", draft-fossati-core-monitor-option-00 (work in
progress), July 2012.
- [I-D.fossati-core-publish-option]
Fossati, T., Giacomini, P., and S. Loreto, "Publish Option
for CoAP", draft-fossati-core-publish-option-00 (work in
progress), July 2012.
- [I-D.giacomini-core-sleepy-option]
Fossati, T., Giacomini, P., Loreto, S., and M. Rossini,
"Sleepy Option for CoAP", draft-giacomini-core-sleepy-
option-00 (work in progress), February 2012.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained
Application Protocol (CoAP)", draft-ietf-core-coap-18
(work in progress), June 2013.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource
Directory", draft-ietf-core-resource-directory-00 (work in
progress), June 2013.

[I-D.rahman-core-sleeping]

Rahman, A., Zuniga, J., and G. Lu, "Sleeping and Multicast Considerations for CoAP", draft-rahman-core-sleeping-00 (work in progress), June 2010.

[I-D.rahman-core-sleepy-problem-statement]

Rahman, A., Fossati, T., Loreto, S., and M. Vial, "Sleepy Devices in CoAP - Problem Statement", draft-rahman-core-sleepy-problem-statement-01 (work in progress), October 2012.

[I-D.rahman-core-sleepy]

Rahman, A., "Enhanced Sleepy Node Support for CoAP", draft-rahman-core-sleepy-04 (work in progress), October 2013.

[I-D.vial-core-mirror-server]

Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-server-01 (work in progress), April 2013.

[IETF87-Poll]

Rahman, A., "Do we need a CORE charter item for CoAP support of Sleepy Nodes?", August 2013, <<http://www.ietf.org/mail-archive/web/core/current/msg04750.html>>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

Author's Address

Akbar Rahman
InterDigital Communications, LLC
Montreal, Quebec H3A 3G4
Canada

Phone: +1-514-585-0761

Email: akbar.rahman@interdigital.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: March 20, 2014

L. Seitz
SICS Swedish ICT AB
S. Gerdes
Universitaet Bremen TZI
G. Selander
Ericsson
September 16, 2013

Use cases for CoRE security
draft-seitz-core-sec-usecases-00

Abstract

This document presents use cases for security measures in scenarios involving constrained RESTful devices. Special focus is placed on access control and authentication. Where specific details are relevant, it is assumed that the devices use CoAP as communication protocol, however most conclusions apply generally.

A number of security requirements are derived from the use cases, which are intended as a guideline for developing a comprehensive authentication and authorization approach for this class of scenarios.

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 <http://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 March 20, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents
(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. Use Cases	4
2.1. Container monitoring	4
2.1.1. Bananas for Munich	4
2.1.2. Requirements	5
2.2. Home Automation	5
2.2.1. Remotely letting in a visitor	5
2.2.2. Requirements	6
2.3. Personal Health Monitoring	6
2.3.1. John and the heart rate monitor	7
2.3.2. Requirements	7
2.4. Building Automation	8
2.4.1. Fire Alarm	8
2.4.2. Requirements	9
2.5. Industrial Control Systems	9
2.5.1. Water treatment plant	10
2.5.2. Requirements	10
3. Requirements From The Use Cases	12
3.1. General Security Requirements	12
3.2. Authentication Requirements	13
3.3. Access Control Requirements	13
4. Security Considerations	15
5. Acknowledgments	16
6. IANA Considerations	17
7. Informative References	18
Authors' Addresses	19

1. Introduction

This document presents use cases in an attempt to analyze the security requirements especially on authentication and access control in an Internet of Things setting. This setting features constrained devices [I-D.ietf-lwig-terminology] communicating over the Internet. Some of these devices may have very low capacity in terms of memory and processing power, and may additionally be limited by the fact that they run on battery power.

These devices offer resources such as sensor data and actuators, which are accessed by clients, that may be users or other devices.

Where specific detail is necessary it is assumed that the devices communicate using the CoAP protocol [I-D.ietf-core-coap], although most conclusions are generic. Currently CoAP proposes to use DTLS [RFC6347] for authentication, and access control lists on the devices, that specify which clients may initiate a DTLS connection. One goal of this document is to point out use cases where this approach is not satisfactory.

1.1. Terminology

Resource Server (RS): The constrained device which hosts resources the Client wants to access.

Client (C): A device which wants to access a resource on the Resource Server. This could also be a constrained device.

Resource Owner (RO): The subject who owns the resource and controls its access permissions.

2. Use Cases

This section lists use cases involving constrained devices with security requirements. Each use case first presents a general description of the application area, then one or more specific use cases, and finally the resulting security requirements.

2.1. Container monitoring

The ability of sensors to communicate environmental data wirelessly opens up new application areas. The use of such sensor systems makes it possible to transmit specific characteristics such as temperature, humidity and gas content during transportation and storage of goods.

The proper handling of the sensors in this scenario is not easy to accomplish. They have to be associated to the appropriate pallet of the respective container. Moreover, the goods and the corresponding sensors belong to specific customers.

During the shipment to their destination the goods often pass stops where they are transloaded to other means of transportation, e.g. from ship transport to road transport.

2.1.1. Bananas for Munich

A Munich supermarket chain buys bananas from a Costa Rican fruit vendor. It instructs a transport company to deliver the goods via ship to Rotterdam where they are picked up by their own company trucks.

The supermarket's quality management wants to assure the quality of their products and thus uses the fruit vendor's service of equipping the bananas with sensors. The state of the goods is monitored consistently during the shipment and abnormal sensor values are recorded. Additionally, the sensor values are used to control the climate within the cargo containers.

The personnel of the transport company and the supermarket's delivery service has to be able to locate the proper goods and match them to the corresponding customer. The state of the cargo must not be disclosed to them, however.

When the goods arrive at the supermarket in Munich, their state is checked.

If no anomalies occurred during the transport, the bananas are admitted for sale.

2.1.2. Requirements

- o U1.1 The supermarket chain must be able to allow the transport company and the delivery service to access the position data on the monitoring devices. Other state information must not be accessible.
- o U1.2 The climate regulation system in the containers must be able to access the monitoring devices' state information to regulate the climate accordingly.
- o U1.3 The integrity and availability of the sensor data must be assured for proper operation of climate control.
- o U1.4 The supermarket chain must be able to allow the supermarket's quality management to access the recorded state information on the monitoring devices.
- o U1.5 The supermarket chain will not want other companies to be able to read sensor information so the confidentiality of the monitoring devices' state information must be assured.

2.2. Home Automation

Automation of the home, housework or household activity is propagated as a future market for the Internet of Things. A home automation system integrates electrical devices in a house with each other, such as heating, ventilation, lighting, home entertainment and home security.

Such a system needs to accommodate a number of regular users (inhabitants, close friends, cleaning personnel) as well as a heterogeneous group of dynamically varying users (visitors, repairmen, delivery men).

The security required by the systems integrated in a automated home varies, however it is clear that the security system controlling e.g. the doors, alarms, and other critical systems needs to be at least as secure as for a comparable unautomated home.

As the users are not typically trained in security (or even computer use), the configuration must use secure default settings, and the interface must be well adapted to novice users.

2.2.1. Remotely letting in a visitor

Jane is the owner of an automated home, that allows her to remotely control all electrical devices through a web interface or mobile

application.

Jane has invited over her acquaintance Jeffrey for dinner, but is stuck in traffic and can not arrive in time, while Jeffrey who uses the subway arrives punctually. Jane calls Jeffrey and offers him to let him in remotely, so he can make himself comfortable and use Jane's home entertainment system.

Jeffrey downloads an application that lets him communicate with Jane's home, and Jane remotely instructs the door to open for him, and the alarm to shut down. She gives Jeffrey access to lighting and HVAC and also limited access to the home entertainment system, allowing Jeffrey to all services except those that are pay-per-use or those that Jane has marked as private.

2.2.2. Requirements

- o U2.1 Jane needs to be able to spontaneously provision authentication means to Jeffrey
- o U2.2 Jane must be able to spontaneously change the access control policies
- o U2.3 Jane needs to be able to apply different rights for different devices and users
- o U2.4 Jane must be able to apply local conditions (presence, time) to authorizations, and the device (e.g. the door) needs to be able to verify these conditions
- o U2.5 The different devices in Jane's home need to be able to communicate with each other and with different control devices
- o U2.6 The configuration of Jane's home needs to be secure by default
- o U2.7 It must be easy for Jane to edit the access control policies for her home

2.3. Personal Health Monitoring

The use of wearable health monitoring technology is expected to grow strongly, as a multitude of novel devices are developed and marketed. These devices are typically battery driven, and located physically on the user. They monitor some bodily function, such as e.g. temperature, blood pressure, or pulse. They are connected to the Internet, either through an intermediary base-station or directly, using wireless technologies. Through this connection they report the

monitored data to some entity, which may either be the user herself, or some medical personnel in charge of the user.

Medical data has always been considered as very sensitive, and therefore requires good protection against unauthorized disclosure. A frequent, conflicting requirement is the capability for medical personnel to gain emergency access, even if no specific access rights exist. As a result, the importance of secure audit logs increases in such scenarios.

Since the users are not typically trained in security (or even computer use), the configuration must use secure default settings, and the interface must be well adapted to novice users. Also the system must require very little maintenance, so e.g. frequent changes of battery are unacceptable.

2.3.1. John and the heart rate monitor

John has a heart condition, that can result in sudden cardiac arrests. He therefore uses a device called HeartGuard that monitors his heart rate and his position. In case of a cardiac arrest it automatically sends an alarm to an emergency service, transmitting John's current location. The device also functions as a implanted cardioverter defibrillator, i.e. it can deliver a shock in order to try and normalize John's heart rate.

John can configure additional persons that get notified in an emergency, for example his daughter Jill. Furthermore the device stores data on John's heart rate, which can later be accessed by a physician to assess the condition of John's heart.

However John is a rather private person, and is worried that Jill might use HeartGuard to monitor his location while there is no emergency. Furthermore he doesn't want his health insurance to get access to the HeartGuard data, since they might refuse to renew his insurance if they decided he was too big a risk for them.

2.3.2. Requirements

- o U3.1 John must be able to selectively allow different persons or groups to access the position data on condition that there is an emergency.
- o U3.2 John must be able to selectively allow different persons or groups to access the heart rate data.
- o U3.3 John must be able to block access to specific persons or groups, if he mistrusts them.

- o U3.4 The device must operate in a way that does not require frequent battery changes
- o U3.5 The device must ensure that both incoming and outgoing communication is confidentiality and integrity protected
- o U3.6 The device must have secure settings by default
- o U3.7 The device's security settings must be easy to configure

2.4. Building Automation

Buildings for commercial use such as shopping malls or office buildings nowadays are equipped increasingly with semi-automatic components to enhance the overall living quality and save energy where possible. This includes for example heating, ventilation and air condition (HVAC) as well as illumination and fire alarm systems.

These buildings are often used by more than one company who share some parts of the building while other areas are used by each of them exclusively. Accordingly, a company must be able to control the light and HVAC system of its own part of the building and must not have access to rooms that belong to other companies.

Some parts of the building automation system such as entrance illumination and fire alarm systems are controlled either by all parties together or by a service company.

The building automation system has to provide for a security mechanism which allows for authentication and fine-grained authorization.

2.4.1. Fire Alarm

The Companies A and B share an office building which is equipped with a fire alarm system. It is triggered by several smoke detectors which are spread out across the building.

It is a really hot day and James who works for company A turns on the air condition in his office. Lucy who works for company B wants to make tea using an electric kettle. After she turned it on she goes outside to talk to a colleague until the water is boiling. Unfortunately, her kettle has a malfunction which causes overheating and results in a smoldering fire of the kettle's plastic case.

Due to the smoke coming from the kettle the fire alarm is triggered. Alarm sirens throughout the building are notified and alert the staff of both companies. Additionally, the ventilation system of the whole

building is closed off to prevent the smoke from spreading and to withdraw oxygen from the fire. The smoke cannot get into James' office although he turned on his air condition because the fire alarm overrides the manual setting.

The fire department is notified of the fire automatically and arrives within a short time. After inspecting the damage and extinguishing the smoldering fire a fire fighter resets the fire alarm because only the fire department is authorized to do that.

2.4.2. Requirements

- o U4.1 The building control devices of company A must be able to interoperate with those of company B. The devices might be produced by different vendors and might be operated by different service providers.
- o U4.2 Only the smoke detectors must be able to trigger an alarm.
- o U4.3 The availability and integrity of the smoke detector's alarm messages have to be assured.
- o U4.4 Only the fire department must be able to reset the fire alarm.
- o U4.5 James must be able to control the air conditioning in his office.
- o U4.6 The emergency system must be able to automatically close off the ventilation system.
- o U4.7 During fire alarm, the personnel must not be allowed to regulate the climate control.
- o U4.8 No unauthorized device must be able to access building control devices.
- o U4.9 Since replacing devices in the building is very work intensive and thus expensive (there are many devices, and some are in places that are hard to access), the devices and their batteries should function for a very long time without maintenance.

2.5. Industrial Control Systems

Industrial control systems (ICS) and especially supervisory control and data acquisition systems (SCADA) use a multitude of sensors and actuators in order to monitor and control industrial processes in the

physical world. Example processes include manufacturing, power generation, and refining of raw materials.

Since the advent of the Stuxnet worm it has become obvious to the general public how vulnerable this kind of systems are, especially when connected to the Internet. These severity of these vulnerabilities are exacerbated by the fact that many ICS are used to control critical public infrastructure, such as power, water treatment of traffic control. Nevertheless the economical advantages of connecting such systems to the Internet can be significant if appropriate security measures are put in place.

2.5.1. Water treatment plant

The communal water treatment plant of a mid-sized city is controlled by a networked ICS. Spread across the city are numerous nodes, sensors (e.g. pollution meters, pressure indicators) and actuators (e.g. valves, pumps) communicating via a wireless network. Since the range of the network is limited, many nodes communicate through intermediary proxies that relay communications to the administration clients of the ICS.

Jenny is a technician whose job it is to monitor the plant and take appropriate measure, if abnormal conditions are detected (e.g. if water pollution is detected, or the pressure in a pump reaches critical levels).

When Jenny is on holiday or sick-leave, the service company sends a replacement worker from a pool of available, qualified persons.

Joshuah is a young, computer savvy kid with too much time at his hands. He spends time wardriving and stumbles upon the wireless network, used by the plant's sensors and actuators. Joshuah tries to interact with the devices on this network and manages to stall a valve controlling water flow to a part of the city. Jenny quickly discovers the intrusion and is able to take appropriate measures to prevent damage to the value and quickly restore normal service conditions.

2.5.2. Requirements

- o U5.1 The Integrity of the messages sent between the nodes in the ICS must be protected.
- o U5.2 The nodes must be resilient to denial of service attacks.
- o U5.3 The security measures must cope with the presence of intermediary proxies between the Resource Server and the Client.

- o U5.4 Since most of the processing capacity of the nodes and the network load capacity must go towards production tasks, the security measures must use minimal resources, both on the network and on the nodes.
- o U5.5 Since replacement workers can spontaneously jump in for Jenny, the system needs to be able to handle authorization updates without re-provisioning each node individually.
- o U5.6 After a replacement worker has finished taking care of the system, the corresponding authorization and authentication means need to be revoked remotely.

3. Requirements From The Use Cases

This section lists requirements derived from the use cases above. Note that not every single requirement applies to every Resource Server, however protocols should allow for all of these requirements to be fulfilled.

3.1. General Security Requirements

The following requirements refer to general security measures, not directly linked to authentication and authorization which are listed in detail in the next sections.

- o Integrity, confidentiality and replay protection of the message exchanges between the Resource Server, the Client and other involved parties (U1.3, U1.5, U3.5, U4.3, U5.1).

This may be achieved by either establishing a secure channel (such as e.g. DTLS [RFC6347]) or object security applied to the payloads (e.g. JWS/JWE [I-D.ietf-jose-json-web-signature], [I-D.ietf-jose-json-web-encryption]).

- o Protect the Resource Server against denial of service (U3.4, U4.3, U5.2) - Minimize the number of protocol steps that an attacker can induce a Resource Server to perform without proper authentication and authorization.

- * Note well that for constrained devices this includes attacks that aim to drain the battery of the target.

- o Security measures must work when traffic from the Client to the Resource Server goes through intermediary nodes (U5.3).

Rationale: In many deployments, there will be gateways, proxies, firewalls etc. between a Client and a Resource Server. Security measures should therefore not require the Client to be directly connected to the Resource Server.

- o Use minimal resources for security measures (U3.4, U4.9, U5.4)

- * Minimize battery usage

- + Minimize message exchanges only for security

- + Minimize the size of authorization and authentication data that is transmitted

- + Minimize the size of required software libraries
- + Minimize memory and stack usage on the devices
- o Enable security by default (U2.6, U3.6)

Rationale: Many attacks exploit insecure default settings, and experience shows that default settings are rarely changed by the end users. Therefore the protocols for constrained devices should require secure modes of use by default.

- o Interoperability (U1.1, U1.2, U2.5, U4.1)

Rationale: Resource Owners may interact with Clients from various manufacturers and vice-versa. In order to function correctly the security mechanisms need to work together. This is best achieved by standardization

- o Usability (U2.7, U3.7)
 - * Keep response times reasonable
 - * Make the security measures transparent for human users where possible
 - * Make the administration of security as simple as possible

3.2. Authentication Requirements

- o Enable mutual authentication between the Client and the Resource Server (U1.1, U1.2, U2.1, U4.4, U4.5, U4.6)
- o Provision authentication means to Clients and Resource Servers (U1.1, U1.2, U2.1, U4.4, U4.5, U4.6, U5.5)
 - * Optionally allow for remote provisioning.
- o Enable remote revocation of authentication means (U3.3, U4.9, U5.6)

3.3. Access Control Requirements

- o Enforce the access control policies of the Resource Owner (U1.1, U1.2, U1.4, U3.1, U3.2, U4.2, U4.4, U4.5, U4.6, U4.8) – Provision access control policies set by the Resource Owner to the Policy Decision Point (which may be on the Resource Server or on another trusted entity) [RFC2904]. – Apply the access control policies to incoming requests (this may be done by the Resource Server or by

another trusted entity).

- o Do not send additional messages just for access control (U3.4, U5.4) Rationale: Sending and receiving is a much bigger battery drainer, compared to processing on the device.
- o Apply different rights for different requesting entities (U1.1, U1.2, U2.3, U4.2, U4.4, U4.5, U4.6) Rationale: In some cases different types of users require different access rights, as opposed to all-or-nothing access control.
- o Allow for fine-grained access control (U1.1, U1.2, U4.2, U4.4, U4.5, U4.6) Resource Servers can host several resources, and a resource (e.g. an actuator) can have different settings. In some cases access rights need to be different at this level of granularity.
- o Apply local conditions (U2.4, U3.1, U4.7) Access may depend on local conditions e.g. access to health data in an emergency. The Policy Decision Point must be able to take such conditions into account.
- o Enable policy updates without re-provisioning the device (U2.2, U4.9, U5.5, U5.6) Rationale: Clients can change rapidly and re-provisioning might be prohibitively expensive.

4. Security Considerations

This document lists security requirements for constrained devices, motivated by specific use cases. Therefore the whole document deals with security considerations.

5. Acknowledgments

The authors would like to thank Olaf Bergmann, and Mohit Sethi for contributing to the discussion and giving helpful comments. Also, thanks to Markus Becker for his input on the container monitoring use case.

6. IANA Considerations

This document has no IANA actions.

7. Informative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-jose-json-web-encryption]
Jones, M., Rescorla, E., and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-16 (work in progress), September 2013.
- [I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-16 (work in progress), September 2013.
- [I-D.ietf-lwig-terminology]
Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-05 (work in progress), July 2013.
- [RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, August 2000.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

Authors' Addresses

Ludwig Seitz
SICS Swedish ICT AB
Scheelevaegen 17
Lund 22370
Sweden

Email: ludwig@sics.se

Stefanie Gerdes
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63906
Email: gerdes@tzi.org

Goeran Selander
Ericsson
Farogatan 6
Kista 16480
Sweden

Email: goran.selander@ericsson.com

CoRE Working Group
Internet-Draft
Intended Status: Standards Track
Expires: April 24, 2014

L. Seitz
SICS Swedish ICT
G. Selander
Ericsson

October 21, 2013

Additional Security Modes for CoAP
draft-seitz-core-security-modes-00

Abstract

The CoAP draft defines how to use DTLS as security mechanism. In order to establish which nodes are trusted to initiate a DTLS session with a device, the following security modes are defined: NoSec, PreSharedKey, RawPublicKey, and Certificate. These modes require either to provision a list of keys of trusted clients, or to handle heavyweight certificates. This memo proposes two intermediate security modes involving a trusted third party that are very similar to PreSharedKey and RawPublicKey respectively, but which do not require out-of-band provisioning of client keys to the device.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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".

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1 Terminology	4
2. DerivedKey Mode	4
2.1 Generating The Nonce	5
2.2 Calculating The Derived Key	5
2.3 Generating PSK_IDENTITY	6
2.4 Key Expiration, Anti-replay, And Revocation	6
3. AuthorizedPublicKey Mode	8
3.1 Structure of the Authorization Certificate Extension	9
3.2 Client and Server Handshake Behavior	10
3.3 Payload Verification Procedure	10
3.4 Key Expiration, Anti-replay, And Revocation	11
4. Access Control Lists	11
5. Security Considerations	11
6. IANA Considerations	12
7. Acknowledgements	12
8. References	13
8.1 Normative References	13
8.2 Informative References	13
Authors' Addresses	14

1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a light-weight web transfer protocol suitable for applications in embedded devices used in services such as smart energy, smart home, building automation, remote patient monitoring etc. Due to the nature of these use cases including critical, unattended infrastructure and the personal sphere, security and privacy are critical components.

CoAP message exchanges can be protected with different security protocols. The CoAP specification defines a DTLS [RFC6347] binding for CoAP, which provides communication security including authentication, encryption, integrity, and replay protection. In order to bootstrap trust relations, the CoAP specification defines four security modes that are the result of different provisioning procedures (see section 9 of [I-D.ietf-core-coap]):

- o NoSec
- o PreSharedKey (PSK)
- o RawPublicKey (RPK)
- o Certificate

The NoSec alternative assumes security measures at another protocol layer and provides no security at all. PSK and RPK modes rely on a pre-provisioned list of keys that the device can initiate a DTLS session with. Certificate mode requires provisioning of certain root trust anchor public keys (equivalent to CA certificates) that can be used to validate previously unknown X.509 certificates, before using them to establish a DTLS session.

Given a setting where security is required, and where at least some devices are too resource constrained to handle X.509 certificates, devices would have to use either the PSK or the RPK mode. If the set of nodes that a device would communicate with varies dynamically (e.g. a pay-per-use scenario) this would in turn require constant re-provisioning of lists of trusted clients to the individual devices.

Such an approach will obviously not scale well and make consistent management of security policies over a set of devices very difficult.

Therefore we propose two additional security modes that take advantage of the low resource consumption of the PSK and the RPK modes, but also allow to manage dynamic trust relations without having to re-provision the individual nodes. The basic idea is to provision a symmetric key of a trust anchor to the devices. A node wishing to connect to the device can obtain either a derived secret key, or a Message Authentication Code (MAC) of its public key from one of the trust anchors, and the device can verify that this derived

secret key, or MAC is generated by a trust anchor. The derived key or public key is then used by the device as in PSK or RPK mode, respectively.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Certain security-related terms are to be understood in the sense defined in [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "encryption", "data integrity", "message authentication code", and "verify".

Terminology for constrained environments is defined in [I-D.ietf-lwig-terminology] e.g. "constrained device".

Furthermore this memo refers to the following entities:

- o The constrained device providing resources is called the Resource Server (RS)
- o The node connecting to the Resource Server in order to request some resource is called Client (C)
- o The entity having an a-priori trust relation with RS is called the Trust Anchor (TA)

2. DerivedKey Mode

This mode addresses similar use cases as the PSK mode, but without the requirement for out-of-band provisioning of shared keys between C and RS. Instead each resource server is configured with secret, symmetric keys shared with its trust anchors. For simplicity of explanation we assume here, that each RS only has a single TA, and that they share the key K_{RS-TA} . A client wishing to establish a connection to a RS needs to obtain a symmetric key K_{RS-C} and a nonce from the TA, where K_{RS-C} is derived from K_{RS-TA} and that nonce. C transmits the nonce in the `psk_identity` field of the `ClientKeyExchange` message of the DTLS protocol. The RS then derives K_{RS-C} from the nonce and K_{RS-TA} , and then both proceed using K_{RS-C} as a pre-shared key [RFC4279]. Figure 1 illustrates this procedure.

Client

Trust Anchor

Resource Server

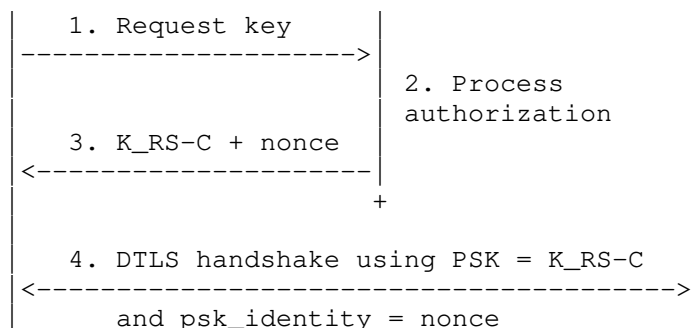


Figure 1: The message flow for DerivedKey mode

How C authenticates with the TA, and how the TA authorizes the request for a key K_{RS-C} is out of scope for this memo.

2.1 Generating The Nonce

Upon request, the trust anchor verifies if C is authorized to connect to the resource server. How this is done is out of scope for this memo. If the verification succeeds, the TA generates the nonce as follows:

```
nonce = 'DK.' + TA_id + '.' C_id + '.' + sequence_number
```

where '+' indicates concatenation,

'TA_id' is an identifier that the RS can use to select the correct K_{RS-TA} ,

'C_id' is an identifier of C, and

'sequence_number' is a sequence number maintained by the RS and the TA.

The TA then generates the shared key K_{RS-C} as described in section 2.2 and transfers the nonce and K_{RS-C} to C via a secure channel.

2.2 Calculating The Derived Key

K_{RS-C} is derived from K_{RS-TA} by the trust anchor and the resource server through a data expansion step, as defined in [RFC5246]:

```
P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) +
                      HMAC_hash(secret, A(2) + seed) +
                      HMAC_hash(secret, A(3) + seed) + ...
```

where '+' indicates concatenation and A() is defined as:

```
A(0) = seed
A(i) = HMAC_hash(secret, A(i-1))
```

In the present case:

- o hash is SHA-256,
- o 'secret' is the shared key K_RS-TA, and
- o 'seed' is the nonce.

The nonce, associated to the connection request, is generated by the trust anchor (see 2.1). A nonce SHALL NOT be reused with the same shared key K_RS-TA.

With one iteration of the P_SHA256, the output data D of 256 bits can be used to define K_RS-C either as one 256 bit key, or as one 128 bit key using the first 128 bits of D and discarding the rest.

2.3 Generating PSK_IDENTITY

The nonce is used as the psk_identity field of the DTLS ClientKeyExchange message. Upon receiving a psk_identity in the ClientKeyExchange message, an RS can determine by the 'DK' prefix that C wants to use the DerivedKey security mode, and select the corresponding key K_RS-TA by using the nonce in order to calculate the key K_RS-C as specified in 2.2.

2.4 Key Expiration, Anti-replay, And Revocation

The key K_RS-C enables the client to open a DTLS connection to the resource server, but in many cases one does not want this key to be valid forever. Furthermore an attacker can reuse a stolen key to gain access to the RS. Therefore the sequence_number part of the nonce can be used to expire the key K_RS-C (i.e. make it invalid for setting up new DTLS sessions) and protect against reuse of a {key, nonce} pair in a DTLS handshake.

The sequence number is a 32-bit number that is specific to a TA and an RS.

The TA keeps a list of sequence numbers per RS it is responsible for. A RS's sequence number is incremented by 1 for each new shared key K_RS-C generated for this RS.

For each TA an RS has (typically only one), it keeps a window of most recently verified sequence numbers. Sequence number verification SHOULD be performed using the following sliding window procedure,

borrowed from Section 3.4.3 of [RFC2402] (see also [RFC6347] section 4.1.2.6).

The sequence number MUST be initialized to zero when an association between a TA and an RS is established. For each received DTLS handshake using the DerivedKey Mode, the RS MUST verify that the nonce contains a fresh sequence number. This SHOULD be the first check applied to a nonce after it has been received in the ClientKeyExchange message, to speed rejection of duplicate or old records.

Freshness is checked through the use of a sliding receive window. (How the window is implemented is a local matter, but the following text describes the functionality that the implementation must exhibit.) A minimum window size of 32 MUST be supported, but a window size of 64 is preferred and SHOULD be employed as the default. Another window size (larger than the minimum) MAY be chosen by the RS.

The "right" edge of the window represents the highest validated Sequence Number value received on this RS. DTLS handshakes, using this security mode, that contain Sequence Numbers lower than the "left" edge of the window are rejected. Handshakes falling within the window are checked against a list of received handshakes with sequence numbers within the window. An efficient means for performing this check, based on the use of a bit mask, is described in Appendix C of [RFC2401].

If the sequence number falls within the window and is new, or if the sequence number is to the right of the window the RS proceeds to generate the shared key K_{RS-C} . If the handshake succeeds the RS updates the window.

On some occasions one may want to explicitly revoke a key K_{RS-C} before its expiration. In these cases the trust anchor has to send a message to the RS specifying the sequence number of the key K_{RS-C} it wants to revoke. The RS can then update the receive window to mark this key as used.

If a server is in use for a long period of time and able to process DTLS handshakes rapidly, the sequence number range may get exhausted within the lifetime of the server. In that case a new shared key K_{RS-TA} must be provisioned to the server and the TA, and the sequence number counters must be reset.

Note: If we make the very optimistic assumption that a DTLS handshake takes very roughly 1 second for a constrained device, a 32-bit sequence number can last roughly 136 years, before it needs to be

reset ($60 \times 60 \times 24 \times 365 = \text{max } 31,536,000$ handshakes per year,
 $2^{32}/31,536,000 > 136$).

3. AuthorizedPublicKey Mode

This security mode addresses similar use cases as the RPK mode, but without the need for out-of-band validation of public keys. As in the DerivedKey mode, we assume that the resource servers are configured with a symmetric key K_{RS-TA} for each of their trust anchors. In order to run this mode, the client needs to get its public key authorized for DTLS with the RS by one of the TA. The TA does this by creating an authorization certificate protected by a message authentication code (MAC) using the key K_{RS-TA} . The TA also provides C with the public key of RS for use in DTLS. The client then performs the DTLS handshake in RPK mode, but replaces the RawPublicKey ClientCertificate with the authorization certificate. The RS verifies the certificate, and if it is valid, proceeds with the DTLS handshake as if the client public key had been provisioned out of band. Furthermore the RS sends an empty certificate_list in the ServerCertificate message, since the key has already been provided to C by the TA.

The authorization certificate is essentially the RawPublicKey certificate of [I-D.ietf-tls-oob-pubkey] with an additional MAC. As with the RPK mode, this security mode benefits from a significantly smaller size of the client's Certificate message in the DTLS handshake. The verification of a MAC is also less resource consuming than verifying a digital signature. A considerable reduction in message size compared with the RPK mode, is that the RS does not have to send any certificate.

Figure 2 illustrates the message flow of this mode.

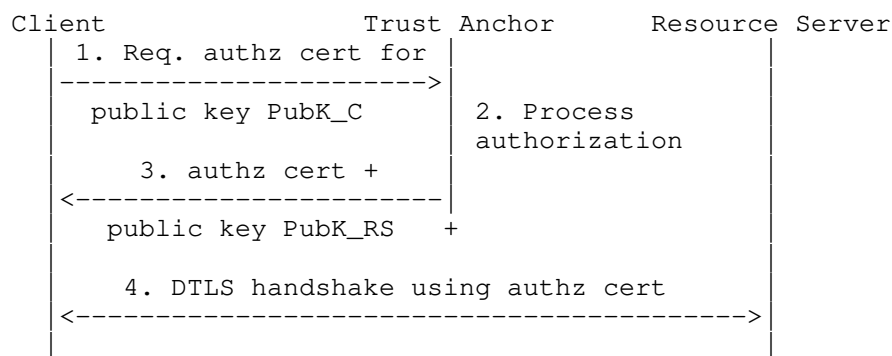


Figure 2: The message flow for AuthorizedPublicKey mode

How C authenticates with the TA, and how the TA authorizes the request for an authorization certificate out of scope for this memo.

3.1 Structure of the Authorization Certificate Extension

This section outlines the changes to the DTLS handshake message contents for the AuthorizedPublicKey mode. The procedure is analogous to the one in [I-D.ietf-tls-oob-pubkey], using the new certificate_type 'AuthzCert' and the new structure 'MacCert'.

```
struct {
  select(certificate_type) {
    // certificate type defined in this document
    case AuthzCert:
      MacCert certificate;

    // certificate type defined in [I-D.ietf-tls-oob-pubkey]
    case RawPublicKey:
      opaque ASN.1_subjectPublicKeyInfo<1..2^24-1>;

    // X.509 certificate defined in RFC 5246
    case X.509:
      ASN.1Cert certificate_list<0..2^24-1>;

    // Additional certificate type based on TLS
    // Certificate Type Registry
  };
} Certificate;
```

The MacCert structure is defined as follows:

```
struct {
  opaque ASN.1_subjectPublicKeyInfo<1..2^24-1>
  opaque trust_anchor_id;
  uint32 sequence_number;
  MACAlgorithm mac_algorithm;
  uint8 mac_length;
  opaque MAC[mac_length];
} MacCert;
```

Where ASN.1_subjectPublicKeyInfo is defined in section 3 of [I-D.ietf-tls-oob-pubkey], and the MACAlgorithm type is defined in [RFC5246]. The 'mac_algorithm' parameter specifies a function MAC = M(key, message), where K_RS-TA is used as the key, and the certificate with an empty MAC value as the message.

Note that the size of the MacCert structure is only marginally larger than the RawPublicKey certificate used in RPK mode.

The extended `client_hello` and extended `server_hello` defined in section 3 of [I-D.ietf-tls-oob-pubkey] are also used here, with the new `certificate_type` 'AuthzCert'.

3.2 Client and Server Handshake Behavior

Section 4 of [I-D.ietf-tls-oob-pubkey] shows the use of the client and server certificate types in TLS. The AuthorizedPublicKey mode uses a variant of the handshake exemplified in section 5.3 of [I-D.ietf-tls-oob-pubkey] as illustrated by figure 4.

```

client_hello,
client_certificate_type=(AuthzCert) //(1)
->
<- server_hello,
    server_certificate_type=(X.509) //(2)
    certificate, //(3)
    client_certificate_type=(AuthzCert) //(4)
    certificate_request, //(5)
    server_key_exchange,
    server_hello_done
certificate, //(6)
client_key_exchange,
change_cipher_spec,
finished
->
<- change_cipher_spec,
    finished

```

Application Data <-----> Application Data

Figure 4: Example of a DTLS handshake with Authorization
Certificate provided by the Client

This handshake starts with the client indicating its ability to use AuthorizedPublicKey mode (1). Since the client has already received the server's public key from the TA, the server sends an empty `certificate_list` in the certificate message (3), using the indication for X.509 certificates in (2). This indication is only used, because it allows to send an empty `certificate_list`. For client authentication the server indicates in (4) that it selected the AuthorizedPublicKey mode and requests a certificate from the client in (5). The client provides a MacCert structure (6) after receiving and processing the server hello message.

3.3 Payload Verification Procedure

After negotiating `client_certificate_type="AuthzCert"` in the

ClientHello/ServerHello steps of the DTLS protocol, and receiving the ClientCertificate message, the RS proceeds to verify the C's public key using the following steps:

- o Check if the trust_anchor_id identifies a trust anchor
- o Check if the sequence_number is valid
- o Check that the ASN.1_subjectPublicKeyInfo contains a valid SubjectPublicKeyInfo structure
- o Check the mac with the shared key K_RS-TA.

If any of these checks fail, the DTLS handshake is aborted and the RS MUST send a bad_certificate alert.

3.4 Key Expiration, Anti-replay, And Revocation

The rationale and procedures for handling sequence numbers are the same as described in section 2.4.

4. Access Control Lists

The CoAP specification uses Access Control Lists to keep track of pre-shared symmetric keys, raw public keys, and root trust anchors for X.509 certificates, used in the corresponding security modes (see section 9 and especially 9.1.3.2.1 of [I-D.ietf-core-coap]). An implementation supporting one or both of the security modes specified above MUST be extended to support storing lists of identifiers and secret keys of the trust anchors.

5. Security Considerations

All security consideration from [RFC6347] and [RFC4279] also apply to this approach. Furthermore the trust anchors used for authorizing the use of keys in the two proposed security modes are valuable targets for attacks since they potentially allow access to many devices. They should be protected accordingly.

The nonce used to generate the shared key for the DK mode is static except for the sequence number, an attacker could exploit this for a dictionary attack. If such an attack is considered feasible, an additional random seed should be added to the nonce to increase the variable part. The client identifier and other information in the nonce cannot be trusted until the client is authenticated using the key derived from the nonce.

The sequence number mechanism for expiration can potentially lead to keys being valid for a longer time than expected. This will be the

case if the number of requests for a device drops significantly, and it therefore takes longer to fill the sliding window. Trust anchors can monitor this and explicitly revoke keys if the frequency of requests drops significantly. It is also possible to use timers in the device to implement complementing expiry mechanisms.

An attacker can induce a server to perform the DTLS handshake up to Flight 4, without having any legitimate key material from a trust anchor. This could be used for denial of service attacks against the server. However these problems are also present with any of the standard CoAP security modes and respective DTLS handshakes.

6. IANA Considerations

IANA is asked to register a new value in the "TLS Certificate Types" registry of Transport Layer Security (TLS) Extensions [TLS-Certificate-Types-Registry], as follows:

Value: 3	Description: Authorized Public Key Certificate
(AuthzCert)	Reference: [[THIS RFC]]

7. Acknowledgements

The authors would like to thank Stefanie Gerdes, Mats Naeslund, and John Mattsson for contributions and helpful comments.

8. References

8.1 Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-tls-oob-pubkey]
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and
T. Kivinen, "Out-of-Band Public Key Validation for
Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-10 (work in progress), October 2013.
- [TLS-Certificate-Types-Registry]
"TLS Certificate Types Registry", February 2013,
<<http://www.iana.org/assignments/tls-extensiontype-values#tls-extensiontype-values-2>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4279] Eronen, P., Ed., and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

8.2 Informative References

- [I-D.ietf-lwig-terminology]
Bormann, C., Ersue, M., and Keranen, A., "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-05 (work in progress), July 2013.
- [RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998. Obsoleted by RFC4301.
- [RFC2402] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998. Obsoleted by RFC4302, RFC4305.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI
36, RFC 4949, August 2007.

Authors' Addresses

Ludwig Seitz
SICS Swedish ICT AB
Scheelevagen 17
22370 Lund
SWEDEN
EMail: ludwig@sics.se

Goeran Selander
Ericsson
Farogatan 6
16480 Kista
SWEDEN
EMail: goran.selander@ericsson.com

CoRE Working Group
Internet-Draft
Intended Status: Informational
Expires: April 24, 2014

G. Selander
M. Sethi
Ericsson
L. Seitz
SICS Swedish ICT
October 21, 2013

Access Control Framework for Constrained Environments
draft-selander-core-access-control-01

Abstract

The Constrained Application Protocol (CoAP) is a light-weight web transfer protocol designed to be used in constrained nodes and constrained networks. Communication security support for CoAP, including authentication, encryption, integrity protection, is specified by means of a DTLS binding for CoAP, but authorization and access control are not described in detail.

This document describes a generic and dynamic access control framework suitable for constrained environments e.g. using CoAP. The framework builds on standards and well known paradigms for access control, externalizing authorization decision making to unconstrained nodes while performing authorization decision enforcement and verification of local conditions in constrained devices.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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".

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at

<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1 Terminology	5
2. Scope and Requirements	5
2.1 Resource Authorization and Protocol Authorization	5
2.2 Requirements	6
3. Outline of Access Control Framework	7
3.1 Rationale	7
3.2 Roles	8
3.3 Message flow	9
4. Access Tokens	10
4.1 Requirements	10
4.2 Access Token Protection	11
4.3 Access Token Transport	11
4.4 Access Token Reception	12
4.5 Access Token Enforcement	13
5. Intermediary processing and notifications	13
5.1 Intermediary nodes	13
5.2 Mirror Server	14
5.3 Observe	14
6. Security Considerations	15
7. IANA Considerations	15
8. Acknowledgements	16
9. References	17
9.1 Normative References	17
9.2 Informative References	17
Appendix A. Example Token Syntax	18

Appendix B. Changelog	19
Authors' Addresses	20

1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a light-weight web transfer protocol, suitable for applications in embedded devices used in services such as smart energy, smart home, building automation, remote patient monitoring etc. Due to the nature of these use cases including critical, unattended infrastructure and the personal sphere, security and privacy are critical components. Use cases for CoRE security are discussed in [I-D.seitz-core-sec-usecases].

CoAP message exchanges can be protected with different security protocols. The CoAP specification defines a DTLS binding for CoAP, which provides communication security services including authentication, encryption, integrity, and replay protection.

Authorization and access control - i.e. controlling who has access to what - is addressed with access control lists, which are assumed to have been provisioned to the devices and which contain lists of identifiers that may start DTLS sessions with the devices.

There are some limitations inherent to such an approach:

1. By restricting the scope of access control to the granularity of identifiers of requesting clients, it is not possible to give different privileges to different entities that are allowed to access the same device. For example, it may be desirable to give some clients the right to GET resources but others the right to POST or PUT resources to the same device; or to give the same client different access rights for different resources on the same device.
2. There are use cases [I-D.seitz-core-sec-usecases] where the granularity of GET/PUT/POST/DELETE is not sufficient to specify the relevant access restrictions. For example, an access policy may depend on local conditions of the device such as date and time, proximity, geo-location, detected effort (press 3 times), or other aspects of the current state of the device.
3. It is not defined how to change access privileges except by re-provisioning. How such changes would be authorized is also unclear.

This document proposes a framework that allows fine-grained and flexible access control, applicable to a generic setting including use cases with constrained devices [I-D.ietf-lwig-terminology].

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Certain security-related terms are to be understood in the sense defined in [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "access control", "confidentiality", "credential", "encryption", "sign", "signature", "data integrity", and "verify".

Terminology for constrained environments is defined in [I-D.ietf-lwig-terminology]. These terms include, but are not limited to, "constrained device", "constrained network", and "device class".

2. Scope and Requirements

This section defines the scope and gives an overview of the requirements that form the basis for the proposed Access Control Framework.

2.1 Resource Authorization and Protocol Authorization

Access control is protection of system resources against unauthorized access. There are different kinds of "system resources" that needs protection and different kinds of protection mechanisms.

For the purpose of this memo, we distinguish between two types of authorization: "Resource Authorization" and "Protocol Authorization".

- o Resource Authorization (RA) deals with the question whether the server should allow a client requesting GET/PUT/POST/DELETE to a resource (where "resource" is as defined in RFC 2616).
- o Protocol Authorization (PA) deals with the question whether the server should allow a client to run a certain protocol with it.

RA is mainly about granting only authorized requests for a resource, and protecting resource related communication between authorized client and server

PA is mainly about protecting the device hosting the resource and avoiding unnecessary protocol processing e.g. to save battery / computing resources or to protect against certain DoS attacks.

Although there are overlapping and degenerate cases, we believe this distinction is useful to understand the purpose of authorization.

There may be dependencies between PA and RA:

- o RA typically imply some PA: If a client is authorized to access a resource hosted on a server, then the client should be allowed to run a protocol (say DTLS) with the server for accessing the resource.
- o PA access does not necessarily imply RA: Just because a client is authorized to execute a protocol (say DTLS) with the server, the client is not necessarily authorized to access any resources hosted on the server.

The CoAP Security Modes [I-D.ietf-core-coap] and the Additional Security Modes for CoAP [I-D.seitz-core-security-modes] define Access Control Lists with information about what clients are allowed to run DTLS with an origin server. This is by definition Protocol Authorization. However, PA can be used to define RA: For example, by allowing access to all resources for all clients successfully executing the protocol.

The scope of the Access Control Framework defined in this draft is primarily RA, but as is noted above, RA implies that complementing PA needs to be defined.

2.2 Requirements

The Access Control Framework SHALL support

- o access control in a constrained environment with constrained devices or networks, in particular,
 - additional messages exclusively for performing access control SHOULD be kept at a minimum to reduce power consumption on the constrained devices.
- o access control applicable to a variety of use cases and access purposes, in particular
 - differentiated access rights for different requesting entities,
 - access control at least at the granularity of RESTful resources,
 - access policies based on local conditions (e.g. state of device, time, position),
- o changes to access policies without re-provisioning, and

- o interoperability between different system components and providers, which is best achieved by the use of state-of-the-art security standards and best practices (in particular end-to-end security between resource and authorized client).

The Access Control Framework SHALL be compatible with a large variety of client-server authentication methods, message protection mechanisms (communication/object security), device key management procedures and trust anchors (secret keys, raw public keys, certificates).

3 Outline of Access Control Framework

In this section we present the rationale leading to our access control framework, the resulting roles, the message flow, and the access control procedure.

3.1 Rationale

Consider a generic setting where a CoAP client wants to access a resource hosted on a CoAP server, which is potentially a constrained device, and where the access rights are determined by the owner of the resource. In this section we introduce some of the terms and procedures and provide some rationale for those.

Managing and evaluating arbitrary access control policies is in general too heavyweight for constrained devices. As a consequence the main authorization decision is externalized to a less constrained node, called the "authorization server", acting on behalf of the resource owner.

On the other hand, access control enforcement should be performed in a trusted environment associated to the resource and as close to the resource as possible, in order to provide end-to-end security between resource and authorized client.

Moreover, verifications of any local conditions should be performed in conjunction with accessing the resource for the following reasons:

- o Transporting information about local conditions in the device to an authorization server for each policy decision (or on a regular basis) introduces delays and/or adds additional messages exclusively for the purpose of performing access control.
- o Local conditions may have changed at the time of enforcement.

We therefore target enforcement and local decisions to take place in the constrained device hosting the resource, or in a proxy-type device offloading a severely constrained device hosting the resource.

We express local conditions as constraints under which an externally granted authorization decision is valid, and which are verified at the time and location of enforcement.

In order to convey the authorization decisions (including local conditions) from the authorization server to the device where access control is enforced, we use integrity protected data objects, which we call "access tokens" (or just "tokens").

Access tokens are acquired from the authorization server and used to gain access to the device. We denote by "access manager" the function of requesting and receiving access tokens from an authorization server. Constrained clients may need support to acquire tokens, in which case the access manager is implemented on a separate node.

NOTES

1. The authorization server must be trusted by all involved parties, in particular by the resource owner. We assume that this trust relationship is manifested through trusted keys established a priori in the constrained device and the authorization server.
2. The existence of such a trust relationship, once introduced to support authorization and access control, can be utilized to optimize key establishment, authentication and message protection between a client and the origin server.

3.2 Roles

The relevant roles involved in this access control framework are:

- o A Resource Owner specifying the policies for access to the resources.
- o An Authorization Server (AS) performing the authorization decision making, based on the access control policies, and provisioned with one or more trusted keys from the Resource Server.
- o A potentially constrained Resource Server (RS) hosting resources and provisioned with one or more trusted keys from the AS.

- o A potentially constrained Origin Client (OC) wishing to access to a resource. As there may be client intermediaries, e.g. forward proxies, the actual CoAP client requesting the RS may be different from the origin client. When there are no other clients to confuse with, we refer to the origin client simply as "the client".
- o An Access Manager (AM) which requests and receives access tokens from an AS. The AM may be a standalone node or integrated/co-located with the OC.

3.3 Message flow

The default procedure for resource access is described in Figure 1, and works as follows:

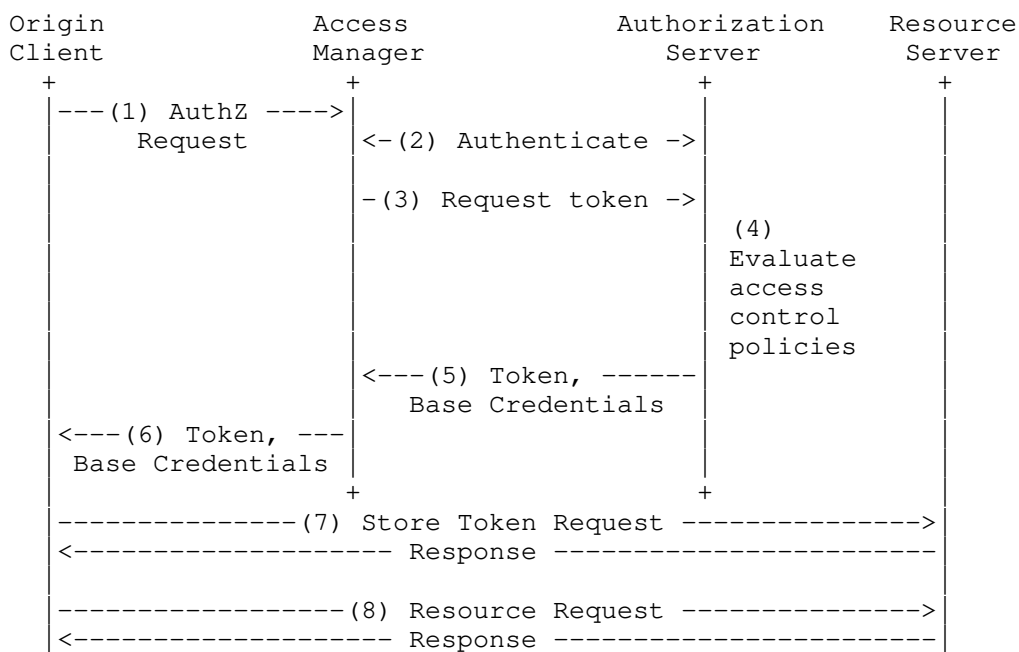


Figure 1: Roles and access control procedure

The OC sends an authorization request to the AM (1).

The AM authenticates to the AS (2) on behalf of the OC. The AM then requests an access token and optionally base credentials for a specific security mode (3). The request contains the OC's subject

identifier which is used to evaluate the access control policies.

The AS makes the authorization decision on behalf of the resource owner (4) and, if granted, responds (5) to the AM with an access token bound to the OC's subject identifier. Optionally it also sends Base Credentials to be used in the message exchange between OC and RS. A base credential may e.g. be the public key of the RS, a public key certificate generated for the OC, or a derived key bootstrapping the trust relation between the AS and RS [I-D.seitz-core-security-modes].

The AM forwards the access token and base credentials to the OC (6).

The OC stores the token on the RS (7). The RS provides a well-known resource for submitting and storing tokens used to authorize future requests. After the token is verified by the RS it is stored and the RS responds appropriately to the OC.

The OC submits Resource Request(s) (8), which are verified against the stored tokens by the RS. If the RS finds a matching token, and all local conditions are met, the request is processed and a response is sent.

Request and Response messages need to be protected, either using communication security, such as DTLS [RFC6347], or object security, such as JWE [I-D.ietf-jose-json-web-encryption] and JWS [I-D.ietf-jose-json-web-signature]. The base credentials that AS optionally provides, can be used to establish the cryptographic keys for the message protection scheme, and protocol authorization for communication security establishment.

4. Access Tokens

The access token is a secure object containing authorization information passed from the AS via the AM and OC to the RS. In this section the content, protection and transport of an access token is discussed.

4.1 Requirements

In order to enable the RS to enforce the authorization decision, the access token MUST provide the following information:

- o Which resource does the decision apply to.
- o Which action (GET, PUT, POST, DELETE) does the decision apply

to.

- o Which OC does the decision apply to (subject identifier), and how can this OC be authenticated (if necessary).
- o Which AS has created this access token (issuer). This information MAY be implicit from the signature of the token.
- o Under what other conditions is the access token valid (local conditions evaluated by the resource server at access time, e.g. expiration, number of uses).

The access token SHALL include a sequence number which together with the issuer, is unique for a given RS. The token MAY state a specific allowed payload value, where applicable (e.g. for PUT/POST requests).

The access token MUST be integrity protected by the AS such that it can be verified by the RS using a trusted key. An access token MAY be protected with a message authentication code. The corresponding symmetric key is then called the Access token Key (AK). An access token MAY be signed with a private key in an asymmetric signature scheme, for example the AS's private key (PrivK_AS). The RS, or other nodes verifying the access token, MUST have access to the relevant key (AK or PubK_AS) at the time of performing the verification.

Using an asymmetric signature scheme is RECOMMENDED if intermediary nodes, between OC and RS, are expected to verify the access token, since it is less security critical to provision PubK_AS to the intermediary nodes, rather than AK.

4.2 Access Token Protection

Since access tokens are to be consumed by constrained devices, the protection of the access token must be lightweight and compact. This specification RECOMMENDS the use of JSON Web Signatures (JWS) [I-D.ietf-jose-json-web-signature] as a means of signing access tokens. It is furthermore RECOMMENDED to use the JWS Compact Serialization in order to further reduce the size of the protected access token object.

In an object security setting, where the token may be transferred over an insecure channel, it can be encrypted and integrity protected using JWE [I-D.ietf-jose-json-web-encryption].

4.3 Access Token Transport

The access token can be transported from the OC to the RS in

different ways.

- A. One possibility is to extend the communication security establishment protocol (e.g. using TLS authorization extensions [RFC5878] in the DTLS/TLS handshake).
- B. Another possibility is to use the application protocol (e.g. CoAP) and send the tokens as regular requests.

In either case the access token is verified upon reception, and if it is valid (see 4.4), stored for being used in a subsequent resource request. If the access token is not valid (see Section 4.4) the RS aborts the corresponding protocol to avoid unnecessary processing. This saves resources in the case A above, since the communication between RS and OC is still in a very early stage. However, early abort of communication establishment can also be achieved by protocol authorization, see e.g. [I-D.seitz-core-security-modes]. Moreover one drawback with case A. is that a new session has to be established if the same OC needs to submit a new access token to the RS.

For these reasons implementations SHALL at least support the transport of access tokens in the application protocol. For this to work, there needs to be a well-known location on the RS to which the OC can send the access token. This resource SHALL have the local URI-path `'./well-known/core/authz/'`. Writing to this location SHALL NOT require Resource Authorization (i.e. no access token is required).

4.4 Access Token Reception

Upon receiving an access token which is not already stored the RS SHALL perform the following processing:

- o Verify if the token is revoked
- o Verify if the token is from a trusted issuer (i.e. the AS known to the RS)
- o Verify the signature of the token

In order to support token revocation the RS SHALL maintain a list of sequence numbers per issuer, specifying the revoked tokens. If the access token passes the verifications, we denote it 'valid'. The RS SHALL only store valid access tokens. Revoked tokens SHALL be removed from storage.

Optionally the RS can use the sequence number of the token, to enforce token expiration. This can be done by rejecting sequence

numbers that are significantly lower than the highest sequence number the RS has received so far.

Optionally the RS can use the time lapse since received to enforce token expiration. This can be done by storing together with the token the local time as measured by the RS upon reception.

4.5 Access Token Enforcement

Upon receiving a request, the RS SHALL perform the following processing on the relevant stored token:

- o If there is information about expiry, verify if the stored token has expired
- o Verify that the stored token is bound to the requesting subject
- o Verify that the stored token authorizes the received request (including local conditions)

If no matching token is found, the request MUST be rejected using the response code 4.03 Forbidden.

Keys or identifiers established in the communication security protocol can be used to support subject binding verification. Table 1 shows examples of token subject identifiers based on different CoAP security modes (see also section 9 of [I-D.ietf-core-coap], [RFC4279] and [I-D.seitz-core-security-modes]).

CoAP security mode	Token subject identifier
PreSharedKey	psk_identity
RawPublicKey	public key fingerprint
Certificate	Subject DN
DerivedKey	psk_identity
AuthorizedPublicKey	public key fingerprint

Table 1: DTLS parameters as token subject identifiers

5. Intermediary processing and notifications

This section describes the security implications of intermediary processing and notifications for access control.

5.1 Intermediary nodes

There may be intermediary nodes between OC and RS, including forward proxies, reverse proxies, cross-proxies, gateways, etc. From an access control point of view the RS SHOULD be able to verify that a received CoAP request is originating from the OC referenced in the received access token. This has implications on the access token and message protection profiles.

We distinguish between the end-to-end security setting where no intermediary nodes need be trusted and the hop-by-hop security setting where at least one intermediary node must be trusted.

DTLS generally needs to be hop-by-hop in case of proxies, this requires some degree of trust in a proxy which may not be acceptable for some applications. A RS sending back the response via the forward proxy trusts the forward proxy with the plain text response (e.g. a GET response) and that the proxy has established secure communication with the OC.

In the hop-by-hop case, neither DTLS nor CoAP offers any means for RS to authenticate the OC.

If the RS has established DTLS with a forward proxy which proxies requests from an OC, then the access token MUST be signed by the OC in addition to the AS integrity protection. The RS can not authenticate the OC directly, but it can infer from a correctly signed valid and fresh access token that the OC is authorized and has an intent to perform the request.

5.2 Mirror Server

The access control framework can also be applied to the scenario where a mirror server as defined in [I-D.vial-core-mirror-proxy] is present. In such a scenario, each RS behaves as a client of the mirror server. The access control enforcement in this case, would be made at the mirror server instead of in a constrained RS, and the trusted AS keys would have to be provisioned to the mirror server. However, to a client wishing to access a resource, the mirror server behaves as any other RS and is indistinguishable (transparent), thereby requiring no change for the communication between client and the mirror server. The communication between the mirror server and the constrained RS may or may not be secured, and is oblivious to the protocols used between the client and the mirror server.

5.3 Observe

The access control framework can also be applied, as it is, in the case where the CoAP observe option [I-D.ietf-core-observe] is used. With the observe option, clients can register an interest in a

particular resource by sending a CoAP request containing the observe option to a RS. The RS would in this case maintain the state information for this expressed interest and send responses on state changes only as long as the access token and local conditions presented in the original interest request are valid. The local conditions may need to be verified at each state change. Once the access token expires, the RS will remove any state information for the interest expressed. The OC would then have to send a new CoAP request with an observe option expressing interest and a new access token for demonstrating that it is allowed access.

6. Security Considerations

The present framework aims to protect the resources on RS, the servers themselves, and the services offered. The means proposed to protect these assets is to enforce granular access restrictions on accessing the devices. Due to the setup of the framework, there is also a need to protect the authorization decisions and the keys used to protect the entire resource access procedure.

The AS is a Trusted Third Party from the point of view of the resource owner. If the AS is compromised, it could e.g. issue access tokens to unauthorized parties.

Since the AM requests tokens on behalf of the OC, the AS must be able to verify that it really represents the OC.

In order to enforce a policy decision, the RS must authenticate the OC, and match the identifier of the authenticated entity with the subject identifier of the access token.

While DTLS offers bundled encryption and integrity protection of both payload and headers, an object security approach allows for a trade-off between protection against performance. Depending on the trust model, access token and payload may need to be encrypted because eavesdropping will reveal information about the OC's request, which may be privacy sensitive. Wrapping of the payloads as secure objects allows differentiated protection of the content based on its sensitiveness.

A typical access token has a size in the order of hundreds of bytes. If tokens can be sent to the RS by unauthenticated clients, care must be taken to prevent that the processing and storage of the token opens for Denial of Service attacks.

7. IANA Considerations

<TBD: IANA considerations text>

8. Acknowledgements

The authors would like to thank Stefanie Gerdes, Mats Naeslund, and John Mattsson for contributions and helpful comments.

9. References

9.1 Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)", draft-ietf-
core-coap-18 (work in progress), June 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2 Informative References

[I-D.seitz-core-sec-usecases]
Seitz, L., Gerdes, S., and Selander, G., "Use cases for
CoRE security", draft-seitz-core-sec-usecases-00 (work in
progress), September 2013.

[I-D.ietf-lwig-terminology]
Bormann, C., Ersue, M., and Keranen, A., "Terminology for
Constrained Node Networks", draft-ietf-lwig-terminology-05
(work in progress), July 2013.

[I-D.seitz-core-security-modes]
Seitz, L., and Selander G., "Additional Security Modes for
CoAP", draft-seitz-core-security-modes-00 (work in
progress), October 2013

[I-D.ietf-jose-json-web-encryption]
Jones, M., Rescorla, E., and Hildebrand J., "JSON Web
Encryption (JWE)", draft-ietf-jose-json-web-encryption-17
(work in progress), October 2013.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and Sakimura N., "JSON Web
Signature (JWS)", draft-ietf-jose-json-web-signature-17
(work in progress), October 2013.

[I-D.vial-core-mirror-proxy]
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-
proxy-01 (work in progress), July 2012.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-

core-observe-11 (work in progress), October 2013.

[I-D.bormann-cbor] Bormann, C., and Hoffman P., "Concise Binary Object Representation (CBOR)", draft-bormann-cbor-09 (work in progress), September 2013.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, August 2007.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC5878] Brown, M. and R. Housley, "Transport Layer Security (TLS) Authorization Extensions", RFC 5878, May 2010.

[RFC4279] Eronen, P., Ed., and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

Appendix A. Example Token Syntax

In this section we give an example of an access token using a compact JSON notation.

```

01 {
02   "SN": "081d5ff7bb2c2d08",
03   "IS": "6f",
04   "SI": "435143a1b5fc8bb70a3aa9b10f6673a8",
05   "LCO": {
06     "NB": "09:00:00Z",
07     "NA": "17:00:00Z"
08   },
09   "ACT": "POST",
10   "VAL": "open",
11   "RES": "node346/doorLock"
12 }
```

+-----+-----+	
Token element	Encoding
+-----+-----+	
Sequence number	SN
Issuer	IS

Subject identifier	SI
Local conditions	LCO
Action	ACT
Allowed payload value	VAL
Resource	RES

Table 2: Token elements encoding

In this example the issuer is identified by a single byte, this is possible because the token is for a specific RS, which is not expected to have more than 256 distinct trusted AS.

The subject identifier is a public key fingerprint binding the token to the corresponding public key, which in turn could be used to establish a DTLS connection to the RS using the RawPublicKey security mode (see section 9 of [I-D.ietf-core-coap]).

The local condition specifies a time frame during which the token is valid (NB = not before, NA = not after). The syntax and semantics of such conditions must be pre-defined on the consuming RS so that it can parse and enforce them.

The action specifies the RESTful action (DELETE, GET, POST, PUT) that this token authorizes, while the resource specifies the URI host and URI path from the CoAP requests.

For actions including a payload (typically PUT and POST), the token can specify a restriction on the allowed payload value.

Note that JSON is used here because it gives a human readable token format, for production deployments one should consider using a more compact representation format such as CBOR [I-D.bormann-cbor] to reduce the token size.

Appendix B. Changelog

Changes from -00 to -01:

- o The draft is significantly shortened, content is moved to separate drafts and much informational content has been removed.
- o The limited use case descriptions are greatly expanded and moved into a separate draft [I-D.seitz-core-sec-usecases].
- o The key provisioning schemes are generalized to alternate CoAP security modes and described in a separate draft [I-D.seitz-core-security-modes]

- o The ACL categories are replaced by the distinction between protocol authorization and resource authorization.
- o The Access Manager functionality originally defined in draft-gerdes-core-dcaf-authorize-00 is introduced.
- o The communication security profile description is removed. For a detailed DTLS based access control setting see [I-D.draft-gerdes-core-dcaf-authorize].
- o The object security profile is planned for a future draft.

Authors' Addresses

Goeran Selander
Ericsson
Farogatan 6
16480 Kista
SWEDEN

Email: goran.selander@ericsson.com

Mohit Sethi
Ericsson
Hirsalantie 11
02420 Jorvas
FINLAND

Email: mohit.m.sethi@ericsson.com

Ludwig Seitz
SICS Swedish ICT AB
Scheelevagen 17
22370 Lund
SWEDEN

Email: ludwig@sics.se

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

B. Silverajan
Tampere University of Technology
T. Savolainen
Nokia Technologies
March 5, 2018

CoAP Communication with Alternative Transports
draft-silverajan-core-coap-alternative-transports-11

Abstract

The aim of this document is to provide a way forward to best decide upon how alternative transport information can be expressed in a CoAP URI. This draft examines the requirements for a new URI format for representing CoAP resources over alternative transports. Various potential URI formats are presented. Benefits and drawbacks of embedding alternative transport information in various ways within the URI components are also discussed. From all listed formats, the document finds scheme-based model to be the most technically feasible.

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 September 6, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Conformance and Design Considerations	4
3. Situating Transport Information in CoAP URIs	5
3.1. Using the URI scheme component	5
3.1.1. Analysis	6
3.2. Using the URI authority component	6
3.2.1. Analysis	7
3.3. Using the URI path component	7
3.3.1. Analysis	7
3.4. Using the URI query component	7
3.4.1. Analysis	8
4. Discussion	8
5. IANA Considerations	8
6. Security Considerations	9
7. Acknowledgements	9
8. References	9
8.1. Normative References	9
8.2. Informative References	9
Appendix A. Expressing transport in the URI in other ways	10
A.1. Transport information as part of the URI authority	10
A.1.1. Usage of DNS records	11
A.2. Making CoAP Resources Available over Multiple Transports	12
A.3. Transport as part of a 'service:' URL scheme	13
Authors' Addresses	14

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a lightweight, binary application layer protocol designed for constrained environments. Owing to its operating environment, CoAP uses UDP and DTLS as its underlying transports between communicating endpoints. However, with an increase in deployment experiences as well as its popularity, compelling reasons exist for extending CoAP messaging to work over alternative transports. These allow CoAP to better address firewall and NAT traversal issues, to operate in Web browser-based and HTML5 applications as well as for energy-constrained M2M communication in cellular networks. At the time of writing, these transports are:

- o TCP, TLS and Websockets [RFC8323]

- o SMS for cellular networks[I-D.becker-core-coap-sms-gprs]
- o SLIP for serial interfaces[I-D.bormann-t2trg-slipmux]

CoAP uses a REST-based model similar to HTTP, where URIs are used to identify resources at servers. An important factor of allowing CoAP communication over alternative transports, is to express not only the resource identifier, but also the alternative transport information in the URI.

CoAP URIs contain information, such as the endpoint address as well as the location of the resource hosted at the endpoint. CoAP URIs beginning with "coap://" are using UDP, while those beginning with "coaps://" are using DTLS.

```

coap :// server.example.org /sensors/temperature
  \____/      \_____/ \_____/ \_____/
   |           \      \      \      \
  URI scheme   URI authority  URI path

```

Figure 1: A CoAP URI

Figure 1 shows the structure of a simple example CoAP URI, in which the various URI components can be interpreted as follows:

- o The URI scheme component (e.g. "coap") contains an application-level identifier which typically identifies the protocol being used as well as its transport and network level protocol configurations. Such configurations are defined by convention or standardisation of the protocol using the scheme.
- o The URI authority component ("server.example.com") contains the endpoint identification, which is typically a fully qualified domain name or a network-level host address.
- o The URI path component ("/sensors/temperature") contains a parameterised resource identifier providing the location and identity of the resource at the endpoint.

In addition to these URI components, Figure 2 shows how specific queries on resource representations are provided by CoAP clients to servers, by specifying one or more URI query components in the URI.

```
coap :// server.example.org /sensors/temperature ?u=cel
                                     \_____/
                                     |
                               URI query
```

Figure 2: A CoAP URI with query

This document focuses on how CoAP URIs can be extended to contain information about alternative transports. For deriving the new URI format, the main design considerations are presented in the next section. Following that, various potential URIs are presented. These URIs provide examples of how transport identifiers can be situated in the URI scheme, authority, path or query components. The proposed URIs are analysed to select feasible formats while disqualifying those not meeting the design criteria.

2. Conformance and Design Considerations

In order to understand which URI formats are best suited for expressing transport information, certain considerations firstly need to be taken into account. Doing so eliminates URI formats that do not meet or conform to the stated requirements. The main criteria are:

1. Conformance to the generic syntax for a URI described in [RFC3986]. A URI format needs to be described in which each URI component clearly meets the syntax and percent-encoding rules described.
2. Alignment with best practices for URI design, as described in [RFC7320]. This is particularly important when it pertains to establishing or standardising the structure and usage of URIs with respect to the various URI components.
3. Request messages sent to a CoAP endpoint using a CoAP Transport URI may be responded to with a relative URI reference. [RFC3986] provides an algorithm to establish how relative references can be resolved against a base URI to obtain a target URI. Given this algorithm, a URI format needs to be described in which relative reference resolution does not result in a target URI that loses its transport-specific information
4. The URI can be supplied as a Proxy-Uri option by a CoAP end-point to a CoAP forward proxy. This allows communication with a CoAP end-point residing in a network using a different transport. Section 6.4 of [RFC7252] provides an algorithm for parsing a received URI to obtain the request's options. Conformance to

[RFC3986] is also necessary in order for the parsing algorithm to be successful.

In addition to the above mentioned requirements, where possible, the following considerations need to be borne in mind:

1. The URI format is able to represent a resource and the transport information for use in constrained environments, without requiring the presence of a naming infrastructure, such as DNS or a directory/lookup service.
 2. Alternative transport information can be easily retrieved by computationally constrained nodes. In other words, the URI format does not result in unnecessarily complex code or logic in such nodes to parse and extract the transport to be used, nor the endpoint address.
 3. URIs are designed to uniquely identify resources. When a single resource is represented with multiple URIs, URI aliasing [WWWArchv1] occurs. Avoiding URI aliasing is considered good practice.
 4. CoAP URIs do not support fragment identifiers.
3. Situating Transport Information in CoAP URIs

The following subsections aim to describe potential URI formats in which the alternative transport information is placed in various URI components.

3.1. Using the URI scheme component

Expressing the transport information in the URI scheme component can be achieved by using new schemes. These can conform to an agreed-upon convention such as "coap+alternative_transport_name" for each new alternative transport and/or "coaps+alternative_transport_name" for its secure counterpart.

Examples of such URIs are:

- o coap+tcp://server.example.org/sensors/temperature for using CoAP over TCP
- o coap+sms://0015105550101/sensors/temperature for using CoAP over SMS with the endpoint identifier being a telephone subscriber number

- o coaps+tcp://server.example.org/sensors/temperature for using CoAP over TLS

3.1.1. Analysis

Expressing transport information in the URI scheme delivers a URI which is human-readable and computationally as easy to parse as standard CoAP URIs, to extract transport identification information. The URI syntax conforms to [RFC3986], and relative URI resolution does not result in the loss of transport identification information. However, each new alternative transport requires minting new schemes, and IANA intervention is required for the registration of each scheme name. The registration process follows the guidelines stipulated in [RFC7595]. Additionally, should a CoAP server wish to expose its resources over multiple transports (such as both UDP and TCP) , URI aliasing can occur if the URI scheme components of these multiple URIs differ in describing the same resource.

3.2. Using the URI authority component

Expressing the transport information within the authority component can result in two possible URI formats.

The first approach is to structure the URI authority's host sub-component with a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint_identifier".

Examples of resulting URIs are:

- o coap://tcp-server.example.org/sensors/temperature for using CoAP over TCP
- o coap://sms-0015105550101/sensors/temperature for using CoAP over SMS

The second approach is to hint at the alternative transport information, by explicitly specifying using the URI authority's port sub-component, thereby differentiating them from standard CoAP URIs.

Examples of resulting URIs are:

- o coap://server.example.org:5684/sensors/temperature for using CoAP over TLS
- o coap://server.example.org:80/sensors/temperature for using CoAP over WebSockets

3.2.1. Analysis

Embedding the transport information in the host would violate the guidelines for the structure of URI authorities in section 2.2 of [RFC7320]. Consequently, the host in a URI authority component cannot be used as a basis for a new CoAP URI for alternative transports.

Embedding the transport information in the port, on the other hand, would not violate the guidelines for the structure of URI authorities in section 2.2 of [RFC7320]. It would result in a CoAP URI that is less human-readable, but URI aliasing is minimised.

On the other hand, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form `"/server2.example.org/path/to/another/resource"`, relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI authority component cannot be used as a basis for a new CoAP URI for alternative transports.

3.3. Using the URI path component

Should the URI path component be used, then special characters or keywords need to be supplied in the path to make the transport explicit. Here, many proposals can exist. In general however, this will result in a URI format such as:

- o `coap://server.example.org/sensors/temperature;tcp` for using CoAP over TCP, by appending the transport information at the end of the URI.

3.3.1. Analysis

Embedding the transport information in the URI path directly results in a URI that is human-readable. However, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form `"../..../path/to/another/resource"`, relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI path component cannot be used as a basis for a new CoAP URI for alternative transports.

3.4. Using the URI query component

The alternative transport information, should URI query components be used, would result in a URI format such as:

- o `coap://server.example.org/sensors/temperature?alternative-transport=wss` for using CoAP over secure WebSockets.

3.4.1. Analysis

Embedding the transport information in a URI query also results in a URI that is human-readable. However, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form `"../..path/to/another/resource"`, relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI query component cannot be used as a basis for a new CoAP URI for alternative transports.

4. Discussion

Based on the analysis of the various options for embedding alternative transport information in a CoAP URI, the most technically feasible option is to use the URI scheme component, as described in Section 3.1. To date, this has also been the WG consensus.

A discussion with IESG members during review of [RFC8323] revealed however, that using the URI scheme to express transport information is not desirable, to avoid the proliferation of new URI schemes for the same application-layer protocol. A strategy was instead proposed to preserve the existing CoAP URI and reuse it for alternative transports, by employing a combination of UDP Confirmable messages and timeouts to determine the eventual correct transport to use between a client and server [IESG-feedback]. The undertaken strategy would have obvious implications regarding interoperability, application and protocol logic, resource usage, for both new CoAP and existing CoAP implementations and deployments. Although URI aliasing can theoretically be avoided with this approach, at the time of writing, its technical feasibility over using the simpler strategy of using URI schemes, has yet to be validated. An obvious drawback is therefore that implementers and other SDOs may choose to provisionally or permanently register new URI schemes with IANA, for CoAP over alternative transports anyway, as was done by the Open Connectivity Foundation (OCF) [CoAP-TCP-TLS-registration].

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

New security risks are not envisaged to arise from the guidelines given in this document, for describing a new URI format containing transport identification within the URI scheme component. However, when specific alternative transports are selected for implementing support for carrying CoAP messages, risk factors or vulnerabilities can be present. Examples include privacy trade-offs when MAC addresses or phone numbers are supplied as URI authority components, or if specific URI path components employed for security-specific interpretations are accidentally encountered as false positives. While this document does not make it mandatory to introduce a security mode with each transport, it recommends ascribing meaning to the use of "coap+" and "coaps+" prefixes in the scheme component, with the "coaps+" prefix used for secure transports for CoAP messages.

7. Acknowledgements

Email discussions, comments and ideas from Thomas Fossati, Akbar Rahman, Klaus Hartke, Martin Thomson, Mark Nottingham, Dave Thaler, Graham Klyne, Carsten Bormann and Markus Becker greatly helped previous versions of this draft.

8. References

8.1. Normative References

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [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>>.
- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.

8.2. Informative References

- [CoAP-TCP-TLS-registration]
, <<https://www.iana.org>>.

- [I-D.becker-core-coap-sms-gprs]
Kuladinithi, K., Becker, M., Li, K., and T. Poetsch,
"Transport of CoAP over SMS", draft-becker-core-coap-sms-
gprs-06 (work in progress), February 2017.
- [I-D.bormann-t2trg-slipmux]
Bormann, C. and T. Kaupat, "Slipmux: Using an UART
interface for diagnostics, configuration, and packet
transfer", draft-bormann-t2trg-slipmux-02 (work in
progress), January 2018.
- [IESG-feedback]
, <[https://www.ietf.org/mail-archive/web/core/current/
msg08768](https://www.ietf.org/mail-archive/web/core/current/msg08768)>.
- [RFC2609] Guttman, E., Perkins, C., and J. Kempf, "Service Templates
and Service: Schemes", RFC 2609, DOI 10.17487/RFC2609,
June 1999, <<https://www.rfc-editor.org/info/rfc2609>>.
- [RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines
and Registration Procedures for URI Schemes", BCP 35,
RFC 7595, DOI 10.17487/RFC7595, June 2015,
<<https://www.rfc-editor.org/info/rfc7595>>.
- [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>>.
- [WWWArchv1]
, <<http://www.w3>>.

Appendix A. Expressing transport in the URI in other ways

Other means of indicating the transport as a distinguishable component within the CoAP URI are possible, but have been deemed unsuitable by not meeting the design considerations listed, or are incompatible with existing practices outlined in [RFC7252]. They are however, retained in this section for historical documentation and completeness.

A.1. Transport information as part of the URI authority

A single URI scheme, "coap-at" can be introduced, as part of an absolute URI which expresses the transport information within the authority component. One approach is to structure the component with

a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint_identifier".

Examples of resulting URIs are:

- ```
o coap-at://tcp-server.example.com/sensors/temperature
o coap-at://sms-0015105550101/sensors/temperature
```

An implementation note here is that some generic URI parsers will fail when encountering a URI such as "coap-at://tcp-[2001:db8::1]/sensors/temperature". Consequently, an equivalent, but parseable URI from the ip6.arpa domain needs to be formulated instead. For [2001:db8::1] using TCP, this would result in the following URL:

```
coap-at://tcp-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0
.1.0.2.ip6.arpa:5683/sensors/temperature
```

Usage of an IPv4-mapped IPv6 address such as `::ffff.192.100.0.1` can similarly be expressed with a URI from the `ip6.arpa` domain.

This URI format allows the usage of a single scheme to represent multiple types of transport end-points. Consequently, it requires consistency in ensuring how various transport-specific endpoints are identified, as a single URI format is used. Attention must be paid towards the syntax rules and encoding for the URI host component. Additionally, against a base URI of the form "coap-at://tcp-server.example.com/sensors/temperature", resolving a relative reference, such as "//example.net/sensors/temperature" would result in the target URI "coap-at://example.net/sensors/temperature", in which transport information is lost.

#### A.1.1. Usage of DNS records

DNS names can be used instead of IPv6 address literals to mitigate lengthy URLs referring to the ip6.arpa domain, if usage of DNS is possible.

DNS SRV records can also be employed to formulate a URL such as:

```
coap-at://srv_coap_tcp.example.com/sensors/temperature
```

in which the "srv" prefix is used to indicate that a DNS SRV lookup should be used for `_coap._tcp.example.com`, where usage of CoAP over TCP is specified for `example.com`, and is eventually resolved to a numerical IPv4 or IPv6 address.

## A.2. Making CoAP Resources Available over Multiple Transports

The CoAP URI used thus far is as follows:

```
URI = scheme ":" hier-part ["?" query]
hier-part = "//" authority path-abempty
```

A new URI format could be introduced, that does not possess an "authority" component, and instead defining "hier-part" to instead use another component, "path-rootless", as specified by RFC3986 [RFC3986]. The partial ABNF format of this URI would then be:

```
URI = scheme ":" hier-part ["?" query]
hier-part = path-rootless
path-rootless = segment-nz *("/" segment)
```

The full syntax of "path-rootless" is described in [RFC3986]. A generic URI defined this way would conform to the syntax of [RFC3986], while the path component can be treated as an opaque string to indicate transport types, endpoints as well as paths to CoAP resources. A single scheme can similarly be used.

A constrained node that is capable of communicating over several types of transports (such as UDP, TCP and SMS) would be able to convey a single CoAP resource over multiple transports. This is also beneficial for nodes performing caching and proxying from one type of transport to another.

Requesting and retrieving the same CoAP resource representation over multiple transports could be rendered possible by prefixing the transport type and endpoint identifier information to the CoAP URI. This would result in the following example representation:

```
coap-at:tcp://example.com?coap://example.com/sensors/temperature
 _____/ _____/
 \ \
 Transport-specific CoAP Resource
 Prefix
```

Figure 3: Prefixing a CoAP URI with TCP transport

Such a representation would result in the URI being decomposed into its constituent components, with the CoAP resource residing within the query component as follows:

Scheme: coap-at

Path: tcp://example.com

Query: `coap://example.com/sensors/temperature`

The same CoAP resource, if requested over a WebSocket transport, would result the following URI:

`coap-at:ws://example.com/endpoint?coap://example.com/sensors/temperature`

Transport-specific Prefix                      CoAP Resource

Figure 4: Prefixing a CoAP URI with WebSocket transport

While the transport prefix changes, the CoAP resource representation remains the same in the query component:

Scheme: `coap-at`

Path: `ws://example.com/endpoint`

Query: `coap://example.com/sensors/temperature`

The URI format described here overcomes URI aliasing [WWWArchv1] when multiple transports are used, by ensuring each CoAP resource representation remains the same, but is prefixed with different transports. However, against a base URI of this format, resolving relative references of the form `"//example.net/sensors/temperature"` and `"/sensor2/temperature"` would again result in target URIs which lose transport-specific information.

Implementation note: While square brackets are disallowed within the path component, the '[' and ']' characters needed to enclose a literal IPv6 address can be percent-encoded into their respective equivalents. The ':' character does not need to be percent-encoded. This results in a significantly simpler URI string compared to section 2.2, particularly for compressed IPv6 addresses. Additionally, the URI format can be used to specify other similar address families and formats, such as Bluetooth addresses.

#### A.3. Transport as part of a 'service:' URL scheme

The "service:" URL scheme name was introduced in [RFC2609] and forms the basis of service description used primarily by the Service Location Protocol. An abstract service type URI would have the form `"service:<abstract-type>:<concrete-type>"`

where `<abstract-type>` refers to a service type name that can be associated with a variety of protocols, while the `<concrete-type>`

then providing the specific details of the protocol used, authority and other URI components.

Adopting the "service:" URL scheme to describe CoAP usage over alternative transports would be rather trivial. To use a previous example, a CoAP service to discover a Resource Directory and its base RD resource using TCP would take the form

```
service:coap:tcp://host.example.com/.well-known/core?rt=core-rd
```

The syntax of the "service:" URL scheme differs from the generic URI syntax and therefore such a representation should be treated as an opaque URI as Section 2.1 of [RFC2609] recommends.

#### Authors' Addresses

Bilhanan Silverajan  
Tampere University of Technology  
Korkeakoulunkatu 10  
FI-33720 Tampere  
Finland

Email: bilhanan.silverajan@tut.fi

Teemu Savolainen  
Nokia Technologies  
Hatanpaaen valtatie 30  
FI-33100 Tampere  
Finland

Email: teemu.savolainen@nokia.com

CoRE  
Internet Draft  
Intended status: Standards track  
Expires: April 2014

A. Bhattacharyya  
S. Bandyopadhyay  
A. Pal  
Tata Consultancy Services Ltd.  
October 9, 2013

CoAP option for no server-response  
draft-tcs-coap-no-response-option-04

Abstract

There can be typical M2M scenarios where responses from the data sink to the data source against request/ notification from the source might be considered redundant. This kind of open-loop exchange (with no reverse path from the sink to the source) may be desired while updating resources in constrained systems looking for maximized throughput with minimized resource consumption. CoAP already provides a non-confirmable (NON) mode of exchange where The receiving end-point does not respond with ACK. However, the receiving end-point responds the sender with a status code indicating "the result of the attempt to understand and satisfy the request".

This draft introduces a header option: 'No-Response' to suppress responses from the receiver and discusses exemplary use cases which motivated this proposition based on real experience. This option also provides granularity by allowing suppression of a typical class or a combination of classes of responses. This option may be effective for both unicast and multicast scenarios.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on April 9, 2014.

#### Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### Table of Contents

|                                                                                           |   |
|-------------------------------------------------------------------------------------------|---|
| 1. Introduction .....                                                                     | 3 |
| 1.1. Granular suppression of responses .....                                              | 3 |
| 1.2. Terminology .....                                                                    | 4 |
| 2. Potential benefits .....                                                               | 4 |
| 3. Exemplary application scenarios .....                                                  | 4 |
| 3.1. Frequent update of geo-location from vehicles to backend (Category 1) .....          | 5 |
| 3.1.1. Benefits using No-Response .....                                                   | 5 |
| 3.2. Multicasting traffic congestion information to PDAs/ smart-phones (Category 2) ..... | 6 |
| 3.2.1. Using granular response suppression .....                                          | 6 |

|                                                                  |    |
|------------------------------------------------------------------|----|
| 3.2.2. Benefits using No-Response .....                          | 6  |
| 4. Option Definition .....                                       | 6  |
| 4.1. Achieving granular suppression .....                        | 8  |
| 5. Example .....                                                 | 9  |
| 5.1. Request/response Scenario .....                             | 9  |
| 5.1.1. Using No-Response with PUT .....                          | 9  |
| 5.1.2. Using No-Response with POST .....                         | 10 |
| 5.1.2.1. POST updating a target resource .....                   | 10 |
| 5.1.2.2. POST performing updates through resource creation ..... | 11 |
| 5.2. Resource-observe Scenario .....                             | 12 |
| 6. IANA Considerations .....                                     | 13 |
| 7. Security Considerations .....                                 | 14 |
| 8. Acknowledgments .....                                         | 14 |
| 9. References .....                                              | 14 |
| 9.1. Normative References .....                                  | 14 |

## 1. Introduction

This draft proposes a new header option 'No-Response' for Constrained Application Protocol (CoAP). This option enables the sender end-point to explicitly express its disinterest in getting responses back from the receiving end-point. By default this option expresses disinterest in any kind of response. This option should be applicable along with non-confirmable (NON) updates. At present this option will have no effect if used with confirmable (CON) mode.

Along with the technical details this draft presents some practical application scenarios which should bring out the utility of this option.

### 1.1. Granular suppression of responses

This option enables granularity by allowing the sender to choose the typical class or combination of classes of responses which it is disinterested in. For example, a sender may explicitly tell the receiver that no response is required unless something 'bad' happens and a response of class 4.xx or 5.xx is to be fed back to the sender. No response is required in case of 2.xx classes. A similar scheme is described in Section 3.7 of [I-D.ietf-core-groupcomm] on the server side. Here the server may perform granular suppression for group communication. But in this case the server itself decides whether to suppress responses or not. This option enables the clients to explicitly inform the server about the disinterest in responses.

## 1.2. Terminology

The terms used in this draft are in conformance with those defined in [I-D.ietf-core-coap].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119.

## 2. Potential benefits

If this option is opportunistically used with fitting M2M applications then the concerned systems may benefit in the following aspects:

- \* Reduction in network clogging
- \* Reduction in server-side loading
- \* Reduction in battery consumption at the constrained end-point
- \* Reduction in communication cost at the constrained end-point
- \* May help to satisfy hard real-time requirements (since, waiting due to closed loop latency is completely avoided)

## 3. Exemplary application scenarios

The described scenarios are confined within a communication pattern where there is a direct communication channel between a constrained device (the device may well be a constrained gateway) and an unconstrained backend. Also, we consider only the scenario of data updates which happen through a push to the server by the client using PUT or POST.

The application scenarios are classified into 2 categories as below:

Category 1) Data-source=constrained device; Data-sink=backend.

Category 2) Data-source=backend; Data-sink=constrained device.

Next sub-section describes the user stories and the potential benefits in each of the cases through the use of No-Response option. An Intelligent Traffic System (ITS) is considered as the base application. The application scenarios are formed out of the different aspects of ITS.

### 3.1. Frequent update of geo-location from vehicles to backend (Category 1)

Each vehicle in ITS is equipped with a sensor-gateway comprising sensors like GPS and Accelerometer. The sensor-gateway connects to the Internet using a low-bandwidth cellular (e.g. GPRS) connection. The GPS co-ordinates are periodically updated to the backend server by the gateway. In case of ITS the update rate is adaptive to the motional-state of the vehicle. If the vehicle moves fast the update rate is high as the position of the vehicle changes rapidly. If the vehicle is static or moves slowly then the update rate is low. This ensures that bandwidth and energy is not consumed unnecessarily. The motional-state of the vehicle is inferred by a local analytics running on the sensor-gateway which uses the accelerometer data and the rate of change in GPS co-ordinates. The back-end server hosts applications which use the updates for each vehicle and produce necessary information for remote users.

Retransmitting a location co-ordinate which is already passed by a vehicle is not efficient as it adds redundant traffic to the network. So, the updates are done in NON mode. However, given the thousands of vehicles updating frequently, the NON exchange will also trigger huge number of status responses from the backend. Each response in the air is of 4 bytes of application layer plus several bytes originating from the lower layers. Thus the cumulative load on the network will be quite significant.

On the contrary, if the edge devices explicitly declare that they do not need any status response then significant load will be reduced from the network and the server as well. The assumption is that since the update rate is high stray losses in geo-locations will be compensated with the large update rate and thereby not affecting the end applications.

#### 3.1.1. Benefits using No-Response

Thus mapping the above scenario to the benefits mentioned in section 2 reveals that use of 'No-Response' will help in:

- \* Reduction in network clogging
- \* Reduction in server-side loading
- \* Help in achieving real-time requirements as the application is not bound by any delay due to closed loop latency

### 3.2. Multicasting traffic congestion information to PDAs/ smart-phones (Category 2)

The ITS might have an application which runs some analytics at the backend and determines the instantaneous traffic congestion spots in a city. The analytics is done based on the real-time geo-location updates received from the vehicles registered in the system. The backend application multicasts the instantaneous results of the analytics to the constrained handheld devices which registered to the city authority for real-time updates on congestion points. The backend is not really interested in the delivery status of these updates. In this case the backend uses No-Response option along with NON updates to reduce the traffic generated due to simultaneous status responses from hundreds of subscribed handheld devices.

#### 3.2.1. Using granular response suppression

However, an intelligent application may use the granularity feature of this option such that the responses are fed-back to the backend when updates to particular devices cause errors. So the updates may contain 'No-Response' option indicating that a response is to be suppressed only in success conditions and all responses in case of errors should be fed back. The server might eventually stop sending updates to the devices which responded with consecutive 'bad' responses. This will indirectly help saving network bandwidth.

#### 3.2.2. Benefits using No-Response

Thus mapping the above scenario to the benefits mentioned in section 2 reveals that use of 'No-Response' will help in:

- \* Reduction in network clogging
- \* Reduction in battery consumption at the constrained end-point
- \* Reduction in communication cost at the constrained end-point

## 4. Option Definition

The properties of this option are as in Table 1.

| Number | C | U | N | R | Name        | Format | Length | Default |
|--------|---|---|---|---|-------------|--------|--------|---------|
| TBD    |   | X | - |   | No-Response | uint   | 1      | 0       |

Table 1: Option Properties

This option is Elective and Non-Repeatable. If a proxy happens to encounter this option it should not forward. Hence caching is not applicable. The assumption here is that if an application needs a proxy in between an unconstrained backend and a constrained node then in most cases the leg between the constrained node and the proxy will be constrained in nature. So by restricting this option up to the proxy we can reap the benefits of this option in constrained environment without increasing overall complexity.

This option is presently intended for update requests (e.g., PUT) in NON mode and should have no effect if used with a CON request. This option contains values to indicate interest/ disinterest in all or a particular class or combination of classes of responses as described in the next sub-section.

The following table provides a 'ready-reckoner' on possible applicability of this option for all the four REST methods. This table is prepared in view of the type of application scenarios foreseen so far.

| Method Name | Remarks on applicability                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GET         | Not applicable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| PUT         | Applicable for frequent updates in NON mode on existing fixed resources. Might not be useful when PUT 'creates' a new resource.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| POST        | If POST is used just to update a target resource then No-Response can be used in the same manner as in NON-PUT. May also be applicable when POST performs resource creation and the client does not refer to the resource in future. For example, than updating a fixed resource, POST API may rather contain a query-string with name/value pairs for a defined action (ex. insertion into a database as part of frequent updates). The resources created this way may be 'short-lived' resources which the client will not refer to in future (see section 5.1.2.2). |
| DELETE      | Not applicable. Deletion is usually a permanent action and the client should make sure that the deletion actually happened.                                                                                                                                                                                                                                                                                                                                                                                                                                            |

Table 2: Applicability of No-Response for different methods

#### 4.1. Achieving granular suppression

This option is defined as a bit-map (Table 3) to achieve granular suppression.

| Value | Binary Representation | Description                                   |
|-------|-----------------------|-----------------------------------------------|
| 0     | 00000000              | Suppress all responses (same as empty value). |
| 2     | 00000010              | Allow 2.xx success responses.                 |
| 8     | 00001000              | Allow 4.xx client errors.                     |
| 16    | 00010000              | Allow 5.xx server errors.                     |

Table 3: Option values

XOR of the values defined for allowing particular classes will result in allowing a combination of classes of responses. So, a value of 18 (binary: 00010010) will result in allowing all 2.xx and 5.xx classes of responses. It is to be noted that a value of 26 will indicate that all types of responses are to be allowed (which is as good as not using No-Response at all).

Implementation Note: The use of No-Response option is very much driven by the application scenario and the characteristics of the information to be updated. Judicious use of this option benefits the overall system as explained in sections 2 and 3.

When No-Response is used with empty or 0 value, the updating end-point should cease the listening activity for response against the particular request. On the contrary, opening up at least one class of responses means that the updating end-point can no longer stop listening and must be configured to listen up to some application specific time-out period for the particular request. The updating end-point never knows whether the present update will be a success or a failure. Thus, if the client decides to open up the responses for errors (4.xx & 5.xx) then it has to wait for the entire time-out period even for the instances where the request is successful (and the server is not supposed to send back a response). This kind of situation may arise for the scenario in section 3.2.1. Under such circumstances the use of No-Response may not help improving the performance in terms of

overall latency. However, the advantages in terms of saving network and energy resources will still hold.

A point to be noted in view of the above example is that there may be situations when the response on errors might get lost. In such a situation the sender would wait up to the time-out period but will not receive any response. But this should not lead to the impression to the sender that the request was successful. The situation will worsen if the receiver is no longer active. The application designer needs to tackle such situation. Since this option is conceived for frequent updates, the sender may strategically insert requests without No-Response after N numbers of requests with No-Response (similar to the way [I-D.ietf-core-observe] 'weaves' CON notifications within series of NON notifications to check if the observer is alive).

## 5. Example

This section illustrates few examples of exchanges based on the scenario narrated in section 3.1. Examples for other scenarios can be easily conceived based on these illustrations.

### 5.1. Request/response Scenario

#### 5.1.1. Using No-Response with PUT

Figure 1 shows a typical request with this option. The depicted scenario occurs when the vehicle#n moves very fast and update rate is high. The vehicle is assigned a dedicated resource: vehicle-stat-<n>, where <n> can be any string uniquely identifying the vehicle. The update requests are in NON mode. The No-Response option causes the server not to reply with any status code.

| Client                                                            | Server                                                                                                                                                                                                                                             |
|-------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>+-----&gt; PUT</pre>                                         | <pre>Header: PUT (T=NON, Code=0.03, MID=0x7d38) Token: 0x53 Uri-Path: "vehicle-stat-00" Content Type: text/plain No-Response: 0 Payload: "VehID=00&amp;RouteID=DN47&amp;Lat=22.5658745&amp;Long=88.4107966667&amp; Time=2013-01-13T11:24:31"</pre> |
| <pre>[No response from the server. Next update in 20 secs.]</pre> |                                                                                                                                                                                                                                                    |
| <pre>+-----&gt; PUT</pre>                                         | <pre>Header: PUT (T=NON, Code=0.03, MID=0x7d39) Token: 0x54 Uri-Path: "vehicle-stat-00" Content Type: text/plain No-Response: 0 Payload: "VehID=00&amp;RouteID=DN47&amp;Lat=22.5649015&amp;Long=88.4103511667&amp; Time=2013-01-13T11:24:51"</pre> |

Figure 1: Exemplary unreliable update with No-Response option using PUT.

#### 5.1.2. Using No-Response with POST

POST "usually results in a new resource being created or the target resource being updated". Exemplary uses of 'No-Response' for both these 'usual' actions of POST are given below.

##### 5.1.2.1. POST updating a target resource

In this case POST acts the same way as PUT. The exchanges are same as above. The updated values are carried as payload of POST as shown in Figure 2.

| Client          | Server                                                                                                                                                                                                                                        |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| +-----><br>POST | Header: POST (T=NON, Code=0.02, MID=0x7d38)<br>Token: 0x53<br>Uri-Path: "vehicle-stat-00"<br>Content Type: text/plain<br>No-Response: 0<br>Payload:<br>"VehID=00&RouteID=DN47&Lat=22.5658745&Long=88.4107966667&<br>Time=2013-01-13T11:24:31" |
|                 | [No response from the server. Next update in 20 secs.]                                                                                                                                                                                        |
| +-----><br>POST | Header: PUT (T=NON, Code=0.02, MID=0x7d39)<br>Token: 0x54<br>Uri-Path: "vehicle-stat-00"<br>Content Type: text/plain<br>No-Response: 0<br>Payload:<br>"VehID=00&RouteID=DN47&Lat=22.5649015&Long=88.4103511667&<br>Time=2013-01-13T11:24:51"  |

Figure 2: Exemplary unreliable update with No-Response option using POST as the update-method.

#### 5.1.2.2. POST performing updates through resource creation

In most practical implementations the backend of section 3.1 will have a dedicated database to store the location updates. In such a case the client would send an update string as the POST URI which contains the name/value pairs for each update. Thus frequent updates may be performed through POST by creating such 'short-lived' resources which the client would not refer to in future. Hence 'No-Response' can be used in same manner as for updating fixed resources. The scenario is depicted in Figure 3.

| Client  | Server                                                 |
|---------|--------------------------------------------------------|
| +-----> | Header: POST (T=NON, Code=0.02, MID=0x7d38)            |
| POST    | Token: 0x53                                            |
|         | Uri-Path: "insertInfo"                                 |
|         | Uri-Query: "VehID=00"                                  |
|         | Uri-Query: "RouteID=DN47"                              |
|         | Uri-Query: "Lat=22.5658745"                            |
|         | Uri-Query: "Long=88.41079666667"                       |
|         | Uri-Query: "Time=2013-01-13T11:24:31"                  |
|         | No-Response: 0                                         |
|         | [No response from the server. Next update in 20 secs.] |
| +-----> | Header: POST (T=NON, Code=0.02, MID=0x7d39)            |
| POST    | Token: 0x54                                            |
|         | Uri-Path: "insertInfo"                                 |
|         | Uri-Query: "VehID=00"                                  |
|         | Uri-Query: "RouteID=DN47"                              |
|         | Uri-Query: "Lat=22.5649015"                            |
|         | Uri-Query: "Long=88.4103511667"                        |
|         | Uri-Query: "Time=2013-01-13T11:24:51"                  |
|         | No-Response: 0                                         |

Figure 3: Exemplary unreliable update with No-Response option using POST with a query-string to insert update information to backend database.

## 5.2. Resource-observe Scenario

This option should be treated transparently if used with NON notifications. In other words, just like GET and DELETE, this option will have no effect for observe notifications. The following example demonstrates how optimizations achieved using No-Response may also be achieved using resource-observe mode in certain situations at least in theory.

For example, the scenario of section 3.1 may also be achieved using resource-observe. In that case the backend will have to subscribe to each of the in-vehicle sensor gateway. The gateways will notify the backend with updated geo-locations. However, considering the huge number of vehicles moving around and several being added to the system quite often, this kind of arrangement may not be as

practicable and efficient solution as illustrated in the previous examples.

Figure 4 shows the resource observe variant. The No-Response option has been used intentionally both with GET and the notifications to illustrate the non-applicability of this option in this situation.

```

Server Client
| |
|<-----+ Header : GET (MID=0x5d28)
| GET | Token : 0x53
| | No-Response: 0
| | Uri-Path: vehicle-stat
| | Observe : (empty)
|
|+-----> Header: 2.05 (T=NON, MID=0x7d38)
| 2.05 | Token: 0x53
| | Content Type: text/plain
| | No-Response: 0
| | Payload:
| | "VehID=00&RouteID=DN47&Lat=22.5658745&Long=88.4107966667&
| | Time=2013-01-13T11:24:31"
|
| [Next update in 20 secs.]
|
|+-----> Header: 2.05 (T=NON, MID=0x7d39)
| 2.05 | Token: 0x53
| | Content Type: text/plain
| | No-Response: 0
| | Payload:
| | "VehID=00&&RouteID=DN47&Lat=22.5649015&Long=88.4103511667&
| | Time=2013-01-13T11:24:51"

```

Figure 4: Exemplary unreliable update in resource-observe mode with No-Response option where practically No-Response has no effect.

Note: The reason for keeping this example is to open up the choice to the user when there is a possibility of choosing between resource-observe with NON and updates with No-Response and to show a possible case where the latter option may sound more useful.

## 6. IANA Considerations

The IANA is requested to add the following option number entries:

| Number | Name        | Reference                  |
|--------|-------------|----------------------------|
| TBD    | No-Response | Section 4 of this document |

## 7. Security Considerations

The No-Response option defined in this document presents no security considerations beyond those in Section 11 of the base CoAP specification [I-D.ietf-core-coap].

## 8. Acknowledgments

Thanks to Carsten Bormann, Esko Dijk, Bert Greevenbosch, Akbar Rahman and Claus Hartke for their valuable inputs.

## 9. References

### 9.1. Normative References

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K. and Bormann, C., "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18, June 28, 2013

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-09, July 15, 2013

[I-D.ietf-core-groupcomm]

Rahman, A. and Dijk, E., "Group Communication for CoAP", draft-ietf-core-groupcomm-12, July 30, 2013

Authors' Addresses

Abhijan Bhattacharyya  
Tata Consultancy Services Ltd.  
Kolkata, India

Email: abhijan.bhattacharyya@tcs.com

Soma Bandyopadhyay  
Tata Consultancy Services Ltd.  
Kolkata, India

Email: soma.bandyopadhyay@tcs.com

Arpan Pal  
Tata Consultancy Services Ltd.  
Kolkata, India

Email: arpan.pal@tcs.com



CORE Working Group  
Internet Draft  
Intended status: Experimental

P. Urien  
Telecom ParisTech

August 2013

Expires: February 2014

Remote APDU Call Secure (RACS)  
draft-urien-core-racs-00.txt

## Abstract

This document describes the Remote APDU Call Protocol Secure (RACS) protocol, dedicated to Grid of Secure Elements (GoSE). These servers host Secure Elements (SE), i.e. tamper resistant chips offering secure storage and cryptographic resources.

Secure Elements are microcontrollers whose chip area is about 25mm<sup>2</sup>; they deliver trusted computing services in constrained environments.

RACS supports commands for GoSE inventory and data exchange with secure elements. It is designed according to the representational State Transfer (REST) architecture. RACS resources are identified by dedicated URIs. An HTTP interface is also supported.

## Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 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 <http://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 February 2014.

.

## Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|                                           |    |
|-------------------------------------------|----|
| Abstract.....                             | 1  |
| Requirements Language.....                | 1  |
| Status of this Memo.....                  | 1  |
| Copyright Notice.....                     | 2  |
| 1 Overview.....                           | 4  |
| 1.1 What is a Secure Element.....         | 4  |
| 1.2 Grid Of Secure Elements (GoSE).....   | 5  |
| 1.3 Secure Element Identifier (SEID)..... | 6  |
| 1.4 APDUs.....                            | 6  |
| 1.4.1 ISO7816 APDU request .....          | 6  |
| 1.4.2 ISO7816 APDU response .....         | 7  |
| 2 The RACS protocol.....                  | 7  |
| 2.1 Structure of RACS request.....        | 8  |
| 2.2 Structure of a RACS response.....     | 8  |
| 2.3 RACS commands.....                    | 9  |
| 2.3.1 BEGIN .....                         | 9  |
| 2.3.2 END .....                           | 9  |
| 2.3.3 GET-VERSION .....                   | 9  |
| 2.3.4 SET-VERSION .....                   | 10 |
| 2.3.5 LIST .....                          | 10 |
| 2.3.6 RESET .....                         | 11 |
| 2.3.7 APDU .....                          | 11 |
| 3 URI for the GoSE.....                   | 13 |
| 4 HTTP interface.....                     | 14 |
| 4.1 HTTPS Request.....                    | 14 |
| 4.2 HTTPS response.....                   | 14 |
| 5 Security Considerations.....            | 14 |
| 6 IANA Considerations.....                | 14 |
| 7 References.....                         | 15 |
| 7.1 Normative References.....             | 15 |
| 7.2 Informative References.....           | 15 |
| 8 Authors' Addresses.....                 | 15 |

## 1 Overview

This document describes the Remote APDU Call Protocol Secure (RACS) protocol, dedicated to Grid of Secure Elements (GoSE). These servers host Secure Elements (SE), i.e. tamper resistant chips offering secure storage and cryptographic resources.

Secure Elements are microcontrollers whose chip area is about 25mm<sup>2</sup>; they deliver trusted computing services in constrained environments.

RACS supports commands for GoSE inventory and data exchange with secure elements.

RACS is designed according to the representational State Transfer (REST) architecture [REST], which encompasses the following features:

- Client-Server architecture.
- Stateless interaction.
- Cache operation on the client side.
- Uniform interface.
- Layered system.
- Code On Demand.

### 1.1 What is a Secure Element

A Secure Element (SE) is a tamper resistant microcontroller equipped with host interfaces such as [ISO7816], SPI (Serial Peripheral Interface) or I2C (Inter Integrated Circuit).

The typical area size of these electronic chips is about 25mm<sup>2</sup>. They comprise CPU (8, 16, 32 bits), ROM (a few hundred KB), nonvolatile memory (EEPROM, FLASH, a few hundred KB) and RAM (a few ten KB). Security is enforced by multiple hardware and logical countermeasures.

According to the [EUROSMART] association height billion of such secure devices were shipped in 2013. Secure elements are widely deployed for electronic payment (EMV cards), telecommunication (SIM modules), identity (electronic passports), ticketing, and access control.

Most of secure elements include a Java Virtual Machine and therefore are able to execute embedded program written in the JAVACARD language. Because these devices are dedicated to security purposes they support numerous cryptographic resources such as digest functions (MD5, SHA1, SHA2...), symmetric cipher (3xDES, AES) or asymmetric procedures (RSA, ECC).

A set of Global Platform [GP] standards control the lifecycle of embedded software, i.e. application downloading, activation and deletion.

As an illustration a typical Secure Element has the following characteristics:

- JAVACARD operating system;
- Compliant with the GP (Global Platform) standards;
- 160 KB of ROM;
- 72 KB of EEPROM;
- 4KB of RAM;
- Embedded crypto-processor;
- 3xDES, AES, RSA, ECC;
- Certification according to Common Criteria (CC) EAL5+ level;
- Security Certificates from payment operators.

## 1.2 Grid Of Secure Elements (GoSE)

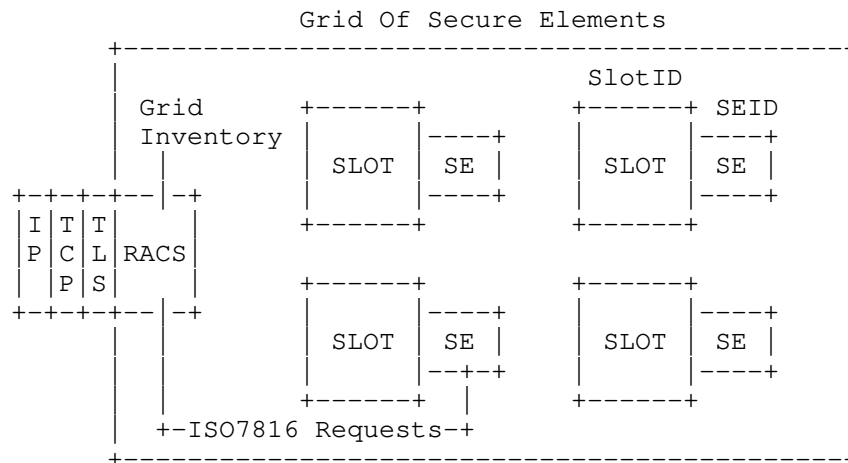


Figure 1. Architecture of a Grid of Secure Elements

A grid of Secure Elements (GoSE) is a server hosting a set of secure elements.

The goal of these platforms is to deliver trusted services over the Internet. These services are available in two functional planes,

- The user plane, which provides trusted computing and secure storage.
- The management plane, which manages the lifecycle (downloading, activation, deletion) of applications hosted by the Secure Element.

A grid of Secure Elements offers services similar to HSM (Hardware Secure Module), but may be managed by a plurality of administrators, dealing with specific secure microcontrollers.

According to this draft all accesses to a GoSE require the TCP transport and are secured by the TLS [TLS 1.0] [TLS 1.1] [TLS 2.0] protocol.

The RACS protocol provides all the features needed for the remote use of secure elements, i.e.

- Inventory of secure elements
- Information exchange with the secure elements

### 1.3 Secure Element Identifier (SEID)

Every secure element needs a physical slot that provides power feeding and communication resources. This electrical interface is for example realized by a socket soldered on an electronic board, or a CAD (Card Acceptance Device, i.e. a reader) supporting host buses such as USB.

Within a GoSE each slot is identified by a SlotID (slot identifier) attribute, which may be a socket number or a CAD name.

The SEID (Secure Element Identifier) is a unique identifier indicating that a given SE is hosted by a GoSE. It also implicitly refers the physical slot (SlotID) to which the SE is plugged.

The GoSE manages an internal table that establishes the relationship between SlotIDs and SEIDs.

Therefore three parameters are needed for remote communication with secure element, the IP address of the GoSE, the associated TCP port, and the SEID.

### 1.4 APDUs

According to the [ISO7816] standards secure element process ISO7816 request messages and return ISO7816 response messages, named APDUs (application protocol data unit).

#### 1.4.1 ISO7816 APDU request

An APDU request comprises two parts: a header and an optional body.

The header is a set of four or five bytes noted CLA INS P1 P2 P3

- CLA indicates the class of the request, and is usually bound to standardization committee (00 for example means ISO request).
- INS indicates the type of request, for example B0 for reading or D0 for writing.
- P1 P2 gives additional information for the request (such index in a file or identifier of cryptographic procedures)
- P3 indicates the length of the request body (from P3=01 to P3=FF), or the size of the expected response body (a null value meaning 256 bytes). Short ISO7816 requests may comprise only 4 bytes

- The body may be empty. Its maximum size is 255 bytes

#### 1.4.2 ISO7816 APDU response

An APDU response comprises two parts an optional body and a mandatory status word.

- The optional body is made of 256 bytes at the most.
- The response ends by a two byte status noted SW. SW1 refers the most significant byte and SW2 the less significant byte.

An error free operation is usually associated to the 9000 status word. Following are some interpretations of the tuple SW1, SW2 according to various standards:

- '9F' 'xx', indicates that xx bytes (modulus 256) are ready for reading. Operation result MUST be fetched by the ISO Get Response APDU (CLA = 'C0', P1=P2= '0', P3= 'XX')
- '67' 'XX', incorrect parameter P3
- '6B' 'XX', incorrect parameter P1 or P2
- '6D' 'XX', unknown instruction code (INS) given in the request
- '6E' 'XX', wrong instruction class (CLA) given in the request
- '6F' 'XX', technical problem, not implemented...
- '61' 'XX', operation result MUST be fetched by the ISO Get Response APDU (CLA = 'C0', P1=P2= '0', P3= 'XX')
- '6C' 'XX', operation must be performed again, with the LE parameter value sets to 'XX'.

## 2 The RACS protocol

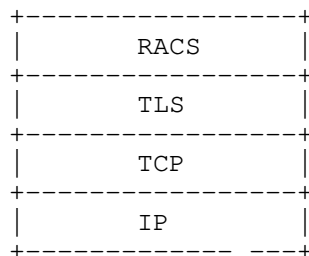


Figure 2. The RACS stack

The RACS protocol works over the TCP transport layer and is secured by the TLS protocol. The TLS client (i.e. the RACS client) MUST be authenticated by a certificate.

One of the main targets of the RACS protocol is to efficiently push a set of ISO7816 requests towards a secure element in order to perform cryptographic operations in the user's plane. In that case a

RACS request typically comprises a prefix made with multiple ISO7816 requests and a suffix that collects the result of a cryptographic procedure.

The use of TLS with mutual authentication based on certificate provides a simple and elegant way to establish the credentials of a RACS client over the GoSE. It also enables an easy splitting between users' and administrators' privileges.

## 2.1 Structure of RACS request

A RACS request is a set of command lines, encoded according to the ASCII format. Each line ends by the CR (carriage Return) and line feed (LF) characters.

Each command is a set of tokens (i.e. words) separated by the space (0x20) character(s).

The first token of each line is the command to be executed.

A command line MAY comprise other tokens, which are called the command parameters.

A RACS request always starts by a BEGIN command and ends by an END command.

The processing of a RACS request is halted after the first error. In that case the returned response contained the error status induced by the last executed command.

## 2.2 Structure of a RACS response

A RACS response is a line, encoded according to the ASCII format, which ends by the CR (carriage Return) and line feed (LF) characters.

The first token is the response status. The first character of the status is either '+' in case of success or '-' if an error occurred during the RACS request execution. It is followed by an ASCII encoded integer, which is the value of the status.

A response line MAY comprise other tokens, which are called the response parameters.

Examples of RACS responses:

```
+000 Success
-300 Error at line 2
-400 Unknown command at line 2
```

-500 Conditions not satisfied at line 2  
-600 Timeout occurred at line 2

## 2.3 RACS commands

### 2.3.1 BEGIN

This command starts a request message. A response message is returned if an error is detected.

### 2.3.2 END

This command ends a request message. It returns the response message triggered by the last command.

#### Example1

=====

Request:

BEGIN CR LF

END CR LF

Response:

+000 Success CR LF

#### Example2

=====

Request:

BEGIN CR LF

APDU ASTERIX-CRYPTO-MODULE [ISO7816-Request] CR LF

END

Response:

+000 [ISO7816-Response] CR LF

### 2.3.3 GET-VERSION

This command requests the current version of the RACS protocol. The returned response is the current version encoded by two integer separated by the '.' character. The first integer indicates the major version and the second integer gives the minor version.

#### Example

=====

Request:

BEGIN CR LF

GET-VERSION CR LF

END

Response:

+000 1.0 CR LF

#### 2.3.4 SET-VERSION

This command sets the version to be used for the RACS request. An error status is returned by the response if an error occurred.

##### Example 1

=====

##### Request:

```
BEGIN CR LF
SET-VERSION 2.0 CR LF
END CR LF
```

##### Response:

```
-400 Error line 2 RACS 2.0 is not supported
```

##### Example 2

=====

##### Request:

```
BEGIN CR LF
SET-VERSION 1.0 CR LF
END CR LF
```

##### Response:

```
+000 RACS 1.0 has been activated CR LF
```

#### 2.3.5 LIST

This command requests the list of SEID plugged in the GoSE.

It returns a list of SEIDs separated by space (0x20) character(s)

Some SEID attributes could be built from a prefix and an integer suffix (such as SE#100 in which SE# is the suffix and 100 is the integer suffix). A list of non-consecutive SEID could be encoded as prefix[i1;i2;...;ip] where i1,i2,ip indicates the integer suffix. A list of consecutive SEID could be encoded as prefix[i1-ip] where i1,i2,ip indicates the integer suffix.

##### Example 1

=====

##### Request:

```
BEGIN CR LF
LIST CR LF
END CR LF
```

##### Response:

```
+000 SEID1 SEID2 CR LF
```

## Example 2

=====

## Request:

BEGIN CR LF

LIST CR LF

END CR LF

## Response:

+000 device[1000-2000] serialnumber[567;789;243] CR LF

## 2.3.6 RESET

This command resets a secure element. The first parameter gives the secure element identifier (SEID). An optional second parameter specifies a warm reset. The default behavior is a cold reset. The response status indicates the success or the failure of the operation.

Syntax: RESET SEID [WARM] CR LF

## Example 1

=====

## Request:

BEGIN CR LF

RESET device#45 CR LF

END CR LF

## Response:

+000 device#45 Reset Done

## Example 2

=====

## Request:

BEGIN CR LF

RESET device#45 WARM CR LF

END CR LF

## Response:

+000 device#45 Warm Reset Done

## 2.3.7 APDU

This command sends an ISO7816 request to a secure element or a set of ISO7816 commands.

The first parameter specified the SEID

The second parameter is an ISO7816 request

Three optional parameters are available; they MUST be located after the second parameter.

- CONTINUE=value, indicates that the next RACS command will be executed only if the ISO7816 status word (SW) is equal to a given value. Otherwise an error status is returned.
- MORE=value, indicates that a FETCH ISO7816 request will be performed (i.e. a new ISO7816 request will be sent) if the first byte of the ISO7816 status word (SW1) is equal to a given value.
- FETCH=value fixes the four bytes of the ISO7816 FETCH request (i.e. CLA INS P1 P2). The default value is 00C00000 (CLA=00, INS=C0, P1=00, P2=00)

When the options CONTINUE and MORE are simultaneously set the SW1 byte is first checked. If there is no match then the SW word is afterwards checked.

#### SYNTAX

APDU SEID ISO7816-REQUEST [CONTINUE=SW] [MORE=SW1] [FETCH=CMD]

The returned response is the ISO7816 response. If multiple ISO7816 requests are executed (due to the MORE option), the bodies are concatenated in the response, which ends by the last ISO7816 status word.

#### Example 1

=====

Request:

```
BEGIN CR LF
APDU SEID ISO7816-REQUEST CR LF
END CR LF
```

Response:

```
+000 ISO7816-RESPONSE CR LF
```

#### Example 2

=====

```
BEGIN CR LF
APDU SEID ISO7816-REQUEST-1 CONTINUE=9000 CR LF
APDU SEID ISO7816-REQUEST-2 CR LF
END CR LF
```

Response:

```
+000 ISO7816-RESPONSE-2 CR LF
```

#### Example 2

=====

```
BEGIN CR LF
APDU SEID ISO7816-REQUEST-1 CONTINUE=9000 CR LF
APDU SEID ISO7816-REQUEST-2 CR LF
END CR LF
```

Response:

-300 Request Error line 1 wrong SW CR LF

Example 3

=====

BEGIN CR LF

APDU SEID ISO7816-REQ-1 CONTINUE=9000 CR LF

APDU SEID ISO7816-REQ-2 CONTINUE=9000 CR LF

APDU SEID ISO7816-REQ-3 CONTINUE=9000 MORE=61 FETCH=00C00000 CR LF

END CR LF

Response:

+000 ISO7816-RESP-3 CR LF

Multiple ISO7816 requests have been performed by the third APDU command according to the following scenario

- the ISO7816-REQ-3 request has been forwarded to the secure element (SEID)

- the ISO 7816 response comprises a body (body0) and a status word (SW0) whose first byte is 0x61

- the FETCH command CLA=00, INS=00, P1=00, P2=00, P3=SW2 is sent to the secure element

- the ISO 7816 response comprises a body (body1) and a status word (SW1) set to 9000

-

The RACS response is set to

+000 body0 || body1 || SW1 CR LF

where || indicates a concatenation operation.

### 3 URI for the GoSE

The URI addressing the resources hosted by the GoSE is represented by the string:

RACS://GoSE-Name:port/?request

where request is the RACS request to be forwarded to a the GoSE

RACS command lines are encoded in a way similar to the INPUT field of an HTML form. Each command is associated to an INPUT name, the remaining of the command line i.e. a set of ASCII characters, is written according to the URL encoding rules. The BEGIN and END commands are omitted; end of line characters, i.e. carriage return (CR) and line feed (LF) are also omitted.

As a consequence request is written to the following syntax

cmd1=cmd1-parameters&cmd2=cmd2-parameters

## 4 HTTP interface

A GoSE SHOULD support an HTTP interface. RACS requests/responses are transported by HTTP messages. The use of TLS is mandatory.

### 4.1 HTTPS Request

`https://GoSE-Name/RACS?request`

where request is the RACS request to be forwarded to a secure element (SEID)

The RACS request is associated to an HTML form whose name is "RACS". The request command lines are encoded as the INPUT field of an HTML form. Each command is associated to an INPUT name, the remaining of the command line i.e. a set of ASCII characters is written according to the URL encoding rules. The BEGIN and END commands are omitted; end of line characters, i.e. carriage return (CR) and line feed (LF) are also omitted.

As a consequence a RACS request is written as

`https://GoSE-Name/RACS?cmd1=cmd1-parameters&cmd2=cmd2-parameters`

### 4.2 HTTPS response

The RACS response is returned in an XML document

The root element of the document is `<RACS-Response>`

The status of the response is content of the `<status>` element

The parameters of the response are the content of the `<parameters>` element

End of line, i.e. carriage return (CR) and line feed (LF) characters are omitted.

As a consequence a RACS response is written as

`<RACS-Response>`

`<status>+000</status>`

`<parameters>parameters of the RACS response</parameters>`

`</RACS-Response>`

## 5 Security Considerations

To be done.

## 6 IANA Considerations

## 7 References

### 7.1 Normative References

[TLS 1.0] Dierks, T., C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999

[TLS 1.1] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006

[TLS 1.2] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", draft-ietf-tls-rfc4346-bis-10.txt, March 2008

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO)

### 7.2 Informative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000,  
<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

[GP] Global Platform Standards, <http://www.globalplatform.org>

[EUROSMART] The EUROSMART association, <http://www.eurosmart.com>

## 8 Authors' Addresses

Pascal Urien  
Telecom ParisTech  
23 avenue d'Italie  
75013 Paris  
France

Phone: NA  
Email: [Pascal.Urien@telecom-paristech.fr](mailto:Pascal.Urien@telecom-paristech.fr)

Core Working Group  
Internet Draft  
Intended status: Informational  
Expires: March 29, 2014

J. Zhu  
M. Qi  
China Mobile  
Sep 29, 2013

Group Authentication  
draft-zhu-core-groupauth-01

Abstract

The group communication is designed for the communication of Internet of Things. A threat is identified in [I-D.ietf-core-groupcomm] that current DTLS based approach is unicast oriented and there is no supporting on group authentication feature. Unicast oriented authentication will causing serious burden when a large number of terminal nodes will be involved inevitably. In another aspect, some terminals will own the same characteristics, such as owning same features, in the same place, working in the same time, etc. With this mechanism, all terminals can be authenticated together with little signaling and calculation at the same time. It will reduce the network burden and save time. This draft describes the security of group authentication and an group authentication implementation method for the Internet of things.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on March 29, 2014.

#### Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the  
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions  
Relating to IETF Documents (<http://trustee.ietf.org/license-info>)  
in effect on the date of publication of this document. Please  
review these documents carefully, as they describe your rights  
and restrictions with respect to this document.

#### Table of Contents

|                                                |    |
|------------------------------------------------|----|
| 1. Introduction .....                          | 2  |
| 2. Conventions used in this document .....     | 3  |
| 2.1. Definitions.....                          | 3  |
| 3. Problem Statement .....                     | 4  |
| 3.1. Use cases .....                           | 4  |
| 3.2. Problem statement .....                   | 5  |
| 4. Requirement .....                           | 6  |
| 5. Group Authentication Solution .....         | 7  |
| 5.1. Introduction .....                        | 7  |
| 5.2. Detailed group scenario description ..... | 7  |
| 5.3. Group scenario procedure .....            | 9  |
| 6. Security Considerations .....               | 10 |
| 7. IANA Considerations .....                   | 11 |
| 8. Conclusions .....                           | 11 |
| 9. Acknowledgement.....                        | 11 |
| 10. References .....                           | 11 |
| 10.1. Normative References .....               | 11 |
| 10.2. Informative References .....             | 11 |

#### 1. Introduction

With the development of Internet of Things, a large number of

terminal nodes will be involved inevitably. The unicast authentication communication from big amount terminals will merge together in the network, and causing serious burden to the server. Although IP multicast technical is introduced for group communication in [I-D.ietf-core-groupcomm], IP multicast relies on the unicast authentication at initial stage. In another aspect, some terminals will own the same characteristics, such as owning same features, in the same place, working in the same time, etc. With this mechanism, all terminals can be authenticated with little signaling and calculation at the same time. It will reduce the network burden and save time.

This draft describes the security of group authentication and an group authentication implementation method for the Internet of things.

## 2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

### 2.1. Definitions

**Terminals:** It is such a constrained device also recognized as group node in this document which communicates with server, through direct or indirect connections, which can be seen by the server. If some constrained devices like Zigbee node only communicates with sink node and it can't be seen by the server, such constrained device is not recognized as terminal.

**Group agent:** It is a device on behalf of group nodes (terminals) to make mutual authentication with network server.

**Network server:** It is the core network server which is responsible for authenticating the legality of terminal and provides specific services for them.

### 3. Problem Statement

#### 3.1. Use cases

Nowadays the normal authentication mechanism in network is a traditional unicast authentication method between a single terminal and a single network entity. The authentication mechanism will be finished based one 1-2 round of challenge-response conversation separately. If there are many terminals, the cost for authentication will be increased.

Now for some M2M service, it may be a large amount of terminal used for an M2M service. These terminals are placed in the same location, will be used for the same purpose, and own same behavior. These terminals can be worked together as a group. In these scenarios, the existing authentication mechanism is no longer appropriate. When a large number of terminals want to access network server, huge number of authentication signaling will be generated by the unicast authentication method. What is more, it will cause network congestion and lead to DoS attack. For some terminal devices which is restricted with limited computing capability and power, the traditional unicast authentication will increase the computational burden of these terminals and drain their poor battery.

The following use cases are identified at this point:

**Smart Metering:** A large amount of smart power meter terminals are deployed in a block. The smart meter uploads meter report frequently through the network to smart meter server. What is more, smart meter server queries all terminals periodically to check whether the terminal is workable or not. So smart meters report at the same time, or the smart meter server need to re-configure all smart meters at the same time. In fact, there are other type of smart metering which has no agent node. This will lead to overload since each apartment needs to equip with a meter and they access network parallel at the same time and it will not be considered in this draft.

**Remote Vehicle Management:** IOT terminals contains GPS location reporter, remote air condition control, etc. would be installed in some special Vehicles like Taxi. It will send information such as position information, navigation, remote diagnosis, on-board communication, news and

entertainment information etc. to the network server in order to make better vehicle scheduling, vehicle monitoring and vehicle controlling. So it needs to connect to the network and make authentication at first. However, such vehicles would gather in a small place like airport, train station, etc. The frequency of connection from these terminals to server will cause overloading.

Intelligent home: various sensors equipped with communication modules are deployed in a house to monitor house conditions and make a control when necessary. These sensors collect and report house related information like the status of door open/close, indoor temperature to its owner through a network, and take actions by following the regulating instructions send by the owner.

### 3.2. Problem statement

In the current smart metering service use cases, a large amount of smart power meter terminals are deployed in a block. The smart meter uploads meter report frequently through the network to smart meter server. What is more, smart meter server queries all terminals periodically to check whether the terminal is workable or not. Therefore, the meter requires frequent and network communication.

In such use cases, when all the meters access network parallel at the same time, or when the server sends message to all meters, the terminals will connect to the network in a short time period (1sec ~ 1min). Assume there are 19 buildings in the block, and each building has 25 floors on average with 10 apartments in each floor. If each apartment is equipped with 1 smart power meter, then 4760 meters will be deployed in total in the block. This will cause pressure to the network.

So an agent node has been introduced to aggregate the message from these meters and then send out these meters data to the server together. After the agent is introduced, the connection between meters and servers is split into two parts: one is the connection between meters, the other is and the one between agent and server. Usually the agent is responsible for the authentication of the meters.

The server is responsible for the authentication of the agent only and gets all information about meters such as ID, data, from agent.

The current security mechanism is:

1. Each meter is authenticated with the agent. Agent will authenticate the meter one by one. After that, agent should make mutual authentication with server. Then server can confirm agent identity.
2. Meter will set up security connection with agent, and agent will also set up security connection with server. When a meter wants to send data to server. It should send the data confidential protected to agent first. Agent will decrypt the data and transfer it to server by using the security protection mechanism between agent and server.

However, this procedure has the following security problems:

1. Since all meters are authenticated by the agent and no direct authentication from server to meter. The server can get meter's ID and data only through agent. So the agent Due to the key position in the authentication, the security protection about agent is very important. Server could not authenticate meters directly. It can only rely on the agent. However, the agent would be placed in un-secure place or owned by different user rather than the server owner. If the agent is compromised or lay to server, agent can act as a middle attacker that makes fake authentication to meters and report fake ID to servers.
2. Another security problem is related with agent and server. Under this scenario, all information from meters will be transferred through agent. So agent will know all information generated by meters. However, under some scenario, agent would be owned and used by different user other than the meters' and servers' owner. So under this assumption, the agent should not get the message from meter to server. So meters should set up an secure end-to-end tunnel with server. It should request another authentication and key generation procedure in addition to authenticate with agent. This will bring complexity and overhead to the system.

#### 4. Requirement

In order to reduce the cost and simplify a lot of overhead with the same characteristics of these groups of meter or sensor node group-based operations, it is needed to provide group authentication. For example, when smart meters perform bulk configuration information updates, it is needed to ensure that the correct identity of the user node within the group, to prevent the configuration information is wrong node receives. In addition, when smart meters report meter readings to the electricity system platform, it is also needed to be able to prove the correctness of the identity of smart meters, to prevent malicious node reporting false readings.

## 5. Group Authentication Solution

### 5.1. Introduction

Group authentication is a kind of authentication technologies that a group of users or terminals can be authenticated together at the same time. Instead of authenticating a number of terminals of a group one by one, group authentication mechanism treats these terminals in the group as a whole, and authenticates them together. Each group has a unique identifier, and an agent, which can be called as group agent, group gateway, etc.

Group authentication comprises following two phases as following:

1. The first phase is that user/terminal should be authenticated whether it belongs to a given group. This can be implemented through the proprietary authentication technology in a group, such as Zigbee or any others.
2. The second phase is that mutual authentication should be made between a given network entity, and a group agent who is responsible to delegate all terminals in the group.

After the authentication, terminals and network entity can generate separated session keys individually if there is some demand to make individual communication between network entity and each terminal.

### 5.2. Detailed group scenario description

For group authentication, there is detailed network description as following. There are 5 nodes inside a given group. They are A1, A2, A3, A4, and A5 which is group agent. And the given group can be named as group A. All nodes in group A can communicate with each other. What is more, A5 is able to communicate with network entity directly. Network entity will store the group information, such as identifiers, root keys used for all nodes inside the group. Network entity is also responsible for generating group authentication vector. The scenario is shown as below.

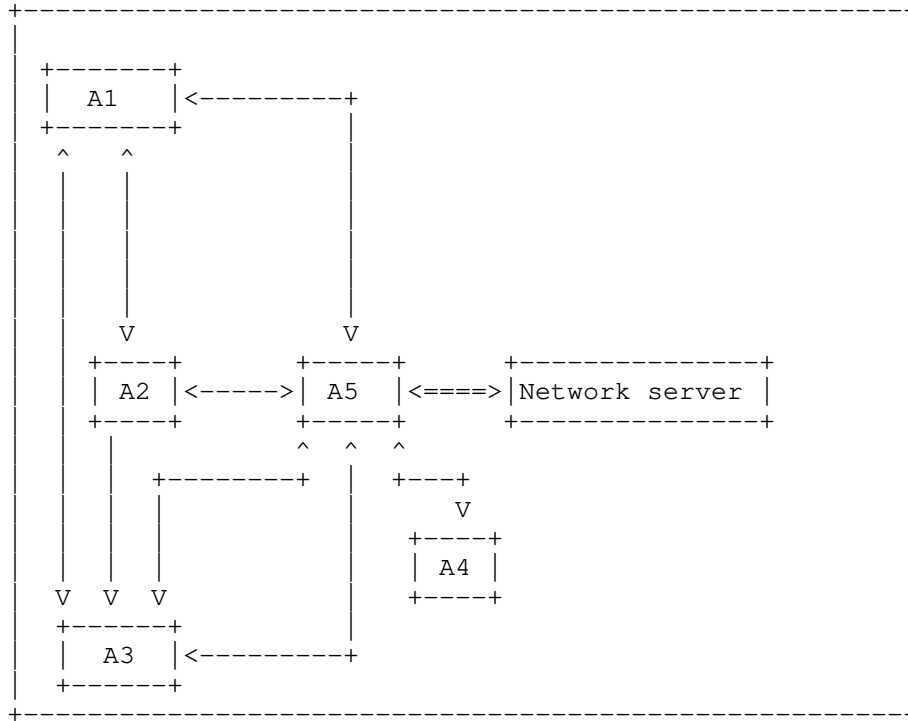


Figure 1 Group Authentication Architecture

- o A5 (group agent) communicates with other nodes, i.e. A1, A2, A3, A4 by inner group protocol. All nodes should contain such models as inner group communication model, group authentication mode. Inner group communication model can be used to sending/receiving

the group authentication message. Group authentication model can be used to generate authentication vectors/response and to authenticate peers.

- o Group agent will make mutual authentication with network entities. There are two kinds of network entities. Network server is responsible for mutual authentication action with group agent. And Network Server is responsible for group authentication vector generation and forwarding AV to network server. After the authentication, terminals and network entity can generate separated session keys individually if there is some demand to make individual communication between network entity and each terminals.

- o Group agent who represents the whole group, communicates with network entity, and generate group session key through authentication with the network server.

- o Pre-configure of the group

All the group nodes should be configured with sub key  $k_1$ ,  $k_2$ ,  $k_3$ ,  $k_4$ ,  $k_g$ , which will be used for mutual authentication in the group and separated communication.

### 5.3. Group scenario procedure

As mentioned above, group authentication can be divided into two phases.

In the first phase, group member, say  $A_i$ , sends authentication request to group agent at first as following.

1. Group member  $A_i$  sends message to trigger authentication at first.
2. Group agent sends authentication request to each group member.
3. Group member  $A_i$  verifies group agent at first. If success,  $A_i$  will generate session key for the communication with group agent, and sends response containing such session key back to group agent. If not success, the authentication is failed and group authentication procedure will be abort.
4. Group agent authenticates each group member  $A_i$  through the response message and record the authentication result in a mapping

table.

After the inner group authentication, all of group members are authenticated by group agent, and second phase can be performed.

5. Group agent sends message to network server to trigger the authentication outside the group.
6. Meanwhile, group agent sends authentication vector request to network server with group agent identity.
7. Network Server will generate authentication vector according to group agent identity.
8. What is more, network server should be able to recognize that is a group authentication is performed based on group agent identity. Network Server will generate session key for each group members by using pre-configured group member information and the same keying material in above step.
9. Network Server will send such authentication vector and session keys together back to network server.
10. Network server will perform mutual authentication with group agent.
11. Group agent authenticates group agent and send authentication response back to network server.
12. Network server authenticates group agent. If success, it can be considered that group agent and all group terminals is authenticated successfully.
13. Group agent will communicate with network server to choose the confidential and integrity protection algorithms.
14. After that, group agent will send keying material, selected algorithms to each group member.
15. Group member will generate session keys.

After these two phases, each terminal is authenticated with network server and generate independently session key with network server.

## 6. Security Considerations

TBD

## 7. IANA Considerations

There are no IANA considerations associated to this memo.

## 8. Conclusions

This memo describes the problem raised by using one-to-one authentication for huge number of Internet of Things terminals.

After that, group authentication requirement is raised and a group authentication mechanism is proposed. By using the proposed group authentication mechanism, the exploited group agent can behalf of all involved group nodes to make mutual authentication with network server, but the agent can not realize the content transmitted between both of them since it is just a intermediate node to forward messages which are encrypted by specific session key between the group node and network server. The trust relationship is between group nodes and network server. After authentication, the trust relationship between group nodes and group agent are not needed.

## 9. Acknowledgement

Thanks very much to Bert Greevenbosch, Stefanie Gerdes, Kepeng Li for their helpful comments and significant suggestions to revise this document.

## 10. References

### 10.1. Normative References

### 10.2. Informative References

## Authors' Addresses

Judy Zhu  
China Mobile  
Unit 2, 32 Xuanwumenxi Ave,  
Xicheng District,  
Beijing 100053, China  
Email: zhuhongru@chinamobile.com

Minpeng Qi  
China Mobile  
Unit 2, 32 Xuanwumenxi Ave,  
Xicheng District,  
Beijing 100053, China  
Email: qiminpeng@chinamobile.com