

CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: January 5, 2015

C. Bormann  
Universitaet Bremen TZI  
July 04, 2014

A TCP transport for CoAP  
draft-bormann-core-coap-tcp-01

Abstract

CoAP (RFC 7252) is defined to be transported over datagram transports such as UDP or DTLS. For a number of applications, it may be useful to channel CoAP messages in a TCP connection. This draft discusses different ways to do that.

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 5, 2015.

Copyright Notice

Copyright (c) 2014 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
1.1. Objectives . . . . .	2
1.2. Terminology . . . . .	2
2. Framing . . . . .	3
2.1. Length prefix . . . . .	3
2.2. Delimiter-based . . . . .	3
2.3. Self-delimiting . . . . .	4
3. Changes to CoAP . . . . .	5
3.1. One Message Type Only . . . . .	5
3.2. Token . . . . .	5
3.3. Message-ID, Fixed Header Format . . . . .	5
3.4. Rejecting messages . . . . .	5
3.5. Resilient variant . . . . .	6
3.6. Signatures . . . . .	6
4. Transport selection . . . . .	6
5. References . . . . .	6
5.1. Normative References . . . . .	6
5.2. Informative References . . . . .	6
Author's Address . . . . .	7

## 1. Introduction

(See Abstract)

The primary use case addressed by this specification is:

- o Aggregation of CoAP streams behind proxies, e.g.:
  - \* Behind a DTLS terminator/load balancer on the cloud side
  - \* As a wide-area interface to a proxy that speaks CoAP over UDP on the constrained side

## 1.1. Objectives

(TBD)

## 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 [RFC2119].

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.

## 2. Framing

The TCP stream needs to be structured into frames in order to delimit CoAP messages.

As the size of CoAP messages is limited, there is no need to split a single CoAP message into multiple frames (no interleaving).

Several alternative frame formats are possible. The current version of this specifications proposes several alternatives, with the understanding that a single one of these is likely to be chosen.

One desirable characteristic of a framing scheme is detection of premature termination of the TCP connection. While TCP in principle distinguishes orderly (FIN) and destructive (RST) termination of a connection, the difference is not always visible from the socket interface; also, a crashing process gives the impression of orderly termination. All schemes proposed here provide this detection.

### 2.1. Length prefix

A popular form of framing for TCP starts each frame with a length indication [RFC1006].

A simple form of length prefix would be an SDNV [RFC6256], which is efficient for large numbers of mostly small (< 128 B) messages. Alternatively, a two-byte prefix could always be used, or the length could be embedded in the CoAP message by using the unused Message-ID field.

The main disadvantage of a length prefix is that the sender needs to know the length before sending the message proper. The main advantage of a length prefix is that the receiver knows the length at the start of receiving the message.

### 2.2. Delimiter-based

Another form of message delimiting uses special byte values for delimiting protocol elements, e.g. CRLF for lines in a text stream. Since CoAP requires full data transparency, introducing a delimiter byte requires escaping occurrences of the delimiter in the data stream, which in turn requires escaping the escape mechanism. In traditional byte-stuffing (called "octet-stuffing" in [RFC1662]), the overhead of this escaping can be up to 100 % on top of the actual data. Cheshire has shown how to combine delimiter-based and length

prefix based encoding in "Consistent Overhead Byte Stuffing" [COBS]; however, this requires at least two bytes per message, achieving full efficiency only for relatively large messages and only if the length of the remaining message is known (see Section 2.1 before). A scheme such as the FSE scheme in [RFC2687] might be simpler to implement (the efficiency of an FSE-style scheme can be quite high by exploiting the fact that CoAP frames never start with a value below 0x40). A good value for FSE-style (or even a non-zero COBS-style) delimiter can be determined by examining a corpus of CoAP messages (TBD).

A major advantage of a COBS-like scheme would be compatibility with schemes that synchronize TCP packet boundaries with message boundaries [MINION].

Requiring a delimiter at the `_end_` of a frame fulfills the requirement for detection of premature TCP connection termination, except for an FSE-style scheme where the FSE starting an escape sequence happens to fall on a packet boundary.

### 2.3. Self-delimiting

Currently, CoAP messages are not self-delimiting, as the payload delimiter is optional and does not contain a payload length.

In the scheme proposed here, the payload delimiter is made required; the payload length is then encoded exactly as in CoAP options. For example:

- o 0xF0 would indicate that a zero-length (absent) payload follows
- o 0xF1 would indicate a single-byte payload
- o 0xFD 0x47 would indicate a 84-byte payload
- o 0xFE 0xFF 0xFF would indicate a 65804-byte payload

One advantage of implementing this scheme is that it could also be used to aggregate multiple CoAP messages into one datagram of a datagram-based transport such as UDP or DTLS, if that is desired, without increasing the overhead for unaggregated messages. For this application, 0xFF could still be used in order to efficiently encode "payload delimited by message boundary" in the final CoAP message in the datagram.

### 3. Changes to CoAP

The content of this section is expressed as a delta on [RFC7252].

#### 3.1. One Message Type Only

As reliability is handled by TCP, there is no need for ACK messages. Similarly, rebooting nodes will drop their TCP connections, so there is no need for RST messages (but see Section 3.4).

Cf. [I-D.savolainen-core-coap-websockets].

There may still be a desire to differentiate CON and NON for the intention behind a TCP-to-UDP proxy. In contrast to [I-D.savolainen-core-coap-websockets], this specification proposes to retain the difference between CON and NON messages as a hint for the reliability requirements placed on a message forwarded through a proxy. There are no ACK or RST messages; ACK messages MUST be encoded as CON messages.

#### 3.2. Token

The Token space is local to the TCP connection. In particular, this means that closing down a TCP connection cancels all outstanding requests (the responses are not sent over a new connection, which is handled like a new endpoint).

#### 3.3. Message-ID, Fixed Header Format

As there are no ACKs, there is no need to correlate an ACK to a CON. As a result, there is no need to carry a Message-ID for this. There is also no danger of duplication of a message, so the Message-ID is entirely without function.

If it seems desirable to maintain the frame format, the message-ID could still be sent empty. Alternatively, it could be used as a space for the frame length.

As does [I-D.savolainen-core-coap-websockets], this specification proposes to elide the Message-ID, i.e. to send bytes 0 and 1 of the CoAP message followed directly by byte 4 and following.

#### 3.4. Rejecting messages

[I-D.ietf-core-observe] now supports Observation Cancellation, reducing the need to support Reset-like messages for cancelling an observation relationship.

### 3.5. Resilient variant

An alternative approach is to treat the TCP connection as ephemeral. If a connection can go away at any point in time, and be replaced by a new one, the delivery of messages is no longer fully reliable. All functions of Message-IDs remain, as well as the functions of ACKs. Tokens retain their meaning beyond a connection.

### 3.6. Signatures

A new TCP connection can send an identifying signature in both directions to facilitate debugging and protocol evolution and to enable detection of mismatches.

E.g., the side opening a connection could send the seven bytes "CoAP1\r\n" and the answering side similarly "cOap1\r\n".

## 4. Transport selection

There may be use cases where the TCP transport should be explicitly selected from a URI. This problem should be solved in a way that doesn't cause the available of different transports to generate aliases for the same resource, i.e. the same "coap://" URI should be used for the same resource. Cf.

[I-D.silverajan-core-coap-alternative-transports].

## 5. References

### 5.1. Normative References

- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

### 5.2. Informative References

- [COBS] Cheshire, S. and M. Baker, "Consistent Overhead Byte Stuffing", ToN Vol.7, No. 2, April 1999.

- [I-D.savolainen-core-coap-websockets]  
Savolainen, T., Hartke, K., and B. Silverajan, "CoAP over WebSockets", draft-savolainen-core-coap-websockets-02 (work in progress), April 2014.
- [I-D.silverajan-core-coap-alternative-transport]  
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transport-05 (work in progress), June 2014.
- [MINION] Iyengar, J., Ford, B., Amin, S., Nowlan, M., and N. Tiwari, "Wire-Compatible Unordered Delivery in TCP and TLS", CoRR abs/1103.0463, February 2011, <<http://arxiv.org/abs/1103.0463v2>>.
- [RFC1006] Rose, M. and D. Cass, "ISO transport services on top of the TCP: Version 3", STD 35, RFC 1006, May 1987.
- [RFC1662] Simpson, W., "PPP in HDLC-like Framing", STD 51, RFC 1662, July 1994.
- [RFC2687] Bormann, C., "PPP in a Real-time Oriented HDLC-like Framing", RFC 2687, September 1999.
- [RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, May 2011.

## Author's Address

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

Phone: +49-421-218-63921  
Email: [cabo@tzi.org](mailto:cabo@tzi.org)

CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 4, 2015

C. Bormann  
Universitaet Bremen TZI  
A. Betzler  
C. Gomez  
I. Demirkol  
Universitat Politecnica de Catalunya/Fundacio i2CAT  
July 03, 2014

CoAP Simple Congestion Control/Advanced  
draft-bormann-core-cocoa-02

Abstract

The CoAP protocol needs to be implemented in such a way that it does not cause persistent congestion on the network it uses. The CoRE CoAP specification defines basic behavior that exhibits low risk of congestion with minimal implementation requirements. It also leaves room for combining the base specification with advanced congestion control mechanisms with higher performance.

This specification defines some simple advanced CoRE Congestion Control mechanisms, Simple CoCoA. In the present version -02, it is making use of input from simulations and experiments in real networks. The specification might still benefit from simplifying it further.

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 4, 2015.



## Copyright Notice

Copyright (c) 2014 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
1.1. Terminology . . . . .	3
2. Context . . . . .	3
3. Advanced CoAP Congestion Control: RTO Estimation . . . . .	4
3.1. Blind RTO Estimate . . . . .	4
3.2. Measured RTO Estimate . . . . .	5
3.2.1. Modifications to the algorithm of RFC 6298 . . . . .	5
3.2.2. Discussion . . . . .	6
3.3. Lifetime, Aging . . . . .	6
4. Advanced CoAP Congestion Control: Non-Confirmables . . . . .	6
4.1. Discussion . . . . .	7
5. IANA Considerations . . . . .	7
6. Security Considerations . . . . .	7
7. Acknowledgements . . . . .	7
8. References . . . . .	8
8.1. Normative References . . . . .	8
8.2. Informative References . . . . .	8
Authors' Addresses . . . . .	9

## 1. Introduction

(See Abstract.)

Extended rationale for this specification can be found in [I-D.bormann-core-congestion-control] and [I-D.eggert-core-congestion-control], as well as in the minutes of the IETF 84 CoRE WG meetings.

## 1.1. Terminology

This specification uses terms from [RFC7252]. In addition, it defines the following terminology:

**Initiator:** The endpoint that sends the message that initiates an exchange. E.g., the party that sends a confirmable message, or a non-confirmable message conveying a request.

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] 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.

(Note that this document is itself informational, but it is discussing normative statements.)

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

## 2. Context

In the Vancouver IETF 84 CoRE meeting, a path forward was defined that includes a very simple basic scheme (lock-step with a number of parallel exchanges of 1) in the base specification together with performance-enhancing advanced mechanisms.

The present specification is based on the approved text in the [RFC7252] base specification. It is making use of the text that permits advanced congestion control mechanisms and allows them to change protocol parameters, including NSTART and the binary exponential backoff mechanism. Note that Section 4.8 of [RFC7252] limits the leeway that implementations have in changing the CoRE protocol parameters.

The present specification also assumes that, outside of exchanges, non-confirmable messages can only be used at a limited rate without an advanced congestion control mechanism (this is mainly relevant for -observe). It is also intended to address the [RFC5405] guideline about combining congestion control state for a destination; and to clarify its meaning for CoAP using the definition of an endpoint.

The present specification does not address multicast or dithering beyond basic retransmission dithering.

### 3. Advanced CoAP Congestion Control: RTO Estimation

For an initiator that plans to make multiple requests to one destination endpoint, it may be worthwhile to make RTT measurements in order to obtain a better RTO estimation than that implied by the default initial timeout of 2 to 3 s. This is based on the usual algorithms for RTO estimation [RFC6298], with appropriately extended default/base values, as proposed in Section 3.2.1. Note that such a mechanism must, during idle periods, decay RTO estimates that are shorter or longer than the basic RTO estimate back to the basic RTO estimate, until fresh measurements become available again, as proposed in Section 3.3.

One important consideration not relevant for TCP is the fact that a CoAP round-trip may include application processing time, which may be hard to predict, and may differ between different resources available at the same endpoint. Also, for communications with networks of constrained devices that apply radio duty cycling, large and variable round-trip times are likely to be observed. Servers will only trigger their early ACKs (with a non-piggybacked response to be sent later) based on the default timers, e.g. after 1 s. A client that has arrived at a RTO estimate shorter than 1 s SHOULD therefore use a larger backoff factor for retransmissions to avoid expending all of its retransmissions in the default interval of 2 to 3 s. A proposal for a mechanism with variable backoff factors is presented in Section 3.2.1.

It may also be worthwhile to do RTT estimates not just based on information measured from a single destination endpoint, but also based on entire hosts (IP addresses) and/or complete prefixes (e.g., maintain an RTT estimate for a whole /64). The exact way this can be used to reduce the amount of state in an initiator is for further study.

#### 3.1. Blind RTO Estimate

The initial RTO estimate for an endpoint is set to 2 seconds.

If only the initial RTO estimate is available, the RTO estimate for each of up to NSTART exchanges started in parallel is set to 2 s times the number of parallel exchanges, e.g. if two exchanges are already running, the initial RTO estimate for an additional exchange is 6 seconds.

### 3.2. Measured RTO Estimate

The RTO estimator runs two copies of the algorithm defined in [RFC6298], as modified in Section 3.2.1: One copy for exchanges that complete on initial transmissions (the "strong estimator"), and one copy for exchanges that have run into retransmissions, where only the first two retransmissions are considered (the "weak estimator"). For the latter, there is some ambiguity whether a response is based on the initial transmission or the retransmissions. For the purposes of the weak estimator, the time from the initial transmission counts. Responses obtained after the third retransmission are not used to update an estimator.

The overall RTO estimate is an exponentially weighted moving average ( $\alpha = 0.5$ ) computed of the strong and the weak estimator, which is evolved after each contribution to the weak estimator (1) or to the strong estimator (2), from the estimator that made the most recent contribution:

$$\text{RTO\_overall\_} := 0.25 * \text{RTO\_weak\_} + 0.75 * \text{RTO\_overall\_} \quad (1)$$
$$\text{RTO\_overall\_} := 0.5 * \text{RTO\_strong\_} + 0.5 * \text{RTO\_overall\_} \quad (2)$$

(Splitting this update into the two cases avoids making the contribution of the weak estimator too big in naturally lossy networks.)

#### 3.2.1. Modifications to the algorithm of RFC 6298

This subsection presents three modifications that must be applied to the algorithm of [RFC6298] as per this document. The first two recommend new parameter settings. The third one is the variable backoff factor mechanism.

The initial value for each of the two RTO estimators is 2 s.

For the weak estimator, the factor K (the RTT variance multiplier) is set to 1 instead of 4. This is necessary to avoid a strong increase of the RTO in the case that the RTTVAR value is very large, which may be the case if a weak RTT measurement is obtained after one or more retransmissions.

If an RTO estimation is lower than 1 s or higher than 3 s, instead of applying a binary backoff factor in both cases, a variable backoff factor is used. For RTO estimations below 1 s, the RTO for a retransmission is multiplied by 3, while for estimations above 3 s, the RTO is multiplied only by 1.5 (this updated choice of numbers to be verified by more simulations). This helps to avoid that exchanges

with small initial RTOs use up all retransmissions in a short interval of time and exchanges with large initial RTOs may not be able to carry out all retransmissions within `MAX_TRANSMIT_WAIT` (93 s).

The binary exponential backoff is truncated at 32 seconds. Similar to the way retransmissions are handled in the base specification, they are dithered between  $1 \times \text{RTO}$  and  $\text{ACK\_RANDOM\_FACTOR} \times \text{RTO}$ .

### 3.2.2. Discussion

In contrast to [RFC6298], this algorithm attempts to make use of ambiguous information from retransmissions. This is motivated by the high non-congestion loss rates expected in constrained node networks, and the need to update the RTO estimators even in the presence of loss. Additional investigation is required to determine whether this is indeed justified.

### 3.3. Lifetime, Aging

The state of the RTO estimators for an endpoint **SHOULD** be kept as long as possible. If other state is kept for the endpoint (such as a DTLS connection), it is very strongly **RECOMMENDED** to keep the RTO state alive at least as long as this other state. It **MUST** be kept for at least 255 s.

If an estimator has a value that is lower than 1 s, and it is left without further update for 16 times its current value, the RTO estimate is doubled. If an estimator has a value that is higher than 3 s, and it is left without further update for 4 times its current value, the RTO estimate is set to be

$$\text{RTO\_overall\_} := 1 \text{ s} + (0.5 * \text{RTO\_overall\_})$$

(Note that, instead of running a timer, it is possible to implement these RTO aging calculations cumulatively at the time the estimator is used next.)

## 4. Advanced CoAP Congestion Control: Non-Confirmables

(TO DO: Align this with final consensus on -observe!)

A CoAP endpoint **MUST NOT** send non-confirmables to another CoAP endpoint at a rate higher than defined by this document. Independent of any congestion control mechanisms, a CoAP endpoint can always send non-confirmables if their rate does not exceed 1 B/s.

Non-confirmables that form part of exchanges are governed by the rules for exchanges.

Non-confirmables outside exchanges (e.g., [I-D.ietf-core-observe] notifications sent as non-confirmables) are governed by the following rules:

1. Of any 16 consecutive messages towards this endpoint that aren't responses or acknowledgments, at least 2 of the messages must be confirmable.
2. The confirmable messages must be sent under an RTO estimator, as specified in Section 3.
3. The packet rate of non-confirmable messages cannot exceed  $1/\text{RTO}$ , where RTO is the overall RTO estimator value at the time the non-confirmable packet is sent.

#### 4.1. Discussion

This is relatively conservative. More advanced versions of this algorithm could run a TFRC-style Loss Event Rate calculator [RFC5348] and apply the TCP equation to achieve a higher rate than  $1/\text{RTO}$ .

#### 5. IANA Considerations

This document makes no requirements on IANA. (This section to be removed by RFC editor.)

#### 6. Security Considerations

(TBD. The security considerations of, e.g., [RFC5681], [RFC2914], and [RFC5405] apply. Some issues are already discussed in the security considerations of [RFC7252].)

#### 7. Acknowledgements

The first document to examine CoAP congestion control issues in detail was [I-D.eggert-core-congestion-control], to which this draft owes a lot.

Michael Scharf did a review of CoAP congestion control issues that asked a lot of good questions. Several Transport Area representatives made further significant inputs this discussion during IETF84, including Lars Eggert, Michael Scharf, and David Black. Andrew McGregor, Eric Rescorla, Richard Kelsey, Ed Beroaset, Jari Arkko, Zach Shelby, Matthias Kovatsch and many others provided very useful additions.

Authors from Universitat Politecnica de Catalunya have been supported in part by the Spanish Government's Ministerio de Economia y Competitividad through projects TEC2009-11453 and TEC2012-32531, and FEDER.

## 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.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

### 8.2. Informative References

- [I-D.bormann-core-congestion-control]  
Bormann, C. and K. Hartke, "Congestion Control Principles for CoAP", draft-bormann-core-congestion-control-02 (work in progress), July 2012.
- [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-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, September 2008.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

Authors' Addresses

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

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

August Betzler  
Universitat Politecnica de Catalunya/Fundacio i2CAT  
Departament d'Enginyeria Telematica  
C/Jordi Girona, 1-3  
Barcelona 08034  
Spain

Email: august.betzler@entel.upc.edu

Carles Gomez  
Universitat Politecnica de Catalunya/Fundacio i2CAT  
Escola d'Enginyeria de Telecomunicacio i Aeroespacial  
de Castelldefels  
C/Esteve Terradas, 7  
Castelldefels 08860  
Spain

Phone: +34-93-413-7206  
Email: carlesgo@entel.upc.edu

Ilker Demirkol  
Universitat Politecnica de Catalunya/Fundacio i2CAT  
Departament d'Enginyeria Telematica  
C/Jordi Girona, 1-3  
Barcelona 08034  
Spain

Email: ilker.demirkol@entel.upc.edu



CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: December 24, 2014

A. Castellani  
University of Padova  
S. Loreto  
Ericsson  
A. Rahman  
InterDigital Communications, LLC  
T. Fossati  
KoanLogic  
E. Dijk  
Philips Research  
June 22, 2014

Advanced Guidelines for HTTP-CoAP Mapping Implementations  
draft-castellani-core-advanced-http-mapping-04

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 December 24, 2014.

Copyright Notice

Copyright (c) 2014 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. URI Mapping via HTTP Cache Control Extensions . . . . .	6
5. Multiple Message Exchanges Mapping . . . . .	6
5.1. Relevant Features of Existing Standards . . . . .	6
5.1.1. Multipart Messages . . . . .	6
5.1.2. Immediate Message Delivery . . . . .	7
5.1.3. Detailing Source Information . . . . .	7
5.2. Multicast Mapping . . . . .	7
5.2.1. URI Identification and Mapping . . . . .	8
5.2.2. Request Handling . . . . .	8
5.2.3. Examples . . . . .	9
5.3. Multicast Response Caching . . . . .	11
5.4. Observe Mapping . . . . .	12
5.4.1. Identification . . . . .	12
5.4.2. Notification(s) Mapping . . . . .	14
5.4.3. Examples . . . . .	15
6. HTML5 Scheme Handler Registration . . . . .	21
7. Placement and Deployment . . . . .	21
8. Examples . . . . .	22
9. Acknowledgements . . . . .	24
10. IANA Considerations . . . . .	24
11. Security Considerations . . . . .	24
11.1. Cross-protocol Security Policy Mapping . . . . .	25
11.2. Subscription . . . . .	25
12. References . . . . .	25
12.1. Normative References . . . . .	25
12.2. Informative References . . . . .	26
Appendix A. Internal Mapping Functions (from an Implementer's Perspective) . . . . .	27
A.1. URL Map Algorithm . . . . .	28
A.2. Security Policy Map Algorithm . . . . .	29
A.3. Content-Type Map Algorithm . . . . .	30
Authors' Addresses . . . . .	31

## 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.

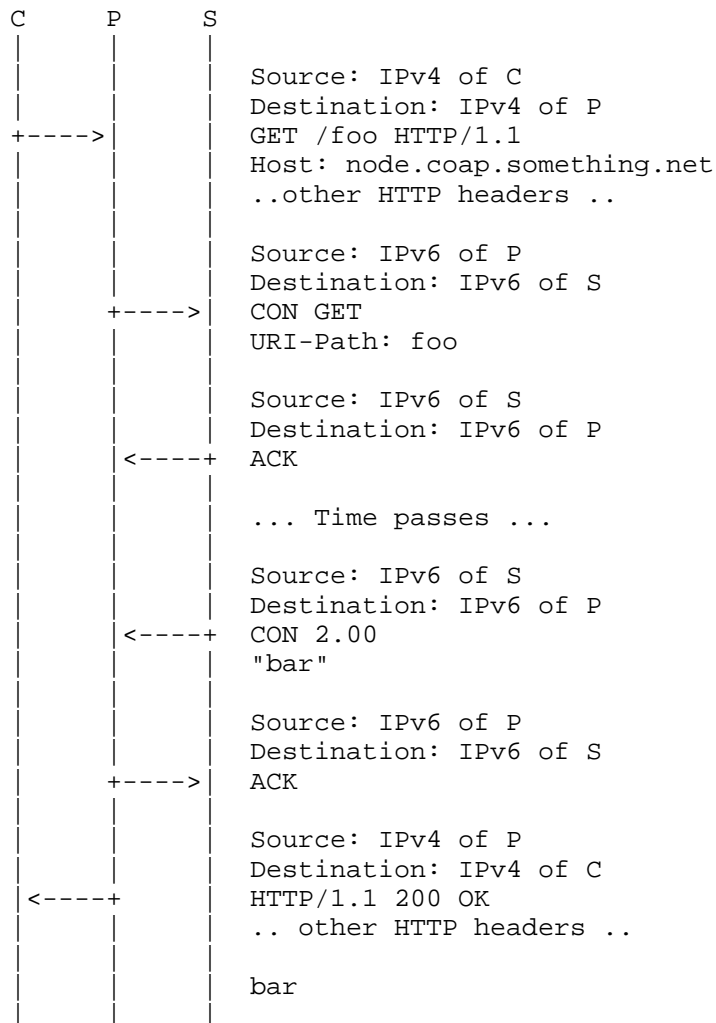


Figure 1: HTTP/IPv4 to CoAP/IPv6 Mapping

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. URI Mapping via HTTP Cache Control Extensions

An advanced strategy for triggering the cross-proxy that a translation is needed can be done via the HTTP Cache Control Extensions described in Section 5.2.3 of [RFC7234]. Specifically two new extensions can be defined, i.e. cross-coap and cross-coaps, that when included in a request to an HC forward cross-proxy translate the request to coap or coaps.

#### 5. 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.

##### 5.1. Relevant Features of Existing Standards

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

##### 5.1.1. 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.

### 5.1.2. 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 5.4, while describing its application to an observe session.

### 5.1.3. 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.

## 5.2. 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.

### 5.2.1. 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.

### 5.2.2. 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.



### 5.2.3. 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.

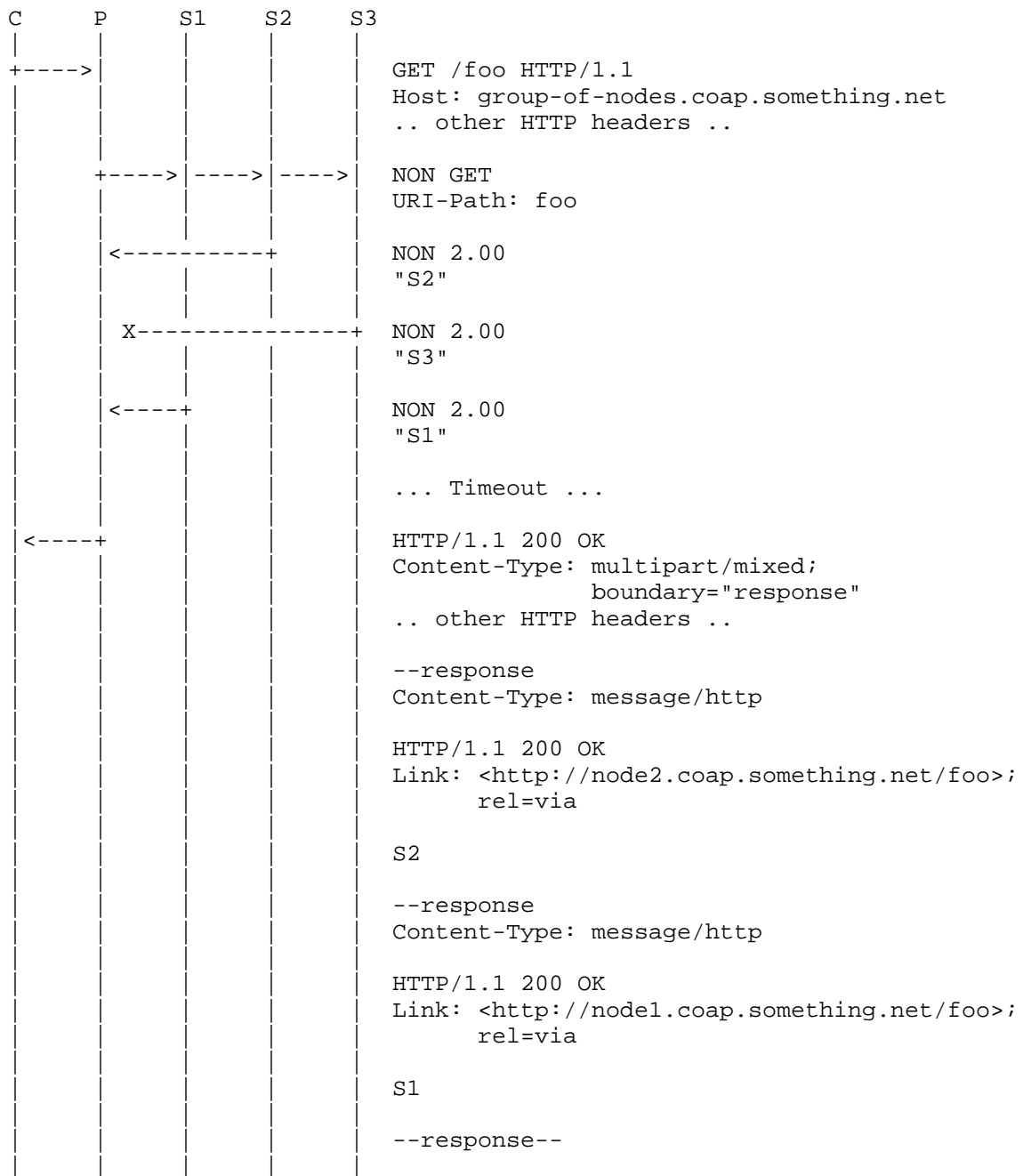


Figure 2: Unicast HTTP to Multicast CoAP Mapping

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 .

### 5.3. 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.

#### 5.4. 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.

##### 5.4.1. 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.

##### 5.4.1.1. 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.

#### 5.4.1.1.1. 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]).

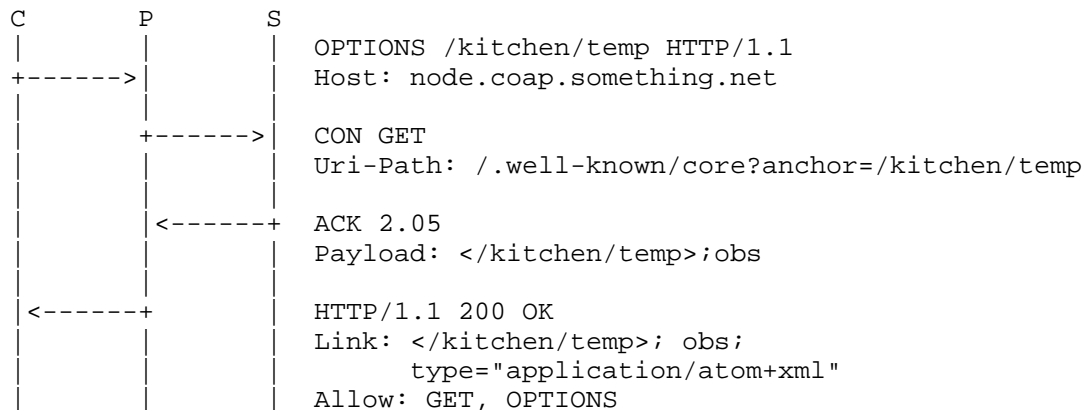


Figure 3: Discover Observability with HTTP OPTIONS

#### 5.4.1.2. 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.

#### 5.4.1.2.1. 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.

#### 5.4.1.2.2. 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.

#### 5.4.2. 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.

#### 5.4.2.1. Multipart Messaging

As already discussed in Section 5.1.1 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.

#### 5.4.2.2. 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.

#### 5.4.3. 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.

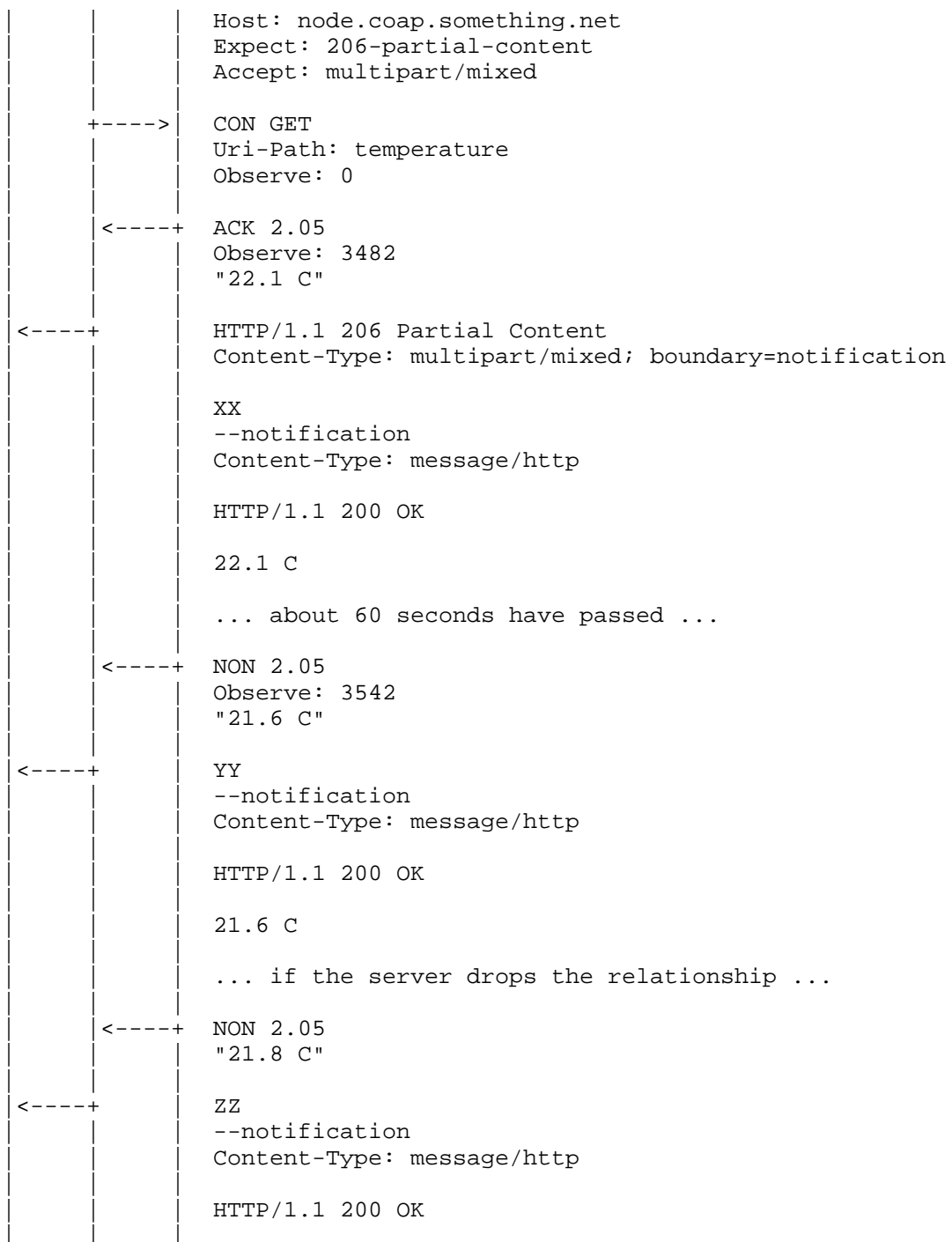
C manifests its intention to observe T by including the Expect Header in the request; if P or S do not support this interaction, the request MUST fail with "417 Expectation Failed" return code. In the presented example, both P and C support this interaction, and the subscription is successful, as stated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

```

C      P      S
|      |      |
+---->|      | GET /temperature HTTP/1.1

```





			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.

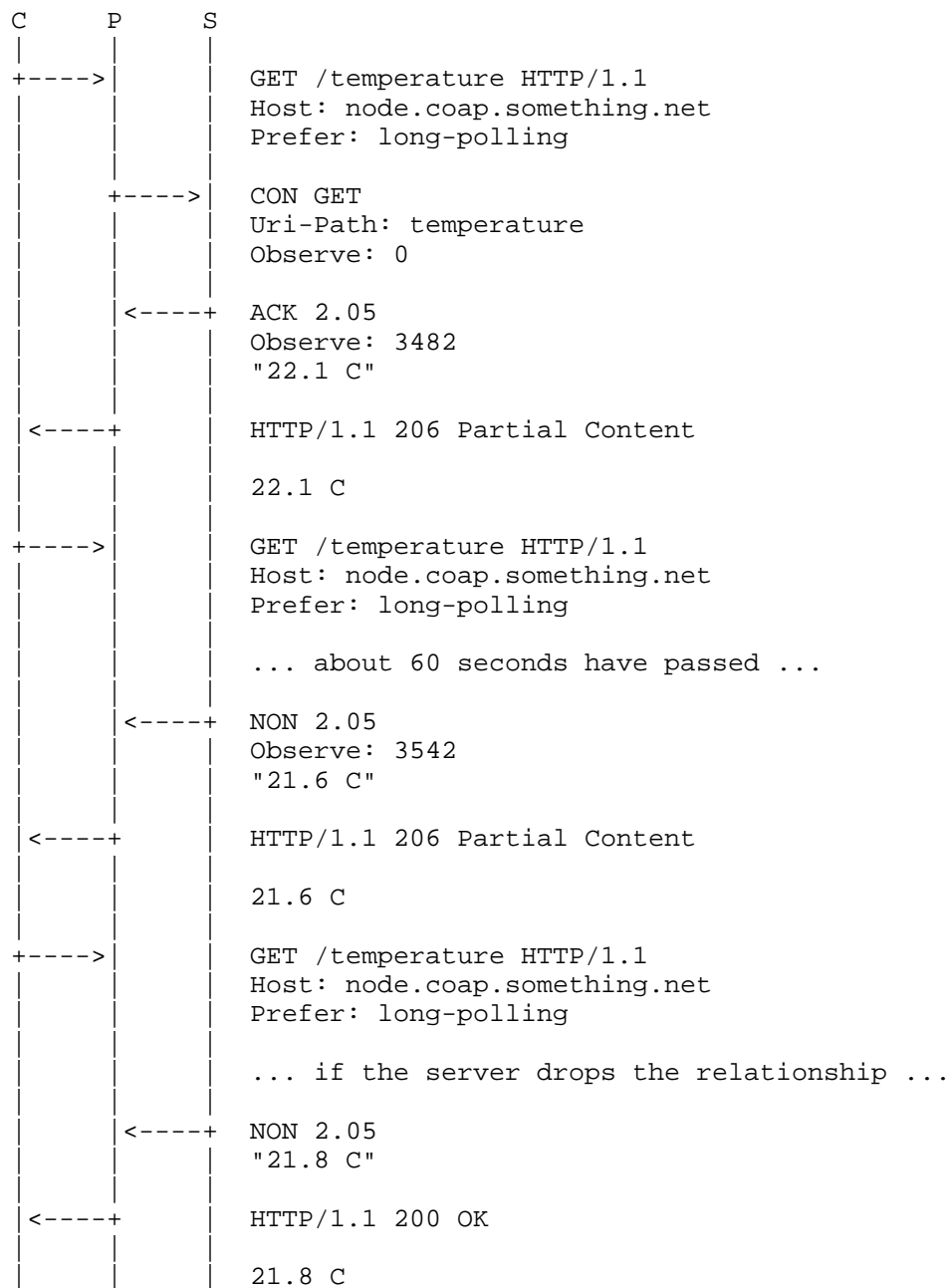


Figure 5: HTTP Long Polling to CoAP Observe

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.

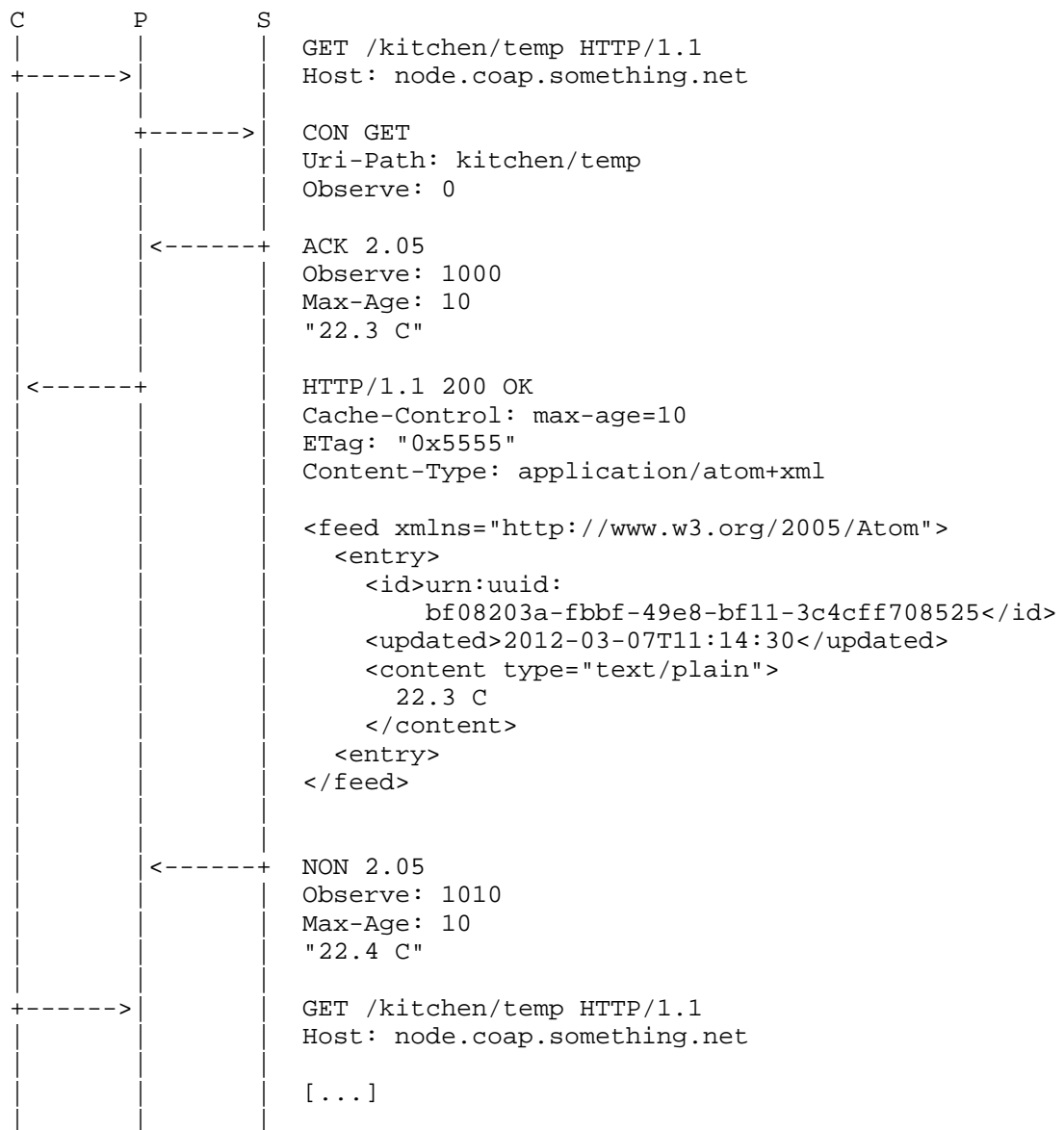


Figure 6: Observation via Atom feeds

## 6. 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"`.

## 7. 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.

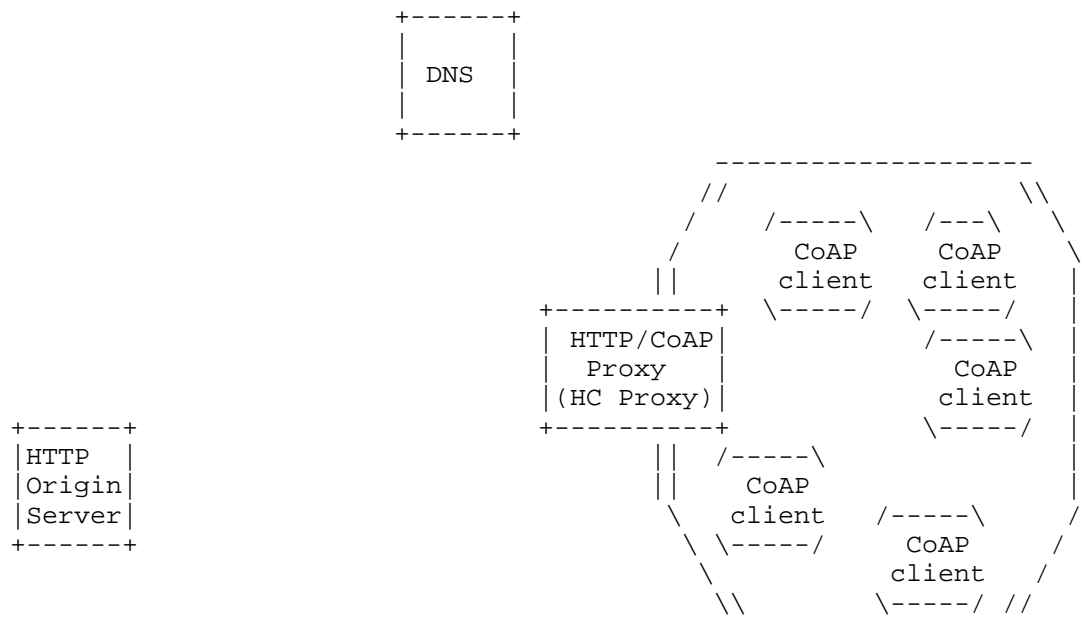


Figure 7: Client-side HC Proxy Deployment Scenario

## 8. 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.

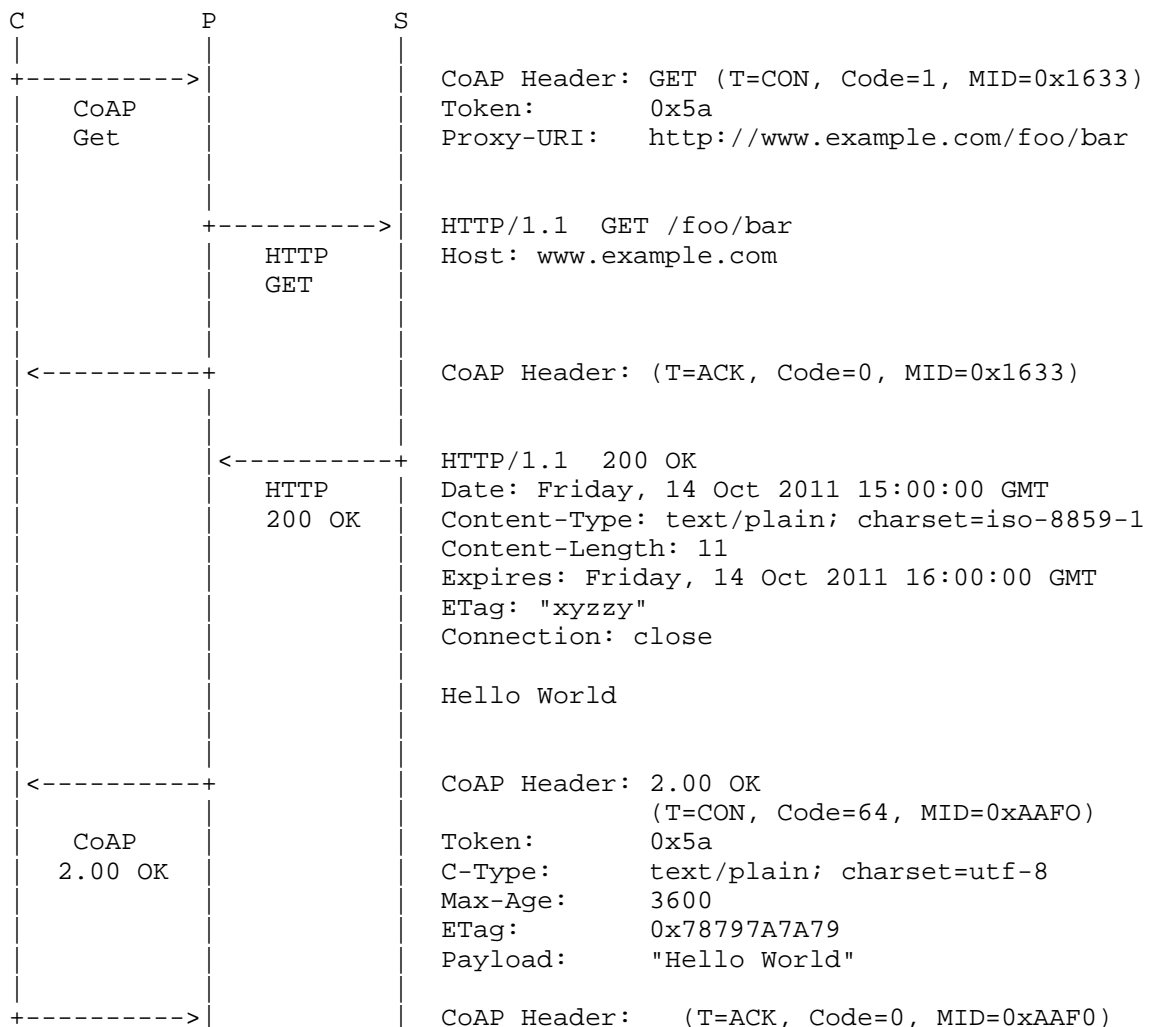


Figure 8: A Basic CoAP-HTTP GET Request

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.

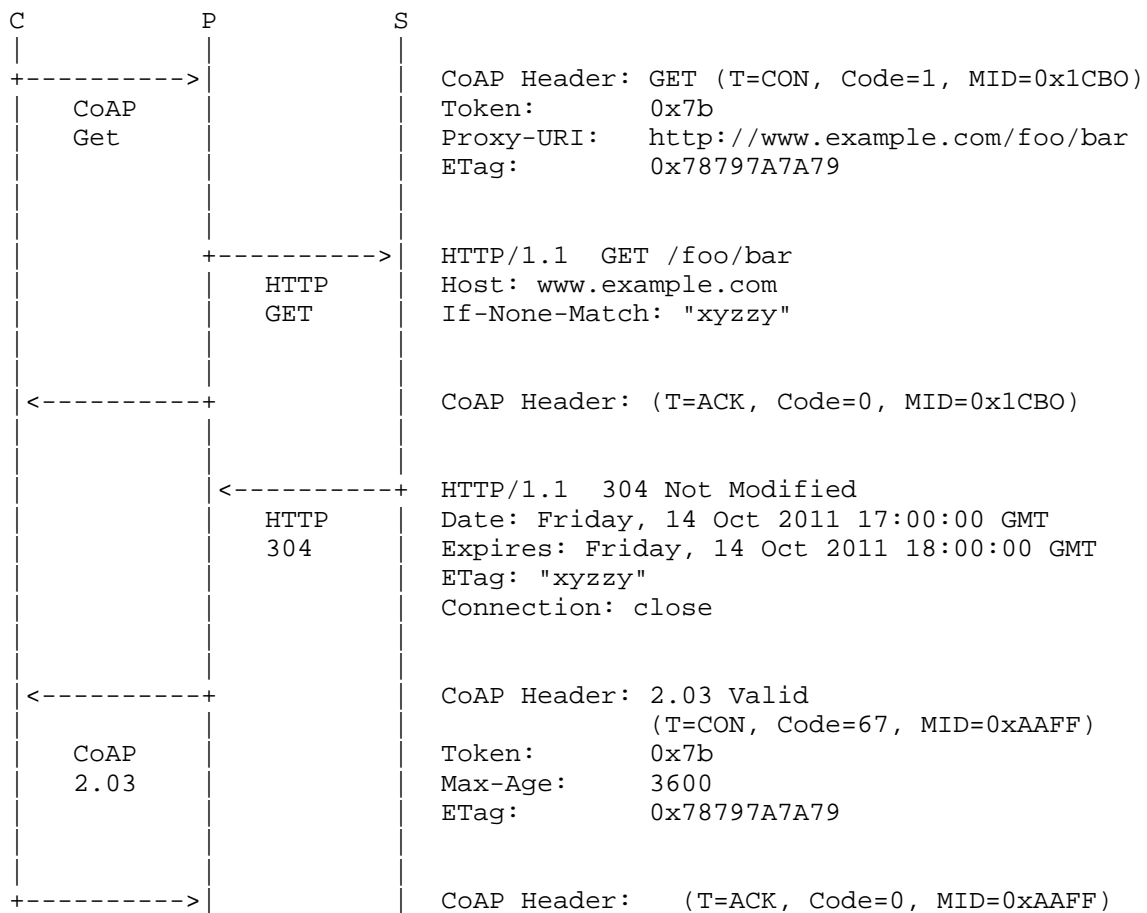


Figure 9: A CoAP-HTTP GET Request with an ETag Option

## 9. Acknowledgements

TBD.

## 10. IANA Considerations

This memo includes no request to IANA.

## 11. Security Considerations



### 11.1. 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.)

### 11.2. 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.

## 12. References

### 12.1. 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.

## 12.2. 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.
- [RFC7234] Fielding, R., Nottingham, M., and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, June 2014.
- [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](mailto:angelo@castellani.net)

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

Email: [salvatore.loreto@ericsson.com](mailto: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](mailto: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](mailto:tho@koanlogic.com)

Esko Dijk  
Philips Research

Email: [esko.dijk@philips.com](mailto:esko.dijk@philips.com)

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 5, 2015

Y. Hong  
Y. Choi  
ETRI  
D. Kim  
M. Khan  
W. JIN  
Jeju Nat. Univ.  
July 4, 2014

CoAP Endpoint Unit Identification for Multiple Sensor and Actuator in a  
Node  
draft-hong-core-coap-endpoint-unit-id-00

Abstract

The Constrained Application Protocol (CoAP) is a protocol intended towards devices which are constrained in terms of memory, processing and power i.e. small low power sensors, switches and valves etc. The CoAP allows such devices to interactively communicate over the Internet. This document is motivated by the concept of a composite CoAP node, a single CoAP entity which integrates multiple CoAP resources (sensors, actuators) and the scheme to allow the identification of individual integrated resources while using the Unit ID as a new CoAP option. The Unit ID option in the CoAP enables the usage of composite nodes consisting of multiple sensors and actuators while having a single IP address for communication. The integrated resources can be individually or collectively communicated with and/or controlled using CoAP messages with additional options of UnitSize and UnitID. The UnitSize is basically a numeric value indicating the number of sub-resources in a composite CoAP node while the UnitID option has the string identifiers for the sub-resource(s) for which the message is intended. These options will enable the CoAP to communicate and control multiple resources by using single composite messages i.e. UnitID = "\*", efficiently utilize IP addresses i.e. one IP multiple IDs, reduce communication traffic and hence conserve power among the CoAP resources.

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 5, 2015.

#### Copyright Notice

Copyright (c) 2014 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. Conventions and Terminology . . . . .	4
3. The use case of multiple CoAP unit identification and control	4
4. IP and Endpoint Unit ID mapping architecture . . . . .	5
5. Benefits of the Endpoint Unit Identification . . . . .	7
6. The Extended CoAP Header . . . . .	7
7. Procedure of the Endpoint Unit Identification . . . . .	8
7.1. Sub Unit(s) Registration with RD . . . . .	8
7.2. Sub Unit Lookup in RD . . . . .	9
7.3. CoAP Client Server Interaction (Single Unit) . . . . .	10
7.4. CoAP Client Server Interaction (Multiple Units) . . . . .	11
8. Security Considerations . . . . .	13
9. IANA Considerations . . . . .	13
10. References . . . . .	13
10.1. Normative References . . . . .	13
10.2. Informative References . . . . .	13
Authors' Addresses . . . . .	14

#### 1. Introduction

This draft presents a conceptual architecture and design features of multiple Unit IDs in a node for resource discovery, registration and lookup. The concept of node ID has been presented in [I-D.li-coap-nodeid]. This draft presents the idea of nodes having

multiple integrated sensing and/or actuating devices. Each of these devices is separately identifiable via a Unit ID. The Unit ID for a given resource must be unique among all the integrated resources in a single node while the same ID can represent a resource integrated in another node.

The integrated resources inside a node are separately identified by node ID and Unit ID together. Every node has an IP address through which it can communicate with clients or other modules of the system (Resource Directory). A detailed description of the purpose and features of Resource Directory have been presented in [I-D.ietf-core-rd].

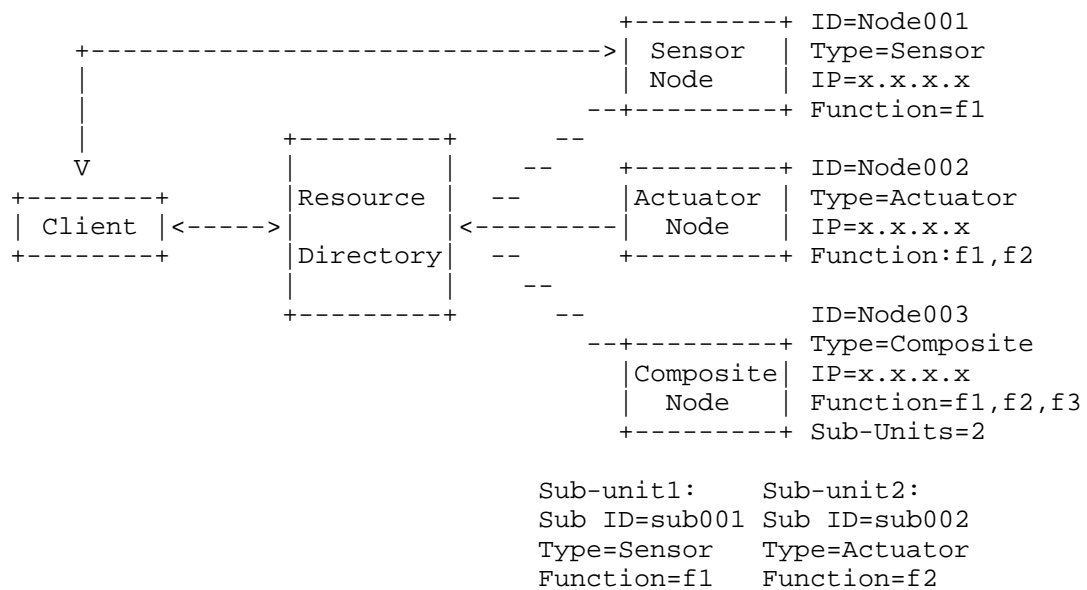


Figure 1: Endpoint Unit ID and Resource Directory

Figure 1 shows that a node may contain a single or multiple integrated resources i.e. multiple sensors, multiple actuators or sensors and actuators in a single node. The nodes register these resources with the Resource Directory. The Resource Directory defines its own function sets for discovery, registration and lookup etc. Once a node had registered all its integrated resources with the Resource Directory, the clients may lookup single or multiple resources and may interact with them directly. The Resource Directory helps in the automated discovery and lookup of resources

while the multi-Unit IDs provide an efficient utilization of a single IP for interacting with multiple resources.

As described in [RFC7252], there are two entities required for CoAP communication i.e. CoAP Client and CoAP Server. A CoAP Server may also act as client and vice versa if both of these entities have resources to share and require certain resources from each other. The CoAP server discovers a Resource Directory (RD) [I-D.ietf-core-rd]. The discovery of RD means finding location of the register function set in the RD using which a CoAP server may register the resources which it wants to share.

Once a complete path is obtained for a register function set in the RD, the CoAP server may then register (publish) resources to the RD. The CoAP clients then requests the RD to look up for registered resources. The RD then returns the access paths for the registered resources according to the request of the client. The returned resources may include simple or composite resources and the client can communicate with these resources. If a single CoAP node has multiple integrated sub devices, then the composite interaction with the resources is based on UnitID(s). The client can interact with individual sub devices or collectively interact with all the sub devices of a composite node. It is important to note that the description and discovery of resources hosted by a constrained web server is specified by the CoRE Link Format [RFC6690] which is based on the Web Linking [RFC5988] for the discovery of resources hosted by an HTTP Web Server.

## 2. Conventions and 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].

## 3. The use case of multiple CoAP unit identification and control

Figure 2 shows the use case scenario for a CoAP composite node which integrates a light sensor and two switches to control the lights in a room. The composite node is accessed via a single IP address assigned to it while the sub-resources of the composite node are accessed with Unit IDs. The composite node like a normal CoAP Endpoint, registers its resources in the form of sub units with the RD. The RD, thus have a single IP address for the composite node and Unit IDs for the sub units of the composite node.

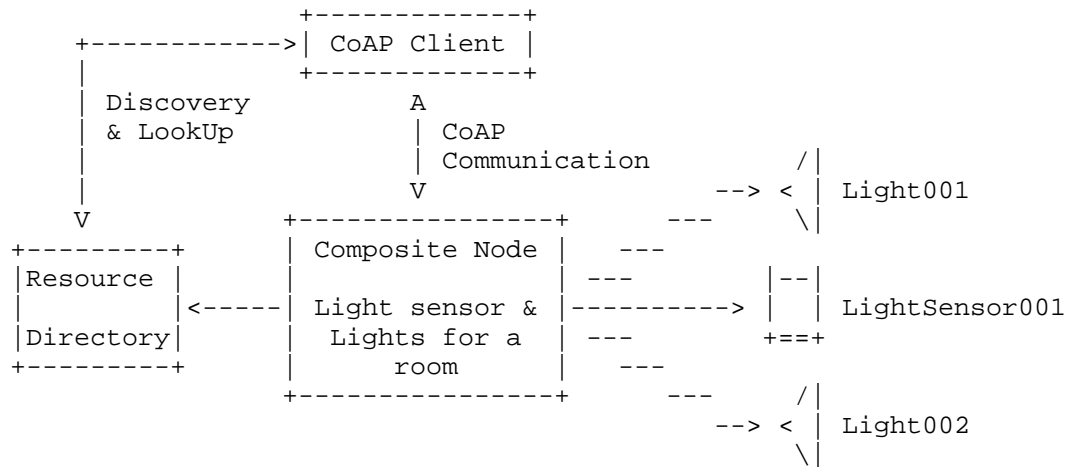


Figure 2: Multiple UnitID based composite CoAP node interaction use case

A CoAP client performs look up on the RD and gets the required resource information. Suppose the client wants to interact with the composite node, the information regarding all its sub units is also provided to the client by the RD. The client then use this information (i.e. UnitSize, UnitID) to create CoAP messages in order to interact with a single or multiple sub units of the composite node. For example, the user may send a CoAP message with UnitSize=1 and UnitID= "lightSensor001" to request data from the light sensor. The Composite node will return an ACK message with UnitID parameter and sensor reading as message payload. The client may also send a CoAP message with UnitSize=2 and UnitID= "Light001", UnitID="Light002" options to turn-on or off the lights with a single message.

#### 4. IP and Endpoint Unit ID mapping architecture

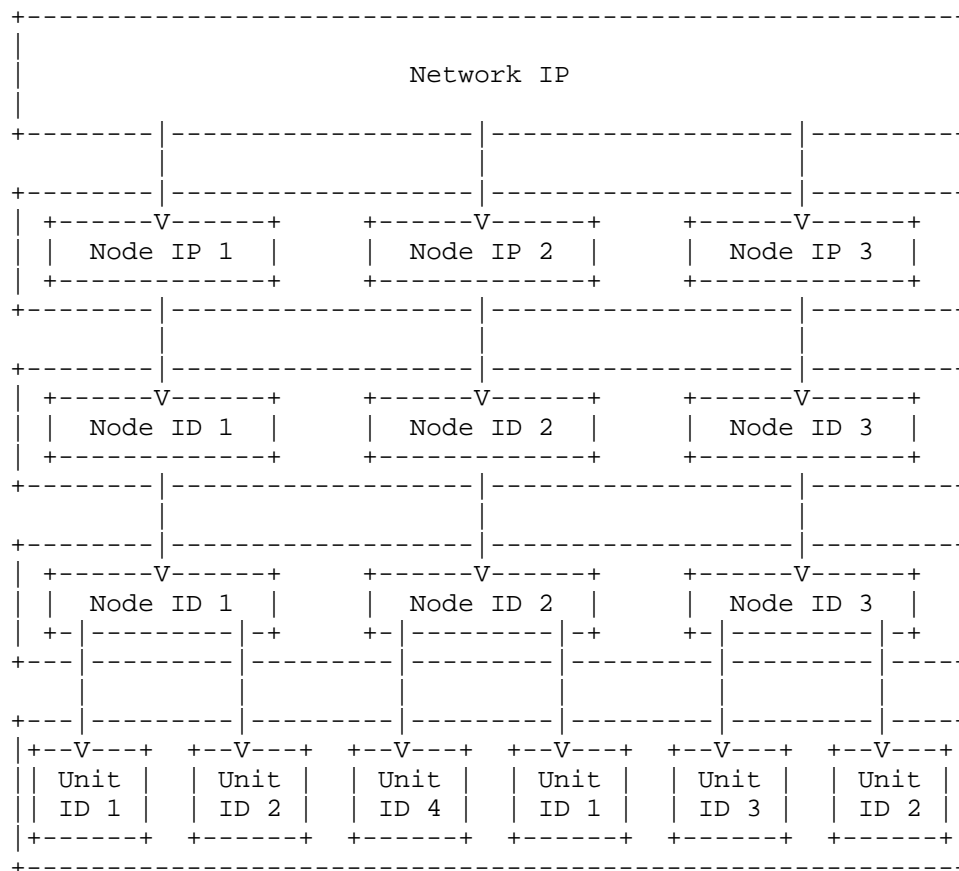


Figure 3: IP address and Endpoint Unit ID mapping architecture

Figure 3 presents a generalized architecture for IP and ID mapping in the proposed Endpoint Unit ID scenario. The network IP and local IP addresses are used to access the network of the node and the physical node respectively. In the CoAP a node ID is used to insure the consistency of the communication when an IP address change at the client or the server occurs during a communication session. Thus a node IP address and node ID pair used to communicate with a single resource. We propose that a single node may have multiple integrated resources and each of these resources can be represented by multiple sub-identifiers (IDs). The sub-identifier for the integrated resource is called as the Unit ID and a node may have more than one Unit IDs.

This scheme enables the use of single IP address for communicating with multiple resources (units) and each resource may be treated as a separate entity having its own address. Thus the result is efficient utilization of addressing space by combining the Node IP and Unit ID pairs. Group registration, lookup etc. and group communication for CoAP resources have been described in [I-D.ietf-core-rd] and [I-D.ietf-coap-group] respectively but both these drafts consider every resource in a group as a unique addressable entity hence no benefits when it comes to controlling IP address space usage or communication traffic load.

## 5. Benefits of the Endpoint Unit Identification

The Unit ID concept for composite endpoint (Node) provides the following major benefits.

- a. A composite node with multiple integrated sub-unit resources will require only one IP address and using the IP address and Unit ID pairs, individual resources can be separately accessed without the need to have a separate IP address for each resource. Thus the proposed scheme efficiently utilizes IP address space to represent more devices with lesser number of IP addresses.
- b. A single CoAP message with Unit ID parameter may be used to control sub-devices collectively using special characters. For example, a given composite endpoint may have sensors and actuators and all these sub-unit devices can be controlled with a single message using "\*" as the Unit ID parameter value.
- c. Using composite messages for Unit ID may also benefit in reducing traffic flow between client and endpoints (CoAP Server) and may also help in conserving energy in the constrained devices.

## 6. The Extended CoAP Header

Figure 4 shows the CoAP message header format. The header for the CoAP message is all the same with fields such as version, Type, and Token length etc. The change can be seen in the options section where the UnitSize field specifies the number of sub-unit integrated into a single composite node and the UnitID option which can hold a string ID for UnitID representing a sub-unit in a composite node. The UnitID field can be repeated multiple times according to the value of the UnitSize parameter and every time representing a single string ID for a sub-unit related to a specific composite node.

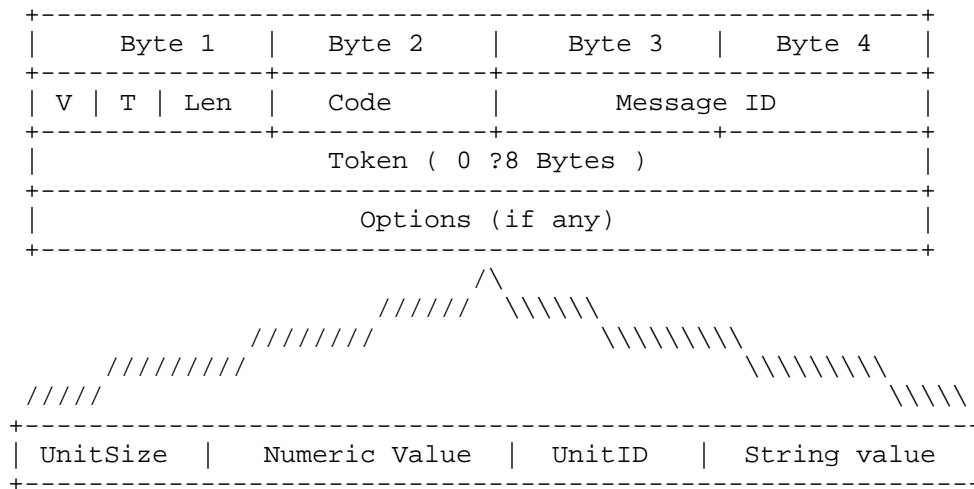


Figure 4: Multi-ID CoAP message format

## 7. Procedure of the Endpoint Unit Identification

### 7.1. Sub Unit(s) Registration with RD

Figure 5 shows the sequence of activities involved in the registration of Endpoint Unit ID nodes with integrated resources in the RD. In order for a node to register its integrated resources with the RD, the node uses the RD's registration function set and sends a CoAP POST message to the RD. The message payload contains the list of all the Unit IDs associated with the node. The RD receives the message and checks whether the request is valid. If the RD receives a valid request from the node, the source IP address and Port number from the CoAP request parameters or the message source address portion (default). The RD then extracts the Unit IDs from the message payload and creates a resource location for all the resources and returns a response message to the node. If the registration process is successful then a location URI is returned to the requesting node so it may update the registration or remove the location entry thus cancelling the registration of its integrated resources otherwise an error message is returned mentioning the cause of the failure.

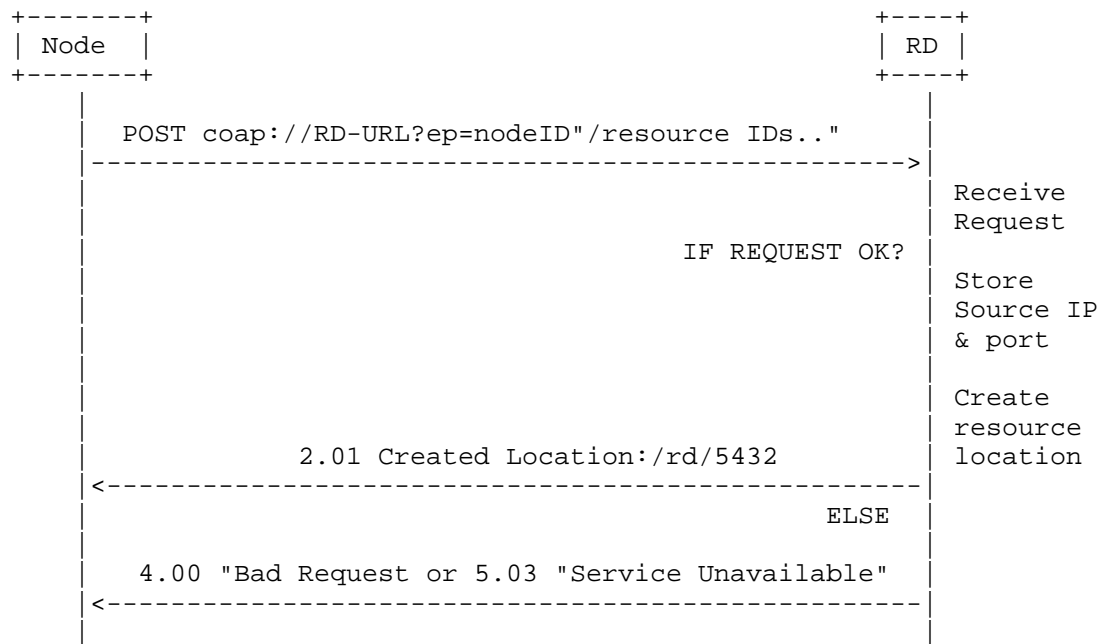


Figure 5: Endpoint Unit ID resource registration with RD

## 7.2. Sub Unit Lookup in RD

Figure 6 presents the RD based lookup process for Endpoint Unit ID resources integrated into a single node i.e. single IP address. The diagram shows a client requesting for a specific type (Temperature) of resources registered with the RD. For this purpose, it sends GET request to the RD with the type of resources the client wants to lookup in the directory. The RD receives the message, checks if the message is a valid CoAP request and then gets the IDs for all the registered resources with the resource type value equivalent to the one requested by the client (Temperature). The RD then creates a response message with the list of node IP address and resource IDs and sends it to the client. The client may then choose a specific resource from this list and communicate with it directly using the CoAP protocol.



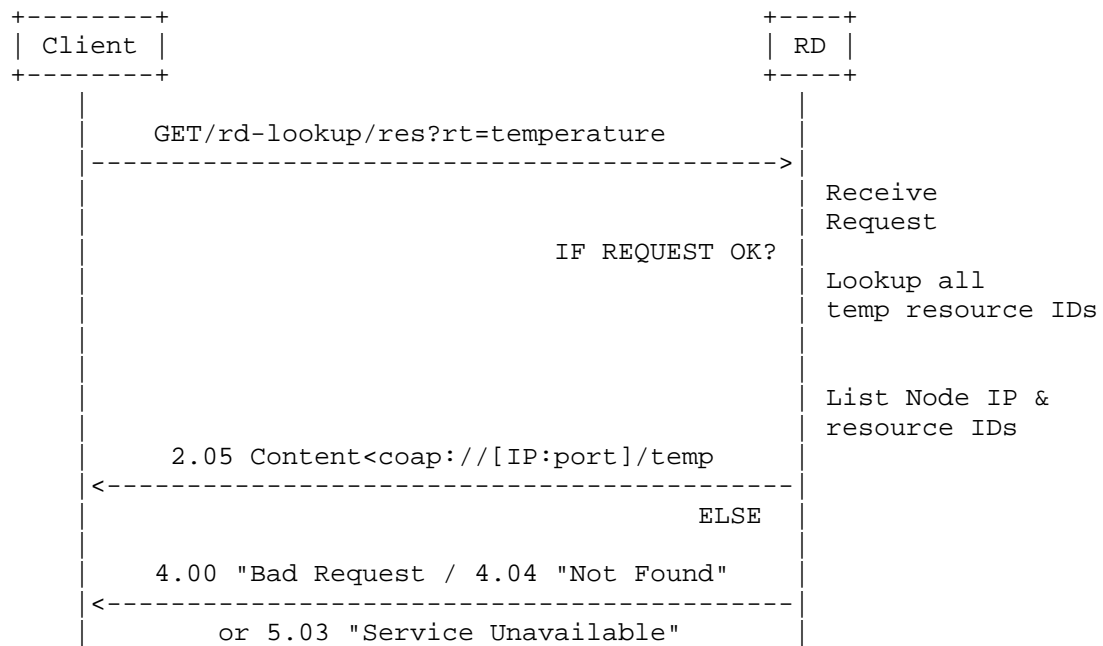


Figure 6: RD based resource lookup

### 7.3. CoAP Client Server Interaction (Single Unit)

Figure 7 shows the interaction among a client and resource (CoAP Server). As mentioned previously, the client performs lookup on the RD for a specific resource type and gets the list of all the resource IDs (node ID and Unit ID) registered with the RD. The following figure shows the process of client selecting a resource from that list and communicating with it directly.

Once the client decides to interact with a resource, it gets the resource complete URI i.e. Node IP address, Port number and Unit ID if it is a composite node. For a simple resource i.e. sensor or actuator, the node ID is used in conjunction with the IP address to perform the interaction between the CoAP client and server while for a composite node i.e. with multiple integrated resources (multiple IDs), the client creates a Unit ID, Token pair and sends a GET request to the integrated resource of a node using the complete URI. Here the Token means the CoAP token sent with a normal GET request. The node (CoAP Server), checks the request's validity and responds back to the client with an ACK, consisting of the Token and data from

the integrated resource. The client checks the source of the data by comparing the Token of the ACK with the stored Unit ID, Token pair.

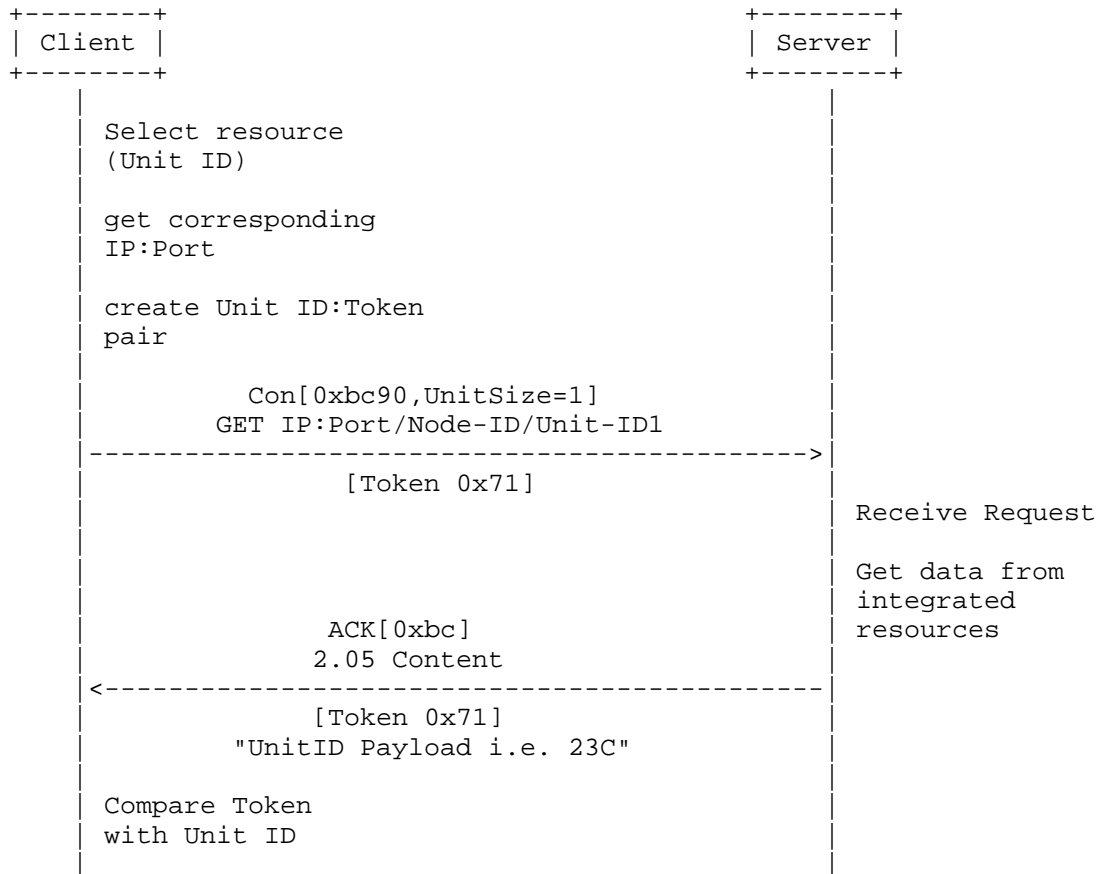


Figure 7: CoAP based client server interaction (single Unit ID)

#### 7.4. CoAP Client Server Interaction (Multiple Units)

Figure 8 shows the interaction among a client and multiple resources i.e. multiple Unit IDs. The example shown in the figure suggests that both Unit IDs belong to a single node but the Unit IDs may also belong to more than one CoAP nodes. As mentioned previously, the client performs lookup on the RD for a specific resource type and gets the list of all the resource IDs (nodes ID and Unit ID) registered with the RD. The following figure shows the process of

client choosing to interact with multiple unit resources (integrated resources) from the list provided by the RD.

Once the client selects the resources' complete URI i.e. Node IP address, Port number and Unit IDs for communication, the client creates and stores the Unit ID, Token pairs.

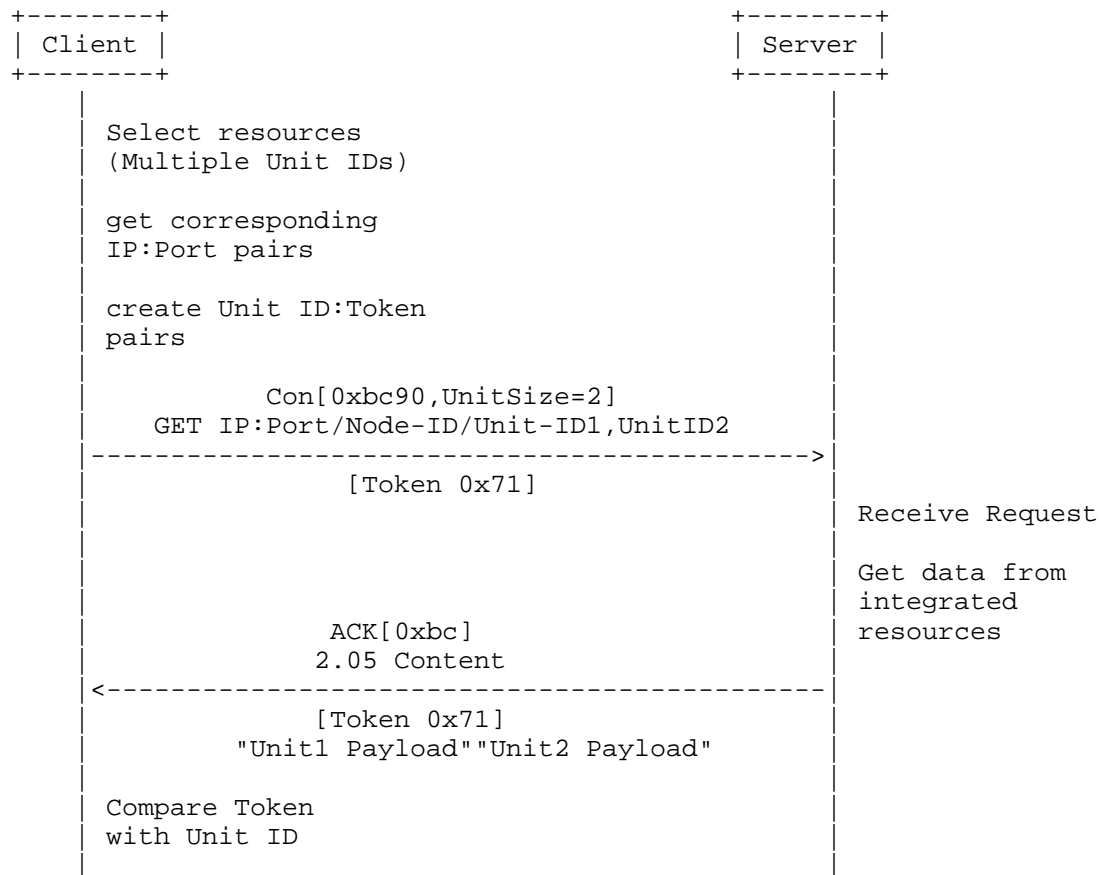


Figure 8: CoAP based client server interaction (Endpoint multiple Unit ID)

Here the Token means the CoAP token sent with a normal GET request. The client then sends a GET request to the integrated resources belonging to one or more nodes using the complete URIs (Node IP address, Port number, Node ID, Unit IDs). The GET request with multiple Unit IDs also has the Unit size parameter, mentioning the

number of integrated resources from which the client requests data. The node (CoAP Server), checks the request's validity and responds back to the client with an ACK, consisting of the Token and data from the integrated resources. The client checks the source of the data by comparing the Token of the ACK with the stored Unit ID, Token pairs.

## 8. Security Considerations

TBD.

## 9. IANA Considerations

TBD

## 10. References

### 10.1. 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.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

### 10.2. Informative References

- [I-D.ietf-coap-group]  
Rahman, A. and E. Dijk, "Group Communication for CoAP", ID draft-ietf-core-groupcomm-19, June 2014.
- [I-D.ietf-core-rd]  
Shelby, Z., Bormann, C., and S. Krco, "CoRE Resource Directory", ID draft-ietf-core-resource-directory-01 , December 2013.
- [I-D.li-coap-nodeid]  
Li, K. and G. Wei, "CoAP Option Extension: NodeId", ID draft-li-core-coap-node-id-option-01 , June 2014.

## Authors' Addresses

Yong-Geun Hong  
ETRI  
218 Gajeong-ro Yuseung-Gu  
Daejeon 305-700  
Korea

Phone: +82 42 860 6557  
Email: yghong@etri.re.kr

Younghwan Choi  
ETRI  
218 Gajeong-ro Yuseung-Gu  
Daejeon 305-700  
Korea

Phone: +82 42 860 1429  
Email: yhc@etri.re.kr

DoHyeun Kim  
Jeju Nat. Univ.  
Jeju  
Korea

Phone: +82 64 754 3658  
Email: kimdh@jejunu.ac.kr

Mohammad Sohail Khan  
Jeju Nat. Univ.  
Jeju  
Korea

Phone: +82 64 754 3658  
Email: sohail.khan@nwfpuet.edu.pk

WENQUAN JIN  
Jeju Nat. Univ.  
Jeju  
Korea

Phone: +82 64 754 3658  
Email: pluskml2@live.com

CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 5, 2015

C. Bormann  
Universitaet Bremen TZI  
Z. Shelby, Ed.  
ARM  
July 04, 2014

Blockwise transfers in CoAP  
draft-ietf-core-block-15

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 January 5, 2015.

#### Copyright Notice

Copyright (c) 2014 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 . . . . .	5
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 . . . . .	15
3.1. Block2 Examples . . . . .	15
3.2. Block1 Examples . . . . .	19
3.3. Combining Block1 and Block2 . . . . .	20
3.4. Combining Observe and Block2 . . . . .	22
4. The Size2 and Size1 Options . . . . .	25
5. HTTP Mapping Considerations . . . . .	26
6. IANA Considerations . . . . .	27
7. Security Considerations . . . . .	28
7.1. Mitigating Resource Exhaustion Attacks . . . . .	29

7.2. Mitigating Amplification Attacks . . . . .	29
8. Acknowledgements . . . . .	30
9. References . . . . .	30
9.1. Normative References . . . . .	30
9.2. Informative References . . . . .	30
Authors' Addresses . . . . .	31

## 1. Introduction

The work on Constrained RESTful Environments (CoRE) aims at realizing the REST architecture in a suitable form for the most constrained nodes (such as microcontrollers with limited RAM and ROM [RFC7228]) and networks (such as 6LoWPAN, [RFC4944]) [RFC7252]. The CoAP protocol is intended to provide RESTful [REST] services not unlike HTTP [RFC7230], 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) for the transport of larger representations would be 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 individual 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", "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 (except for the last one) 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

No.	C	U	N	R	Name	Format	Length	Default
23	C	U	-	-	Block2	uint	0-3	(none)
27	C	U	-	-	Block1	uint	0-3	(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 [RFC7252], 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 [RFC7252]).

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 specific 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 Section 3.2 of [RFC7252]). 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.)

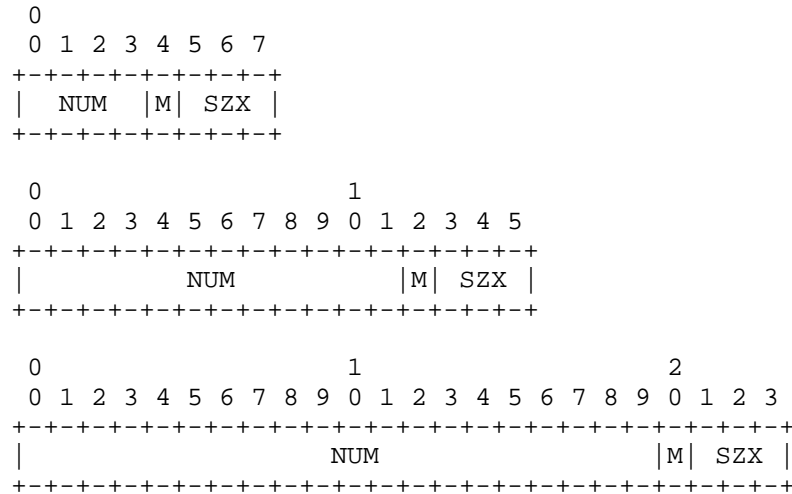


Figure 1: Block option value

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 first byte 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 implied 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 [RFC7252], 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 as the initial request 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 MAY also use the Block2 Option on the initial request, 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 [RFC7252], and another response code is extended in its meaning.

### 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 [RFC7252], 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". [RFC7252] 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 block size proposal. All ACK messages except for the last carry 64 bytes of payload; the last one carries between 1 and 64 bytes.

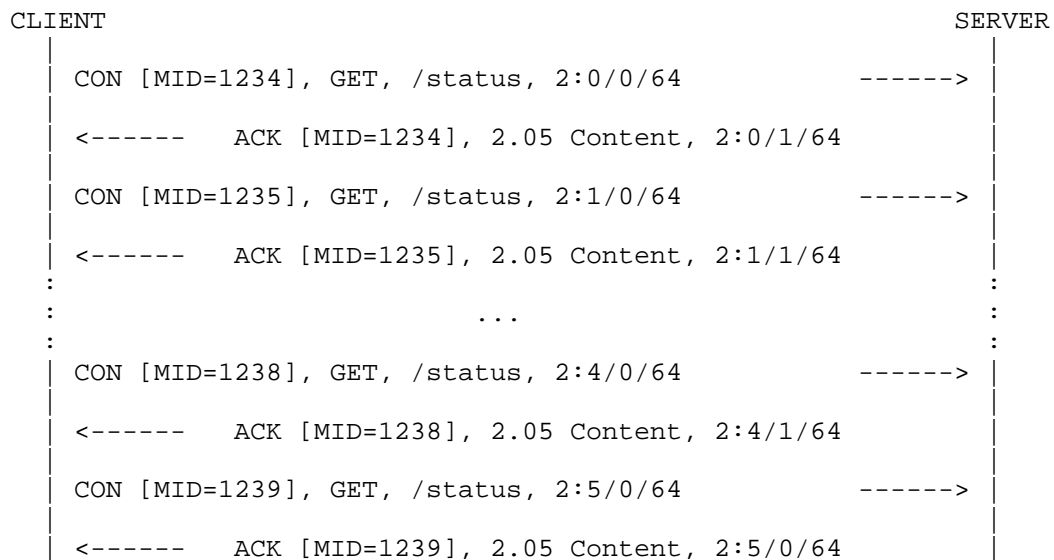


Figure 3: Blockwise GET with early negotiation

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.

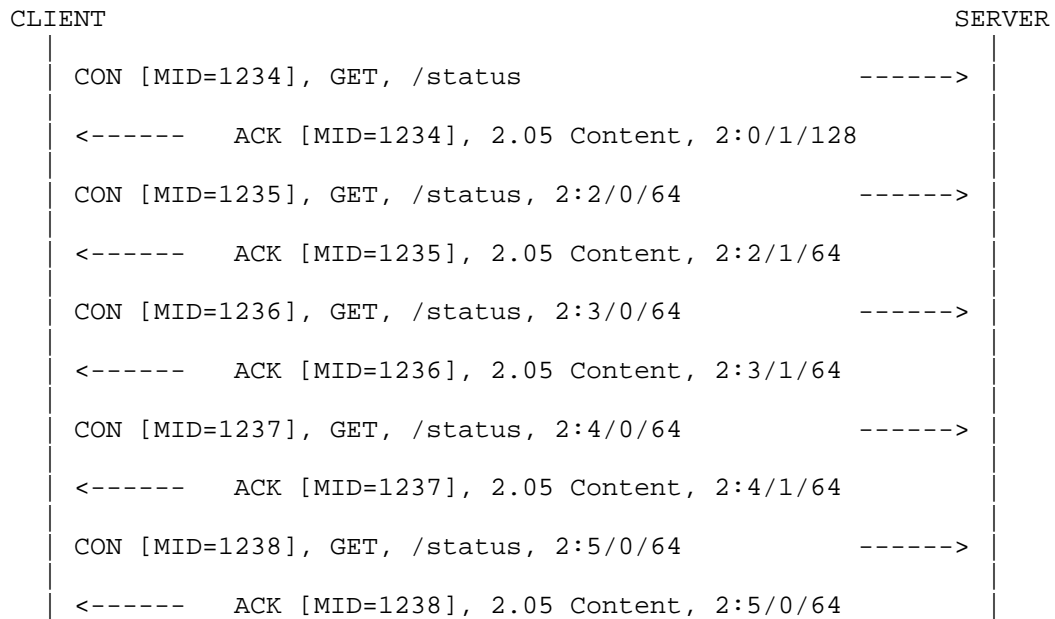


Figure 4: Blockwise GET with late negotiation

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).

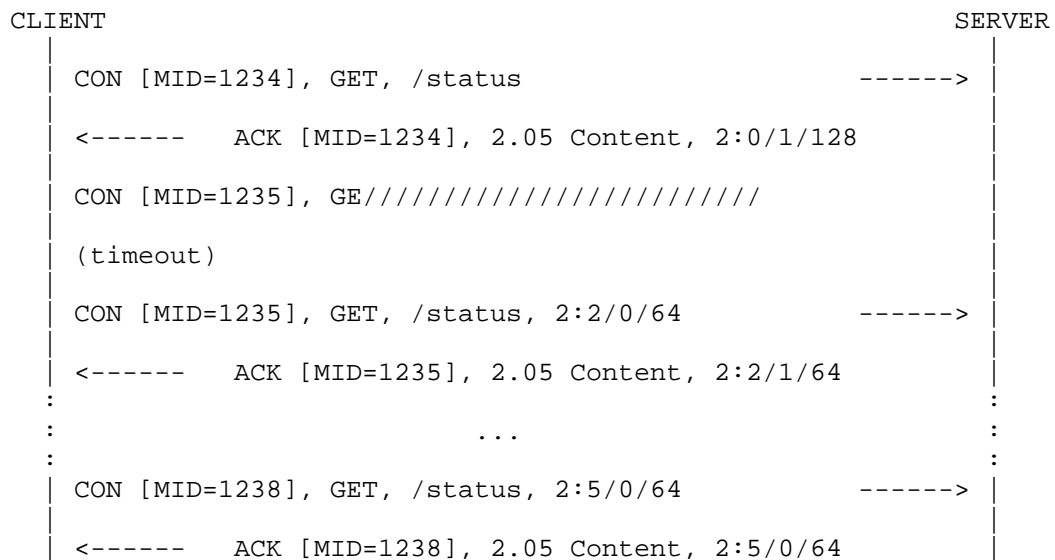


Figure 5: Blockwise GET with late negotiation and lost CON

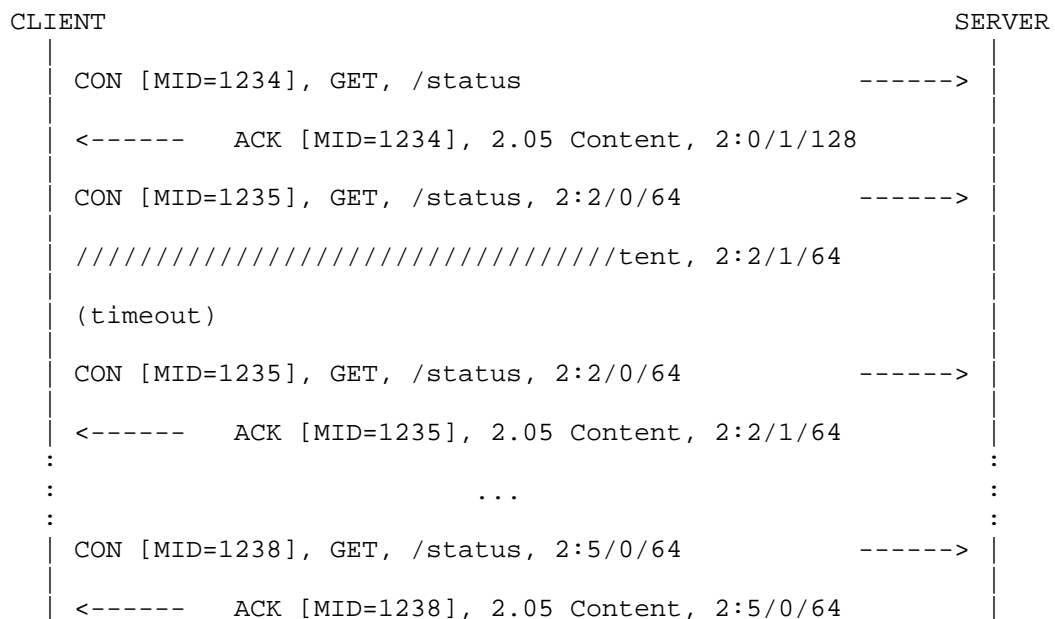


Figure 6: Blockwise GET with late negotiation and lost ACK

### 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)	
<----- ACK [MID=1237], 2.04 Changed, 2:1/1/128	
CON [MID=1238], POST, /soap, 2:2/0/128	----->
<----- ACK [MID=1238], 2.04 Changed, 2:2/1/128	
CON [MID=1239], POST, /soap, 2:3/0/128	----->
<----- ACK [MID=1239], 2.04 Changed, 2:3/0/128	

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
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, 2:0/0/64 ----->	
<----- ACK [MID=1236], 2.04 Changed, 1:2/0/128, 2:0/1/64	
CON [MID=1237], POST, /soap, 2:1/0/64 -----> (no payload for requests with Block2 with NUM != 0)	
<----- ACK [MID=1237], 2.04 Changed, 2:1/1/64	
CON [MID=1238], POST, /soap, 2:2/0/64 ----->	
<----- ACK [MID=1238], 2.04 Changed, 2:2/1/64	
CON [MID=1239], POST, /soap, 2:3/0/64 ----->	
<----- ACK [MID=1239], 2.04 Changed, 2:3/0/64	

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.

CLIENT	SERVER
+----->	Header: GET 0x41011636
GET	Token: 0xfb
	Uri-Path: status-icon
	Observe: (empty)
<-----+	Header: 2.05 0x61451636
2.05	Token: 0xfb

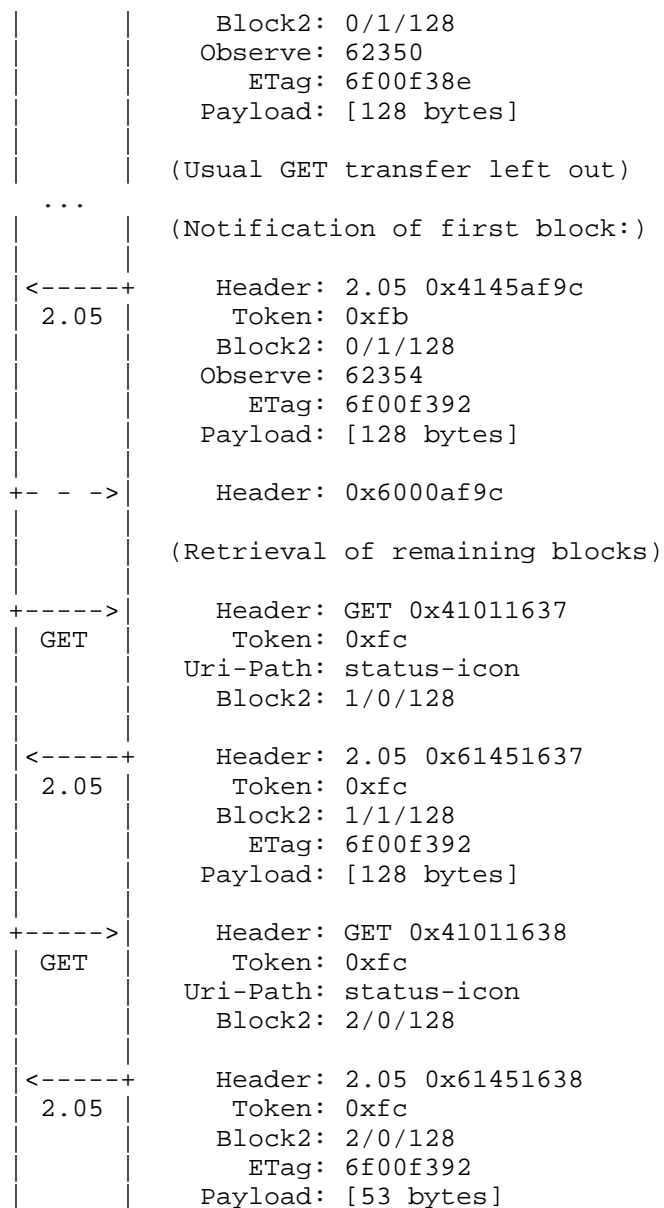


Figure 12: Observe sequence with blockwise response

In the following example, the client also uses early negotiation to limit the block size to 64 bytes.

CLIENT	SERVER
+----->	Header: GET 0x41011636
GET	Token: 0xfb
	Uri-Path: status-icon
	Observe: (empty)
	Block2: 0/0/64
<-----+	Header: 2.05 0x61451636
2.05	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:)
<-----+	Header: 2.05 0x4145af9c
2.05	Token: 0xfb
	Block2: 0/1/64
	Observe: 62354
	ETag: 6f00f392
	Payload: [64 bytes]
+ - - ->	Header: 0x6000af9c
	(Retrieval of remaining blocks)
+----->	Header: GET 0x41011637
GET	Token: 0xfc
	Uri-Path: status-icon
	Block2: 1/0/64
<-----+	Header: 2.05 0x61451637
2.05	Token: 0xfc
	Block2: 1/1/64
	ETag: 6f00f392
	Payload: [64 bytes]
....	
+----->	Header: GET 0x41011638
GET	Token: 0xfc
	Uri-Path: status-icon
	Block2: 4/0/64

<-----+	Header: 2.05 0x61451638
2.05	Token: 0xfc
	Block2: 4/0/64
	ETag: 6f00f392
	Payload: [53 bytes]

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. (Size1 is already defined in [RFC7252] for the narrow case of indicating in 4.13 responses the maximum size of request payload that the server is able and willing to handle.)

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 [RFC7252], 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.

No.	C	U	N	R	Name	Format	Length	Default
60			x		Size1	uint	0-4	(none)
28			x		Size2	uint	0-4	(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 for either Size Option.
- 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, historically there was more variation in how HTTP servers might implement ranges; recently, [RFC7233] has defined that Range header fields received with a request method other than GET are not to be interpreted. So, in general, the CoAP-to-HTTP intermediary will have to try sending the payload of all the blocks of a block-wise transfer for these other methods 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 [RFC7252]:



Number	Name	Reference
23	Block2	[RFCXXXX]
27	Block1	[RFCXXXX]
28	Size2	[RFCXXXX]

Table 3: CoAP Option Numbers

This draft adds the following response code to the CoAP Response Codes registry of [RFC7252]:

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 specific security associations, 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), for instance 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

[RFC7252] 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 [RFC7252] 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-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

### 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](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.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.
- [RFC7233] Fielding, R., Lafon, Y., and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, June 2014.

#### Authors' Addresses

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

Phone: +49-421-218-63921  
Email: [cabo@tzi.org](mailto:cabo@tzi.org)

Zach Shelby (editor)  
ARM  
150 Rose Orchard  
San Jose, CA 95134  
USA

Phone: +1-408-203-9434  
Email: zach.shelby@arm.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.

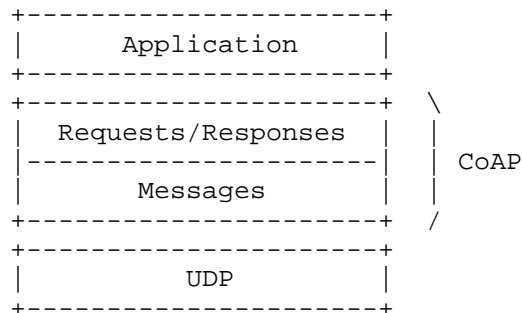


Figure 1: Abstract layering of CoAP

## 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).

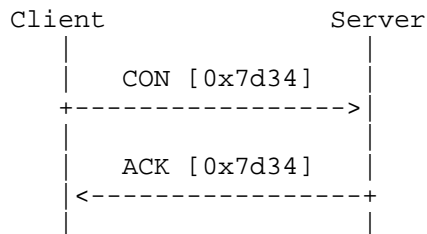


Figure 2: Reliable message transmission

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).

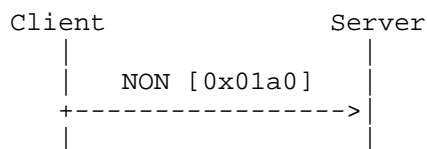


Figure 3: Unreliable message transmission

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.

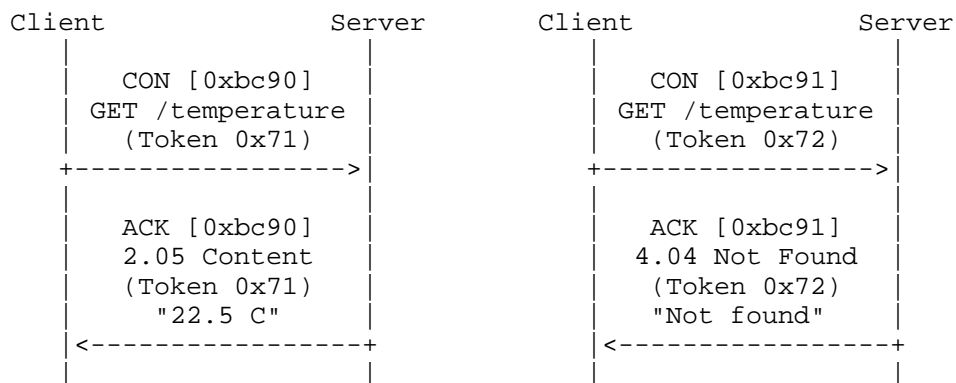


Figure 4: Two GET requests with piggy-backed responses

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.

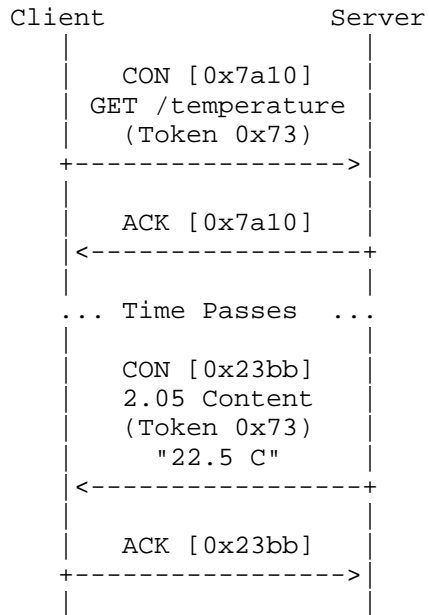
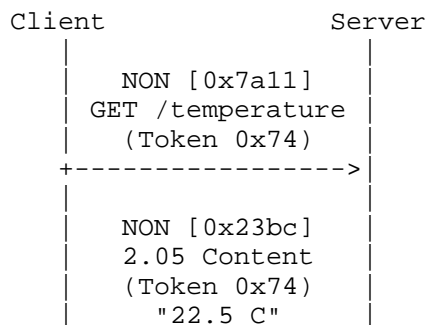


Figure 5: A GET request with a separate response

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.



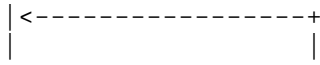


Figure 6: A NON request and response

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.

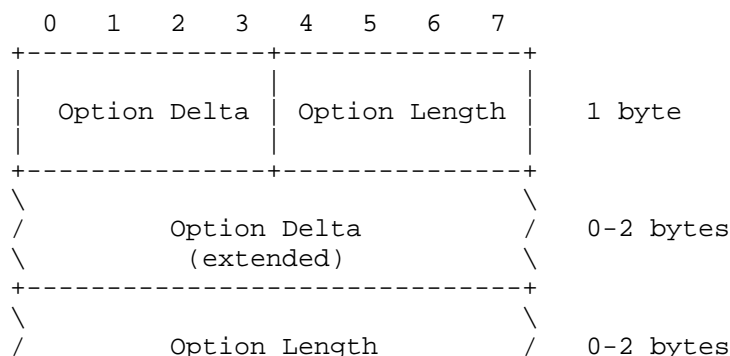
Implementation Note: The byte value 0xFF may also occur within an option length or value, so simple byte-wise scanning for 0xFF is not a viable technique for finding the payload marker. The byte 0xFF has the meaning of a payload marker only where the beginning of another option could occur.

### 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.



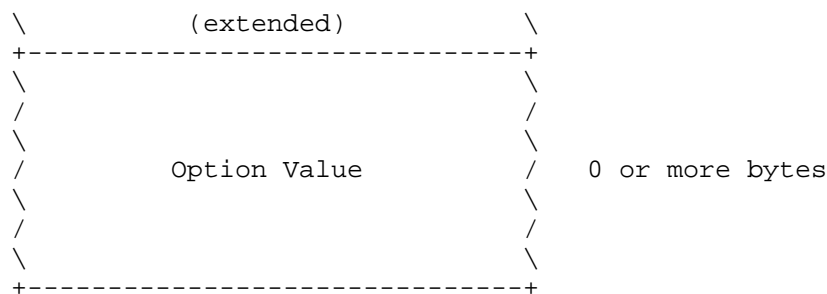


Figure 8: Option Format

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 `(ACK_TIMEOUT * 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.

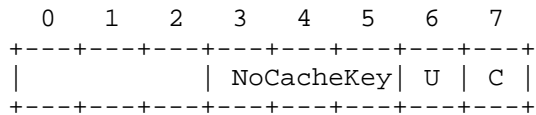


Figure 10: Option Number Mask (Least Significant Byte)

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/%7Esensors/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.

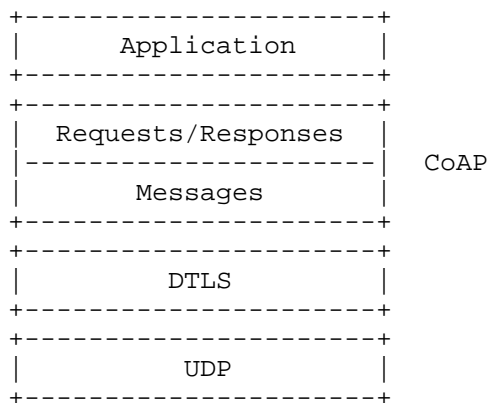


Figure 13: Abstract layering of DTLS-secured CoAP

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.

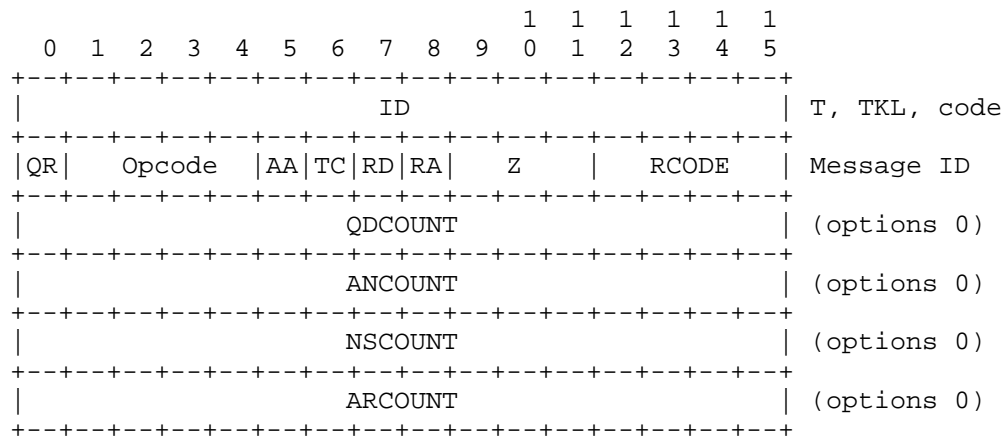


Figure 15: DNS Header vs. CoAP Message

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.mcgregw-tls-aes-ccm-ecc]  
McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM ECC Cipher Suites for TLS", draft-mcgregw-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](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)



Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

[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.

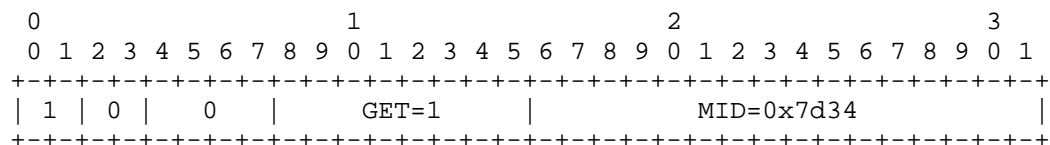
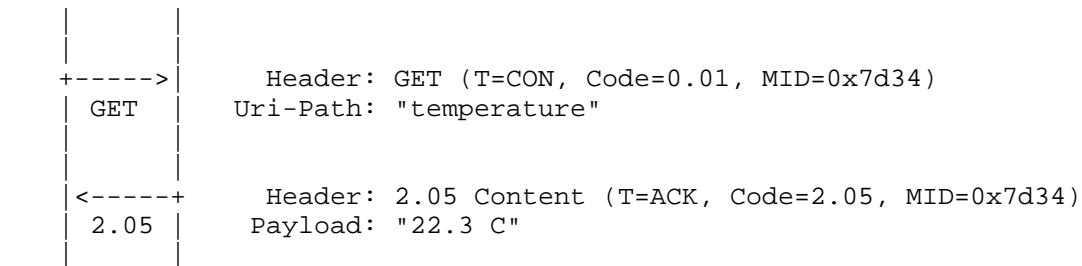
[W3CXMLESEC] Wenning, R., "Report of the XML Security PAG", October 2012, <<http://www.w3.org/2011/xmlsec-pag/pagreport.html>>.

## Appendix A. Examples

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.

Figure 16 shows a basic GET request causing a piggy-backed response: The client sends a Confirmable GET request for the resource `coap://server/temperature` to the server with a Message ID of 0x7d34. The request includes one Uri-Path Option (Delta 0 + 11 = 11, Length 11, Value "temperature"); the Token is left empty. This request is a total of 16 bytes long. A 2.05 (Content) response is returned in the Acknowledgement message that acknowledges the Confirmable request, echoing both the Message ID 0x7d34 and the empty Token value. The response includes a Payload of "22.3 C" and is 11 bytes long.

Client    Server



```

| 11 | 11 | "temperature" (11 B) ...
+-----+
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
+-----+
| 1 | 2 | 0 | 2.05=69 | MID=0x7d34 |
+-----+
| 1 1 1 1 1 1 1 | "22.3 C" (6 B) ...
+-----+

```

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.

Client	Server
+----->	Header: GET (T=CON, Code=0.01, MID=0x7d35)
GET	Token: 0x20
	Uri-Path: "temperature"
<-----+	Header: 2.05 Content (T=ACK, Code=2.05, MID=0x7d35)
2.05	Token: 0x20
	Payload: "22.3 C"

```

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
+-----+
| 1 | 0 | 1 | GET=1 | MID=0x7d35 |
+-----+
| 0x20 |
+-----+
| 11 | 11 | "temperature" (11 B) ...
+-----+

```

```

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
+-----+
| 1 | 2 | 1 | 2.05=69 | MID=0x7d35 |
+-----+

```

```

+-----+
| 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.

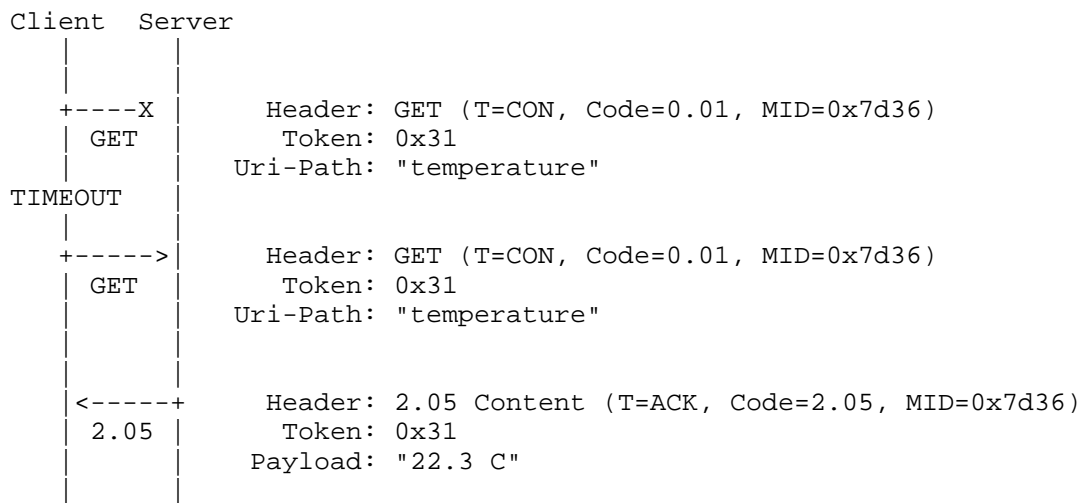
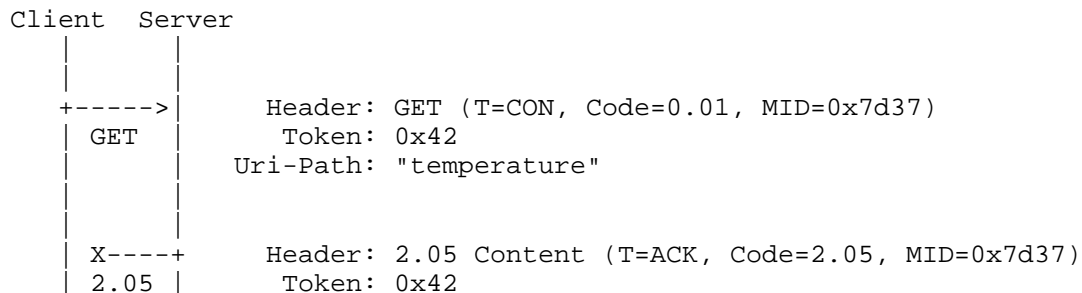


Figure 18: Confirmable request (retransmitted); piggy-backed response

In Figure 19, the first Acknowledgement message from the server to the client is lost. After ACK\_TIMEOUT seconds, the client retransmits the request.



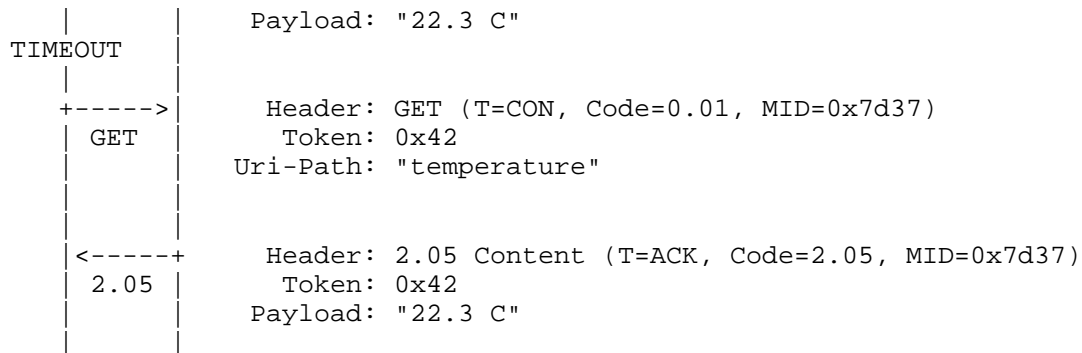


Figure 19: Confirmable request; piggy-backed response (retransmitted)

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.

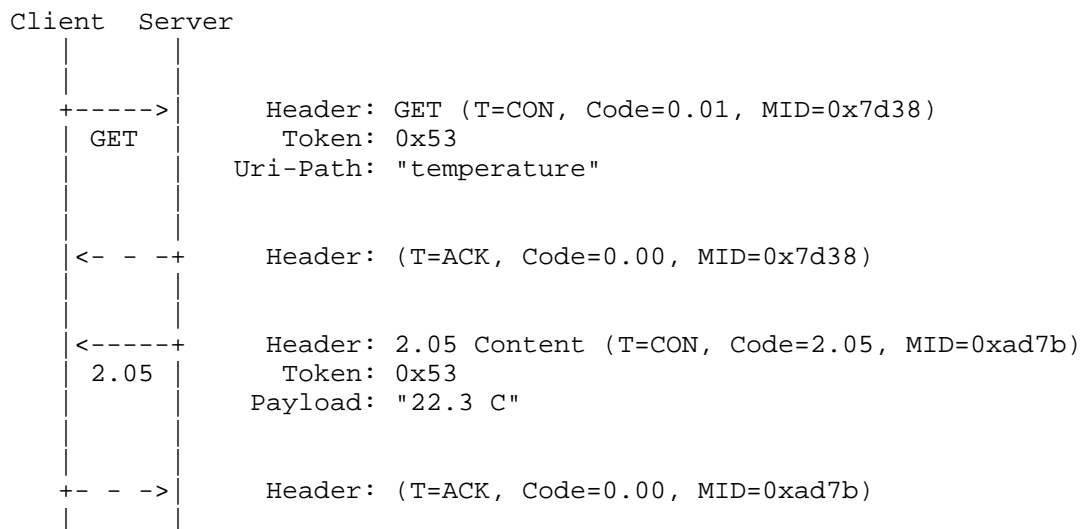


Figure 20: Confirmable request; separate response

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.

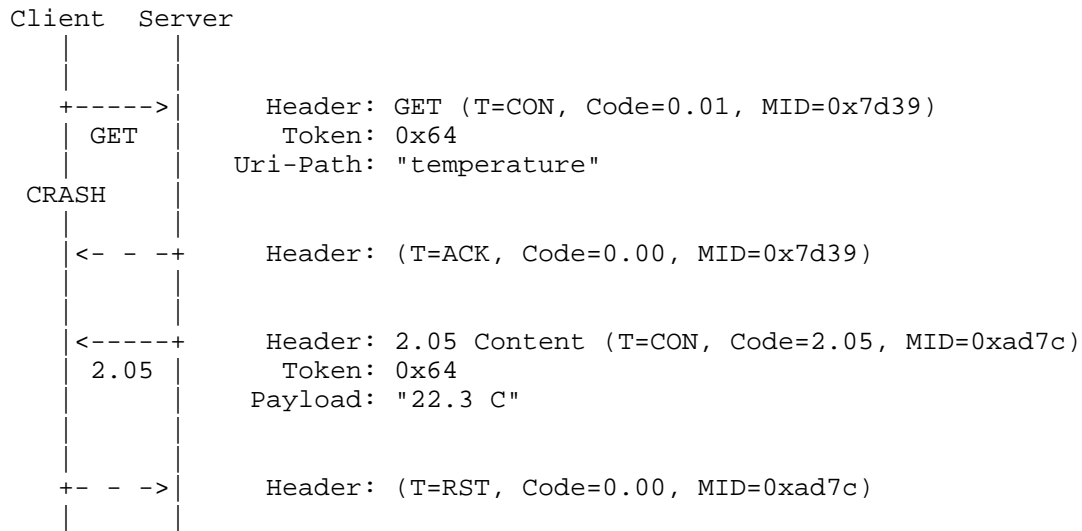


Figure 21: Confirmable request; separate response (unexpected)

Figure 22 shows a basic GET request where the request and the response are Non-confirmable, so both may be lost without notice.

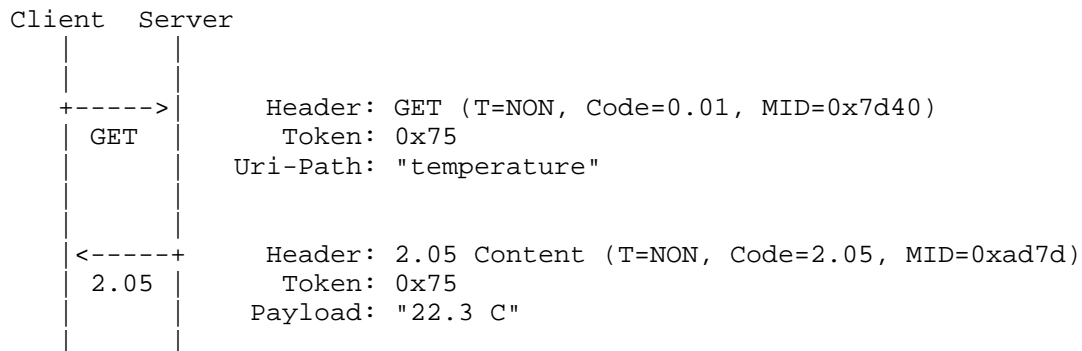


Figure 22: Non-confirmable request; Non-confirmable response

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.

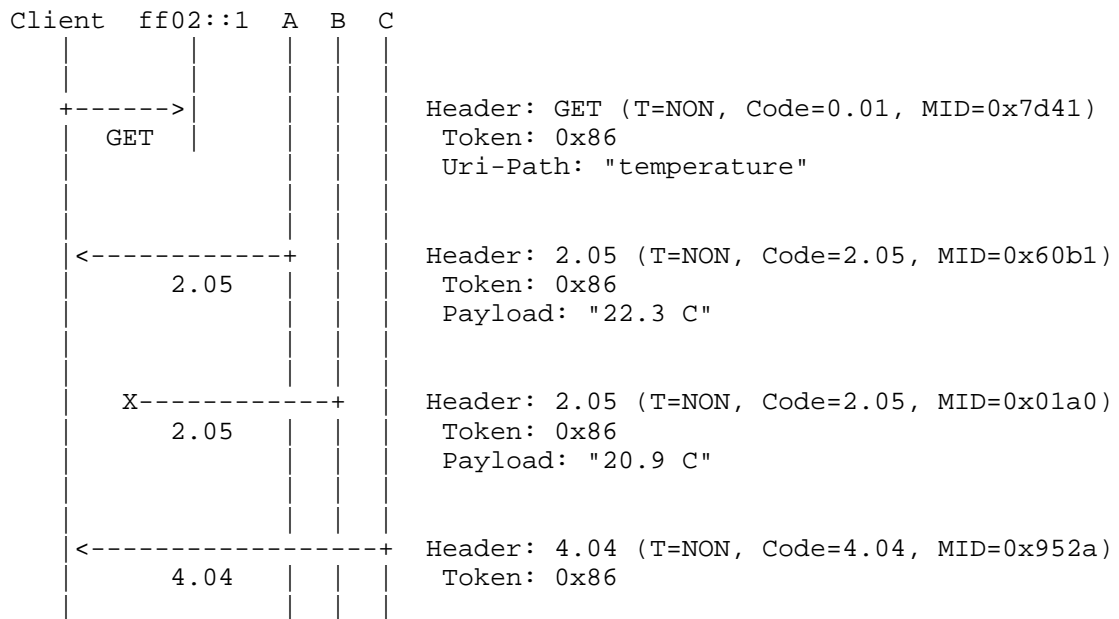


Figure 23: Non-confirmable request (multicast); Non-confirmable response

#### 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: January 21, 2015

A. Rahman, Ed.  
InterDigital Communications, LLC  
E. Dijk, Ed.  
Philips Research  
July 20, 2014

Group Communication for CoAP  
draft-ietf-core-groupcomm-20

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 January 21, 2015.

Copyright Notice

Copyright (c) 2014 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 . . . . .	5
2.1. IP Multicast Background . . . . .	5
2.2. Group Definition and Naming . . . . .	6
2.3. Port and URI Configuration . . . . .	7
2.4. RESTful Methods . . . . .	8
2.5. Request and Response Model . . . . .	9
2.6. Member Discovery . . . . .	10
2.7. Membership Configuration . . . . .	10
2.7.1. Background . . . . .	10
2.7.2. Membership Configuration RESTful Interface . . . . .	10
2.8. Request Acceptance and Response Suppression Rules . . . . .	15
2.9. Congestion Control . . . . .	17
2.10. Proxy Operation . . . . .	18
2.11. Exceptions . . . . .	20
3. Use Cases and Corresponding Protocol Flows . . . . .	20
3.1. Introduction . . . . .	20
3.2. Network Configuration . . . . .	20
3.3. Discovery of Resource Directory . . . . .	22
3.4. Lighting Control . . . . .	24
3.5. Lighting Control in MLD Enabled Network . . . . .	28
3.6. Commissioning the Network Based On Resource Directory . . . . .	29
4. Deployment Guidelines . . . . .	30
4.1. Target Network Topologies . . . . .	30
4.2. Networks Using the MLD Protocol . . . . .	31
4.3. Networks Using RPL Multicast Without MLD . . . . .	31
4.4. Networks Using MPL Forwarding Without MLD . . . . .	32
4.5. 6LoWPAN Specific Guidelines for the 6LBR . . . . .	33
5. Security Considerations . . . . .	33
5.1. Security Configuration . . . . .	33
5.2. Threats . . . . .	34
5.3. Threat Mitigation . . . . .	34
5.3.1. WiFi Scenario . . . . .	34
5.3.2. 6LoWPAN Scenario . . . . .	34
5.3.3. Future Evolution . . . . .	35
5.4. Pervasive Monitoring Considerations . . . . .	35

6.	IANA Considerations . . . . .	35
6.1.	New 'core.gp' Resource Type . . . . .	36
6.2.	New 'coap-group+json' Internet Media Type . . . . .	36
7.	Acknowledgements . . . . .	37
8.	References . . . . .	37
8.1.	Normative References . . . . .	37
8.2.	Informative References . . . . .	39
Appendix A.	Multicast Listener Discovery (MLD) . . . . .	40
Appendix B.	Change Log . . . . .	40
Authors' Addresses	. . . . .	50

## 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 [RFC7252]. CoAP has many similarities to HTTP [RFC7230] but also has some key differences. Constrained devices can be large in numbers, but are often related to each other in function or by 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 can simultaneously get (or set) resources 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 sending CoAP requests on top of IP multicast, and processing the (unicast IP) responses are given in Section 8 of [RFC7252]. The main contribution of this document lies in providing additional guidance for key CoAP group communication concepts. Among the topics covered are group definition, group RESTful methods, and group request and response processing (see Section 2). Also, proxy operation and minimizing network congestion for group communication is discussed (see Section 2). Finally,

specific use cases (see Section 3) and deployment guidelines (see Section 4) for 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 [RFC7252]. In addition, this document defines the following terminology:

#### Group Communication

A source node sends a single application layer (e.g. CoAP) 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 mechanisms for CoAP group communication are UDP/IP multicast for the requests, and unicast UDP/IP for the responses. The network involved may be a constrained network such as a low-power, lossy network.

#### Reliable Group Communication

A special case of group communication where for each destination node it is guaranteed that the node either 1) eventually receives the message sent by the source node, or 2) does not receive the message and the source node is notified of the non-reception event.

#### 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 approach 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]. A complete IP multicast solution may

include support for managing group memberships, and IP multicast routing/forwarding (see Section 2.1).

#### 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).

IP multicast is generally classified as an unreliable service in that packets are not guaranteed to be delivered to each and every member of the group. In other words, it cannot be directly used as a basis for "reliable group communication" as defined in Section 1.3. However, the level of reliability can be increased by employing a



multicast protocol that performs periodic retransmissions as is done for example in MPL.

## 2.2. Group Definition and Naming

A CoAP group is defined as a set of CoAP endpoints, where each endpoint is configured to receive CoAP group communication requests that are sent to the group's associated IP multicast address. The individual response by each endpoint receiver to a CoAP group communication request is always sent back as unicast. An endpoint may be a member of multiple groups. Group membership of an endpoint may dynamically change over time.

All CoAP server nodes SHOULD join the "All CoAP Nodes" multicast group [RFC7252], Section 12.8) by default to enable CoAP discovery. For IPv4, the address is 224.0.1.187 and for IPv6 a server node joins at least both the link-local scoped address FF02::FD and the site-local scoped address FF05::FD. IPv6 addresses of other scopes MAY be enabled.

A CoAP 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 regular CoAP URI syntax [RFC7252].

Note: A group URI is needed to initiate CoAP group communications. For CoAP implementations it is recommended to use the URI composition method of Section 6.5 of [RFC7252] in such way that, from a group URI, a CoAP group communication request is generated.

For sending nodes, it is recommended to use the IP multicast address literal in a group URI. However, in case a group hostname is used, it can be uniquely mapped to an 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:

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 ([RFC1033]). Reverse mapping is important, for example, in troubleshooting to translate IP multicast addresses back to human readable hostnames to show in a diagnostics user interface.

### 2.3. 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. These rules imply that different ports (for the same IP multicast address) cannot be used to specify different CoAP groups.

CoAP 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 group 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 group communication 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 group.
3. Use the default CoAP UDP port (5683).

For a CoAP server node that supports resource discovery, the default port 5683 MUST be supported (Section 7.1 of [RFC7252] for the "All CoAP Nodes" group).

All CoAP group communication requests SHOULD operate on group URI paths in one of the following ways:

1. Pre-configured group 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 ([RFC7320]). 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 group 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 it can use this URI in a CoAP group communication request. For example, a commissioning tool may instruct a sensor device in this way to which target group (group URI) it should report sensor events.

#### 2.4. RESTful Methods

Idempotent CoAP RESTful methods (i.e., GET, PUT, and DELETE) SHOULD be used for group communication, with one exception as follows. A non-idempotent CoAP method (i.e., POST) MAY be used for group communication if the resource being POSTed to has been designed to cope with the unreliable and lossy nature of IP multicast. Note that not all group members are guaranteed to receive the IP multicast request, and the sender cannot readily find out which group members did not receive it.

## 2.5. Request and Response Model

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

A server MAY send back a unicast response to the CoAP group communication request (e.g., response "2.05 Content" to a group GET request). The unicast responses received by the CoAP client may be a mixture of success (e.g., 2.05 Content) and failure (e.g., 4.04 Not Found) codes depending on the individual server processing results. Detailed processing rules for IP multicast request acceptance and unicast response suppression are given in Section 2.8.

A CoAP request sent over IP multicast and any unicast response it causes must take into account the congestion control rules defined in Section 2.9.

The CoAP client can distinguish the origin of multiple server responses by source IP address of the UDP message containing the CoAP response, or any other available unique identifier (e.g. contained in the CoAP payload). In case a CoAP client sent multiple group requests, the responses are as usual matched to a request using the CoAP Token.

For multicast CoAP requests there are additional constraints on the re-use of Token values, compared to the unicast case. In the unicast case, receiving a response effectively frees up its Token value for re-use since no more responses will follow. However, for multicast CoAP the number of responses is not bounded a-priori. Therefore the reception of a response cannot be used as a trigger to "free up" a Token value for re-use. Re-using a Token value too early could lead to protocol error i.e. a wrong response/request matching in the client. Therefore the time between re-use of Token values (for Token values used in multicast requests) must be at least:

$$\text{NON\_LIFETIME} + \text{MAX\_LATENCY} + \text{MAX\_SERVER\_RESPONSE\_DELAY}$$

where NON\_LIFETIME and MAX\_LATENCY are defined in Section 4.8 of [RFC7252]. MAX\_SERVER\_RESPONSE\_DELAY is defined here as the expected maximum response delay over all servers that the client can send a multicast request to. This delay includes the maximum Lease time period as defined in Section 8.2 of [RFC7252]. Using the CoAP default protocol parameters the re-use time becomes at least 250 seconds, but may need to be much longer in practice since there is no time limit defined in CoAP for generation of responses by a server.

## 2.6. Member Discovery

CoAP Groups, and the membership of these groups, can be discovered via the lookup interfaces in the Resource Directory (RD) defined in [I-D.ietf-core-resource-directory]. An example of doing some of these RD lookups is given in Section 3.6.

## 2.7. Membership Configuration

### 2.7.1. 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 using a specific service discovery means. Third, it may be configured by another node (e.g., a commissioning device).

In the first case, the pre-configured group information may be either an IP multicast address or a hostname (FQDN) which is resolved later (during operation) to an 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 case is detailed more in Section 3.6.

In the third case, typical in scenarios such as building control, a dynamic 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.

### 2.7.2. Membership Configuration RESTful Interface

To achieve better interoperability between endpoints from different manufacturers, an OPTIONAL CoAP membership configuration RESTful interface for configuring 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 CoAP methods (GET/PUT/POST/DELETE) only as it is a method of configuring group information in individual endpoints.

Also, a form of authorization (making use of DTLS-secured CoAP [RFC7252]) SHOULD be used such that only authorized controllers are allowed by an endpoint to configure its group membership.

It is important to note that other approaches may be used to configure CoAP endpoints with relevant group information. These alternate approaches may support a subset or super-set of the membership configuration RESTful interface described in this document. For example, a simple interface to just read the endpoint group information may be implemented via a classical Management Information Base (MIB) approach (e.g. following approach of [RFC3433]).

#### 2.7.2.1. CoAP-Group Resource Type and Media Type

CoAP endpoints implementing the membership configuration RESTful interface MUST support the CoAP 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 client uses this media type to query/manage group membership of a CoAP endpoint as defined in the following subsections.

The group configuration resource and its sub-resources have a JSON-based content format (as indicated by the "application/coap-group+json" media type). The resource includes zero or more group membership JSON objects in a format as defined in Section 2.7.2.4. A group membership JSON object contains one or more key/value pairs as defined below. It represents a single IP multicast group membership for the CoAP endpoint.

Examples of four different group membership objects are:

```
{ "n": "All-Devices.floor1.west.bldg6.example.com",  
  "a": "[ff15::4200:f7fe:ed37:abcd]:4567" }  
  
{ "n": "sensors.floor2.east.bldg6.example.com" }  
  
{ "n": "coap-test",  
  "a": "224.0.1.187:56789" }  
  
{ "a": "[ff15::c0a7:15:c001]" }
```

The OPTIONAL "n" key/value pair stands for "name" and identifies the group with a hostname, for example a FQDN. The OPTIONAL "a" key/value pair specifies the IP multicast address (and optionally the port number) of the group. It contains an IPv4 address (in dotted decimal notation) or an IPv6 address. The following ABNF rule can be used for parsing the address, referring to the definitions in Section 6 of [RFC7252] 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). In a response, the "a" key/value pair SHOULD be included if the IP address is known at the time of generating the response, and MUST NOT be included if unknown. If the "a" value is not provided in a request, the "n" value in the same group membership object SHOULD be a valid hostname with optional port number that can be translated to an IP multicast address via DNS resolution, as follows:

```
group-name = host [ ":" port ]
```

If the port number is not provided then it is assumed to be the default CoAP port (5683). 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.

#### 2.7.2.2. Creating a new multicast group membership (POST)

Method: POST  
URI Template: /{+gp}  
Location-URI Template: /{+gp}/{index}  
URI Template Variables:  
    gp - Group Configuration Function Set path (mandatory).  
    index - Group index, SHOULD be a string of 1 or 2 alphanumerical characters. It MUST be generated as locally unique.

Example:

```
Req: POST /coap-group  
    Content-Format: application/coap-group+json  
    { "n": "All-Devices.floor1.west.bldg6.example.com",  
      "a": "[ff15::4200:f7fe:ed37:abcd]:4567" }  
Res: 2.01 Created  
    Location-Path: /coap-group/12
```

For the 'gp' variable it is recommended to use the path "coap-group" by default. If the "a" key/value 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.

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" an IP multicast address to which the endpoint is already subscribed, no change to that subscription is needed.

#### 2.7.2.3. Deleting a single group membership (DELETE)

Method: DELETE  
URI Template: {+location}  
URI Template Variables:  
location - The Location-Path returned by the CoAP server as a result of a successful group creation.

Example:

Req: DELETE /coap-group/12  
Res: 2.02 Deleted

#### 2.7.2.4. Reading all group memberships at once (GET)

A (unicast) GET on the CoAP-group resource returns a JSON object containing multiple keys and values, the keys being group indices and the values the corresponding group objects. Each group object is a group membership JSON object that indicates one IP multicast group membership. So, the group index is used as a JSON key to point to the group membership object, as shown below.

Method: GET  
URI Template: /{+gp}  
URI Template Variables:  
gp - see earlier definition

Example:

Req: GET /coap-group  
Res: 2.05 Content  
Content-Format: application/coap-group+json  
{ "8" :{ "a": "[ff15::4200:f7fe:ed37:14ca]" },  
"11":{ "n": "sensors.floor1.west.bldg6.example.com",  
"a": "[ff15::4200:f7fe:ed37:25cb]" },  
"12":{ "n": "All-Devices.floor1.west.bldg6.example.com",  
"a": "[ff15::4200:f7fe:ed37:abcd]:4567" }  
}

Note: the returned IPv6 address may be a different string from the one originally submitted in group membership creation, due to different choices in IPv6 string representation formatting that may be allowed for the same address (see [RFC5952]).



## 2.7.2.5. Reading a single group membership (GET)

Method: GET  
URI Template 1: {+location}  
URI Template 2: /{+gp}/{index}  
URI Template Variables:  
location, gp, index - see earlier definitions

## Example:

Req: GET /coap-group/12  
Res: 2.05 Content  
Content-Format: application/coap-group+json  
{ "n": "All-Devices.floor1.west.bldg6.example.com",  
 "a": "[ff15::4200:f7fe:ed37:abcd]:4567" }

## 2.7.2.6. Creating/updating all group memberships at once (PUT)

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. This operation SHOULD only be used to delete or update group membership objects for which the CoAP client, invoking this operation, is responsible. The responsibility is based on application level knowledge. For example, a commissioning tool will be responsible for any group membership objects that it created.

Method: PUT  
URI Template: /{+gp}  
URI Template Variables:  
gp - see earlier definition

## Example: (replacing all existing group memberships with two new groups)

Req: PUT /coap-group  
Content-Format: application/coap-group+json  
{ "1":{ "a": "[ff15::4200:f7fe:ed37:1234]" },  
 "2":{ "a": "[ff15::4200:f7fe:ed37:5678]" }  
}  
Res: 2.04 Changed

## Example: (clearing all group memberships at once)

Req: PUT /coap-group  
Content-Format: application/coap-group+json  
{ }  
Res: 2.04 Changed

After a successful PUT on the Group configuration resource, the endpoint MUST effect registration to any new IP multicast group(s) and de-registration from any previous IP multicast group(s), i.e. not

anymore present in the new memberships, as soon as possible. Also it MUST take into account the group indices present in the new resource during the generation of any new unique group indices in the future.

#### 2.7.2.7. Updating a single group membership (PUT)

A (unicast) PUT with a group membership JSON object will replace an existing group membership in the endpoint with the new one defined in the PUT request. This can be used to update the group membership.

Method: PUT

URI Template 1: {+location}

URI Template 2: /{+gp}/{index}

URI Template Variables:

location, gp, index - see earlier definitions

Example: (group name and IP multicast port change)

Req: PUT /coap-group/12

Content-Format: application/coap-group+json

```
{ "n": "All-My-Devices.floor1.west.bldg6.example.com",  
  "a": "[ff15::4200:f7fe:ed37:abcd]" }
```

Res: 2.04 Changed

After a successful PUT on the Group configuration resource, the endpoint MUST effect registration to any new IP multicast group(s) and de-registration from any previous IP multicast group(s), i.e. not anymore present in the new membership, as soon as possible.

#### 2.8. Request Acceptance and Response Suppression Rules

CoAP [RFC7252] and CoRE Link Format [RFC6690] define normative behaviors for:

1. IP multicast request acceptance - in which cases a CoAP request is accepted and executed, and when not.
2. IP 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 an IP multicast CoAP request. This section first summarizes these normative behaviors and then presents additional guidelines for response suppression. Also a number of IP 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 IP multicast by making use of APIs such as IPV6\_RECVPKTINFO [RFC3542].

For IP multicast request acceptance, the REQUIRED behaviors are:

- o A server SHOULD NOT accept an IP multicast request that cannot be "authenticated" in some way (cryptographically or by some multicast boundary limiting the potential sources) [RFC7252]. See Section 5.3 for examples of multicast boundary limiting methods.
- o A server SHOULD NOT accept an IP 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 an IP multicast request that acts on a specific resource for which IP multicast support is not required. (Note that for the resource `"/.well-known/core"`, IP multicast support is required if "multicast resource discovery" is supported as specified in section 1.2.1 of [RFC6690]). Implementers are advised to disable IP multicast support by default on any other resource, until explicitly enabled by an application or by configuration.)
- o Otherwise accept the IP multicast request.

For IP multicast response suppression, the REQUIRED behaviors are:

- o A server SHOULD NOT respond to an IP 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 an IP multicast request, if there's nothing useful to respond (e.g., error or empty response).
- o Otherwise respond to the IP 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 IP 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 CoAP 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 group communication 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. Note that in this case the client implements a "reliable group communication" (as defined in Section 1.3) function using additional, non-standardized functions above the CoAP layer.
- o IP multicast firmware update by sending blocks of data: Suppress all 2.xx and 5.xx responses. After having sent all IP 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 group URI query: Suppress all 2.05 responses with empty payload (i.e., if a query produces no matching results).

## 2.9. Congestion Control

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

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

- o A server MAY choose not to respond to an IP multicast request if there's nothing useful to respond (e.g., error or empty response). See Section 2.8 for more detailed guidelines on response suppression.
- o A server SHOULD limit the support for IP multicast requests to specific resources where multicast operation is required.

- o An IP multicast request **MUST** be Non-confirmable.
- o A response to an IP multicast request **SHOULD** be Non-confirmable (Section 5.2.3 of [RFC7252]).
- o A server does not respond immediately to an IP 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 group communication GET for resources that are small. For example, the payload of the response is limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size so it fits into a single link-layer frame in case 6LoWPAN [RFC4944] is used.
- o A server can minimize the payload length in response to a group communication 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 group communication 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 an IP multicast CoAP request to `"/.well-known/core"` **SHOULD** support core-block.
- o A client should use CoAP group communication with the smallest possible IP multicast scope that fulfills the application needs. As an example, site-local scope is always preferred over global scope IP multicast if this fulfills the application needs.

More guidelines specific to use of CoAP in 6LoWPAN networks [RFC4944] are given in Section 4.5.

## 2.10. Proxy Operation

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

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

- o Aggregation of (unicast) responses to a CoAP group communication request in a proxy is difficult. This is because the proxy does not know how many members there are in the group, or how many group members will actually respond. Also the proxy does not know how long to wait before deciding to send back the aggregated response to the client.
- 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 [RFC7252], the proxy would simply forward all the individual (unicast) responses to a CoAP 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 group URI 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 an IP 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 IP 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.

### 2.11. Exceptions

CoAP group communication using IP multicast offers improved network efficiency and latency amongst other benefits. However, group communication may not always be implementable in a given network. The primary reason for this will be that IP multicast is not (fully) supported in the network.

For example, if only the RPL protocol [RFC6550] is used in a network with its optional multicast support disabled, there will be no IP multicast routing at all. The only multicast that works in this case is link-local IPv6 multicast. This implies that any CoAP group communication request will be delivered to nodes on the local link only, regardless of the scope value used in the IPv6 destination address.

## 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 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 IPv6 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.



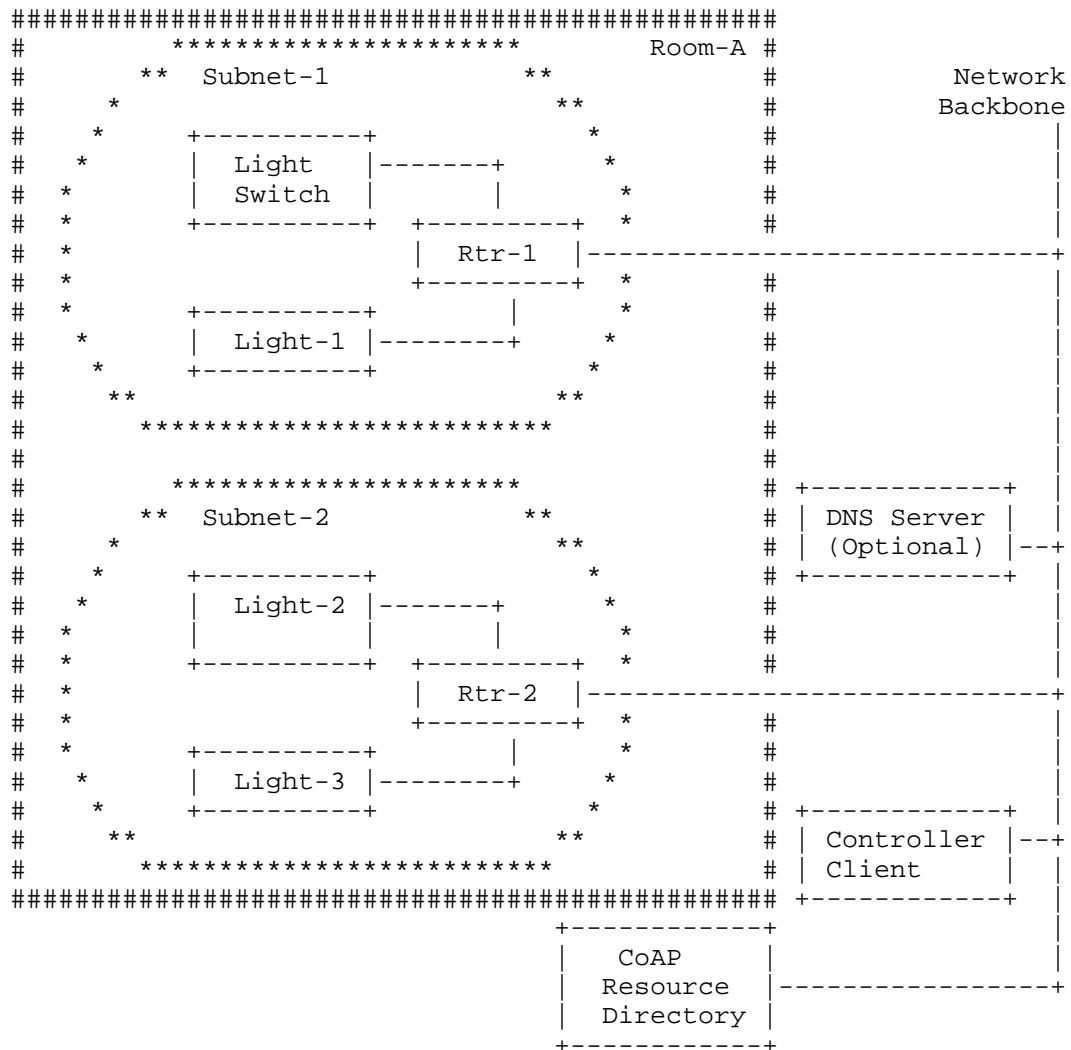


Figure 1: Network Topology of a Large Room (Room-A)

### 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 group communication GET request (with the `"/.well-known/core?rt=core.rd"` request URI) to the site-local "All CoAP Nodes" multicast address (`FF05:::FD`).
- o This multicast message will then go to each node in subnet-2. Rtr-2 will then forward it into to the Network Backbone where it will be received by the CoAP RD. All other nodes in subnet-2 will ignore the group communication GET request because it is qualified by the query string `"?rt=core.rd"` (which indicates it should only be processed by the endpoint if it contains a resource of type `"core.rd"`).
- 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, CoAP group communication can be used in a similar fashion as in the above use case. For example, Link-Local (LL), admin-local or site-local scoped discovery can be done this way.

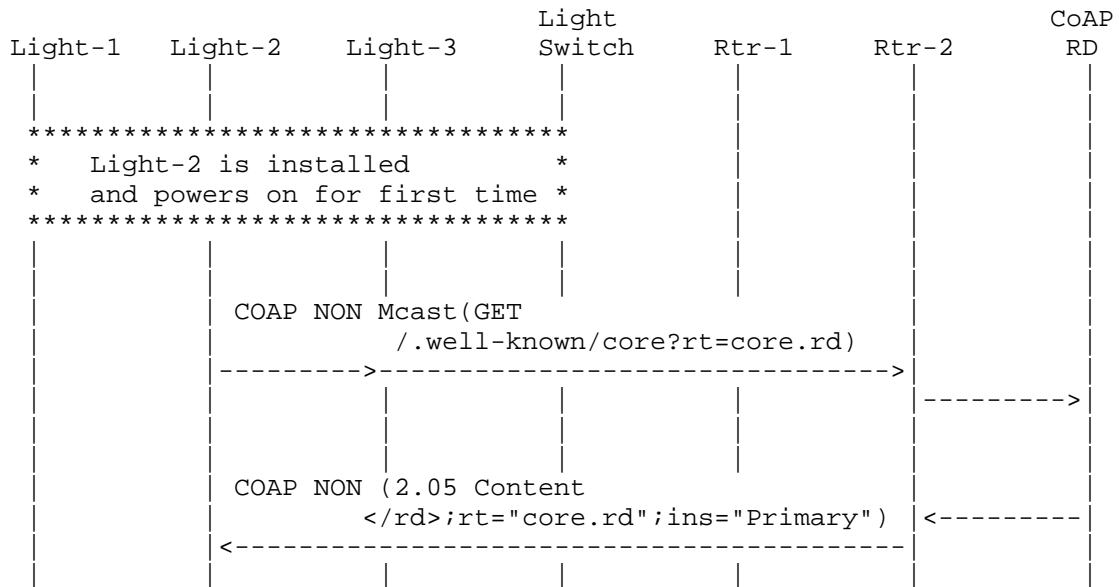


Figure 2: Resource Directory Discovery via Multicast Request

### 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 IP multicast CoAP requests related to lighting control. (See Section 2.8 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 IP multicast request with (unicast) CoAP NON responses. There are two success responses and one 5.00 error response. In this particular case, the Light Switch does not check that all Lights in the group received the IP multicast request by examining the responses. This is because the Light Switch is not configured with an exhaustive list of the IP addresses of all Lights belonging to the group. However, based on received error responses it could take additional action such as logging a fault or alerting the user via its LCD display. In case a CoAP message is delivered multiple times to a Light, the subsequent CoAP messages can be filtered out as duplicates, based on the CoAP Message ID.

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

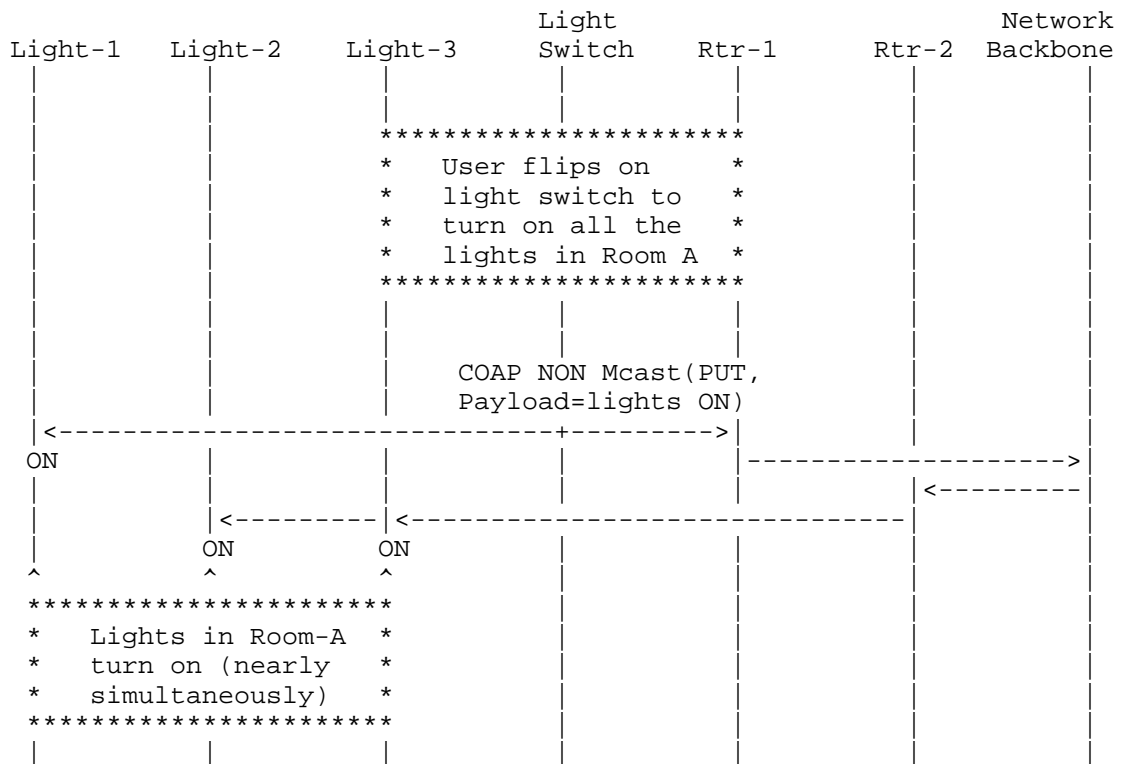


Figure 3: Light Switch Sends Multicast Control Message

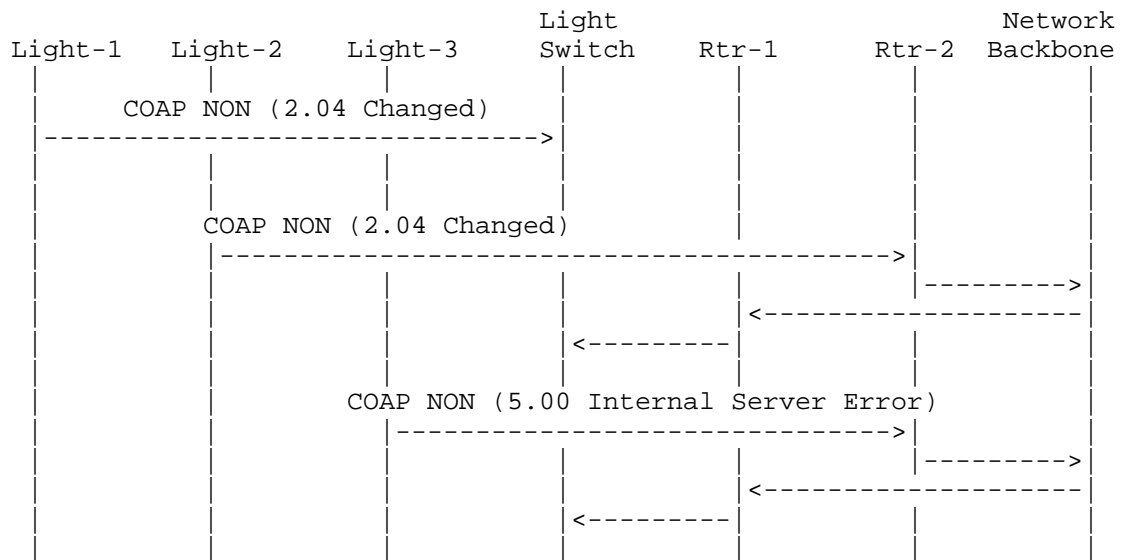


Figure 4: Lights (Optionally) Respond to Multicast CoAP Request

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 group communication request to turn on all lights in Room-A. Every Light sends back a CoAP response to the Controller after being turned on.

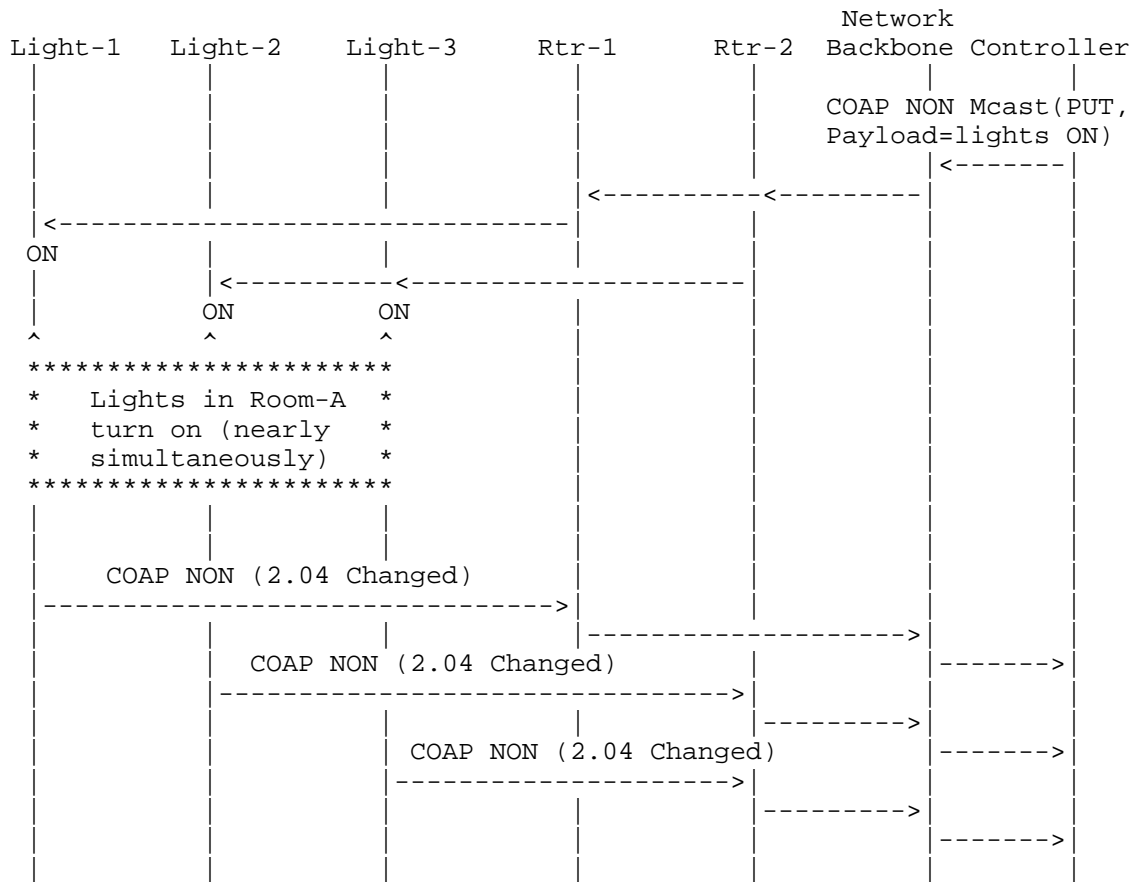


Figure 5: Controller On Backbone Sends Multicast Control Message

### 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 IP 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.

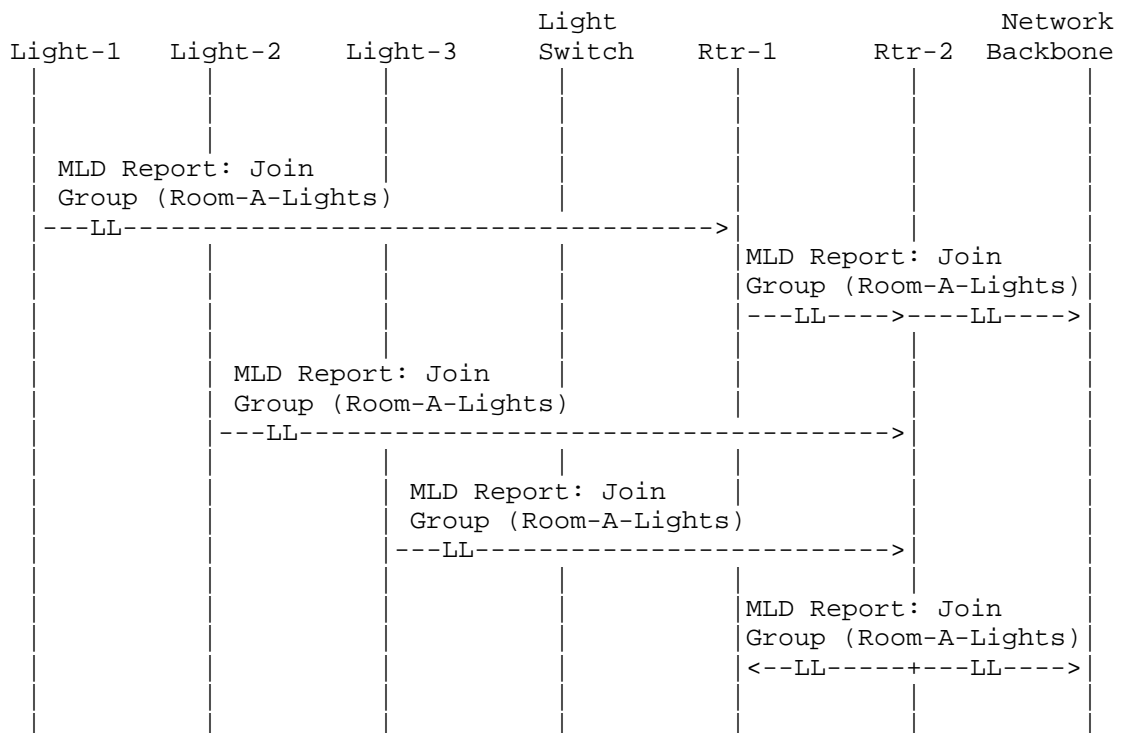


Figure 6: Joining Lighting Groups Using MLD

### 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 IP 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 IP multicast address. For each light in the list the tool learns the IP address of the light and instructs the RD with three (unicast) 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 IP multicast address in the Light endpoints with, for example, the (unicast) POST command discussed in Section 2.7.2.2.

The light switch can discover the group in RD and thus learn the IP multicast address of the group. The light switch will use this address to send CoAP group communication requests to the members of the group. When the message arrives the Lights should recognize the IP 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 both features.

#### 4.2. Networks Using the MLD Protocol

CoAP nodes that are IP hosts (i.e., not IP routers) are generally unaware of the specific IP 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 IP multicast routers which IP 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 IPv6 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 RPL Routers, or are RPL Leaf Nodes, to advertise IP multicast group membership to parent routers. Then, the RPL protocol is used to route IP multicast CoAP requests over multiple hops to the correct CoAP servers.

The same DAO 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" helps to avoid congestion of a 6LoWPAN subnet by IP multicast traffic from the traditional backbone IP 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 IPv6 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 Multicast Listener Discovery (MLD) is not implemented in the network, for example due to resource limitations in an LLN.

The purpose of MPL is to let a predefined group of Forwarders collectively work towards the goal of distributing an IPv6 multicast packet throughout an MPL Domain. (A Forwarder node may be associated to multiple MPL Domains at the same time.) 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 just outside the MPL Domain, the request will not be propagated by MPL. An example of such a case is the network topology of Figure 1 where the Subnets are 6LoWPAN subnets and per 6LoWPAN subnet one Realm-Local ([I-D.droms-6man-multicast-scopes]) MPL Domain is defined. The backbone network in this case is not part of any MPL Domain.

This situation can become a problem in building control use cases. For example, when the Controller Client needs to send a single IP multicast request to the 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 IP 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 each of the two MPL Domains. Or in above example, create just a single MPL Domain that includes both 6LoWPAN subnets plus the backbone link, which is possible since MPL is not tied to a single link-layer technology.
2. Manual configuration of edge router(s) as MPL Seed(s) for specific IP multicast traffic. E.g. in above example, first configure Rtr-1 and Rtr-2 to act as MLD Address Listeners for the Room-A-Lights IP multicast group. This step allows any (other) routers on the backbone to learn that at least one node on the backbone link is interested to receive any IP multicast traffic

to Room-A-Lights. Second, configure both routers to "inject" any IP multicast packets destined to group Room-A-Lights into the (Realm-Local) MPL Domain that is associated to that router. Third, configure both routers to propagate any IPv6 multicast packets originating from within their associated MPL Domain to the backbone, if at least one node on the backbone has indicated interest to receive such IPv6 packets (for which MLD is used on the backbone).

3. Use an additional protocol/mechanism for injection of IP multicast traffic from outside an MPL Domain into that MPL Domain, based on IP multicast group subscriptions of Forwarders within the MPL Domain. Such protocol is currently not defined in [I-D.ietf-roll-trickle-mcast].

Concluding, MPL can be used directly in case all sources of IP 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 on how IP multicast packets can be injected from outside into an MPL Domain.

#### 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 [RFC7252], 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 Will use "coap" scheme mode. The "coaps" scheme should only be used when a future group security solution is developed (see also Section 5.3.3).

## 5.2. Threats

Essentially the above configuration means that there is currently 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 [RFC7252] for IP multicast.

## 5.3. Threat Mitigation

The [RFC7252] identifies various threat mitigation techniques for CoAP group communication. 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 IP multicast filters set so that it enforces multicast scope boundaries to isolate local multicast groups from the rest of the Internet (e.g., as per [RFC6092]). In addition, the scope of the IP multicast should be set to be site-local or smaller 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 communications (see [I-D.keoh-dice-multicast-security]) or similar approaches should be considered. This will allow introduction of a secure mode of CoAP group communication, and use of the "coaps" scheme for that purpose.

### 5.4. Pervasive Monitoring Considerations

A key additional threat consideration for group communication is pointed to by [RFC7258] which warns of the dangers of pervasive monitoring. CoAP group communication which is built on top of IP multicast should pay particular heed to these dangers. This is because IP multicast is easier to intercept (e.g. and to secretly record) compared to unicast traffic. Also, CoAP traffic is meant for the Internet of Things. This means that CoAP traffic is often used for the control and monitoring of critical infrastructure (e.g. lights, alarms, etc.) which may be prime targets for attack.

For example, an attacker may attempt to record all the CoAP traffic going over the smart grid (i.e. networked electrical utility) of a country and try to determine critical control nodes for further attacks. CoAP multicast traffic is inherently more vulnerable (compared to a unicast packet) as the same packet may be replicated over many links so there is a much higher probability of it getting captured by a pervasive monitoring system.

One useful mitigation to pervasive monitoring is to restrict the scope of the IP multicast to the minimal scope that fulfills the application need. Thus, for example, site-local IP multicast scope is always preferred over global scope IP multicast if this fulfills the application needs. This approach has the added advantage that it coincides with the guidelines for minimizing congestion control (see Section 2.9).

In the future, even if all the CoAP multicast traffic is encrypted (e.g. [I-D.keoh-dice-multicast-security]), an attacker may still attempt to capture the traffic and perform an off-line attack. Though of course having the multicast traffic protected is always desirable as it significantly raises the cost to an attacker (e.g. to break the encryption) versus unprotected multicast traffic.

## 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 2.7.2.

### 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 [RFC7252].

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 (re-)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 a form of authorization (making use of DTLS-secured CoAP) be used such that only authorized controllers are allowed by an endpoint 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 2.7.2.

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, Thomas Fossati, Bjoern Hoehrmann, Matthias Kovatsch, Guang Lu, Salvatore Loreto, Kerry Lynn, Andrew McGregor, Dale Seed, Zach Shelby, Peter van der Stok, Gengyu Wei, 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.
- [RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, October 2002.
- [RFC3433] Bierman, A., Romascanu, D., and K. Norseth, "Entity Sensor Management Information Base", RFC 3433, December 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.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 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.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, May 2014.
- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, July 2014.

## 8.2. Informative References

- [RFC1033] Lottor, M., "Domain administrators operations guide", RFC 1033, November 1987.
- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-15 (work in progress), July 2014.
- [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-09 (work in progress), April 2014.
- [I-D.keoh-dice-multicast-security]  
Keoh, S., Kumar, S., Garcia-Morchon, O., Dijk, E., and A. Rahman, "DTLS-based Multicast Security in Constrained Environments", draft-keoh-dice-multicast-security-08 (work in progress), July 2014.
- [I-D.ietf-core-resource-directory]  
Shelby, Z., Bormann, C., and S. Krco, "CoRE Resource Directory", draft-ietf-core-resource-directory-01 (work in progress), December 2013.

[I-D.droms-6man-multicast-scopes]

Droms, R., "IPv6 Multicast Address Scopes", draft-droms-6man-multicast-scopes-02 (work in progress), July 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 IP 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 IP multicast listeners on directly attached links, and to discover which IP multicast addresses are of interest to those listening nodes. MLD was specifically designed to cope with fairly dynamic situations in which IP 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-19 to ietf-20:

- o Replaced obsolete reference [RFC 2616] by [RFC 7230].
- o Replaced outdated reference draft-ietf-appsawg-uri-get-off-my-lawn by [RFC 7320] and moved to Normative reference.
- o Replaced outdated reference draft-ietf-core-coap by [RFC 7252].
- o Moved [RFC 1033] to Informative reference.
- o Updated to latest revision numbers for informative draft references by regenerating file through xml2rfc tool.
- o Re-ran IETF spell check tool and corrected some minor spelling errors.
- o Various minor editorial updates.

Changes from ietf-18 to ietf-19:

- o Added guideline on Token value re-use in section 2.5.
- o Updated section 5.1 (Security Configuration) and 5.3.3 (Future Security Evolution) to point to latest security developments happening in DICE WG for support of group security.
- o Added Pervasive Monitoring considerations in section 5.4.
- o Various editorial updates for improved readability.

Changes from ietf-17 to ietf-18:

- o Extensive editorial updates based on WGLC comments by Thomas Fossati and Gengyu Wei.
- o Addressed ticket #361: Added text for single membership PUT section 2.7.2.7 (Updating a single group membership (PUT)).
- o Addressed ticket #360: Added text for server duties upon all-at-once PUT section 2.7.2.6 (Creating/updating all group memberships at once (PUT)).
- o Addressed ticket #359: Fixed requirements text for Section 2.7.2.2 (Creating a new multicast group membership (POST)).
- o Addressed ticket #358: Fixed requirements text for Section 2.7.2.1 (CoAP-Group Resource Type and Media Type).
- o Addressed ticket #357: Added that "IPv6 addresses of other scopes MAY be enabled" in section 2.2 (Group Definition and Naming).
- o Various editorial updates for improved readability.

Changes from ietf-16 to ietf-17:

- o Added guidelines on joining of IPv6/IPv4 "All CoAP Nodes" multicast addresses (#356).
- o Added MUST support default port in case multicast discovery is available.
- o In section 2.1 (IP Multicast Background), clarified that IP multicast is not guaranteed and referenced a definition of Reliable Group Communication (#355).

- o Added section 2.5 (Messages and Responses) to clarify how responses are identified and how Token/MID are used in multicast CoAP.
- o In section 2.6.2 (RESTful Interface for Configuring Group Memberships), clarified that group management interface is an optional approach for dynamic commissioning and that other approaches can also be used if desired.
- o Updated section 2.6.2 (RESTful Interface for Configuring Group Memberships) to allow deletion of individual group memberships (#354).
- o Various editorial updates based on comments by Peter van der Stok. Removed reference to expired draft-vanderstok-core-dna at request of its author.
- o Various editorial updates for improved readability.

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 [RFC 3542]".
- 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 draft-ietf-core-coap in section 2.2 (Scope).
- o Removed IPSec related requirement, as IPSec is not part of draft-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 draft-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 draft-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: January 5, 2015

A. Castellani  
University of Padova  
S. Loreto  
Ericsson  
A. Rahman  
InterDigital Communications, LLC  
T. Fossati  
Alcatel-Lucent  
E. Dijk  
Philips Research  
July 4, 2014

Guidelines for HTTP-CoAP Mapping Implementations  
draft-ietf-core-http-mapping-04

Abstract

This draft provides reference information for HTTP-CoAP protocol translation proxy implementation, focusing on the reverse proxy case. It details deployment options, defines a template 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 January 5, 2015.

Copyright Notice

Copyright (c) 2014 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. Terminology . . . . .	4
3. Cross-Protocol Usage of URIs . . . . .	4
4. Use Cases . . . . .	5
5. URI Mapping . . . . .	5
5.1. URI Terminology . . . . .	6
5.2. Default Mapping . . . . .	6
5.2.1. Optional scheme . . . . .	7
5.2.2. Encoding Caveats . . . . .	7
5.3. URI Mapping Template . . . . .	7
5.3.1. Simple Form . . . . .	8
5.3.2. Enhanced Form . . . . .	9
5.4. Discovery . . . . .	10
5.4.1. Examples . . . . .	11
6. HTTP-CoAP Reverse Proxy . . . . .	12
6.1. Proxy Placement . . . . .	13
6.2. Response Code Translations . . . . .	14
6.3. Media Type mapping . . . . .	16
6.3.1. Loose Media Type Mapping . . . . .	18
6.3.2. Internet Media Type to Content Format Mapping Algorithm . . . . .	18
6.3.3. Content Transcoding . . . . .	19
6.4. Caching and Congestion Control . . . . .	20
6.5. Cache Refresh via Observe . . . . .	21
6.6. Use of CoAP Blockwise Transfer . . . . .	21
6.7. Security Translation . . . . .	22
6.8. Other guidelines . . . . .	22
7. IANA Considerations . . . . .	23
8. Security Considerations . . . . .	23
8.1. Traffic overflow . . . . .	24
8.2. Handling Secured Exchanges . . . . .	24
8.3. URI Mapping . . . . .	25
9. Acknowledgements . . . . .	25
10. References . . . . .	25
10.1. Normative References . . . . .	25
10.2. Informative References . . . . .	26
Appendix A. Change Log . . . . .	27
Authors' Addresses . . . . .	28

## 1. Introduction

CoAP [RFC7252] 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 [RFC7230] through an intermediary proxy which performs cross-protocol conversion.

Section 10 of [RFC7252] 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 describes the properties of the HTTP-to-CoAP reverse proxy;
- o Section 8 discusses possible security impact related to HTTP-CoAP protocol mapping.



## 2. Terminology

This document assumes readers are familiar with the terms Reverse Proxy as defined in [RFC7230] and Interception Proxy as defined in [RFC3040]. In addition, the following terms are defined:

HC 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" [RFC7230] 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 only support 'http' and 'https' schemes and cannot directly access CoAP servers (which support 'coap' and/or 'coaps'). In this situation, communication is enabled by a HC 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: Any smartphone can access directly a home sensor using an authenticated 'https' request, if its home router contains a HTTP-CoAP proxy. For this use-case an HTML5 application can be built providing a friendlier UI to the user.
2. Legacy building control application without CoAP: A building control application that uses HTTP but not CoAP, can check the status of sensors and/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. In this way even unattended HTTP clients, such as web crawlers, may index sensor data as regular web pages.

#### 5. URI Mapping

Though, in principle, a CoAP URI could be directly used by a HTTP user agent to de-reference a CoAP resource through a HC Proxy, the reality is that all major web browsers and command line tools do not allow making HTTP requests using URIs with a scheme different from "http" or "https".

Thus, there is a need for web applications to "pack" a CoAP URI into a HTTP URI so that it can be (non-destructively) transported from the user agent to the HC Proxy. The HC Proxy can then "unpack" the CoAP

URI and finally de-reference it via a CoAP request to the target Server.

URI Mapping is the process through which the URI of a CoAP resource is transformed in to an HTTP URI so that:

- o the requesting HTTP user agent can handle it;
- o the receiving HC Proxy can extract the intended CoAP URI unambiguously.

To this end, the remainder of this section will identify:

- o the default mechanism to map a CoAP URI into a HTTP URI;
- o the URI template format to express a class of CoAP-HTTP URI mapping functions;
- o the discovery mechanism based on [RFC6690] through which clients of a HC Proxy can dynamically discover information about the supported URI Mapping Template(s), as well as the base URI where the HC Proxy function is anchored.

#### 5.1. URI Terminology

In the remainder of this section, the following terms will be used with a distinctive meaning:

Target CoAP URI:

URI which refers to the (final) CoAP resource that has to be de-referenced. It conforms to syntax defined in section 6 of [RFC7252]. Specifically, it has a scheme of "coap" or "coaps".

Hosting HTTP URI:

URI that conforms to syntax in section 2.7 of [RFC7230]. Its authority component refers to an HC Proxy, whereas path (and query) component(s) embed the information used by an HC Proxy to extract the Target CoAP URI.

#### 5.2. Default Mapping

The default is for the Target CoAP URI to be appended as-is to a base URI provided by the HC Proxy to form the Hosting HTTP URI.

For example: given a base URI `http://p.example.com/hc` and a Target CoAP URI `coap://s.example.com/light`, the resulting Hosting HTTP URI would be `http://p.example.com/hc/coap://s.example.com/light`.

Provided a correct Target CoAP URI, the Hosting HTTP URI resulting from the default mapping is always syntactically correct. Furthermore, the Target CoAP URI can always be extracted in an unambiguous way from the Hosting HTTP URI. Also worth noting that, using the default mapping, a query component in the target CoAP resource URI is naturally encoded into the query component of the Hosting URI, e.g.: `coap://s.example.com/light?dim=5` becomes `http://p.example.com/hc/coap://s.example.com/light?dim=5`.

There is no default for the base URI. Therefore it is either known in advance, e.g. as a configuration preset, or dynamically discovered using the mechanism described in Section 5.4.

The default URI mapping function is RECOMMENDED to be implemented and activated by default in a HC Proxy, unless there are valid reasons, e.g. application specific, to use a different mapping function.

#### 5.2.1. Optional scheme

When found in a Hosting HTTP URI, the scheme (i.e. "coap" or "coaps"), the scheme component delimiter (":"), and the double slash ("//") preceding the authority MAY be omitted. In such case, a local default - not defined by this document - applies.

So, `http://p.example.com/hc/s.coap.example.com/foo` could either represent the target `coap://s.coap.example.com/foo` or `coaps://s.coap.example.com/foo` depending on application specific presets.

#### 5.2.2. Encoding Caveats

When the authority of the Target CoAP URI is given as an IPv6 address, then the surrounding square brackets MUST be percent-encoded in the Hosting HTTP URI, in order to comply with the syntax defined in Section 3.3. of [RFC3986] for a URI path segment. E.g.: `coap://[2001:db8::1]/light?on` becomes `http://p.example.com/hc/coap://%5B2001:db8::1%5D/light?on`.

Everything else can be safely copied verbatim from the Target CoAP URI to the Hosting HTTP URI.

#### 5.3. URI Mapping Template

This section defines a format for the URI template used by a HC Proxy to inform its clients about the expected syntax for the Hosting HTTP URI.

When instantiated, an URI Mapping Template is always concatenated to a base URI provided by the HC Proxy via discovery (see Section 5.4), or by other means.

A simple form (Section 5.3.1) and an enhanced form (Section 5.3.2) are provided to fit different users' requirements.

Both forms are expressed as level 2 URI template's to take care of the expansion of values that are allowed to include reserved URI characters.

#### 5.3.1. Simple Form

The simple form MUST be used for mappings where the Target CoAP URI is going to be copied verbatim at some fixed position into the Hosting HTTP URI.

The following template variables MUST be used in mutual exclusion in a template definition:

```
cu = coap-URI    ; from [RFC7252], Section 6.1
su = coaps-URI   ; from [RFC7252], Section 6.2
tu = cu / su
```

The same considerations done in Section 5.2.1 apply.

##### 5.3.1.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use `http://p.example.com/hc` as the base URI.

1. "coap" URI is a query argument of the Hosting HTTP URI:

```
?coap_target_uri={+cu}
```

```
coap://s.example.com/light
```

```
http://p.example.com/hc?coap_target_uri=coap://s.example.com/light
```

2. "coaps" URI is a query argument of the Hosting HTTP URI:

```
?coaps_target_uri={+su}
```

```
coaps://s.example.com/light
```

```
http://p.example.com/hc?coaps_target_uri=coaps://s.example.com/light
```

3. Target CoAP URI as a query argument of the Hosting HTTP URI:

```
?target_uri={+tu}
```

```
coap://s.example.com/light
```

```
http://p.example.com/hc?target_uri=coap://s.example.com/light
```

or

```
coaps://s.example.com/light
```

```
http://p.example.com/hc?target_uri=coaps://s.example.com/light
```

4. Target CoAP URI in the path component of the Hosting HTTP URI (i.e. the default URI Mapping template):

```
/ {+tu}
```

```
coap://s.example.com/light
```

```
http://p.example.com/hc/coap://s.example.com/light
```

or

```
coaps://s.example.com/light
```

```
http://p.example.com/hc/coaps://s.example.com/light
```

#### 5.3.2. Enhanced Form

The enhanced form can be used to express more sophisticated mappings, i.e. those that do not fit into the simple form.

There MUST be at most one instance of each of the following template variables in a template definition:

```
s = "coap" / "coaps" ; from [RFC7252], Sections 6.1 and 6.2
hp = host [ ":" port ] ; from [RFC3986] Sections 3.2.2 and 3.2.3
p = path-abempty ; from [RFC3986] Section 3.3.
q = [ "?" query ] ; from [RFC3986] Section 3.4
```

##### 5.3.2.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use `http://p.example.com/hc` as the base URI.

1. Target CoAP URI components in path segments, and optional query in query component:

`{+s}{+hp}{+p}{+q}`

`coap://s.example.com/light`

`http://p.example.com/hc/coap/s.example.com/light`

or

`coap://s.example.com/light?on`

`http://p.example.com/hc/coap/s.example.com/light?on`

2. Target CoAP URI components split in individual query arguments:

`?s={+s}&hp={+hp}&p={+p}&q={+q}`

`coap://s.example.com/light`

`http://p.example.com/hc?s=coap&hp=s.example.com&p=/light&q`

or

`coaps://s.example.com/light?on`

`http://p.example.com/hc?s=coaps&hp=s.example.com&p=/light&q=on`

#### 5.4. Discovery

In order to accommodate site specific needs while allowing third parties to discover the proxy function, the HC Proxy SHOULD publish information related to the location and syntax of the HC Proxy function using the CoRE Link Format [RFC6690] interface.

To this aim a new Resource Type, "core.hc", is associated with a base URI, and can be used as the value for the "rt" attribute in a query to the /.well-known/core in order to locate the base URI where the HC Proxy function is anchored.

Along with it, the new target attribute "hct" MAY be returned in a "core.hc" link to provide the associated URI Mapping Template. The default template given in Section 5.2, i.e. `{+tu}`, MUST be assumed if no "hct" attribute is found in the returned link. If an "htc" attribute is present in the returned link, then a compliant client MUST use it to create the Hosting HTTP URI.

Discovery SHOULD be available on both the HTTP and the CoAP side of the HC proxy, with one important difference: on the CoAP side the link associated to the "core.hc" resource needs an explicit anchor referring to the HTTP origin, while on the HTTP interface the link context is already the HTTP origin carried in the request's Host header, and doesn't have to be made explicit.

#### 5.4.1. Examples

- o The first example exercises the CoAP interface, and assumes that the default template, {+tu}, is used:

```
Req:  GET coap://[ff02::1]/.well-known/core?rt=core.hc
```

```
Res:  2.05 Content
      </hc>;anchor="http://p.example.com";rt="core.hc"
```

- o The second example - also on the CoAP side of the HC Proxy - uses a custom template, i.e. one where the CoAP URI is carried inside the query component, thus the returned link carries the URI template to be used in an explicit "hct" attribute:

```
Req:  GET coap://[ff02::1]/.well-known/core?rt=core.hc
```

```
Res:  2.05 Content
      </hc>;anchor="http://p.example.com";rt="core.hc";hct="?uri={+tu}"
```

On the HTTP side link information can be serialised in more than one way:

- o using the 'application/link-format' content type:

```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com
```

```
Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format
      Content-Length: 18
```

```
</hc>;rt="core.hc"
```

- o using the 'application/link-format+json' content type:



```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com

Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format+json
      Content-Length: 31

      [{"href":"/hc","rt":"core.hc"}]
```

- o using the Link header:

```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com

Res:  HTTP/1.1 200 OK
      Link: </hc>;rt="core.hc"
```

- o An HC Proxy may expose two different base URIs to differentiate between Target CoAP resources in the "coap" and "coaps" scheme:

```
Req:  GET /.well-known/core?rt=core.hc
      Host: p.example.com

Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format+json
      Content-Length: 111

      [
        {"href":"/hc/plaintext","rt":"core.hc","htc":"+cu"},
        {"href":"/hc/secure","rt":"core.hc","htc":"+su"}
      ]
```

## 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" [RFC7230] 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 [RFC7252]. 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 HC 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.)

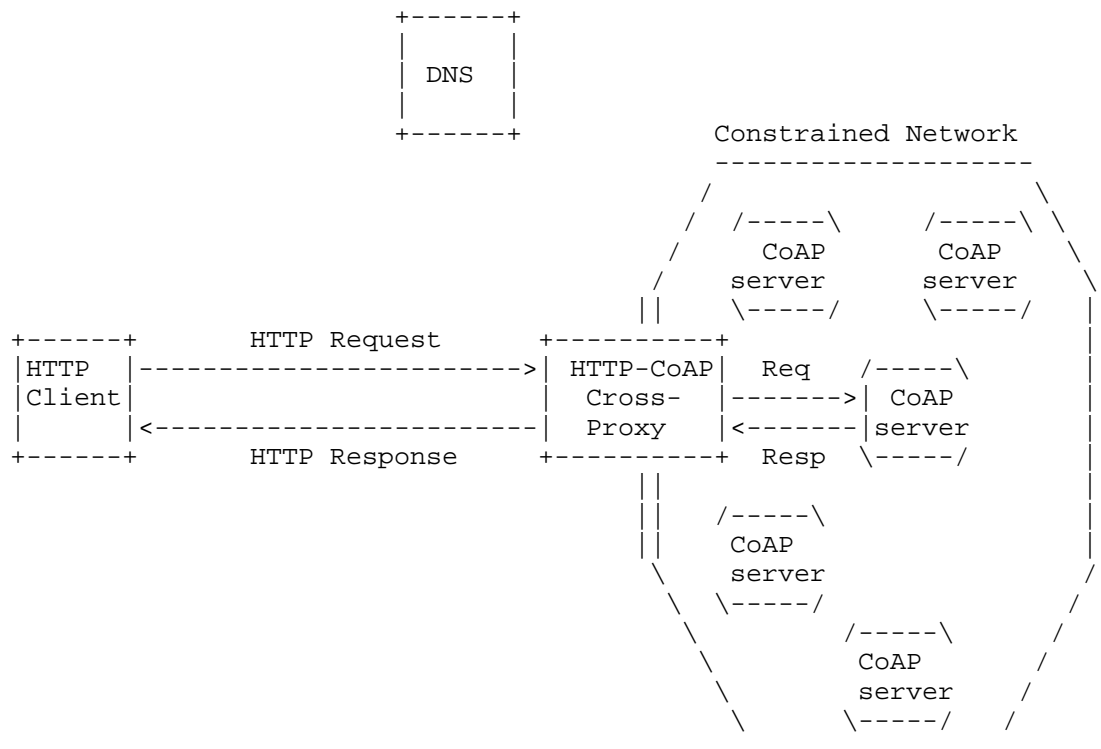


Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

## 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 [RFC7252] 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 [RFC7231] 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. [RFC7231] 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 [RFC7235]).
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 recognised 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 mapping

A HC Proxy translates HTTP media types (Section 3.1.1.1 of [RFC7231]) and content encodings (Section 3.1.2.1 of [RFC7231]) into CoAP content formats (Section 12.3 of [RFC7252]).

Media type translation can happen in GET, PUT or POST requests going from HTTP to CoAP, and in 2.xx (i.e. successful) responses going from CoAP to HTTP. Specifically, PUT and POST need to map the Content-Type and Content-Encoding HTTP headers into a CoAP Content-Format option, whereas GET needs to map Accept and Accept-Encoding HTTP

headers into a CoAP Accept option. On the way back, the CoAP Content-Format option is renormalised into a suitable HTTP Content-Type and Content-Encoding combination.

An HTTP request carrying a Content-Type and Content-Encoding combination which the HC Proxy is unable to map to an equivalent CoAP Content-Format, SHALL elicit a 415 (Unsupported Media Type) response by the HC Proxy.

If the HC Proxy receives a CoAP response with a Content-Format that it does not recognise (for example because the value has been registered after the proxy has been deployed), then it is allowed to either return a HTTP entity without a Content-Type header, or examine the data to determine its type on the fly.

On the content negotiation side, failing to map Accept and Accept-Encoding headers SHOULD be silently ignored: the HC Proxy SHOULD therefore forward the request with no Accept option.

While the CoAP to HTTP direction has always a well defined mapping, the HTTP to CoAP direction is more problematic because the source set, i.e., potentially 1000+ IANA registered media types, is much bigger than the destination set, i.e. the mere 6 values initially defined in Section 12.3 of [RFC7252].

Depending on the tight/loose coupling with the application(s) for which it proxies, the HC Proxy could implement different media-type mappings.

When tightly coupled, the HC Proxy knows exactly which content formats are supported by the applications, and can be strict when enforcing its forwarding policies in general, and the media-type mapping in particular.

On the other side, when the HC Proxy is a general purpose application layer gateway, being too strict could significantly reduce the amount of traffic that it'd be able to successfully forward. In this cases, the "loose" media-type mapping detailed in Section 6.3.1 MAY be implemented.

The latter grants unconstrained evolution of the surrounding ecosystem, at the cost of allowing more attack surface. In fact, as a result of such strategy, payloads would be forwarded more liberally across the unconstrained/constrained boundary of the communication path. Therefore, when applied, other forms of access control must be set in place to avoid unauthorised users to deplete or abuse systems and network resources.

### 6.3.1. Loose Media Type Mapping

By structuring the type information in a super-class (e.g. "text") followed by a finer grained sub-class (e.g. "html"), and optional parameters (e.g. "charset=utf-8"), Internet media types provide a rich and scalable framework for encoding the type of any given entity.

This approach is not applicable to CoAP, where Content Formats conflate an Internet media type (potentially with specific parameters) and a content encoding into one small integer value.

To remedy this loss of flexibility, we introduce the concept of a "loose" media type mapping, where media-types that are specialisations of a more generic media-type can be aliased to their super-class and then mapped (if possible) to one of CoAP content formats. For example, "application/soap+xml" can be aliased to "application/xml", which has a known conversion to CoAP. In the context of this "loose" media type mapping, "application/octet-stream" can be used as a fall back when no better alias is found for a specific media-type.

Table 2 defines the default lookup table for the "loose" media-type mapping. Given an input media-type, the table returns its best generalised media-type using longest prefix match.

Internet media-type	Generalised media-type
application/*+xml	application/xml
application/*+json	application/json
text/xml	application/xml
text/*	text/plain
*/*	application/octet-stream

Table 2: Media type generalisation

The "loose" media-type mapping is an OPTIONAL feature. Implementations supporting this kind of mapping SHOULD provide a flexible way to define the set of media-type generalisations allowed.

### 6.3.2. Internet Media Type to Content Format Mapping Algorithm

This section defines the algorithm used to map an Internet media type to its correspondent CoAP content format.

The algorithm uses the mapping table defined in Section 12.3 of [RFC7252] plus, possibly, any locally defined extension of it. Optionally, the table and lookup mechanism described in Section 6.3.1 can be used if the implementation chooses so.

Note that the algorithm may have side effects on the associated representation (see also Section 6.3.3).

In the following:

- o C-T, C-E, and C-F stand for the values of the Content-Type (or Accept), Content-Encoding (or Accept-Encoding) HTTP headers, and Content-Format CoAP option respectively.
- o If C-E is not given it is assumed to be "identity".
- o MAP is the mandatory lookup table, GMAP is the optional generalised table.

```
INPUT:  C-T and C-E
OUTPUT: C-F or Fail

1.  if no C-T: return Fail
2.  C-F = MAP[C-T, C-E]
3.  if C-F is not None: return C-F
4.  if C-E is not "identity":
5.      if C-E is supported (e.g. gzip):
6.          decode the representation accordingly
7.          set C-E to "identity"
8.      else:
9.          return Fail
10. repeat steps 2. and 3.
11. if C-T allows a non-lossy transformation into \
12.     one of the supported C-F:
13.     transcode the representation accordingly
14.     return C-F
15. if GMAP is defined:
16.     C-F = GMAP[C-T]
17.     if C-F is not None: return C-F
18. return Fail
```

Figure 2

### 6.3.3. Content Transcoding

As noted in Section 6.3.2, the process of mapping the type of the resource can have side effects on the forwarded entity body.



This may be caused by the removal or addition of a specific content encoding, or because the HC Proxy decides to transcode the representation to a different (compatible) format. The latter proves useful when an optimised version of a specific format exists. For example an XML-encoded resource could be transcoded to EXI, or a JSON-encoded resource into CBOR [RFC7049], effectively achieving compression without losing any information.

Payload transcoding (see steps 11-14 of Figure 2) is an OPTIONAL feature. Implementations supporting this feature SHOULD provide a flexible way to define the set of transcodings allowed.

#### 6.4. Caching and Congestion Control

A HC 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 HC 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 HC Proxy (closing the TCP connection) after the HTTP request was made, a HC Proxy SHOULD wait for the associated CoAP response and cache it if possible. Further requests to the HC 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 [RFC7252], 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 HC 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 HC 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 HC 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 [RFC7252]. 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 HC 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 [RFC7252] Section 4.6.

A HC 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 HC 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 HC 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 HC 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 HC 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 [RFC7230].

A HC 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 [RFC7252].

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 [RFC7230]) MAY be supported by the proxy and is transparent to the CoAP network: the HC Proxy will sequentially serve the pipelined requests by issuing different CoAP requests.

It is expected that the HC function will often be implemented in software on the proxy. Many different software approaches are possible, including using CGI [RFC3875] as an interface between the HTTP layer and the protocol translation engine.

## 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 HC Proxy scenario. In fact, the HC Proxy is a trusted (not rarely a transparently trusted) component in the network path.

The trustworthiness assumption on the HC 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 HC 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 HC 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 [RFC7252].

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 HC 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 HC 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 HC proxy deployment shown in Figure 1), the HC 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 HC Proxy, i.e. the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The HC 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.

### 8.3. URI Mapping

The following risks related to the URI mapping described in Section 5 have been identified:

DoS attack on the internal network.

Default deny any Target CoAP URI whose authority is (or maps to) a multicast address. Then explicitly whitelist multicast resources that are allowed to be de-referenced.

Leaking information on the internal network resources and topology.

Default deny any Target CoAP URI (especially /.well-known/core is the resource to be protected), and then explicit whitelist resources that are allowed to be seen from outside.

Reduced privacy due to the mechanics of the URI mapping.

The internal CoAP Target resource is totally transparent from outside: an HC Proxy implementing a HTTPS-only interface makes the Target CoAP URI totally opaque to a passive attacker.

## 9. Acknowledgements

An initial version of the table found in Section 6.2 has been provided in revision -05 of [RFC7252]. 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-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.
- [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.
- [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.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.
- [RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.
- [RFC7235] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

## 10.2. Informative References

- [I-D.ietf-core-groupcomm]  
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-19 (work in progress), June 2014.
- [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.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, January 2001.
- [RFC3875] Robinson, D. and K. Coar, "The Common Gateway Interface (CGI) Version 1.1", RFC 3875, October 2004.

[RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, December 2006.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

#### Appendix A. Change Log

[Note to RFC Editor: Please remove this section before publication.]

Changes from ietf-03 to ietf-04:

- o Expanded use case descriptions in Section 4;
- o Fixed/enhanced discovery examples in Section 5.4.1;
- o Addressed Ticket #365 (Add text on media-type conversion by HTTP-CoAP proxy) in new section 6.3.1 (Generalized media-type mapping) and new section 6.3.2 (Content translation);
- o Updated HTTPBis WG draft references to recently published RFC numbers.
- o Various editorial improvements.

Changes from ietf-02 to ietf-03:

- o Closed Ticket #351 "Add security implications of proposed default HC URI mapping";
- o Closed Ticket #363 "Remove CoAP scheme in default HTTP-CoAP URI mapping";
- o Closed Ticket #364 "Add discovery of HTTP-CoAP mapping resource(s)".

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](mailto:angelo@castellani.net)

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

Email: [salvatore.loreto@ericsson.com](mailto: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](mailto:Akbar.Rahman@InterDigital.com)

Thomas Fossati  
Alcatel-Lucent  
3 Ely Road  
Milton, Cambridge CB24 6DD  
UK

Email: [thomas.fossati@alcatel-lucent.com](mailto:thomas.fossati@alcatel-lucent.com)

Esko Dijk  
Philips Research  
High Tech Campus 34  
Eindhoven 5656 AE  
The Netherlands

Email: [esko.dijk@philips.com](mailto:esko.dijk@philips.com)

CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 5, 2015

C. Bormann  
Universitaet Bremen TZI  
July 04, 2014

Representing CoRE Link Collections in JSON  
draft-ietf-core-links-json-02

Abstract

Web Linking (RFC5988) provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format (RFC6690). Outside of constrained environments, it may be useful to represent these collections of Web links in JSON format (RFC7159).

This specification defines a common format for representing Web links in JSON format.

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 5, 2015.

Copyright Notice

Copyright (c) 2014 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
1.1. Objectives . . . . .	3
1.2. Terminology . . . . .	3
2. Web Links in JSON . . . . .	3
2.1. Examples . . . . .	4
3. IANA Considerations . . . . .	5
4. Security Considerations . . . . .	6
5. Acknowledgements . . . . .	6
6. References . . . . .	6
6.1. Normative References . . . . .	6
6.2. Informative References . . . . .	6
Appendix A. Implementation . . . . .	6
Author's Address . . . . .	7

## 1. Introduction

Web Linking [RFC5988] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252].

Outside of constrained environments, it may also be useful to represent the same collections of Web links in the widely used JSON format [RFC7159]. When converting between these two formats, as usual, there are many little decisions that have to be made. If left without guidance, it is likely that a number of slightly incompatible dialects will emerge.

This specification defines a common format for representing CoRE Web Linking in JSON format.

Note that there is a separate question on how to represent Web links out of JSON documents, as discussed e.g. in [MNOT11]. While there are good reasons to stay as compatible as possible to developments in this area, the present specification is solving a different problem.

### 1.1. Objectives

This specification has been designed based on the following objectives:

- o Canonical mapping
  - \* lossless round-tripping with [RFC6690]
  - \* but not trying for bit-preserving (DER-style) round-tripping
- o The simplest thing that could possibly work
  - \* Do not cater for RFC 5988 complications caused by HTTP header character set issues [RFC2047]
- o Consider other work that has links in JSON, e.g.: JSON-LD, JSON-Reference [I-D.pbryan-zyp-json-ref]
  - \* Do not introduce unmotivated differences

### 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 [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.

## 2. Web Links in JSON

The objective of the JSON mapping defined in this document is to contain information of the formats specified in [RFC5988] and [RFC6690]. This specification therefore uses the names of the ABNF productions used in those documents.

An application/link-format document is a collection of web links ("link-value"), each of which is a collection of attributes ("link-param") applied to a "URI-Reference".

We straightforwardly map:

- o the outer collection to an array of links
- o each link to a JSON object.

In the object representing a "link-value", each target attribute or other parameter ("link-param") is represented by a JSON name/value

pair (member). The name is a string representation of the parameter or attribute name (as in "parmname"), the value is a string representation of the parameter or attribute value ("ptoken" or "quoted-string"). "quoted-string" productions are parsed (i.e, the backslash constructions evaluated) as defined in [RFC6690] and its referenced documents, before placing them in JSON strings (where they may gain back additional decorations such as backslashes as defined in [RFC7159]).

If a Link attribute ("parmname") is present more than once in a "link-value", its values are then represented as a JSON array of JSON string values; this array becomes the value of the JSON name/value pair where the attribute name is the JSON name. Attributes occurring just once MUST NOT be represented as JSON arrays but MUST be directly represented as JSON strings. (Note that the most recent version of link-format has cut down on the use of repeated parameter names; they are still allowed by [RFC5988] though. No attempt has been made to decode the possibly space-separated values for rt=, if=, and rel= into JSON arrays.)

The URI-Reference is represented as a name/value pair with the name "href" and the URI-Reference as the value. (Rationale: This usage is consistent with the use of "href" as a query parameter for link-format query filtering and with link-format reserving the link parameter "href" specifically for this use [RFC6690]).

(TBD: Should we do something special with the "hosts" relation? Should we include an anchor where the link-format does not explicitly set one?)

## 2.1. Examples

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/tl23>;anchor="/sensors/temp"
;rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 1: Example from page 15 of [RFC6690]

becomes

```
"[{ "href": "/sensors", "ct": "40", "title": "Sensor Index" }, { "href":
"/sensors/temp", "rt": "temperature-c", "if": "sensor" }, { "href":
"/sensors/light", "rt": "light-lux", "if": "sensor" }, { "href": "http://www.example.com/sensors/
tl23", "anchor": "/sensors/
```

```
temp", "rel": "describedby"}, {"href": "/t", "anchor": "/sensors/  
temp", "rel": "alternate"}] "
```

(More examples to be added.)

### 3. IANA Considerations

This specification registers the following additional Internet Media Types:

Type name: application

Subtype name: link-format+json

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+json" media type are required to conform to the "application/json" Media Type and are therefore subject to the same encoding considerations specified in Section 6 {{RFC7159}}.

Security considerations: As defined in this specification

Published specification: This specification.

Applications that use this media type: None currently known.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): TEXT

Person & email address to contact for further information:  
Carsten Bormann <cabo@tzi.org>

Intended usage: COMMON

Change controller: IESG

#### 4. Security Considerations

The security considerations of [RFC6690] apply.

(TBD.)

#### 5. Acknowledgements

(TBD.)

#### 6. References

##### 6.1. 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.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

##### 6.2. Informative References

- [I-D.pbryan-zyp-json-ref] Bryan, P. and K. Zyp, "JSON Reference", draft-pbryan-zyp-json-ref-03 (work in progress), September 2012.
- [MNOT11] Nottingham, M., "Linking in JSON", November 2011, <[http://www.mnot.net/blog/2011/11/25/linking\\_in\\_json](http://www.mnot.net/blog/2011/11/25/linking_in_json)>.
- [RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

#### Appendix A. Implementation

This appendix provides a simple reference implementation of the mapping between CoRE link format and Links-in-JSON.

(TBD - the reference implementation was used to create the above examples, but I still have to clean it up for readability and paste it in at 69 columns max.)

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: Standards Track  
Expires: January 1, 2015

K. Hartke  
Universitaet Bremen TZI  
June 30, 2014

Observing Resources in CoAP  
draft-ietf-core-observe-14

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 January 1, 2015.

Copyright Notice

Copyright (c) 2014 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. Consistency Model . . . . .	5
1.4. Observable Resources . . . . .	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 . . . . .	12
4.1. Request . . . . .	12
4.2. Notifications . . . . .	12
4.3. Caching . . . . .	13
4.4. Reordering . . . . .	14
4.5. Transmission . . . . .	14
5. Intermediaries . . . . .	17
6. Web Linking . . . . .	18
7. Security Considerations . . . . .	19
8. IANA Considerations . . . . .	19
9. Acknowledgements . . . . .	19
10. References . . . . .	20
10.1. Normative References . . . . .	20
10.2. Informative References . . . . .	20
Appendix A. Examples . . . . .	21
A.1. Client/Server Examples . . . . .	21
A.2. Proxy Examples . . . . .	25
Appendix B. Changelog . . . . .	27

## 1. Introduction

### 1.1. Background

CoAP [RFC7252] is an application protocol for constrained nodes and networks. It is intended to provide RESTful services [REST] not unlike HTTP [RFC7230] 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 [RFC7228].

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 authority 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 retrieves a representation of the resource and requests this representation be updated by the server as long as the client is interested in the resource.

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, 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.

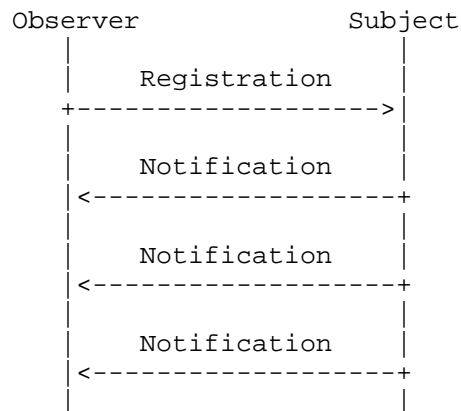


Figure 1: The Observer Design Pattern

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.

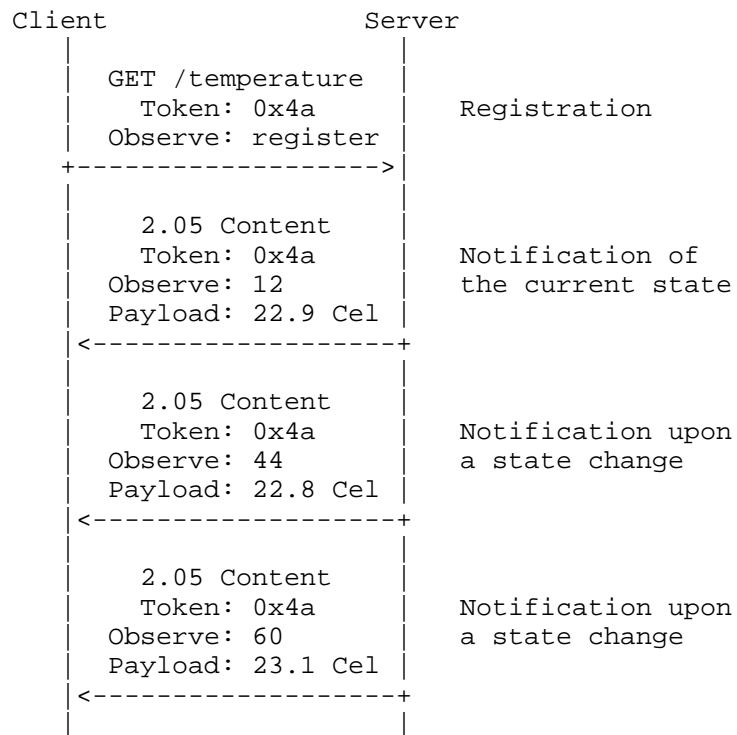


Figure 2: Observing a Resource in CoAP

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. When the client deregisters, rejects a notification, or the transmission of a notification times out after several transmission attempts, the client is considered no longer interested and is removed from the list of observers by the server.

### 1.3. Consistency Model

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 fully avoided that the client and the server become out of sync at times: First, there is always some latency between the change of the resource state and the receipt of the notification. Second, CoAP 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 eventually registers its interest again.

The protocol addresses this issue 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.

#### 1.4. 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 few seconds to a current reading of the temperature sensor;
- o `<coap://server/temperature/felt>`, which changes its state to "COLD" whenever the temperature reading drops below a certain pre-configured threshold, and to "WARM" whenever the reading exceeds a second, slightly higher threshold;
- o `<coap://server/temperature/critical?above=42>`, which changes its state based on the client-specified parameter value: every few

seconds to the current temperature reading if the temperature exceeds the threshold, or to "OK" when the reading drops below;

- o <coap://server/?query=select+avg(temperature)+from+Sensor.window:time(30sec)>, which accepts expressions of arbitrary complexity and changes its state accordingly.

Thus, 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 supplying it continuously with raw sensor data. 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. The application designer therefore can choose exactly the right level of complexity for the application envisioned and devices involved, and is not constrained to a "one size fits all" mechanism built into the protocol.

#### 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	uint	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 or remove an entry in the list of observers of the resource, where the entry consists of the client endpoint and the token specified in the request.

'register' (0) adds the entry to the list, if not present;

'deregister' (1) removes the entry from the list, if present.

The Observe Option is not critical for processing the request. If the server is unwilling or unable to add a new entry to the list of observers, 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 an entry with the client endpoint and request token to the list of observers and that it will notify the client of changes to the resource state. The option value is a 24-bit sequence number for reordering detection (see Section 3.4 and Section 4.4).

The value of the Observe Option is encoded as an unsigned integer in network byte order using a variable number of bytes ('uint' option format); see Section 3.2 of RFC 7252 [RFC7252].

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 with an Observe Option is used to satisfy a normal GET request, the option MUST be removed before the response is returned.

### 3. Client-side Requirements

#### 3.1. Request

A client registers its interest in a resource by issuing a GET request with an Observe Option set to 'register' (0). If the server returns a 2.xx response that includes an Observe Option as well, the server has successfully added an entry with the client endpoint and request token 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 stream of updates resulting from one observation request can be used to satisfy another (observation or normal GET) request if the target resource is the same. A client MUST aggregate such requests 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.

Notifications typically have a 2.05 (Content) response code. They include 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. If the client included one or more ETag Options in the request (see Section 3.3), notifications can also have



a 2.03 (Valid) response code. Such notifications include an Observe Option with a sequence number but no payload.

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. Non-2.xx responses do not include an Observe Option.

### 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 7252 [RFC7252]. 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 the same token as the original at any time. All options MUST be identical to those in the original request, except for the set of ETag Options. 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 at least wait for a random amount of time between 5 and 15 seconds after Max-Age expired 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 re-registers 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 consider a notification fresh 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 (Section 4.8.2 of RFC 7252 [RFC7252]).

### 3.5. Transmission

A notification can be confirmable or non-confirmable, i.e., it can 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 notifications for a resource can simply "forget" the observation. When the server then sends the next notification, the client will not recognize the token in the message and thus will return a Reset message. This causes the server to remove the associated entry from the list of observers. The entries in lists of observers are effectively "garbage collected" by the server.

Implementation Note: Due to potential message loss, the Reset message may not reach the server. The client may therefore have to reject multiple notifications, each with one Reset message, until the server finally removes the associated entry from the list of observers and stops sending notifications.

In some circumstances, it may be desirable to cancel an observation and release the resources allocated by the server to it more eagerly. In this case, a client MAY explicitly deregister by issuing a GET request which has the Token field set to the token of the observation to be cancelled and includes an Observe Option with the value set to 'deregister' (1). All other options MUST be identical to those in the registration request, except for the set of ETag Options. When the server receives such a request, it will remove any matching entry from the list of observers and processes the GET request as usual.

## 4. Server-side Requirements

### 4.1. Request

A GET request with an Observe Option set to 'register' (0) 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 returns 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 registration request 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.

If the Observe Option in a request is set to any other value than 'register' (0), then the server MUST remove any entry with a matching endpoint/token pair from the list of observers and process the GET request as usual. The resulting response MUST NOT include an Observe Option.

### 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 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 associated entry from the list of observers of the resource.

The Content-Format specified in a 2.xx 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 format, it SHOULD send a notification with a 4.06 (Not Acceptable) response code and MUST remove the associated entry from the list of observers of the resource.

A 2.xx notification MUST include an Observe Option with a sequence number as specified in Section 4.4 below; a non-2.xx notification MUST NOT include an Observe Option.

#### 4.3. Caching

As notifications are just additional responses sent by the server in reply to a GET request, they are subject to caching as defined in Section 5.6 of RFC 7252 [RFC7252].

##### 4.3.1. Freshness

After returning the initial response, the server MUST try to keep the returned representation current, i.e., it MUST 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 new notification with the unchanged representation and a new Max-Age just before the Max-Age indicated earlier 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 retransmission.

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 first 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 theoretically 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. Therefore, the maximum notification rate is reduced to 32768 notifications per second. This 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 of a resource is changing frequently. It also MAY choose to send more than one notification for the same resource state. 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 that the client is interested in receiving further notifications. If a client rejects a confirmable or non-confirmable notification with a Reset message, or if the last attempt to retransmit a confirmable notification times out, then the client is considered no longer interested and the server MUST remove the associated entry from the list of observers.

Implementation Note: To properly process a Reset message that rejects a non-confirmable notification, a server needs to remember the message IDs of the non-confirmable notifications it sends. This may be challenging for a server with constrained resources. However, since Reset messages are transmitted unreliably, the client must be prepared that its Reset messages aren't received by the server. A server thus can always pretend that a Reset message rejecting a non-confirmable notification was lost. If a server does this, it could accelerate cancellation by sending the following notifications to that client in confirmable messages.

A server that transmits notifications mostly in non-confirmable messages MUST send a notification in a confirmable message instead of a non-confirmable message at least every 24 hours. This prevents a client that went away or is no longer interested from remaining in the list of observers indefinitely.

#### 4.5.1. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.2 of RFC 7252 [RFC7252] and the limitations in Section 4.7 of RFC 7252 [RFC7252]. However, CoAP places the responsibility of congestion control for simple request/response interactions only on the clients: rate limiting request transmission implicitly controls the transmission of the responses. When a single request yields a potentially infinite number of notifications, additional responsibility needs to be placed on the server.

In order not to cause congestion, servers **MUST** strictly limit the number of simultaneous outstanding notifications/responses that they transmit to a given client to `NSTART` (1 by default; see Section 4.7 of RFC 7252 [RFC7252]). An outstanding notification/response is either a confirmable message for which an acknowledgement has not yet been received and whose last retransmission attempt has not yet timed out, or a non-confirmable message for which the waiting time that results from the following rate limiting rules has not yet elapsed.

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]).

Further congestion control optimizations and considerations are expected in the future with advanced CoAP congestion control mechanisms.

#### 4.5.2. Advanced Transmission

The state of an observed resource may change while the number of the number of simultaneous outstanding notifications/responses to a client on the list of observers is greater than or equal to `NSTART`. In this case, the server cannot notify the client of the new resource state immediately but has to wait for an outstanding notification/response to complete first.

If there exists an outstanding notification/response that the server transmits to the client and that pertains to the changed resource, then it is desirable for the server to stop working towards getting the representation of the old resource state to the client, and to start transmitting the current representation to the client instead, so the resource state observed by the client stays closer in sync with the actual state at the server.



For this purpose, the server MAY optimize the transmission process by aborting the transmission of the old notification (but not before the current transmission attempt completed) and starting a new transmission for the new notification (but with the retransmission timer and counter of the aborted transmission retained).

In more detail, a server MAY supersede an outstanding transmission that pertains to an observation as follows:

1. Wait for the current (re-)transmission attempt to be acknowledged, rejected or to time out (confirmable transmission); or wait for the waiting time to elapse or the transmission to be rejected (non-confirmable transmission).
2. If the transmission is rejected or it was the last attempt to retransmit 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. The new notification is transmitted with a new Message ID and the following transmission parameters: If the previous (re-)transmission attempt timed out, retain its transmission parameters, increment the retransmission counter and double the timeout; otherwise, initialize the transmission parameters as usual (see Section 4.2 of RFC 7252 [RFC7252]).

It is possible that the server later receives an acknowledgement for a confirmable notification that it superseded this way. Even though this does not signal consistency, it is valuable in that it signals the client's further interest in the resource. The server therefore should avoid inadvertently removing the associated entry from the list of observers.

## 5. Intermediaries

A client may be interested in a resource in the namespace of a 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 server, acting as if it was communicating with the 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 to keep the representation updated upon changes to the resource state, 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 server. If the response returned by the next hop doesn't include 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 in notifications, 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 in Section 11 of RFC 7252 [RFC7252] apply.

Observing resources can dramatically increase the negative effects of amplification attacks. That is, not only can notifications messages be much larger than the request message, but the nature of the protocol can cause a significant number of notifications to be generated. Without client authentication, a server therefore **MUST** 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 A. Bigot, Angelo P. Castellani, Gilbert Clark, Esko Dijk, Thomas Fossati, Brian Frank, Bert Greevenbosch, Jeroen Hoebeke, Cullen Jennings, Matthias Kovatsch, Salvatore Loreto, Charles Palmer, Akbar Rahman, 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

- [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.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

### 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](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.
- [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.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

## Appendix A. Examples

### A.1. Client/Server Examples

t	Observed State	CLIENT	SERVER	Actual State
1				
2	unknown			18.5 Cel
3		+----->		Header: GET 0x41011633
4		GET		Token: 0x4a
5				Uri-Path: temperature
6				Observe: 0 (register)
7				
8				
9		<-----+		Header: 2.05 0x61451633
10		2.05		Token: 0x4a
11	18.5 Cel			Observe: 9
12				Max-Age: 15
13				Payload: "18.5 Cel"
14				
15				
16		<-----+		Header: 2.05 0x51457b50
17		2.05		Token: 0x4a
18	19.2 Cel			Observe: 16
19				Max-Age: 15
20				Payload: "19.2 Cel"
21				

Figure 3: A client registers and receives one notification of the current state and one of a new state upon a state change

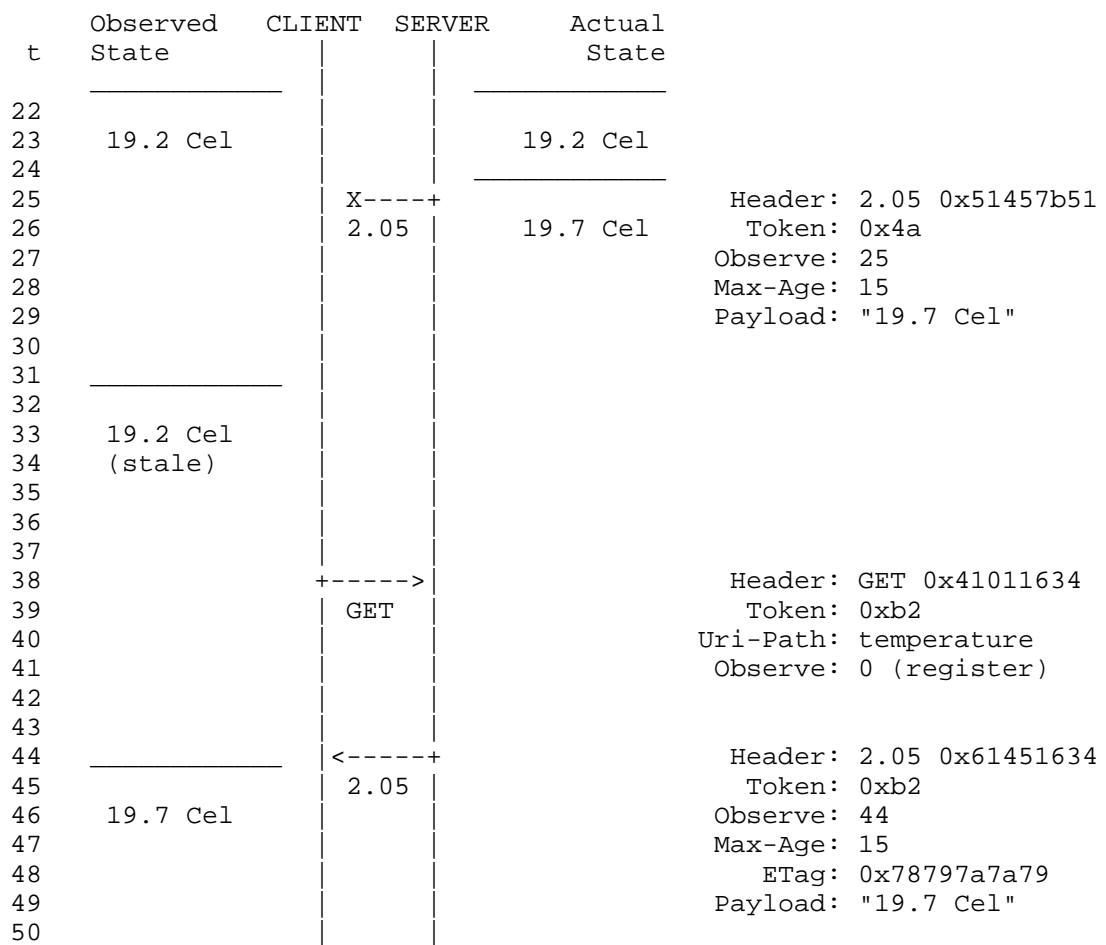


Figure 4: The client re-registers after Max-Age ends

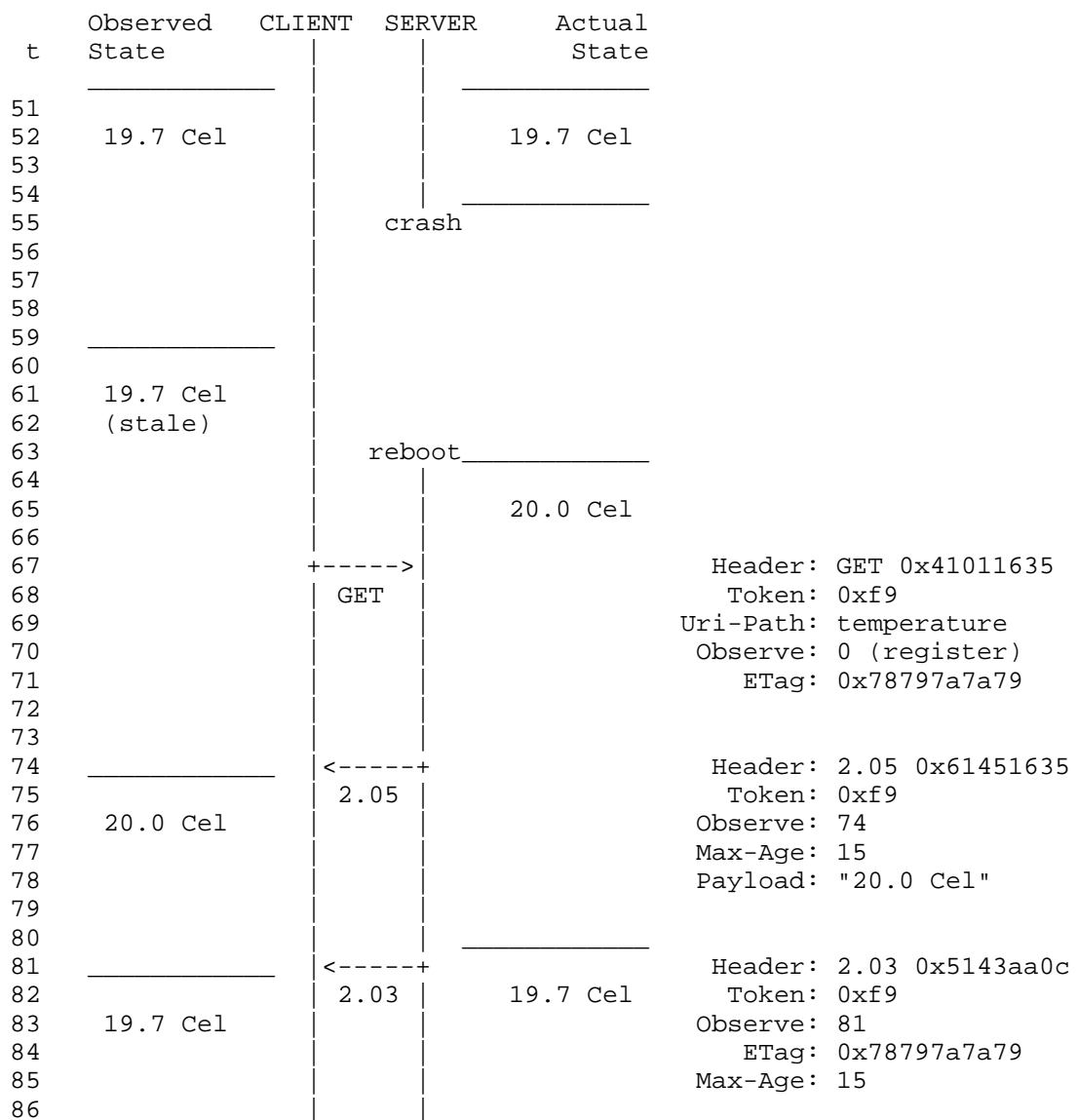


Figure 5: The client re-registers and gives the server the opportunity to select a stored response

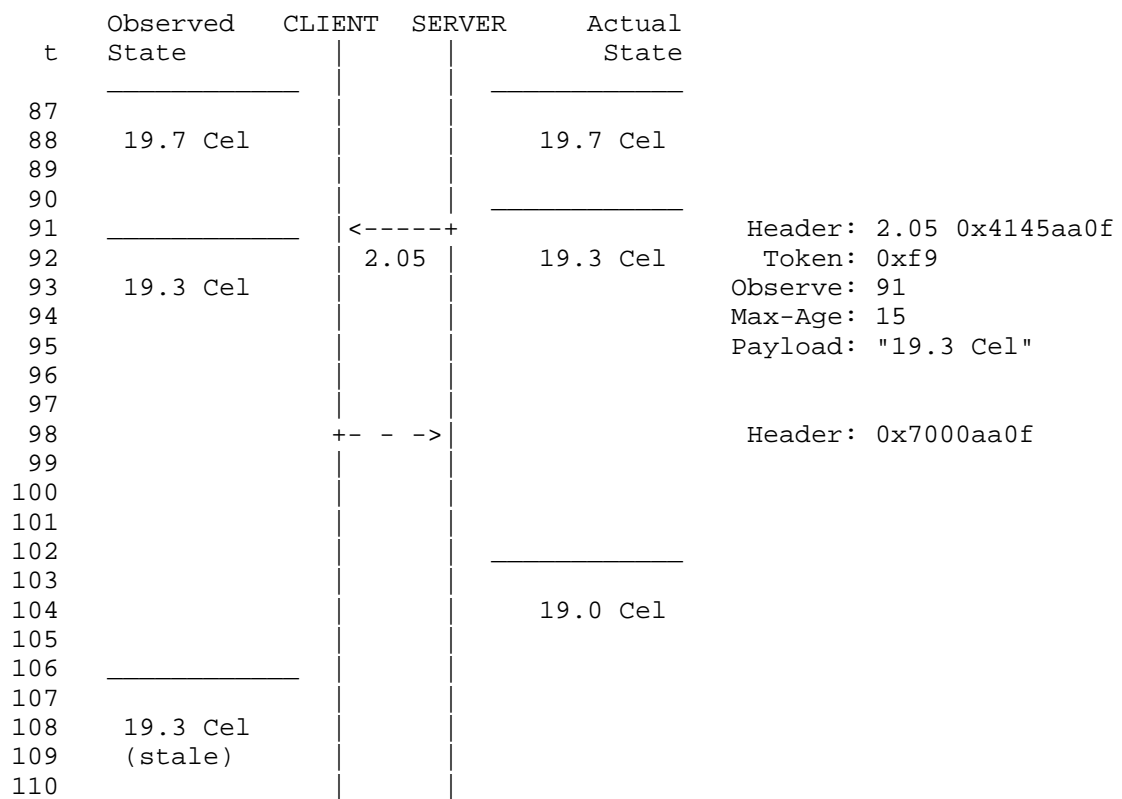


Figure 6: The client rejects a notification and thereby cancels the observation



## A.2. Proxy Examples

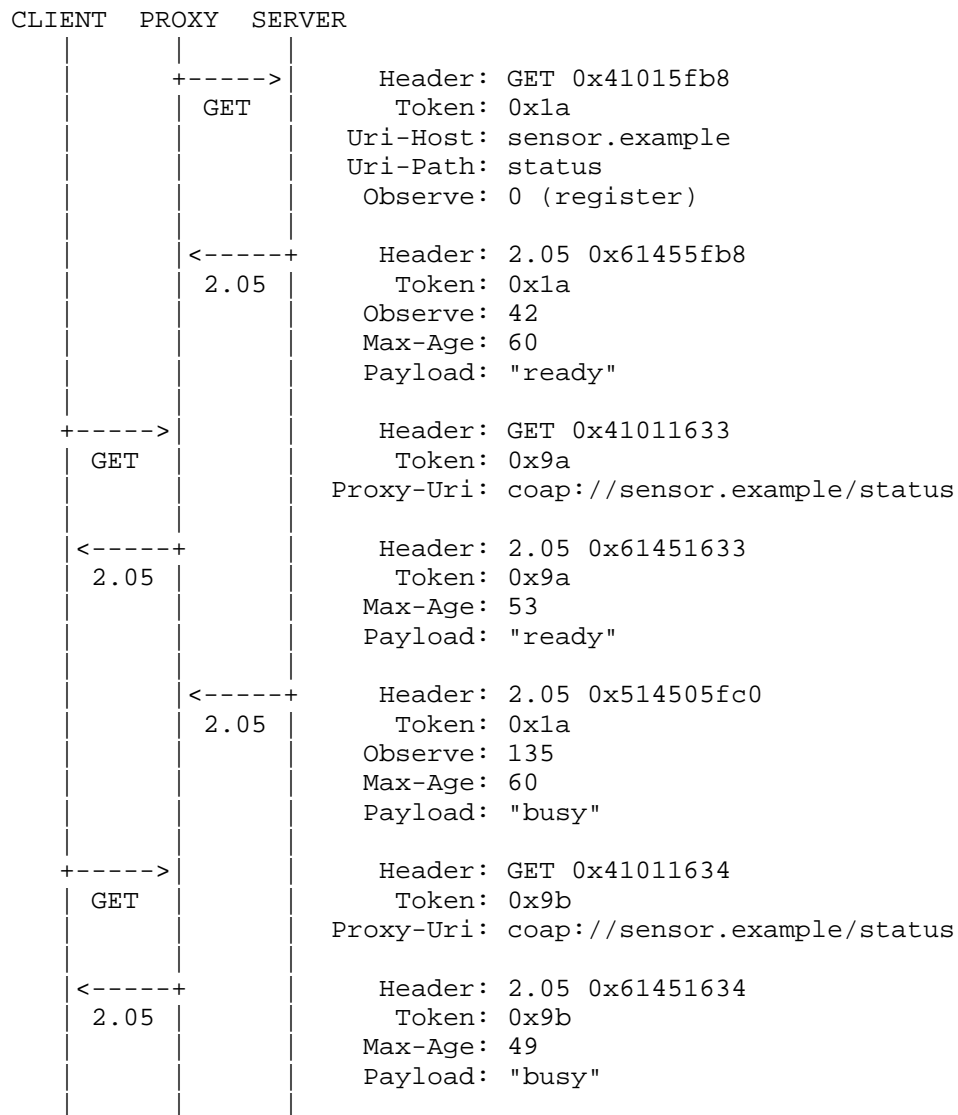


Figure 7: A proxy observes a resource to keep its cache up to date

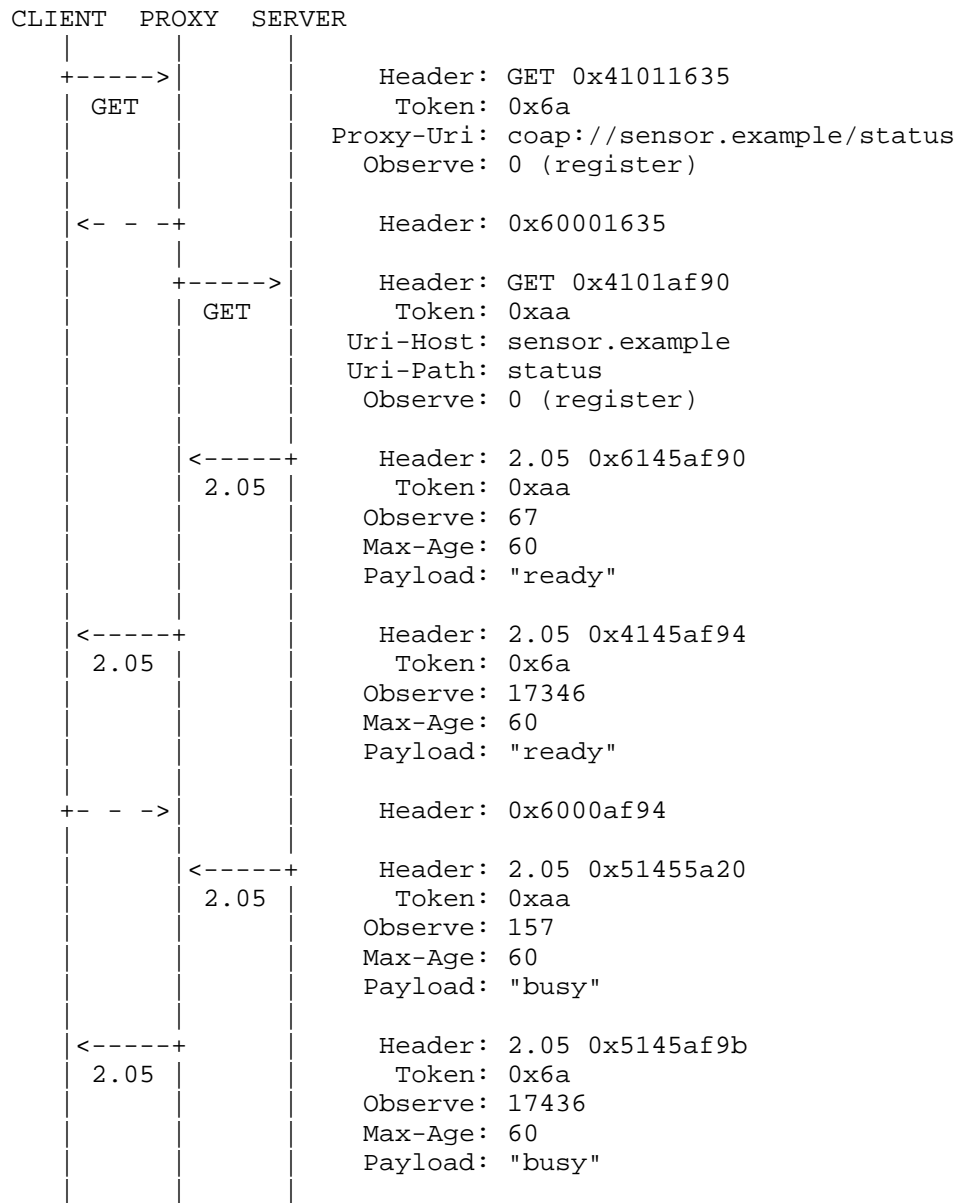


Figure 8: A client observes a resource through a proxy

## Appendix B. Changelog

[Note to RFC Editor: Please remove this section before publication.]

Changes from ietf-13 to ietf-14:

- o Updated references.

Changes from ietf-12 to ietf-13:

- o Extended the Observe Option in requests to not only add but also remove an entry in the list of observers, depending on the option value.

Note: The value of the Observe Option in a registration request may now be any sequence of bytes that encodes the unsigned integer 0, i.e., 0x'', 0x'00', 0x'00 00' or 0x'00 00 00'.

- o Removed the 7.31 Code for cancellation.

Changes from ietf-11 to ietf-12:

- o Introduced the 7.31 Code to request the cancellation of a pending request.
- o Made the algorithm for superseding an outstanding transmission OPTIONAL.
- o Clarified that the entry in the list of observers is removed if the client fails to acknowledge a confirmable notification before the last retransmission attempt times out (#350).
- o Simplified the text on cancellation (#352) and the handling of Reset messages (#353).

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





Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: October 4, 2014

O. Kleine  
University of Luebeck, ITM  
April 2, 2014

CoAP Endpoint Identification  
draft-kleine-core-coap-endpoint-id-00.txt

Abstract

The Constrained Application Protocol (CoAP) is an application layer protocol for constrained devices (e.g. low power, few memory) and networks (e.g. lossy, low bandwidth) which relies on UDP on the transport layer. With CoAP it is often the case, that message exchanges need to extend the common request/response pattern, e.g. for separate responses. This holds, e.g. for CON requests that are confirmed by the server with an empty ACK and answered later with a separate response. According to the CoAP specification the request/response matching is realized using a unique pair of server address and token per client.

Due to the mobile nature of some devices, e.g. smartphones, they are often assigned new IP addresses because of a network change. Thus, the IP address of a CoAP server might change during an ongoing conversation. This draft proposes a method to assign each communication partner with an identifier (endpoint ID) which replaces the IP address as (partial) key to relate requests and responses.

Besides the common separate responses, the proposed method is also useful to handle IP address changes, e.g. during an ongoing observation ([observe]) or a blockwise transfer ([block]).

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 [RFC2119].

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 October 4, 2014.

#### Copyright Notice

Copyright (c) 2014 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. A "Message Exchange" . . . . .	3
3. Endpoint Identification Options . . . . .	6
3.1. ENDPOINT_ID_1 option . . . . .	6
3.2. ENDPOINT_ID_2 option . . . . .	6
3.3. Option syntax and semantics . . . . .	7
3.4. Endpoint IDs for observations . . . . .	7
3.4.1. Client IP changes during observation . . . . .	7
3.4.2. Server IP changes during observation . . . . .	7
4. Examples . . . . .	8
4.1. NON request and NON response . . . . .	8
4.2. NON request, CON response, and empty ACK . . . . .	8
4.3. CON request, empty ACK, and NON response . . . . .	8
4.4. CON request, empty ACK, CON response, and empty ACK . . . . .	9
4.5. Server IP address changes during observation . . . . .	9
4.6. Client IP address changes during observation . . . . .	10
5. Acknowledgements . . . . .	11
6. IANA Considerations . . . . .	11
7. Security Considerations . . . . .	11
8. References . . . . .	12
8.1. Normative References . . . . .	12
8.2. Informative References . . . . .	12
Author's Address . . . . .	12

## 1. Introduction

The concept of confirmable messages (CON) introduced in the main CoAP specification provides reliability in terms of message reception by the remote endpoint, i.e. the recipient of a confirmable message MUST confirm the reception with an acknowledgement (ACK) within 2 seconds. The absence of an ACK causes the sender of the CON message to retransmit the CON message. However, an (empty) ACK just confirms the reception and for confirmable requests this might cause the server to send a separate response containing the actual result of the request processing, i.e. a third message within a single conversation.

According to the CoAP specification the key to match incoming responses with open requests is a token which is defined by the client. This token is set as a part of the requests header and sent to the server. The server includes the same token in the response and by this means enables the client to match the response with a request. The token is unique per communication partner, i.e. a client would use 2 different tokens for 2 parallel requests to a server but may use the same token for 2 parallel requests to different servers. Thus, the client must use the combination of the server address and the token to match incoming responses with open requests.

CoAP servers may run on mobile devices, e.g. smartphones, that are often assigned new IP addresses due to network changes. The assignment of a new IP could happen within an ongoing conversation, i.e. after an empty ACK was sent but before the actual (separate) response. In this case, the client can not match the response with the open request. This draft introduces 2 new CoAP options to deal with this issue and enable ongoing conversations to continue even if one of the endpoints changes its IP address.

Besides the common separate responses, the proposed method is also useful to handle IP address changes, e.g. during an ongoing observation [observe] or a blockwise transfer [block].

## 2. A "Message Exchange"

A message exchange is considered to consist of all messages that are sent between two endpoints as direct consequence of the first message plus this first message. Thus, according to the CoAP specification (without extensions) a message exchange consists of either 1, 2, 3, or 4 messages.

As NON request do not require a response, it is possible, that a message exchange consists only of a single message (see Figure 1).

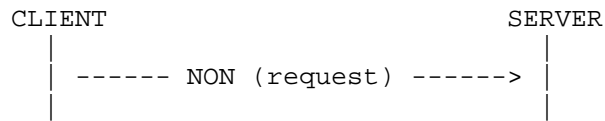


Figure 1: NON request without any response

There are 2 possible types of Message Exchange that consist of 2 messages. Those are depicted in Figure 2 and Figure 3.

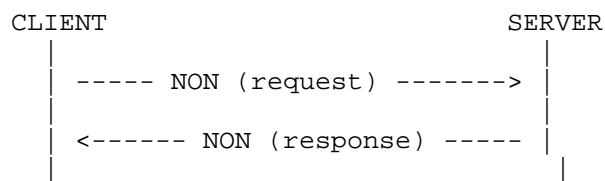


Figure 2: NON requests and NON response

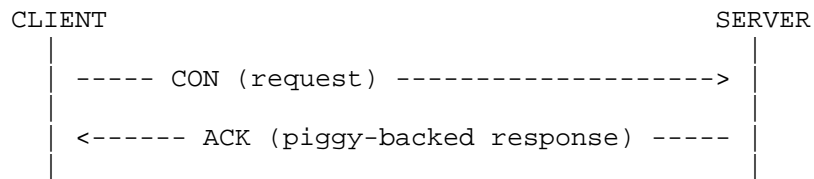


Figure 3: CON request and ACK response (piggy-backed)

Those were the types of Message Exchange that match the common request/response pattern. However, due to the reliability concept of CoAP there are also types of Message Exchange that extends this pattern by consisting of 3 or even 4 messages. The 2 possible types of Message Exchange that consist of 3 messages are depicted in Figure 4 and Figure 5.

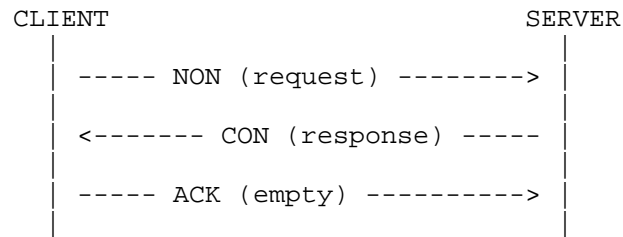


Figure 4: NON requests and CON response

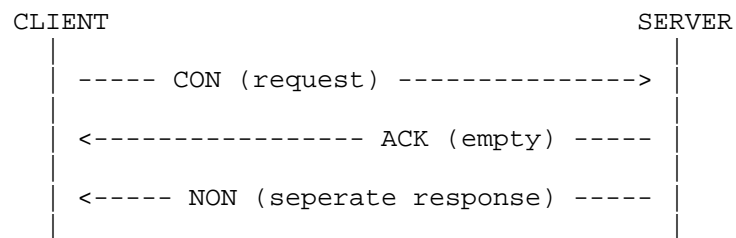


Figure 5: CON request, empty ACK and NON response (seperate)

The last type of Message Exchange consists of 4 messages to be sent and includes reliability for both, request and response (see Figure 6).

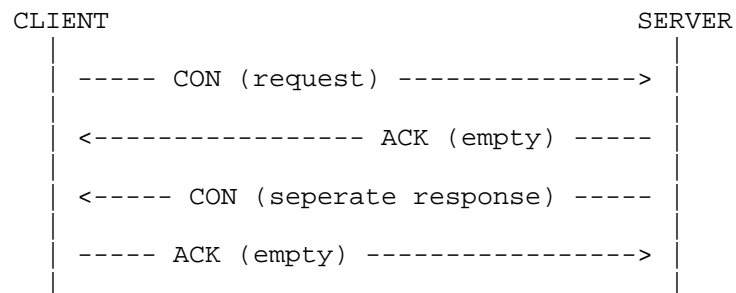


Figure 6: CON request, empty ACK, CON response, empty ACK

### 3. Endpoint Identification Options

The Endpoint Identification Extension introduces 2 new (opaque) options. The first option (ENDPOINT\_ID\_1) is used to assign the communication partner, i.e. the remote CoAP endpoint, a unique ID. The recipient of a CoAP message that contains an ENDPOINT\_ID\_1 option repeats its value in every follow-up message as value of the ENDPOINT\_ID\_2 option.

Furthermore, the sender of a CoAP message with ENDPOINT\_ID\_1 option uses the value as (partial) key for the duration of the conversation instead of the remote endpoints IP address, e.g. for request/response matching in combination with a token. However, this approach does not include CoAPs reliability "layer" as empty ACKs MUST not include any options. Thus, the CON/ACK matching still bases on the combination of remote IP and message ID.

#### 3.1. ENDPOINT\_ID\_1 option

The ENDPOINT\_ID\_1 option is set by the sender of a CoAP message to assign the remote endpoint an ID which is supposed to be used to identify this endpoint for the remaining duration of the actual message exchange (see Section 3.2) whenever possible (this does explicitly not include empty messages, e.g. ACK or RST).

If the recipient of a CoAP message with ENDPOINT\_ID\_1 option does not support the option it MAY ignore that option. As the recipient is supposed to repeat the value of the ENDPOINT\_ID\_1 option as value of the ENDPOINT\_ID\_2 option in every follow-up message within a message exchange, the first message origin can derive the lack of support for that option from the missing ENDPOINT\_ID\_2 option in the follow-up messages.

#### 3.2. ENDPOINT\_ID\_2 option

The ENDPOINT\_ID\_2 option is set by the sender of a CoAP message to identify itself as the message origin. The value of the ENDPOINT\_ID\_2 option repeats the value of the latest ENDPOINT\_ID\_1 option that was received from the intended recipient of the message to be sent.

Thus, a ENDPOINT\_ID\_2 option MUST not be set in a CoAP message if the intended recipient did not send a ENDPOINT\_ID\_1 option in a previous message. If the ENDPOINT\_ID\_2 option is not supported the message MUST be rejected via RST message. Also ENDPOINT\_ID\_2 option values that are unknown to the recipient MUST be rejected with a RST message.

### 3.3. Option syntax and semantics

Type	C	U	N	R	Name	Format	Length	Default
124	E	U	-	-	ENDPOINT_ID_1	opaque	0-4 B	(none)
189	C	U	-	-	ENDPOINT_ID_2	opaque	0-4 B	(none)

Table 1: The endpoint ID option numbers

### 3.4. Endpoint IDs for observations

Observing a CoAP resource means to retrieve multiple responses as a consequence of a single request. If the observe option is set in a request and observing is supported by the addressed resource, the client receives another response (update notification) whenever the status of the observed resource changes [observe].

This leads to a new type of Message Exchange consisting of an arbitrary number of messages. Within the duration of an observation relationship between a client and a server, both, the IP of the client and the IP of the server may change.

#### 3.4.1. Client IP changes during observation

The server MUST set the ENDPOINT\_ID\_1 option in every update notification. By this means, the client is assigned an ID which is independent from its IP address. Whenever the IP address of the client changes during an ongoing observation, the client resends its initial request and adds the assigned ID as value of the ENDPOINT\_ID\_2 option.

By this means, the server is able to update its internal "client ID/IP address - mapping" and continue the observation. However, if the request was a CON request, a server MAY only respond with an empty ACK instead of a full response if the observed resource did not change since the last update notification.

#### 3.4.2. Server IP changes during observation

Due to the ENDPOINT\_ID\_1 option in the request starting the observation, the server is assigned an ID that is independent from its IP address. This ID is to be set as ENDPOINT\_ID\_2 value in every follow-up response (update notification) within this observation relationship.



By this means, a client is able to update its internal "server ID/IP address - mapping" with every update notification.

#### 4. Examples

Within the figures in this section MID refers to message ID, whereas ID\_x refers to the value of the ENDPOINT\_ID\_x option.

##### 4.1. NON request and NON response

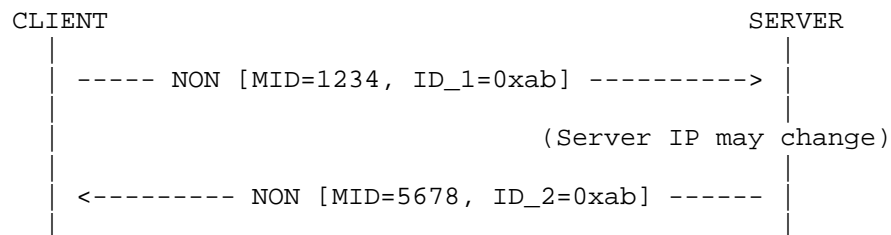


Figure 7: NON requests and NON response

##### 4.2. NON request, CON response, and empty ACK

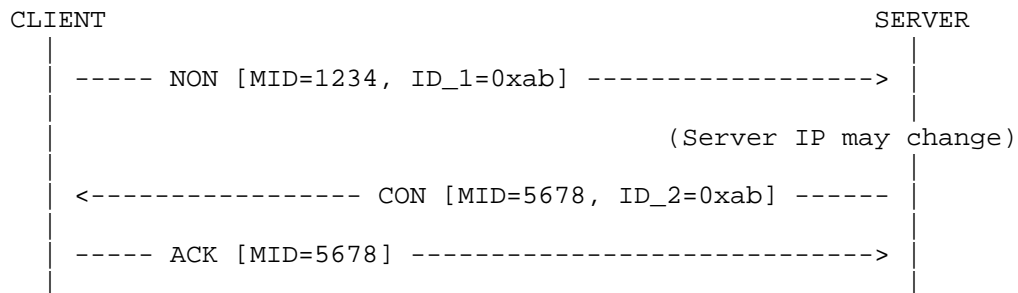


Figure 8: NON requests and NON response

##### 4.3. CON request, empty ACK, and NON response

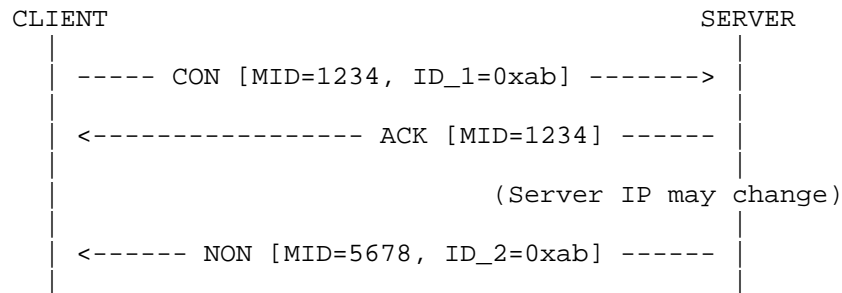


Figure 9: NON requests and NON response

## 4.4. CON request, empty ACK, CON response, and empty ACK

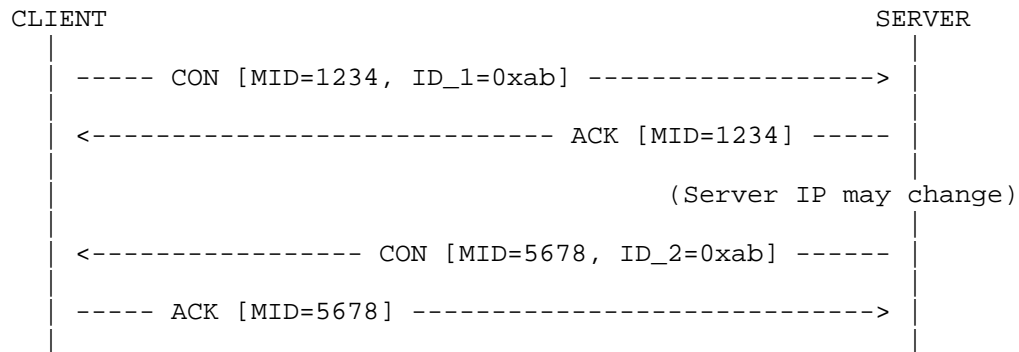


Figure 10: CON request, empty ACK, CON response, and empty ACK

## 4.5. Server IP address changes during observation

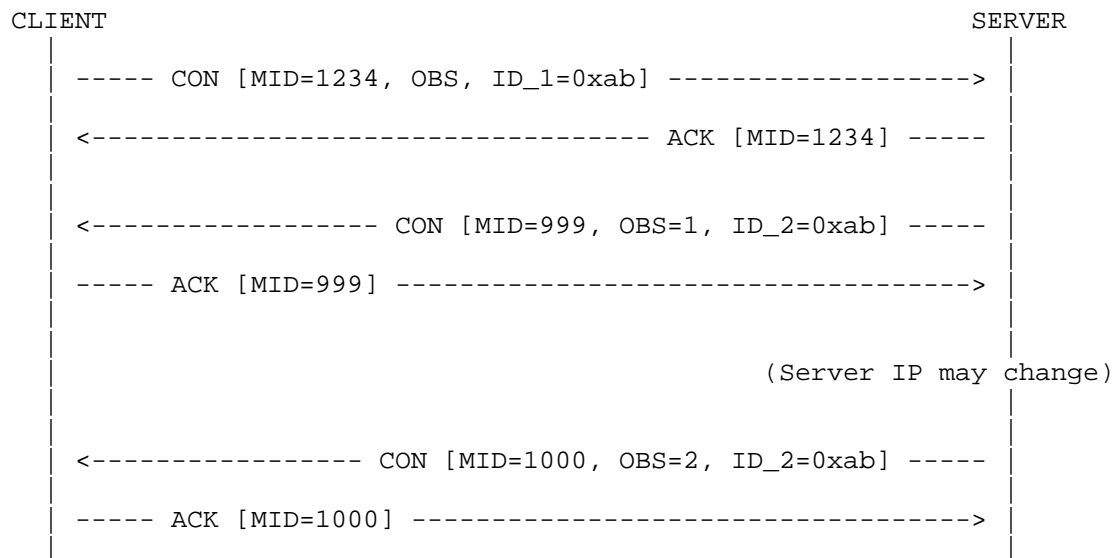


Figure 11: Server IP address changes during observation

## 4.6. Client IP address changes during observation

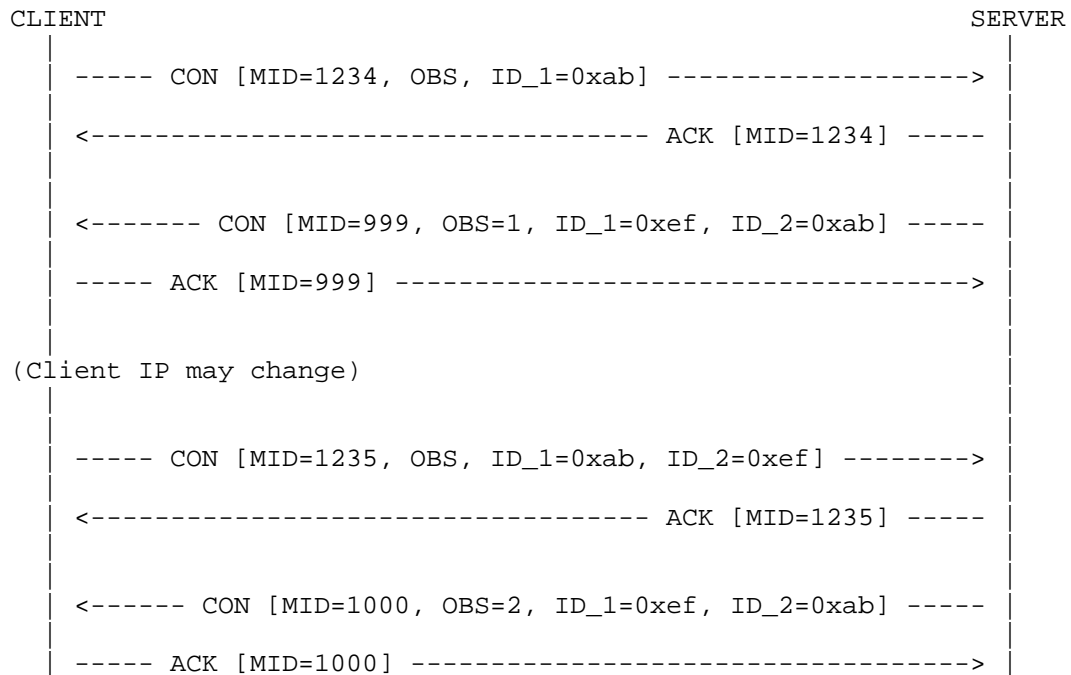


Figure 12: Client IP address changes during observation

## 5. Acknowledgements

No acknowledgements, yet...

## 6. IANA Considerations

This memo includes no request to IANA.

(It's good - indeed pretty much mandatory now - to have an explicit note because otherwise IANA wastes cycles trying to figure out if something is needed..)

## 7. Security Considerations

To avoid an eval interruption of an ongoing Message Exchange, DTLS SHOULD be used to encrypt the CoAP messages.

## 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, <<http://xml.resource.org/public/rfc/html/rfc2119.html>>.
- [coap] Shelby, et.al., , "Constrained Application Protocol", 2013, <<https://datatracker.ietf.org/doc/draft-ietf-core-coap/>>.

### 8.2. Informative References

- [block] Borman, et al., , "Blockwise transfers in CoAP", 2013, <<https://datatracker.ietf.org/doc/draft-ietf-core-block/>>.
- [observe] Hartke, , "Observing Resources in CoAP", 2014, <<https://datatracker.ietf.org/doc/draft-ietf-core-observe/>>.

### Author's Address

Oliver Kleine  
University of Luebeck, Institute of Telematics  
Ratzeburger Allee 160  
Luebeck 23552  
DE

Email: [kleine@itm.uni-luebeck.de](mailto:kleine@itm.uni-luebeck.de)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 5, 2015

M. Koster  
ARM Limited  
A. Keranen  
J. Jimenez  
Ericsson  
July 4, 2014

Message Queueing in the Constrained Application Protocol (CoAP)  
draft-koster-core-coapmq-00.txt

Abstract

The Constrained Application Protocol, CoAP, and related extensions are intended to support machine-to-machine communication in systems where one or more nodes are resource constrained, in particular for low power wireless sensor networks. This document defines publish-subscribe message queuing functionality for CoAP that extends the capabilities for supporting nodes with long breaks in connectivity and/or up-time.

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 5, 2015.

Copyright Notice

Copyright (c) 2014 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 . . . . .	4
3.1. RD Server with associated CoAP-MQ Broker . . . . .	4
3.2. Client Endpoint . . . . .	5
3.3. Server Endpoint . . . . .	5
3.4. Publish-Subscribe Topics . . . . .	5
4. CoAP-MQ Registration and discovery . . . . .	6
4.1. Register PubSub Endpoint . . . . .	6
4.2. Unregister Endpoint . . . . .	7
5. CoAP-MQ Functions and Interactions . . . . .	7
5.1. Client Role Endpoint Functions . . . . .	8
5.1.1. Client Endpoint PUBLISH to CoAP-MQ broker . . . . .	8
5.1.2. Client Endpoint SUBSCRIBE, Broker PUBLISH . . . . .	8
5.1.3. Client Endpoint GET from CoAP-MQ Broker . . . . .	9
5.2. Server Role Endpoint Functions . . . . .	9
5.2.1. CoAP-MQ broker SUBSCRIBES to Server Role EP . . . . .	10
5.2.2. CoAP-MQ Broker Publishes to Server Role Endpoint . . . . .	10
5.2.3. CoAP-MQ Broker GET from Server Role Endpoint . . . . .	11
6. Enabling Multiple Publishers . . . . .	11
6.1. Creating a Topic . . . . .	11
6.2. Publishing a Topic from Multiple Publishers . . . . .	12
6.3. Subscribing to a topic with multiple publishers . . . . .	12
7. Sleep-Wakeup Operation and Message Queueing . . . . .	13
8. Security Considerations . . . . .	13
9. IANA Considerations . . . . .	14
9.1. Resource Type value 'core.pubsub.client' . . . . .	14
9.2. Resource Type value 'core.pubsub.server' . . . . .	14
10. Acknowledgements . . . . .	14
11. References . . . . .	15
11.1. Normative References . . . . .	15
11.2. Informative References . . . . .	15
Authors' Addresses . . . . .	15

## 1. Introduction

The Constrained Application Protocol (CoAP) [6] supports machine to machine communication across networks of constrained devices. One important class of constrained devices includes devices that are intended to run for years from a small battery, or by scavenging

energy from their environment. These devices spend most of their time in a sleeping state with no network connectivity.

Devices may also have limited reachability due to certain middle-boxes, such as Network Address Translators (NATs) or firewalls. Such devices must communicate using a client role, whereby the endpoint is responsible for initiating communication.

This document specifies the means for nodes with limited reachability to communicate using simple extensions to CoAP and the CoRE Resource Directory [4]. The extensions enable publish-subscribe communication using a Message Queue (MQ) broker node that enables store-and-forward messaging between two or more nodes.

The mechanisms specified in this document are meant to address key design requirements from earlier CoRE drafts covering sleepy node support and mirror server.

## 2. Terminology

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this specification are to be interpreted as described in [1].

This specification requires readers to be familiar with all the terms and concepts that are discussed in [8] and [2]. Readers should also be familiar with the terms and concepts discussed in [6] and [4]. The URI template format, see [3], is used to describe the REST interfaces defined in this specification.

The following entities are used in this specification:

**CoAP Message Queue (CoAP-MQ) Service** A service provided by a node or system where CoAP messages sent by one endpoint to another are queued (stored) by intermediate node(s) and forwarded only when suitable, e.g., when the message recipient endpoint is not sleeping.

**CoAP-MQ Broker** A server node capable of storing messages to and from other nodes and able to match subscriptions and publications in order to route messages to right destinations.

**CoAP-MQ Endpoint** An endpoint that implements the MQ function set defined in Section 5. A CoAP-MQ endpoint has two potential roles, CoAP-MQ client and CoAP-MQ server

This specification makes use of the following additional terminology:



**CoAP Message Queue (CoAP-MQ) Service** A service provided by a node or system where CoAP messages sent by one endpoint to another are queued (stored) by intermediate node(s) and forwarded only when suitable, e.g., when the message recipient endpoint is not sleeping.

**CoAP-MQ Broker** A server node capable of storing messages to and from other nodes and able to match subscriptions and publications in order to route messages to right destinations.

**CoAP-MQ function set** A group of well-known REST resources that together provide the CoAP-MQ service.

**CoAP-MQ Endpoint** An endpoint that implements the CoAP-MQ function set. A CoAP-MQ endpoint has two potential roles, CoAP-MQ client and CoAP-MQ server.

**Publish-Subscribe (pub-sub)** A messaging paradigm where messages are published (e.g., to a broker) and potential receivers can subscribe to receive the messages.

**Topic** In Publish-Subscribe systems a topic is a unique identifying string for a particular item or object being published and/or subscribed to.

### 3. Architecture

#### 3.1. RD Server with associated CoAP-MQ Broker

Figure 1 shows an example architecture of a CoAP-MQ capable service. A Resource Directory service accepts registrations and registration updates from one or more endpoints and hosts a resource discovery service for one or more web application clients. State information is updated from the endpoints to the CoAP-MQ broker. Web clients subscribe to the state of the endpoint from the CoAP-MQ broker, and publish updates to the endpoint state through the CoAP-MQ broker. The CoAP-MQ broker performs a store-and-forward function between web clients and the CoAP-MQ capable endpoints. The CoAP-MQ broker is also responsible for acting as a proxy, returning the last published value to web clients or other endpoints on behalf endpoints that are sleeping.

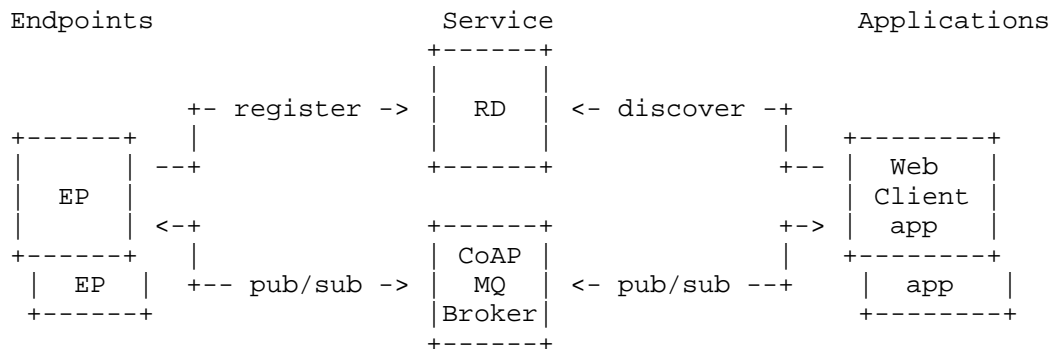


Figure 1: CoAP MQ Architecture

### 3.2. Client Endpoint

Client endpoints initiate all interactions with the RD and MQ broker. If the endpoint is an actuator it will need to either use CoAP Observe [I-D.ietf-core-observe] or periodically poll the MQ broker to check for updates. A CoAP-MQ client endpoint **MUST** use CoAP PUT operations to update its state on the MQ broker. An endpoint **SHOULD** update the RD periodically to indicate that it is still alive even if it has no pending data updates. Endpoints can operate in the client role even if not directly reachable from the CoAP-MQ broker or RD server.

### 3.3. Server Endpoint

Server endpoint interactions require the CoAP-MQ broker to perform the client role, initiating interaction with the server endpoint. The CoAP-MQ broker **MAY** then use PUT operations to update state at the server endpoint, and **MAY** use GET or GET+Observe to subscribe to resources at the endpoint. Server mode endpoints are required to be reachable from the CoAP-MQ broker. In a network containing both client and server endpoints, client endpoints **MAY** subscribe to server endpoints directly, in broker-less configurations, using RD or core-link-format metadata in .well-known/core to discover the CoAP-MQ capabilities and using GET+Observe to subscribe to the desired topics.

### 3.4. Publish-Subscribe Topics

Topic are strings used to identify particular resources and objects in publish-subscribe systems. Topics are conventionally formed as a hierarchy, e.g. "/sensors/weather/barometer/pressure".

Implementations are free to map topics to resources, reusing existing resource addressing schemes.

#### 4. CoAP-MQ Registration and discovery

An endpoint wishing to use a CoAP-MQ broker registers with an RD server that advertises a service having the the "core.mq" attribute. This indicates that there is a CoAP-MQ broker at the location returned by the discovery query as shown in Figure 2. The endpoint registers topics using the "rt=core.pubsub.client" or "core.pubsub.server" (or both) attributes to indicate intention to use CoAP-MQ and which roles are supported.

A server that implements a CoAP\_MQ broker MAY advertize this capability by registering the rt="core.mq" with an associated Resource Directory. If a server advertizes as a CoAP-MQ Broker, it MUST support the transactions described in section 5 of this document. As server that implements the CoAP-MQ Broker MAY also implement sleeping endpoint and message queueing support referred to in Section 6 of this document.

##### 4.1. Register PubSub Endpoint

Figure 2 shows the flow of the registration operation. Discovery proceeds as per CoRE Resource Directory[I-D.ietf-core-resource-directory-01]. When an endpoint wishes to use CoAP-MQ, it discovers the rt="core.mq" attribute at the RD service associated with the CoAP-MQ broker and registers its CoAP-MQ resources with the RD server by registering topics having the rt="core.pubsub" attribute. Topics are created using an initial POST operation to the registered topic or any valid sub-topic. For example, if the registered topic is "/sensors/weather", the sub-topic "/sensors/weather/barometer" is created using a POST to "/mq/sensors/weather/barometer". An implementation MAY mix CoAP-MQ resources and CoAP REST resources on the same endpoint. Endpoint registration proceeds as per normal RD registration.

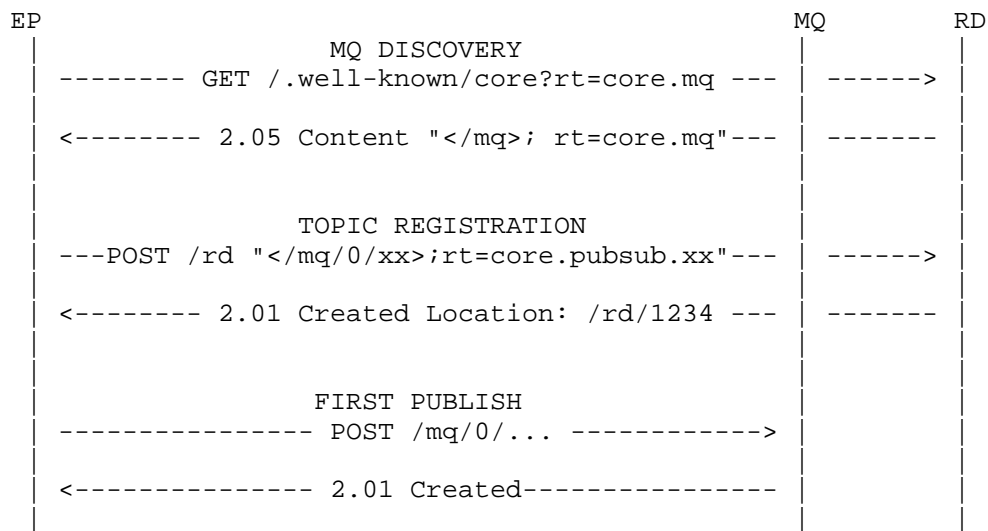


Figure 2: Discovery and Registration

#### 4.2. Unregister Endpoint

CoAP-MQ endpoints indicate the end of their registration tenure by either explicitly unregistering, as in Figure 3, or allowing the lifetime of the previous registration to expire.

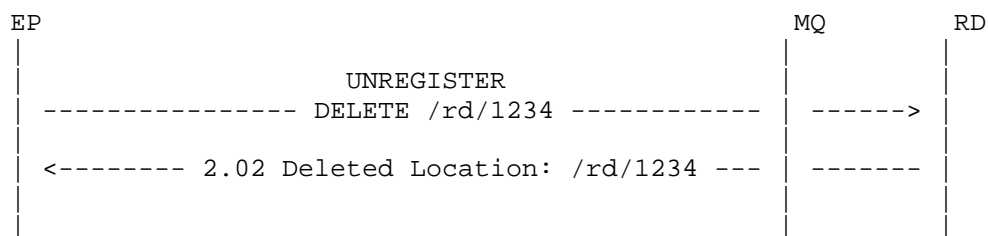


Figure 3: Unregister Endpoint

### 5. CoAP-MQ Functions and Interactions

This section describes the transaction flows and interactions between CoAP-MQ endpoints and CoAP-MQ brokers. Client endpoint functions are used by endpoints implementing the client role, for example to enable sleep/wakeup and partial connectivity. Server role endpoint functions are used by endpoints implementing the server role, for

example always on, reachable, endpoints. An endpoint implementation MAY support both client role and server role at an endpoint. A CoAP-MQ broker MUST implement support for both client role and server role endpoints.

### 5.1. Client Role Endpoint Functions

This section describes the transaction flows and interactions between CoAP-MQ endpoints and CoAP-MQ brokers where the endpoint supports the client role. A client registering the "core.pubsub.client" attribute MUST support the client role endpoint functions and interactions described in this section.

#### 5.1.1. Client Endpoint PUBLISH to CoAP-MQ broker

Client endpoint PUBLISH updates to CoAP-MQ broker.

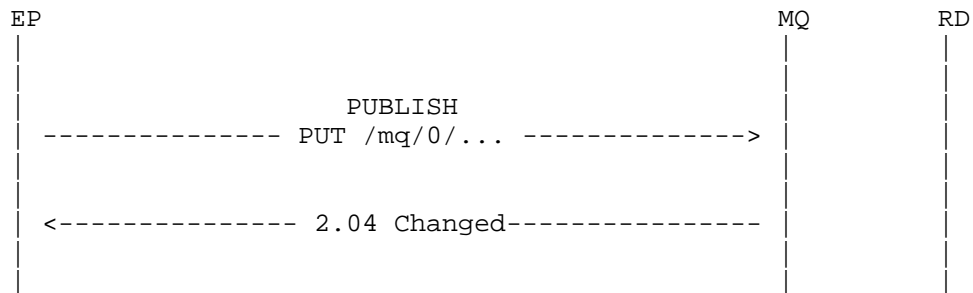


Figure 4: Client Role PUBLISH from EP to Broker

#### 5.1.2. Client Endpoint SUBSCRIBE, Broker PUBLISH

Client mode endpoint subscribes to the topic at the CoAP-MQ broker using GET+Observe. Published updates to the CoAP-MQ broker are published to the Endpoint using Observe response tokens. Client endpoint MAY update actuator or resource based on received values associated with responses. A CoAP-MQ broker MUST publish updates to subscribed endpoints upon receiving published updates on the associated topics.

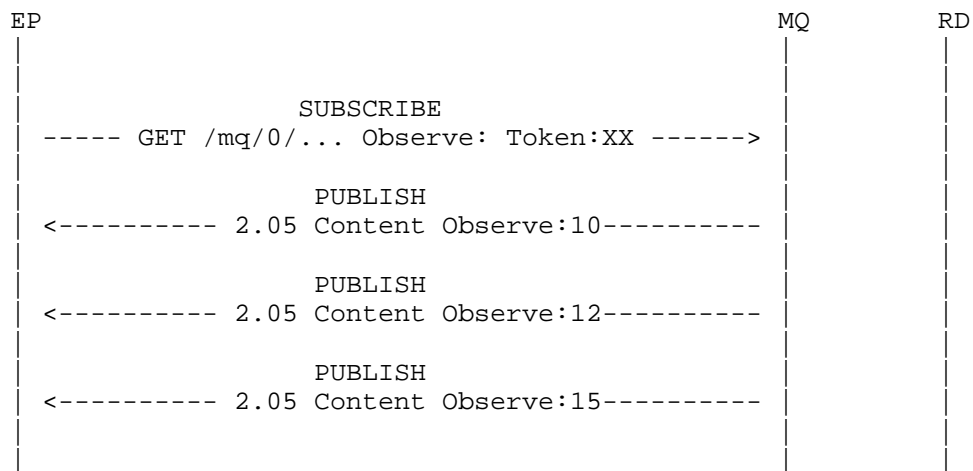


Figure 5: Client Role Endpoint SUBSCRIBE, Broker PUBLISH to Endpoint

#### 5.1.3. Client Endpoint GET from CoAP-MQ Broker

Client mode endpoint MAY issue GET to topic without Observe as needed to obtain last published state from the CoAP-MQ broker.

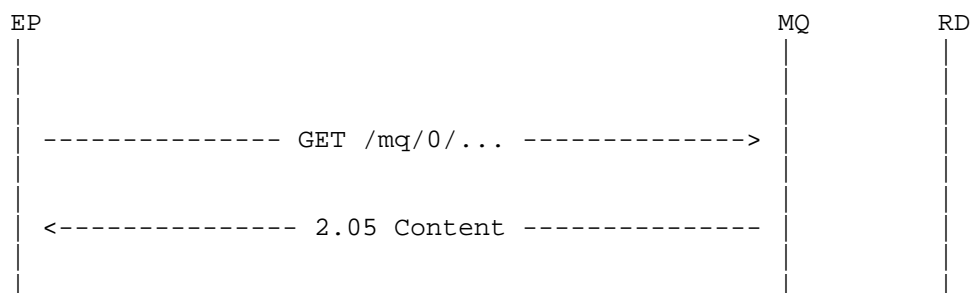


Figure 6: Client EP GET from CoAP-MQ Broker

#### 5.2. Server Role Endpoint Functions

This section describes the transaction flows and interactions between CoAP-MQ endpoints and CoAP-MQ brokers where the endpoint supports the server role. An endpoint registering the "core.pubsub.server" attribute MUST support these functions and interactions.

## 5.2.1. CoAP-MQ broker SUBSCRIBES to Server Role EP

The server mode endpoint requires the CoAP-MQ broker to act as a client and subscribe to a resource on the endpoint using GET + Observe. A CoAP-MQ broker MAY subscribe to topics registered by a server role endpoint at any time. A CoAP-MQ broker MUST subscribe to a topic registered by a server role endpoint upon receiving a subscription on the associated topic. A CoAP-MQ broker MUST forward state updates received from a publishing endpoint to all endpoints subscribed on the associated topic. Figure 7 shows the flow of a CoAP-MQ Broker subscribing to a server role endpoint.

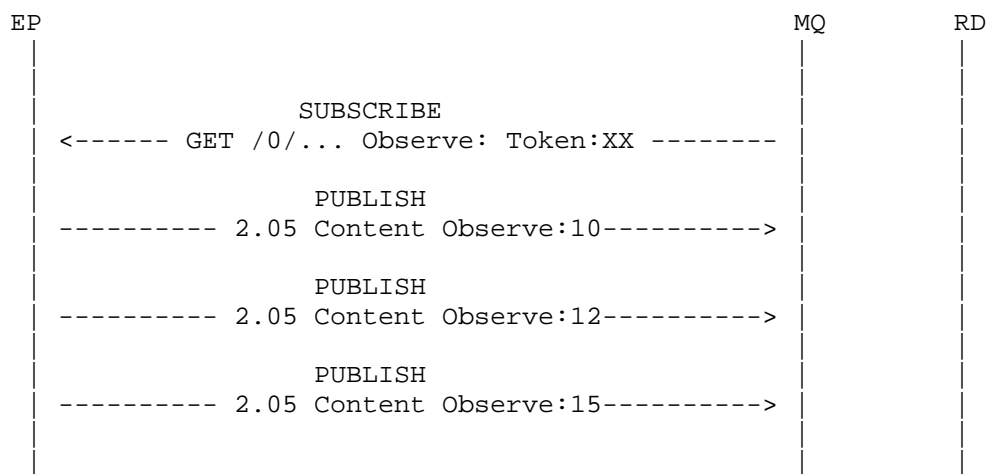


Figure 7: Broker SUBSCRIBE to Server Role EP

## 5.2.2. CoAP-MQ Broker Publishes to Server Role Endpoint

CoAP-MQ broker MUST update server mode endpoint using PUT when upon receiving updates published on the associated topics. Endpoint server MAY update actuator or resource upon receiving published state updates from the broker.

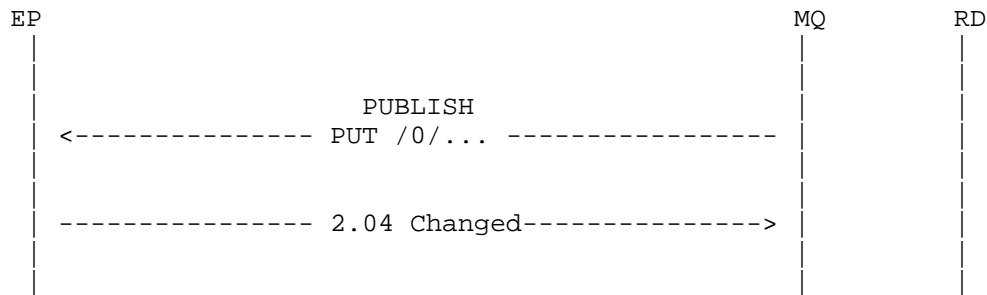


Figure 8: Broker PUBLISH to Server Role EP

### 5.2.3. CoAP-MQ Broker GET from Server Role Endpoint

CoAP-MQ broker MAY issue GET without Observe as needed to obtain state update from the server role endpoint.

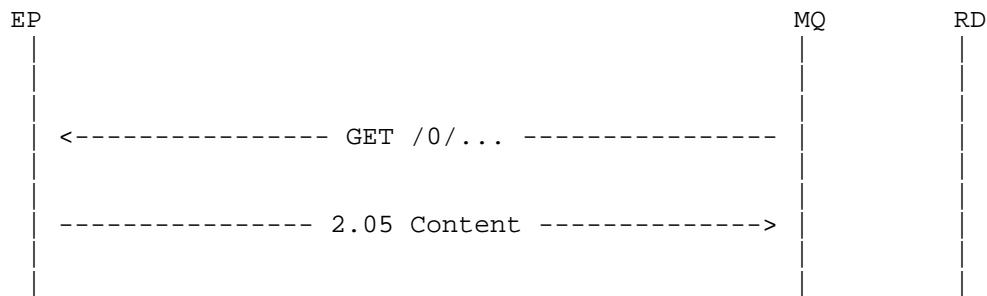


Figure 9: Broker GET from Server Role Endpoint

## 6. Enabling Multiple Publishers

### 6.1. Creating a Topic

After registration of the EP in the RD and discovering the CoAP-MQ function, a designated EP acting as publisher for a particular topic is responsible for creating such topic. To do so, it will have to register the new topic in the RD and create it on the MQ function by doing a first publication as shown in Figure 2.

After the topic has been created in the CoAP-MQ broker, the broker will be responsible of hosting this resource and to queue messages published on it as explained in Section 5



## 6.2. Publishing a Topic from Multiple Publishers

After the topic has been registered in the RD and is created in the CoAP-MQ broker, any device with the right access permissions can publish on that topic by using the topic field. For example in the following diagram, both EP1 and EP2 update the same topic that EP3 has previously subscribed to.

After the topic has been created in the CoAP-MQ Broker, the broker will be responsible of hosting this resource and to queue messages published on it as explained in Section 5

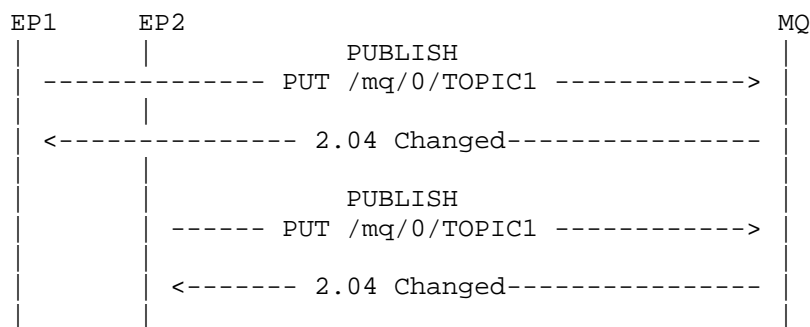


Figure 10: Multiple CoAP-MQ EPs PUBLISH to Broker

## 6.3. Subscribing to a topic with multiple publishers

Subscription to this topic is the same as in Section 5, since it acts as any other resource. Following the previous example, if EP3 is subscribed to the shared topic, it should receive two updates from both EP1 and EP2.

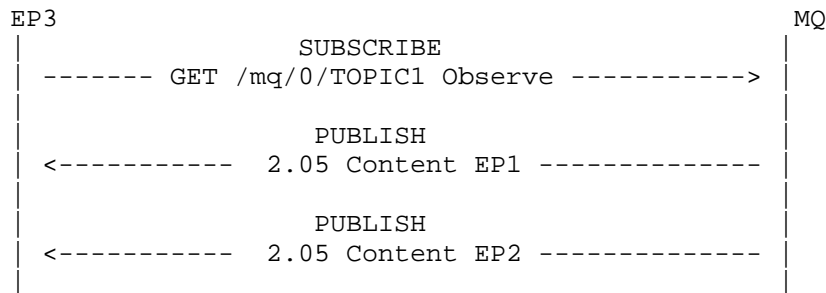


Figure 11: CoAP-MQ Endpoint SUBSCRIBE to Broker

## 7. Sleep-Wakeup Operation and Message Queueing

A CoAP-MQ broker MAY implement support for sleeping endpoints and queueing of messages as provided for in [7]

## 8. Security Considerations

CoAP-MQ re-uses CoAP [6], CoRE Resource Directory [4], and Web Linking [8] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP-MQ broker and the endpoints SHOULD authenticate each other and enforce access control policies. A malicious EP could subscribe to data it is not authorized to or mount a denial of service attack against the broker by publishing a large number of resources. The authentication can be performed using the already standardized DTLS offered mechanisms, such as certificates. DTLS also allows communication security to be established to ensure integrity and confidentiality protection of the data exchanged between these relevant parties. Provisioning the necessary credentials, trust anchors and authorization policies is non-trivial and subject of ongoing work.

The use of a CoAP-MQ broker introduces challenges for the use of end-to-end security between the end device and the cloud-based server infrastructure since brokers terminate the exchange. While running separate DTLS sessions from the EP to the broker and from broker to the web application protects confidentiality on those paths, the client/server EP does not know whether the commands coming from the broker are actually coming from the client web application. Similarly, a client web application requesting data does not know whether the data originated on the server EP. For scenarios where end-to-end security is desirable the use of application layer security is unavoidable. Application layer security would then provide a guarantee to the client EP that any request originated at the client web application. Similarly, integrity protected sensor

data from a server EP will also provide guarantee to the client web application that the data originated on the EP itself. The protected data can also be verified by the intermediate broker ensuring that it stores/caches correct request/response and no malicious messages/requests are accepted. The broker would still be able to perform aggregation of data/requests collected.

Depending on the level of trust users and system designers place in the CoAP-MQ broker, the use of end-to-end encryption may also be envisioned. The CoAP-MQ broken would then only be able to verify the request/response message/commands and store-and-forward without being able to inspect the content. The solution for providing application layer security will depend on the utilized data encoding. For example, with a JSON-based data encoding the work from the JOSE working group could be re-used. Distribution of the credentials for accomplishing end-to-end security might introduce challenges if previously unknown parties need to exchange data.

## 9. IANA Considerations

This document registers two attribute values in the Resource Type (rt=) registry established with RFC 6690 [2].

### 9.1. Resource Type value 'core.pubsub.client'

- o Attribute Value: core.pubsub.client
- o Description: Section X of [[This document]]
- o Reference: [[This document]]
- o Notes: None

### 9.2. Resource Type value 'core.pubsub.server'

- o Attribute Value: core.pubsub.server
- o Description: Section Y of [[This document]]
- o Reference: [[This document]]
- o Notes: None

## 10. Acknowledgements

Hannes Tschofenig, Zach Shelby

## 11. References

### 11.1. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [2] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [3] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.
- [4] Shelby, Z., Bormann, C., and S. Krco, "CoRE Resource Directory", draft-ietf-core-resource-directory-01 (work in progress), December 2013.
- [5] Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.
- [6] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.
- [7] Open Mobile Alliance, "OMA LightweightM2M v1.0", <http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0>, 12 2013.

### 11.2. Informative References

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

## Authors' Addresses

Michael Koster  
ARM Limited

Email: Michael.Koster@arm.com

Ari Keranen  
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez  
Ericsson

Email: [jaime.jimenez@ericsson.com](mailto:jaime.jimenez@ericsson.com)

core  
Internet-Draft  
Intended status: Standards Track  
Expires: December 26, 2014

K. Li  
Huawei Technologies  
G. Wei  
BUPT  
June 24, 2014

CoAP Option Extension: NodeId  
draft-li-core-coap-node-id-option-01

Abstract

CoAP is a RESTful application protocol for constrained nodes and networks. This specification provides a simple extension for CoAP, the NodeId Option. This Option can be used to identify the node, either the client or the server.

Note

Discussion and suggestions for improvement are requested, and should be sent to [core@ietf.org](mailto: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 December 26, 2014.

Copyright Notice

Copyright (c) 2014 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
1.1. Justification . . . . .	2
1.2. Terminology . . . . .	3
2. NodeId Option Extension . . . . .	3
2.1. NodeId Option Definition . . . . .	3
2.2. Using the NodeId Option . . . . .	3
2.2.1. Registration . . . . .	3
2.2.2. Usage . . . . .	4
3. Example . . . . .	4
3.1. Registration example . . . . .	4
3.2. Notification example . . . . .	5
4. Security Considerations . . . . .	6
5. IANA Considerations . . . . .	6
6. Acknowledgements . . . . .	6
7. References . . . . .	6
7.1. Normative References . . . . .	6
7.2. Informative References . . . . .	6
Authors' Addresses . . . . .	7

## 1. Introduction

This specification adds a new option NodeId to CoAP [I-D.ietf-core-coap]. The main purpose is for a node to have a unique identity, named as NodeId. The NodeId is used by the node, as a sender, to identify itself to the recipient, during registration and communications.

### 1.1. Justification

A node is set to have a NodeId and the node nerver changes its NodeId. There are several scenarios to have the NodeId to identify the nodes.

In the network, it is quite common for a node to change its IP address due to rebooting. After the server or client changes its IP address, the peer of the other side lacks a facility to correlate the old IP address and the new IP address as the same node. This will cause the other side to lose some contexts. If the node can use NodeId after its IP address being changed, it is very easy for the node to correlate the old IP address and the new one by NodeId.

In the multicast observation case, after a client sends a multicast observation request to a group URI, e.g. `all.bldg6.example.com`, the client will receive multiple notifications from different servers of the multicast group with the same token as specified in the multicast request. As a result, the client can't use token to correlate multicast request and notification responses. The client may use the IP address extracted from UDP/IP transport/network layers to differentiate servers and responses. If a server changes its IP address and sends back the notification, the client can't determine where the notification message comes from any more. In this case, if `NodeId` is included in the notifications, it can be used to correlate multicast request and subsequent notifications by the node.

The `NodeId` can also be used for authentication and authorization of the node.

## 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 [RFC2119].

## 2. NodeId Option Extension

### 2.1. NodeId Option Definition

Type	C	U	N	R	Name	Format	Length	Default
TBD	-	-	-	-	NodeId	string	1-255 B	(none)

The NodeId option is used to identify the node. The value SHOULD be unique for each node within a Resource Directory server. The value can be in the form of Binary bits, IMEI (International Mobile Equipment Identity number), IEEE 802 MAC Address, or other identifiers which can uniquely identify itself. Usually the value is pre-configured or pre-provisioned in the node.

## 2.2. Using the NodeId Option

### 2.2.1. Registration

When a node registers itself to the Resource Directory server, the registration request SHOULD contain its node identifier. This node identifier MAY be included in the NodeId option in the registration request, or MAY be included in the URI-Query option as specified in [I-D.ietf-core-resource-directory].



### 2.2.2. Usage

This option MAY be used in a CoAP request or response. And it can be used to correlate the messages for a node in case of IP address change. As long as a node changes its IP address, the NodeId SHALL be included in the first request and response and sent in CON message.

Whenever the node reboots or moves, the NodeId MUST NOT change. And the node SHOULD send the updated IP address with the NodeId to the RD server, using the update interface as specified in [I-D.ietf-core-resource-directory]. This informs the RD server a mapping relation between the new IP address and the NodeId identified node.

For the usage of notifications in the observe, when a server in a group receives a multicast observe request, it SHOULD include a NodeId option in the notifications. In this way, even the server changes the IP address, the client can still correlate all the notifications with this server.

It is recommended to use NodeId as identifier during authentication and authorization.

Todo: How to use it for authentication and authorization?

This option is "elective". It MUST NOT occur more than once.

## 3. Example

### 3.1. Registration example

This section gives a short example with a message flow that illustrates the use of the NodeId option in a registration request.

This example (Figure 1) shows that the requester includes its NodeId in the registration request.

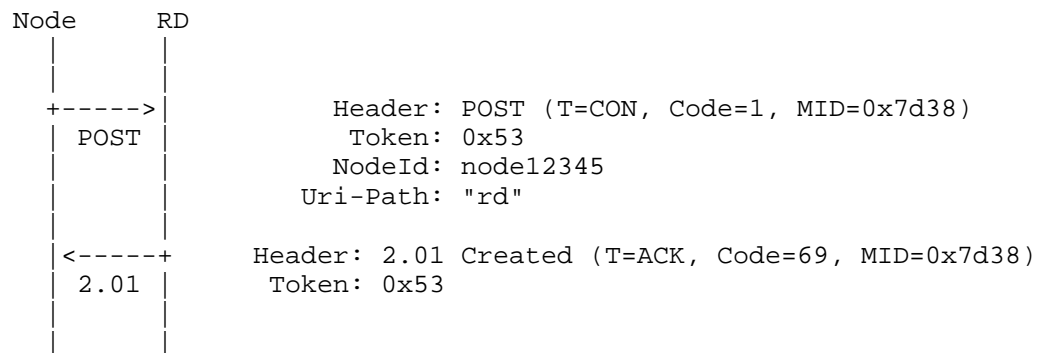


Figure 1: NodeId Option in a registration request

### 3.2. Notification example

This section gives a short example with a message flow that illustrates the use of the NodeId option in an observe notification.

This example (Section 3.2) shows that the server includes its NodeId option in an observe notification after IP address changes.

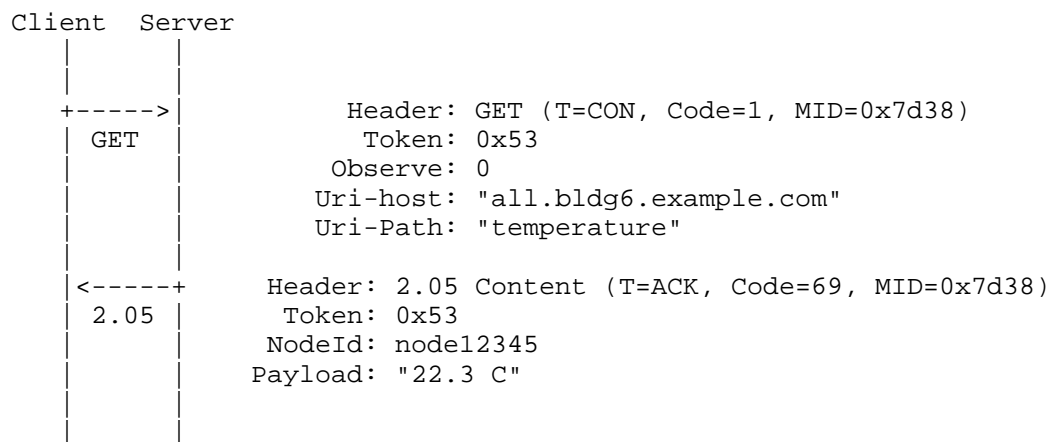


Figure 2: NodeId Option in an observe notification

#### 4. Security Considerations

This presents no security considerations beyond those in section 9 and 11 of the base CoAP specification [I-D.ietf-core-coap].

#### 5. IANA Considerations

The IANA is requested to add the following "CoAP Option Numbers" entry as per Section 12.2 of [I-D.ietf-core-coap].

Number	Name	Reference
TBD	NodeId	Section 2

#### 6. Acknowledgements

The authors of this draft would like to thank the Esko Dijk, Carsten Bormann, Bert Greevenbosch and Klaus Hartke for the email discussions on this issue.

#### 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.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-13 (work in progress), April 2014.
- [I-D.ietf-core-resource-directory]  
Shelby, Z., Bormann, C., and S. Krco, "CoRE Resource Directory", draft-ietf-core-resource-directory-01 (work in progress), December 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

##### 7.2. Informative References

[I-D.bormann-coap-misc]

Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-26 (work in progress), December 2013.

Authors' Addresses

Kepeng Li  
Huawei Technologies  
Huawei Base, Bantian, Longgang District  
Shenzhen, Guangdong 518129  
P. R. China

Phone: +86-755-28974289  
Email: likepeng@huawei.com

Gengyu Wei  
Beijing University of Posts & Telecommunications  
No.10 Xitucheng Road, Haidian District  
Beijing 100876  
P. R. China

Phone: +86-10-62283067  
Email: weigengyu@bupt.edu.cn

core  
Internet-Draft  
Intended status: Standards Track  
Expires: January 5, 2015

K. Li  
B. Greevenbosch  
Huawei Technologies  
E. Dijk  
Philips Research  
S. Loreto  
Ericsson  
July 04, 2014

CoAP Option Extension: Patience  
draft-li-core-coap-patience-option-04

Abstract

CoAP is a RESTful application protocol for constrained nodes and networks. This specification provides a simple extension for CoAP, the Patience option. This option informs a recipient of the preferred time frame for a request or response depending on usage context. In a unicast request, it indicates the patience a client has in waiting for a response. The CoAP server tries to return the response within the specified time frame. In a multicast request, it indicates the patience a server should have in sending its response. The recipient would then try to randomly delay its response within the time frame that the requester indicated or computed by the recipient itself. In a CoAP observe notification, it indicates the patience an observer should have in both waiting for a subsequent notification and in re-establishing an observation relation.

Note

Discussion and suggestions for improvement are requested, and should be sent to [core@ietf.org](mailto: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 January 5, 2015.

#### Copyright Notice

Copyright (c) 2014 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
1.1. Justification . . . . .	3
1.2. Terminology . . . . .	4
2. Patience Option Extension . . . . .	4
2.1. Patience Option Definition . . . . .	4
2.2. Using the Patience Option . . . . .	4
2.2.1. Unicast usage . . . . .	5
2.2.2. Multicast usage . . . . .	6
2.2.3. Observe usage . . . . .	6
2.3. Detection of IP unicast or multicast CoAP request . . . . .	7
3. Example . . . . .	7
3.1. Unicast Usage Example . . . . .	7
3.2. Observe Usage Example . . . . .	8
4. Security Considerations . . . . .	9
5. IANA Considerations . . . . .	9
6. Acknowledgements . . . . .	9
7. References . . . . .	9
7.1. Normative References . . . . .	9
7.2. Informative References . . . . .	9
Authors' Addresses . . . . .	10

#### 1. Introduction

This specification adds a new option Patience to CoAP [RFC7252]. The main purpose is for the requester to inform the recipient of the preferred time frame for a response. In the unicast request case, it is used to indicate the patience it has in waiting for a response. It then indicates "a response is most useful within the specified time frame". In the multicast request case, it indicates the

patience that a server should have in sending a response. In other words, it indicates "if possible please delay your response by a randomly chosen time within the specified time frame". A second purpose is for use by a server when sending CoAP observe [I-D.ietf-core-observe] notifications, to indicate the maximum time an observer should wait (i.e. patience of the observer) before starting any observation relationship recovery.

### 1.1. Justification

In the unicast case, it is useful for the requester (client) to indicate that the response is required to be returned within a certain amount of time. For example, the requester could require a response within 2 seconds, otherwise the response is not of interest anymore. With this indication of the patience for a response, the requester knows how long it should wait for the response, and it needs to keep the state of the request only for the indicated time. After this period, the request will be given up. It can avoid that the recipient wastes resources by sending a response which already exceeds the set patience timeout of the requester.

In the multicast case, if a server decides 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. The length of this period is called the Leisure, and defined in [RFC7252]. The same document specifies how to compute the a rough lower bound for Leisure, as well as the DEFAULT\_LEISURE. A Patience option, if present, can be used as an upper bound for the Leisure, i.e. the server SHOULD respond before the time frame indicated by Patience has been exceeded.

In an observe scenario, it is useful for a server to indicate to an observer that, after the period of time in the Max-Age option has expired, a new notification will be sent within the time interval indicated by the Patience option. The server may use this to send notifications with a dithered delay i.e. randomly chosen within the Patience-specified time interval, when there are many CoAP clients simultaneously observing a resource on the server, avoiding network congestion issues. Another use is for the server to delay sending a new notification because e.g. the resource has not changed. The observer in this case can assume that the server will do its best to deliver a notification at least before the Patience time interval runs out.

If the Patience option is combined with Observe option in a request, currently it indicates the maximum time an observer is prepared to wait for an initial notification.

## 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 [RFC2119].

## 2. Patience Option Extension

### 2.1. Patience Option Definition

No.	C	U	N	R	Name	Format	Length	Default
28			x		Patience	see below	1 B	(none)

The value carried in the Patience Option is in a specific format resembling a pseudo-Floating Point value (as in [I-D.bormann-coap-misc] Appendix B.2):

```

      0
      0 1 2 3 4 5 6 7
+-----+
|         T         | TX|
+-----+
```

T = Time

TX = Time Exponent

where the patience time is calculated as:

Patience time =  $2^{(TX * 4 + 3)} * T$

The value of the Patience option is calculated in milliseconds or alternatively mibiseconds (1/1024s) if this would ease numerical operations on above values on constrained platforms. The minimum non-zero patience time is 8ms, when TX=0, T=1 and a milliseconds time base is used. The maximum patience time is then 2064384ms, around 34 minutes, when TX=3 and T=63.

The Patience option is "elective". It MUST NOT occur more than once.

### 2.2. Using the Patience Option

The semantics of the Patience Option depends on its usage context, as detailed in below sections.



### 2.2.1. Unicast usage

In the unicast case, this option is used by a CoAP client to indicate the maximum time a requester is prepared to wait for a response.

The requester adds the Patience option to any request for which it is prepared to wait for a response. The requester sets the option to the maximum time that it is prepared to wait.

The Patience option applies to both piggy-backed response and separate response. For a separate response, the patience applies to the actual response after the ACK. ACK should be sent immediately upon receipt of the CON message.

TBD: In case a requester retransmits a request, the Patience Option value MAY be decreased by an amount of time equivalent to the time since the previous transmission attempt. In case a requester did not receive an ACK to a confirmable request and a time interval of at least the interval indicated in the Patience Option of the request has passed, the requester SHOULD give up the request.

The recipient interprets this option as the maximum time between receipt of the complete request and the time that it begins sending the response. The requester will observe a longer time interval between request and response, as network transit and processing by proxies add delays. If timing is critical, the requester SHOULD consider the possible delays and choose the value for the option accordingly.

The recipient MAY apply a lower value to the patience timeout based on local policy. A recipient MAY choose to take longer to produce a response, at the risk that the requester is no longer able to use the response.

In case that the CoAP message is transmitted through a proxy, the Proxy MAY reduce the value of a Patience option based on a local policy (e.g. to consider the maximum time that an idle connection is kept open by a local NAT or Firewall). A Proxy MAY add a Patience option if none is present. The value in the Patience option MUST NOT be increased or removed.

If the requester does not receive a response within the indicated response time, the requester SHOULD consider the request as failed. If the recipient can't provide a response within the required time, the recipient SHOULD discard the request.

### 2.2.2. Multicast usage

In the multicast case, Leisure is defined in [RFC7252] to work as a duration for the period of time during which a server intends to respond to a multicast request. The Patience option in a CoAP request can be used as an upper bound for the Leisure.

How to use Leisure is defined in [RFC7252].

### 2.2.3. Observe usage

In a CoAP observe [I-D.ietf-core-observe] scenario, the Patience Option MAY be used in a notification to indicate the maximum time an observer should wait before starting any observation relationship recovery.

The Max-Age Option indicates the maximum time a response (notification) may be cached before it MUST be considered stale. The Max-Age Option of a notification is usually set to a value that estimates when the server will send the next notification. However, in the case the value has not changed, the server can decide not to send a new notification, possibly confusing the observer. It is quite difficult for an observer to discriminate the situation that it has not received a new notification because the value has not changed from situations where the server has lost its state, or for some reason has given up on notification delivery.

The Patience Option in a notification is used to indicate the maximum time a server will try to reach the client before giving up. This is to save the client some effort in re-establishing observation relationships each time max-age is reached. This option is also useful to give a server the time to send out the notifications, in case there are many CoAP clients observing simultaneously a resource, while avoiding network congestion issues.

The server adds the Patience option to any notification related to an observation relationship from which it wants delay an observation refresh request made by the observer. The server sets the option to the maximum time that it is prepared to spend to reach the observer before giving up.

The observer interprets this option as the minimum time between the expiration of a notification (as indicated by its Max-Age Option value) and the moment it MAY start an observation relationship recovery action with the server.

If the observer does not receive a response within the indicated time interval, the observer SHOULD attempt to re-establish the observation

relationship with the server if it is still interested in observing the particular resource.

### 2.3. Detection of IP unicast or multicast CoAP request

A single Patience Option, used to indicate potentially either client patience (in the IP unicast case) or server patience (in the IP multicast case), requires that a CoAP server is able to distinguish between IP unicast and multicast requests. If there exist commonly used IP stacks that do not offer such functionality [to be checked], requiring servers to be able to make the unicast/multicast distinction seems unwise and limits the applicability of the Patience Option.

Approaches for a CoAP server to detect unicast versus multicast requests may include:

- 1) CoAP server application opens a specific socket and sets IP multicast reception using the POSIX `setsockopt` function [to be verified if IP unicast traffic also is received in this case, or not].
- 2) CoAP server checks the IP destination address of incoming packets. If this has the `FF00::/8` IPv6 prefix, then it's treated as multicast otherwise unicast [to be verified if IP stack APIs allow to get IP destination].
- 3) Receiving CoAP multicast requests always occurs on a different port than the standard CoAP port. For example, similar to `coaps://` that uses a different port than `coap://`, a scheme `coapm://` on a different port may be defined for multicast requests.

## 3. Example

### 3.1. Unicast Usage Example

This section gives a short example with a message flow that illustrates the use of the Patience option in a GET request.

This example (Figure 1) shows that the requester wants to get a response within 3200 milliseconds, when  $T=25$ ,  $TX=1$ .

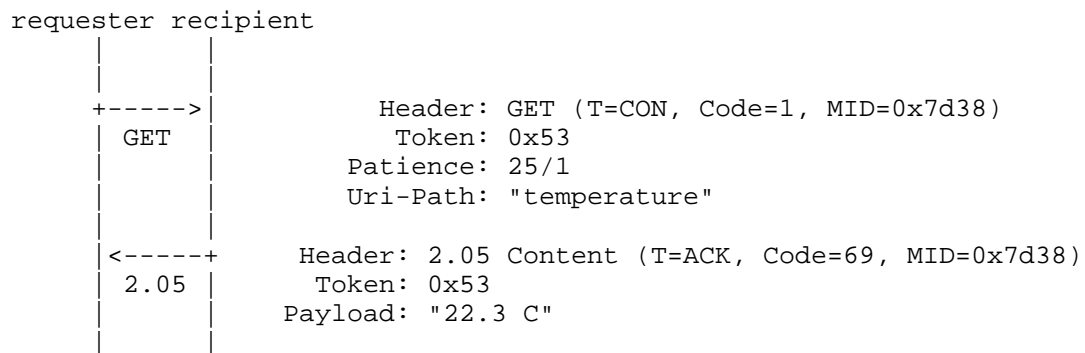


Figure 1: Patience Option in a unicast request

### 3.2. Observe Usage Example

This section gives a short example with a message flow that illustrates the use of the Patience option in an Observe notification.

This example (Figure 2) shows that the server wants the observer to wait 819 seconds ( $T=25$ ,  $TX=3$ ) before starting any observation relationship recovery, even though the Max-Age of the temperature value notification is only 120 seconds.

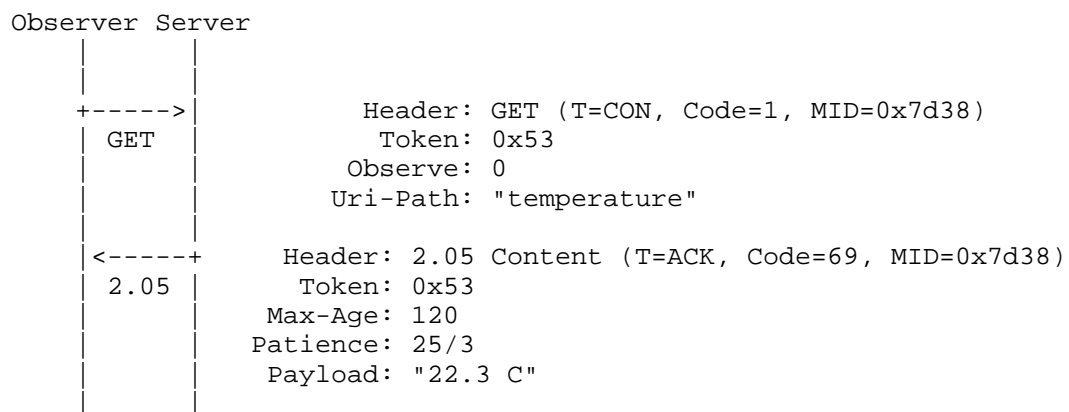


Figure 2: Patience Option in an observe notification

#### 4. Security Considerations

This presents no security considerations beyond those in section 10 of the base CoAP specification [RFC7252].

#### 5. IANA Considerations

The IANA is requested to add the following "CoAP Option Numbers" entry as per Section 12.2 of [RFC7252].

No.	C	U	N	R	Name	Format	Length	Default
28			x		Patience	(ref to this document)	1 B	(none)

#### 6. Acknowledgements

The authors of this draft would like to thank the participants of the email discussion on this issue. Thanks to Carsten Bormann, Peter Bigot, Barry Leiba, Linyi Tian, Gengyu Wei for the reviews and discussions.

#### 7. References

##### 7.1. Normative References

- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-13 (work in progress), April 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

##### 7.2. Informative References

- [I-D.bormann-coap-misc]  
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-26 (work in progress), December 2013.

Authors' Addresses

Kepeng Li  
Huawei Technologies  
Huawei Base, Bantian, Longgang District  
Shenzhen, Guangdong 518129  
P. R. China

Email: [likepeng@huawei.com](mailto:likepeng@huawei.com)

Bert Greevenbosch  
Huawei Technologies  
Huawei Base, Bantian, Longgang District  
Shenzhen, Guangdong 518129  
P. R. China

Email: [bert.greevenbosch@huawei.com](mailto:bert.greevenbosch@huawei.com)

Esko Dijk  
Philips Research  
High Tech Campus 34  
Eindhoven  
The Netherlands

Email: [esko.dijk@philips.com](mailto:esko.dijk@philips.com)

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

Email: [salvatore.loreto@ericsson.com](mailto:salvatore.loreto@ericsson.com)

CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: October 12, 2014

T. Savolainen  
Nokia  
K. Hartke  
Universitaet Bremen TZI  
B. Silverajan  
Tampere University of Technology  
April 10, 2014

CoAP over WebSockets  
draft-savolainen-core-coap-websockets-02

Abstract

This document specifies how to retrieve and update CoAP resources using CoAP requests and responses over the WebSocket Protocol.

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 October 12, 2014.

Copyright Notice

Copyright (c) 2014 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. Overview . . . . .	4
1.2. Terminology . . . . .	6
2. CoAP over WebSockets . . . . .	6
2.1. Opening Handshake . . . . .	6
2.2. Message Format . . . . .	7
2.3. Message Transmission . . . . .	8
2.4. Connection Health . . . . .	8
2.5. Closing the Connection . . . . .	8
3. CoAP over WebSockets URIs . . . . .	9
4. Security Considerations . . . . .	9
5. IANA Considerations . . . . .	10
5.1. URI Scheme Registrations . . . . .	10
5.2. WebSocket Subprotocol Registration . . . . .	12
5.3. Well-Known URI Suffix Registration . . . . .	12
6. Acknowledgements . . . . .	12
7. References . . . . .	12
7.1. Normative References . . . . .	12
7.2. Informative References . . . . .	13
Appendix A. Examples . . . . .	14
Authors' Addresses . . . . .	17



## 1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a web protocol designed for communications between resource constrained nodes. By default, CoAP operates on top of UDP or DTLS, but there is interest in using CoAP also over other types of transports, such as SMS [I-D.becker-core-coap-sms-gprs].

An interesting transport for CoAP could be the WebSocket Protocol [RFC6455]. The WebSocket protocol provides two-way communication between a client and a server after upgrading an HTTP [RFC2616] connection, and may be available in an environment that does not allow transportation of CoAP over UDP. This environment can be, for example, a corporate network with Internet access only via an HTTP proxy, or a CoAP application running in a web browser without access to connectivity means other than HTTP and WebSockets.

This document specifies how to access resources using CoAP requests and responses over the WebSocket Protocol. This allows connectivity-limited applications to obtain end-to-end CoAP connectivity either by communicating CoAP directly with a CoAP server that is accessible over a WebSocket Connection, or via an intermediary that proxies CoAP requests and responses between different transports, such as between WebSockets and UDP.

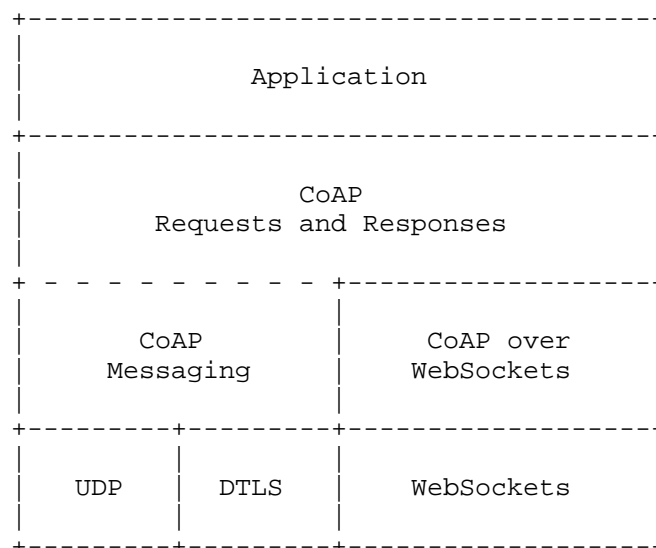


Figure 1: Abstract layering of CoAP extended by WebSockets

### 1.1. Overview

CoAP over WebSockets can be used in a number of configurations. The most basic configuration is a CoAP client seeking to retrieve or update a CoAP resource located at a CoAP server that exposes a WebSocket endpoint (Figure 2). The CoAP client takes the role of the WebSocket client, establishes a WebSocket Connection and sends a CoAP request, to which the CoAP server returns a CoAP response. The WebSocket Connection can be used for any number of requests.

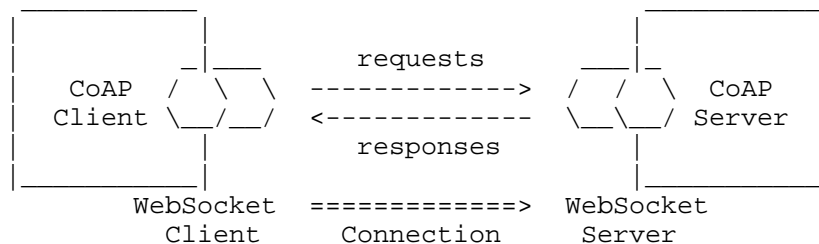


Figure 2: CoAP client (WebSocket client) accesses CoAP server (WebSocket server)

The challenge in this configuration is to identify resource in the namespace of the CoAP server: When the WebSocket Protocol is used by a dedicated client directly (i.e., not from a web page through a web browser), the client can connect to any WebSocket endpoint. This means it is necessary that the client is able to determine both the WebSocket endpoint (identified by a "ws" or "wss" URI) and the path and query of the CoAP resource within that endpoint from the same URI. When the WebSocket Protocol is used from a web page, the choices are more limited [RFC6454], but the challenge persists.

Section 3 proposes a new "coap+ws" URI scheme that identifies both a WebSocket endpoint and a resource within that endpoint as follows:

```

coap+ws://example.org/sensors/temperature?u=Cel
      \_____/ \_____/ \_____/
      \         \         \
ws://example.org/.well-known/coap  Uri-Path: "sensors"
                                   Uri-Path: "temperature"
                                   Uri-Query: "u=Cel"
  
```

Figure 3: The "coap+ws" URI Scheme

Another possible configuration is to set up a CoAP forward proxy at the WebSocket endpoint. Depending on what transports are available to the proxy, it could forward the request to a CoAP server with a CoAP UDP endpoint (Figure 4), an SMS endpoint (a.k.a. mobile phone), or even another WebSocket endpoint. The client specifies the resource to be updated or retrieved in the Proxy-URI Option.

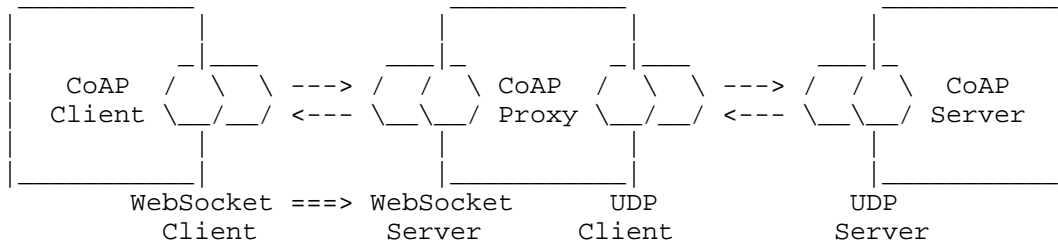


Figure 4: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

In a completely different way, another possible configuration is a CoAP server running inside a web browser (Figure 5). The web browser initially connects to a WebSocket endpoint and is then reachable through the WebSocket server. When no connection exists, the CoAP server is not reachable; it therefore can be considered a Sleepy Endpoint [I-D.dijk-core-sleepy-reqs].

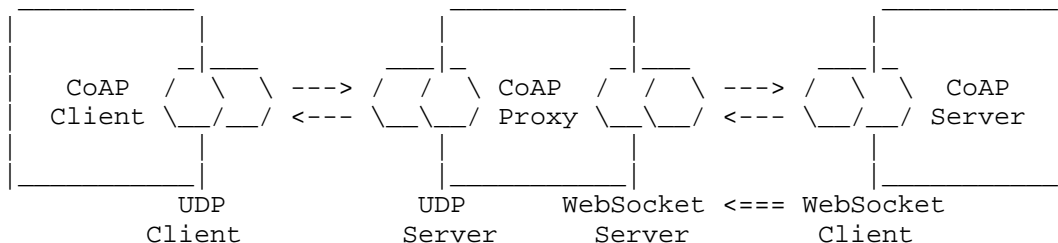


Figure 5: CoAP Client (UDP client) accesses sleepy CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

The challenge, again, is to identify the resource. Since the CoAP server is running inside the web browser, this requires not only to identify the WebSocket client and the path and query, but also the intermediary, which is the only path to reach the server. The

problem can be avoided if the intermediary is turned into a Reverse Proxy or a Mirror Server [I-D.vial-core-mirror-server].

Further configurations are possible, including those where a WebSocket Connection is established through an HTTP proxy.

## 1.2. Terminology

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455] and [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 [RFC2119].

## 2. CoAP over WebSockets

CoAP over WebSockets is intentionally very similar to CoAP as defined over UDP. Therefore, instead of presenting CoAP over WebSockets as a new protocol, this document specifies it as a series of deltas from [I-D.ietf-core-coap].

### 2.1. Opening Handshake

Before CoAP requests and responses can be exchanged, a WebSocket Connection needs to be established as defined in Section 4 of [RFC6455]. The WebSocket client MUST include the subprotocol name "coap.v1" in the list of protocols, which indicates support for the protocol defined in this document. Figure 6 shows an example.

```
GET /.well-known/coap HTTP/1.1
Host: example.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: coap.v1
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: coap.v1
```

Figure 6: Example of an Opening Handshake

## 2.2. Message Format

Once a WebSocket Connection has been established, CoAP requests and responses can be exchanged as WebSocket messages. Since CoAP uses a binary message format, the messages are transmitted in binary data frames as specified in Sections 5 and 6 of [RFC6455].

The message format is very similar to the format specified for CoAP over UDP [I-D.ietf-core-coap]. The differences are as follows:

- o Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. This means the "T" and "Message ID" fields in the CoAP message header can be elided.
- o Furthermore, since the CoAP version is already negotiated during the opening handshake, the "Ver" field can be elided as well.

The resulting message format is shown in Figure 7. The four most-significant bits of the first byte are reserved (R). The remaining fields and structure are the same as defined in [I-D.ietf-core-coap].

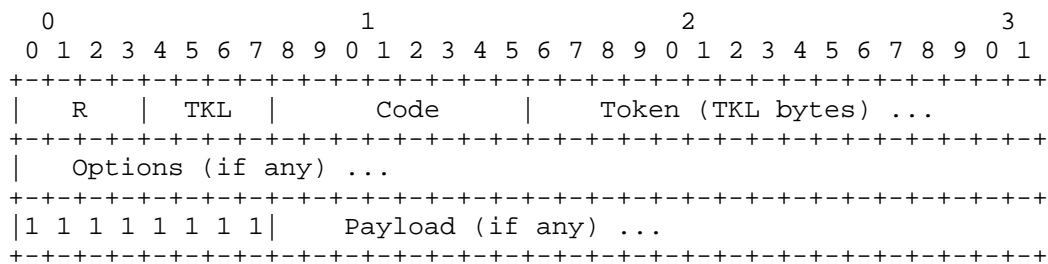


Figure 7: CoAP Message Format over WebSockets

Requests and response messages can be fragmented as specified in Section 5.4 of [RFC6455], though typically they are sent unfragmented as they tend to be small and fully buffered before transmission. The WebSocket protocol does not provide means for multiplexing; if it is not desirable for a large message to monopolize the connection, a multiplexing extension such as [I-D.ietf-hybi-websocket-multiplexing] can be used. Alternatively, requests and responses can be transferred in a blockwise fashion, as defined in [I-D.ietf-core-block].

Messages **MUST NOT** be Empty (Code 0.00), i.e., messages always carry either a request or a response.

### 2.3. Message Transmission

CoAP requests and responses are exchanged asynchronously over the WebSocket Connection, i.e., a CoAP client can send multiple requests without waiting for a response, and the CoAP server can return responses in any order. Responses **MUST** be returned over the same connection as the originating request. Concurrent requests are differentiated by the Token, which is locally scoped to the connection.

The connection is bi-directional, so requests can be sent both by the entity that established the connection and the remote host.

Retransmission and deduplication of messages is provided by the WebSocket Protocol. CoAP over WebSockets therefore does not make a distinction between Confirmable or Non-Confirmable messages, and does not provide Acknowledgement or Reset messages.

Since the WebSocket Protocol provides ordered delivery of messages, the mechanism for reordering detection when observing resources [I-D.ietf-core-observe] is not needed. The value of the Observe Option in notifications therefore **MAY** be empty on transmission and **MUST** be ignored on reception.

### 2.4. Connection Health

When a client does not receive any response for some time after sending a CoAP request (or, similarly, when a client observes a resource and it does not receive any notification), the connection between the WebSocket client and the WebSocket server may be lost or temporarily disrupted without the client being aware of it.

To check the health of the WebSocket Connection (and thereby of all active requests, if any), the client can send a Ping frame or an unsolicited Pong frame, as specified in Section 5.5 of [RFC6455].

### 2.5. Closing the Connection

The WebSocket Connection is closed as specified in Section 7 of [RFC6455].

All requests (if any) for which the CoAP client has not received a response yet, are cancelled when the connection is closed. If the client observes one or more resource over the WebSocket Connection, then the CoAP server (or intermediary in the role of the CoAP server) **MUST** remove all entries associated with the client from the lists of observers when the connection is closed.

### 3. CoAP over WebSockets URIs

For the first configuration discussed in Section 1.1, this document defines two new URI schemes that can be used for identifying CoAP resources and providing a means of locating these resources: "coap+ws" and "coap+wss".

Similar to the "coap" and "coaps" schemes, the "coap+ws" and "coap+wss" schemes organize resources hierarchically under a CoAP origin server. The key difference is that the server is potentially reachable on a WebSocket endpoint instead of a UDP endpoint.

The WebSocket endpoint is identified by an "ws" or "wss" URI that is composed of the authority part of the "coap+ws" or "coap+wss" URI, respectively, and the well-known path `/.well-known/coap` [RFC5785]. The path and query parts of a "coap+ws" or "coap+wss" URI identify a resource within the specified endpoint which can be operated on by the methods defined by the CoAP protocol.

The syntax of the "coap+ws" and "coap+wss" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty" and "query" are the same as in [RFC3986].

```
coap-ws-URI =  
    "coap+ws:" "//" host [ ":" port ] path-abempty [ "?" query ]  
  
coap-wss-URI =  
    "coap+wss:" "//" host [ ":" port ] path-abempty [ "?" query ]
```

The port component is OPTIONAL; the default for "coap+ws" is port 80, while the default for "coap+wss" is port 443.

Fragment identifiers are not part of the request URI and thus MUST NOT be transmitted in a WebSocket handshake or in a CoAP request.

### 4. Security Considerations

CoAP over WebSockets and CoAP over TLS-secured WebSockets do not introduce additional security issues beyond CoAP and DTLS-secured CoAP respectively [I-D.ietf-core-coap].

The security considerations of [RFC6455] apply.

## 5. IANA Considerations

### 5.1. URI Scheme Registrations

#### 5.1.1. "coap+ws"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+ws".

URI scheme name.  
coap+ws

Status.  
Permanent.

URI scheme syntax.  
Defined in Section 3.

URI scheme semantics.  
The "coap+ws" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol.

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 the WebSocket protocol.

Interoperability considerations.  
None.

Security considerations.  
See Section 4.

Contact.  
IETF Chair <chair@ietf.org>

Author/Change controller.  
IESG <iesg@ietf.org>

References.  
This document.



## 5.1.2. "coap+wss"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+wss".

URI scheme name.  
coap+wss

Status.  
Permanent.

URI scheme syntax.  
Defined in Section 3.

URI scheme semantics.  
The "coap+wss" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol secured with Transport Layer Security (TLS).

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 the WebSocket protocol secured with TLS.

Interoperability considerations.  
None.

Security considerations.  
See Section 4.

Contact.  
IETF Chair <chair@ietf.org>

Author/Change controller.  
IESG <iesg@ietf.org>

References.  
This document.

## 5.2. WebSocket Subprotocol Registration

This document requests the registration of the subprotocol name "coap.v1" in the WebSocket Subprotocol Name Registry.

Subprotocol Identifier.  
coap.v1

Subprotocol Common Name.  
Constrained Application Protocol (CoAP).

Subprotocol Definition.  
This document.

## 5.3. Well-Known URI Suffix Registration

This document requests the registration of the Well-Known URI suffix "coap" in the Well-Known URI Registry.

URI suffix.  
coap

Change controller.  
IETF.

Specification document(s).  
This document.

Related information.  
None.

## 6. Acknowledgements

Thanks to Nadir Javed for helpful comments and discussions that have shaped the document.

## 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.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-13 (work in progress), April 2014.

- [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.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

## 7.2. Informative References

- [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.dijk-core-sleepy-reqs]  
Dijk, E., "Sleepy Devices using CoAP - Requirements",  
draft-dijk-core-sleepy-reqs-00 (work in progress),  
June 2013.
- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",  
draft-ietf-core-block-14 (work in progress), October 2013.
- [I-D.ietf-hybi-websocket-multiplexing]  
Tamplin, J. and T. Yoshino, "A Multiplexing Extension for  
WebSockets", draft-ietf-hybi-websocket-multiplexing-11  
(work in progress), July 2013.
- [I-D.vial-core-mirror-server]  
Vial, M., "CoRE Mirror Server",  
draft-vial-core-mirror-server-01 (work in progress),  
April 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.

- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

## Appendix A. Examples

This section gives examples for the first two configurations discussed in Section 1.1.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap+ws" URI might be as follows. Figure 8 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI `<coap+ws://example.org/sensors/temperature?u=Cel>`, for example, from a resource representation that it retrieved previously.
2. It establishes a WebSocket Connection to the endpoint URI composed of the authority "example.org" and the well-known path `"/.well-known/coap"`, `<ws://example.org/.well-known/coap>`.
3. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") options.
4. It waits for the server to return a response.
5. The CoAP client uses the connection for further requests, or the connection is closed.

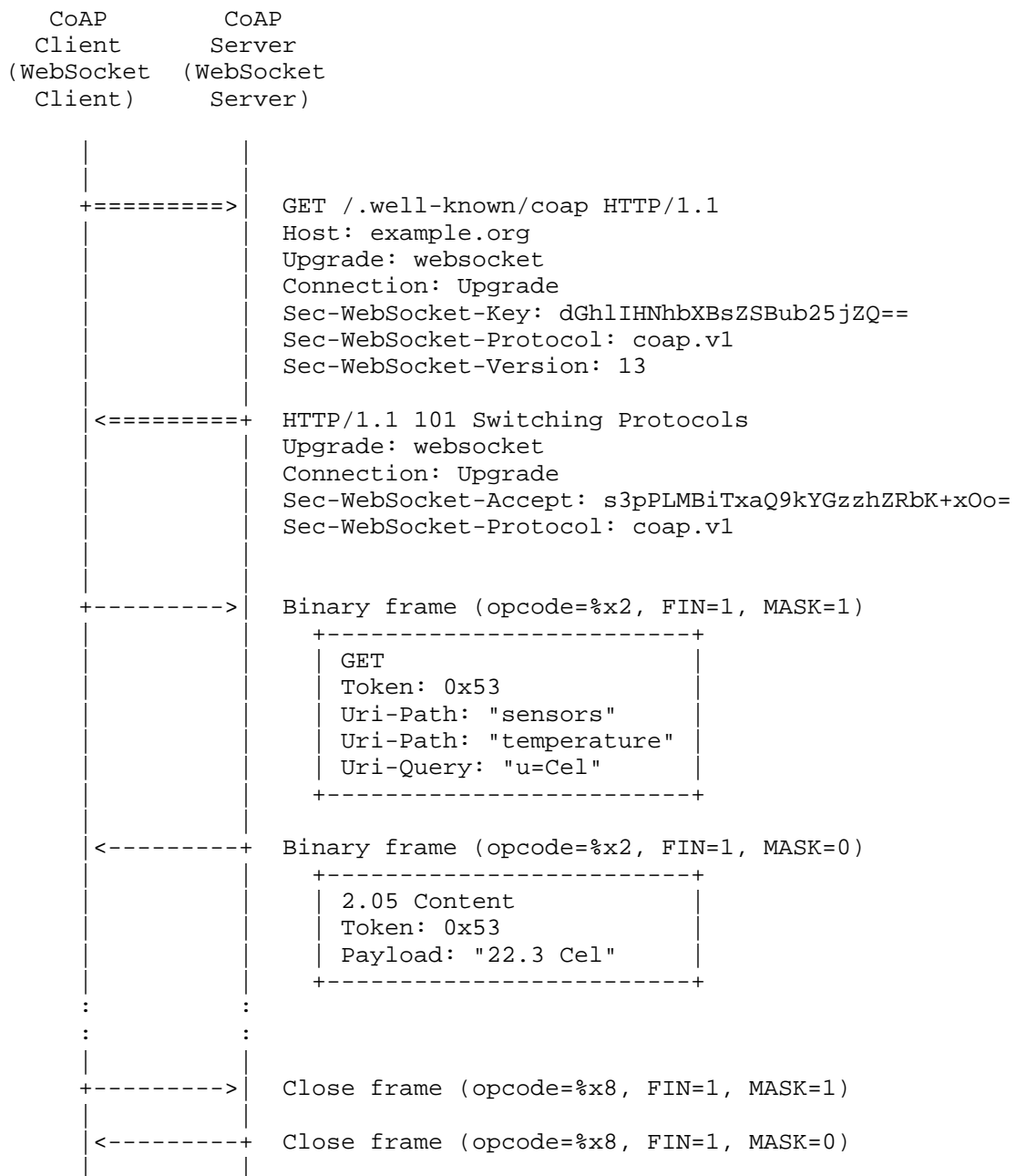


Figure 8: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 9 shows how a CoAP client uses a CoAP forward proxy with a WebSocket endpoint to retrieve the representation of the resource `<coap://[2001:DB8::1]/>`. The use of the forward proxy and the address of the WebSocket endpoint are determined by the client from local configuration rules. The request URI is specified in the Proxy-Uri Option. Since the request URI uses the "coap" URI scheme, the proxy fulfills the request by issuing a Confirmable GET request over UDP to the CoAP server and returning the response over the WebSocket connection to the client.

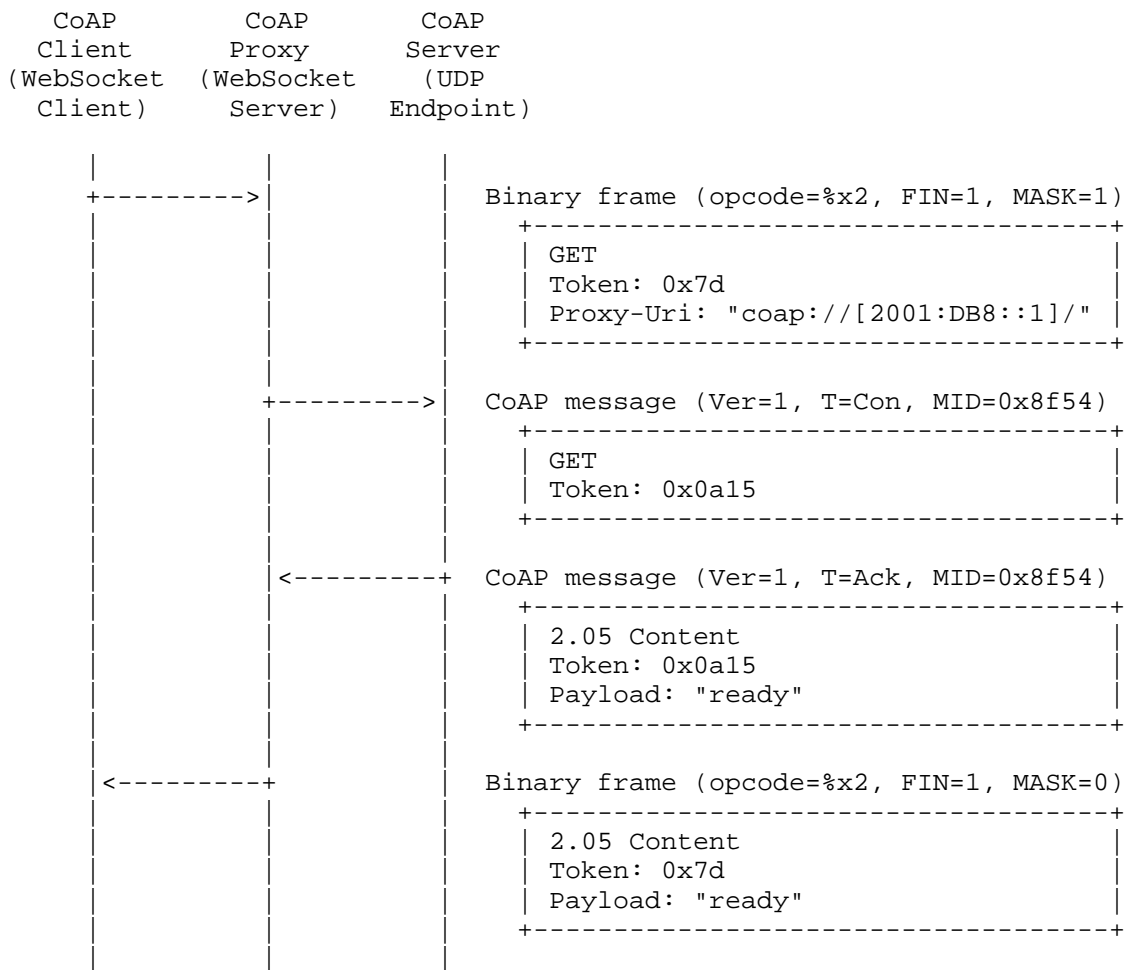


Figure 9: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSockets-enabled CoAP proxy

Authors' Addresses

Teemu Savolainen  
Nokia  
Hermiankatu 12 D  
Tampere FI-33720  
Finland

Email: teemu.savolainen@nokia.com

Klaus Hartke  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63905  
Email: hartke@tzi.org

Bilhanan Silverajan  
Tampere University of Technology  
Korkeakoulunkatu 10  
Tampere FI-33720  
Finland

Email: bilhanan.silverajan@tut.fi





CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 22, 2015

B. Silverajan  
Tampere University of Technology  
T. Savolainen  
Nokia  
July 21, 2014

CoAP Communication with Alternative Transports  
draft-silverajan-core-coap-alternative-transports-06

Abstract

CoAP has been standardised as an application level REST-based protocol. A single CoAP message is typically encapsulated and transmitted using UDP or DTLS as transports. These transports are optimal solutions for CoAP use in IP-based constrained environments and nodes. However compelling motivation exists for understanding how CoAP can operate with other transports, such as the need for M2M communication using non-IP networks, improved transport level end-to-end reliability and security, NAT and firewall traversal issues, and mechanisms possibly incurring a lower overhead to CoAP/HTTP translation gateways. This draft examines the requirements for conveying CoAP messages to end points over such alternative transports. It also provides a new URI format for representing CoAP resources over alternative transports.

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 22, 2015.

Copyright Notice

Copyright (c) 2014 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. Usage Cases	3
2.1. Use of SMS	4
2.2. Use of WebSockets	4
2.3. Use of P2P Overlays	4
2.4. Use of TCP	4
2.5. Others	5
3. Node Types based on Transport Availability	5
4. CoAP Alternative Transport URI	6
4.1. Design Considerations	7
4.2. URI format	8
5. Alternative Transport Analysis and Properties	9
6. IANA Considerations	11
7. Security Considerations	11
8. Acknowledgements	12
9. References	12
9.1. Normative References	12
9.2. Informative References	12
Appendix A. Expressing transport in the URI in other ways	14
A.1. Transport information as part of the URI authority	14
A.1.1. Usage of DNS records	15
A.2. Making CoAP Resources Available over Multiple Transports	15
A.3. Transport as part of a 'service:' URL scheme	17
Authors' Addresses	18

## 1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] has been standardised by the CoRE WG as a lightweight, HTTP-like protocol providing a request/response model that constrained nodes can use to communicate with other nodes, be those servers, proxies, gateways, less constrained nodes, or other constrained nodes.

As the Internet continues taking shape by integrating new kinds of networks, services and devices, the need for a consistent, lightweight method for resource representation, retrieval and

manipulation becomes evident. Owing to its simplicity and low overhead, CoAP is a highly suitable protocol for this purpose. However, the CoAP endpoint can reside in a non-IP network, be separated from its peer by NATs and firewalls or simply has no possibility to communicate over UDP. Consequently in addition to UDP, alternative transport channels for conveying CoAP messages could be considered.

Extending CoAP over alternative transports allows implementations to have a significantly larger relevance in constrained as well as non-constrained networked environments. It leads to better code optimisation in constrained nodes and broader implementation reuse across new transport channels. As opposed to implementing new resource retrieval mechanisms, an application in an end-node can continue relying on using CoAP's REST-based resource retrieval and manipulation for this purpose, while changes in end point identification and the transport protocol can be addressed by a transport-specific messaging sublayer. This simplifies development and memory requirements. Resource representations are also visible in an end-to-end manner for any CoAP client. The processing and computational overhead for conveying CoAP Requests and Responses from one underlying transport to another, would be less than that of an application-level gateway performing protocol translation of individual messages between CoAP and another resource retrieval protocol such as HTTP.

This document first provides scenarios where usage of CoAP over alternative transports is either currently underway, or may prove advantageous in the future. A simple transport type classification for CoAP-capable nodes is provided next. Then a new URI format is described through which a CoAP resource representation can be formulated that expresses transport identification in addition to endpoint information and resource paths. Following that, a discussion of the various transport properties which influence how CoAP Requests and Responses are mapped to transport level payloads, is presented.

This document however, does not touch on application QoS requirements, user policies or network adaptation, nor does it advocate replacing the current practice of UDP-based CoAP communication.

## 2. Usage Cases

Apart from UDP and DTLS, CoAP usage is being specified for the following environments as of this writing:

### 2.1. Use of SMS

CoAP Request and Response messages can be sent via SMS between CoAP end-points in a cellular network [I-D.becker-core-coap-sms-gprs]. A CoAP Request message can also be sent via SMS from a CoAP client to a sleeping CoAP Server as a wake-up mechanism and trigger communication via IP. The Open Mobile Alliance (OMA) specifies both UDP and SMS as transports for M2M communication in cellular networks. The OMA Lightweight M2M protocol being drafted uses CoAP, and as transports, specifies both UDP binding as well as Short Message Service (SMS) bindings [OMALWM2M] for the same reason.

### 2.2. Use of WebSockets

The WebSocket protocol is being proposed as a transport channel between WebSocket enabled CoAP end-points on the Internet [I-D.savolainen-core-coap-websockets]. This is particularly useful as a means for web browsers, especially in smart devices, to allow embedded client side scripts to create new WebSocket connections to various WebSocket-enabled servers, through which CoAP Request and Response messages can be exchanged. This also allows a browser containing an embedded CoAP server to behave as a WebSocket client by opening a connection to a WebSocket enabled CoAP Mirror Server [I-D.vial-core-mirror-server] to register and update its resources.

### 2.3. Use of P2P Overlays

[I-D.jimenez-p2psip-coap-reload] specifies how CoAP nodes can use a peer-to-peer overlay network called RELOAD, as a resource caching facility for storing wireless sensor data. When a CoAP node registers its resources with a RELOAD Proxy Node (PN), the node computes a hash value from the CoAP URI and stores it as a structure together with the PN's Node ID as well as the resources. Resource retrieval by CoAP nodes is accomplished by computing the hash key over the Request URI, opening a connection to the overlay and using its message routing system to contact the CoAP server via its PN.

### 2.4. Use of TCP

Using TCP to facilitate the traversal of CoAP Request and Response messages [I-D.bormann-core-coap-tcp], allows easier communication between CoAP clients and servers separated by firewalls and NATs. This also allows CoAP messages to be transported over push notification services from a notification server to a client app on a smartphone, that may previously have subscribed to receive change notifications of CoAP resource representations, possibly by using CoAP Observe-functionality [I-D.ietf-core-observe].

## 2.5. Others

CoAP could in addition be extended atop other transport channels, such as:

1. The transportation of CoAP messages in Delay-Tolerant Networks [RFC4838], using the Bundle Protocol [RFC5050] for reaching sensors in extremely challenging environments such as acoustic, underwater and deep space networks.
2. Any type of non-IP networks supporting constrained nodes and low-energy sensors, such as Bluetooth and Bluetooth Low Energy (either through L2CAP or with GATT) [BTCorev4.1], ZigBee, Z-Wave, 1-Wire, DASH7 and so on.
3. Instant Messaging and Social Networking channels, such as Jabber and Twitter.

## 3. Node Types based on Transport Availability

The term "alternative transport" in this document thus far has been used to refer to any non-UDP and non-DTLS transport that can convey CoAP messages in its payload. A node however, may in fact possess the capability to utilise CoAP over multiple transport channels at its disposal, simultaneously or otherwise, at any point in time to communicate with a CoAP end-point. Such communication can obviously take place over UDP and DTLS as well. Inevitably, if two CoAP endpoints reside in distinctly separate networks with orthogonal transports, a CoAP proxy node is needed between the two networks so that CoAP Requests and Responses can be exchanged properly.

In [RFC7228], Tables 1, 3 and 4 introduced classification schemes for devices, in terms of their resource constraints, energy limitations and communication power. For this document, in addition to these capabilities, it seems useful to additionally identify devices based on their transport capabilities.

Name	Transport Availability
T0	Single transport
T1	Multiple transports, with one or more active at any point in time
T2	Multiple active transports

Table 1: Classes of Available Transports

Nodes falling under Type T0 possess the capability of exactly 1 type of transport channel for CoAP, at all times. These include both active and sleepy nodes, which may choose to perform duty cycling for power saving.

Type T1 nodes possess multiple different transports, and can retrieve or expose CoAP resources over any or all of these transports. However, not all transports are constantly active and certain transport channels and interfaces could be kept in a mostly-off state for energy-efficiency, such as when using CoAP over SMS (refer to section 2.1)

Type T2 nodes possess more than 1 transport, and multiple transports are simultaneously active at all times. CoAP proxy nodes which allow CoAP endpoints from disparate transports to communicate with each other, are a good example of this.

#### 4. CoAP Alternative Transport URI

Based on the usage scenarios as well as the transport classes presented in the preceding sections, this section discusses the formulation of a new URI for representing CoAP resources over alternative transports.

CoAP is logically divided into 2 sublayers, whereby a request/response layer is responsible for the protocol functionality of exchanging request and response messages, while the messaging layer is bound to UDP. These 2 sublayers are tightly coupled, both being responsible for properly encoding the header and body of the CoAP message. The CoAP URI is used by both logical sublayers. For a URI that is expressed generically as

URI = scheme ":" "://" authority path-abempty ["?"query ]

a simple example CoAP URI, "coap://server.example.com/sensors/temperature" is interpreted as follows:

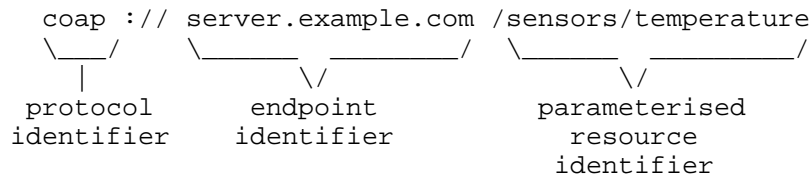


Figure 1: The CoAP URI format

The resource path is explicitly expressed, and the endpoint identifier, which contains the host address at the network-level is also directly bound to the scheme name containing the application-level protocol identifier. The choice of a specific transport for a scheme, however, cannot be embedded with a URI, but is defined by convention or standardisation of the protocol using the scheme. As examples, [RFC5092] defines the 'imap' scheme for the IMAP protocol over TCP, while [RFC2818] requires that the 'https' protocol identifier be used to differentiate using HTTP over TLS instead of TCP.

#### 4.1. Design Considerations

Several ways of formulating a URI which express an alternative transport binding to CoAP, can be envisioned. When such a URI is provided from an end-application to its CoAP implementation, the URI component containing transport-specific information can be checked to allow CoAP to use the appropriate transport for a target endpoint identifier.

The following design considerations influence the formulation of a new URI expressing CoAP resources over alternative transports:

1. A CoAP Transport 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. Also, the generic syntax for a URI is described in [RFC3986]. By ensuring conformance to RFC3986, the need for custom URI parsers as well as resolution algorithms can be obviated. In particular, a URI format needs to be described in

which each URI component clearly meets the syntax and percent-encoding rules described.

2. Request messages sent to a CoAP endpoint using a CoAP Transport URI may be responded to with a relative URI reference, for example, of the form "../..path/to/resource". In such cases, the requesting endpoint needs to resolve the relative reference against the original CoAP Transport URI to then obtain a new target URI to which a request can be sent to, to obtain a resource representation. [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
3. The host component of current CoAP URIs can either be an IPv4 address, an IPv6 address or a resolvable hostname. While the usage of DNS can sometimes be useful for distinguishing transport information (see section 4.3.1), accessing DNS over some alternative transport environments may be challenging. Therefore, a URI format needs to be described which is able to represent a resource without heavy reliance on a naming infrastructure, such as DNS.

#### 4.2. URI format

To meet the design considerations previously discussed, the transport information is expressed as part of the URI scheme component. This is performed by minting new schemes for alternative transports using the form "coap+<transport-name>", where the name of the transport is clearly and unambiguously described. Each scheme name formed in this manner is used to differentiate the use of CoAP over an alternative transport instead of the use of CoAP over UDP or DTLS. The endpoint identifier, path and query components together with each scheme name would be used to uniquely identify each resource.

Examples of such URIs are:

- o coap+tcp://[2001:db8::1]:5683/sensors/temperature for using CoAP over TCP
- o coap+sms://0015105550101/sensors/temperature for using CoAP over SMS or USSD with the endpoint identifier being a telephone subscriber number
- o coap+ws://www.example.com/sensors/temperature for using CoAP over WebSockets



A URI of this format to distinguish transport types is simple to understand and not dissimilar to the CoAP URI format. As the usage of each alternative transport results in an entirely new scheme, IANA intervention is required for the registration of each scheme name. The registration process follows the guidelines stipulated in [I-D.ietf-appsawg-uri-scheme-reg], particularly where permanent URI scheme registration is concerned.

It is also entirely possible for each new scheme to specify its own rules for how resource and transport endpoint information can be presented. However, the URIs and resource representations arising from their usage should meet the URI design considerations and guidelines mentioned in this document. In addition, each new transport being defined should take into consideration the various transport-level properties that can have an impact on how CoAP messages are conveyed as payload. This is elaborated on in the next section.

## 5. Alternative Transport Analysis and Properties

In this section the various characteristics of alternative transports for successfully supporting various kinds of functionality for CoAP are considered. CoAP factors lossiness, unreliability, small packet sizes and connection statelessness into its protocol logic. General transport differences and their impact on carrying CoAP messages here are discussed. Note that Properties 1, 2, and 3 are related.

Property 1: Uniqueness of an end-point identifier.

Transport protocols providing non-unique end-point IDs for nodes may only convey a subset of the CoAP functionality. Such nodes may only serve as CoAP servers that announce data at specific intervals to a pre-specified end point, or to a shared medium.

Property 2: Unidirectional or bidirectional CoAP communication support.

This refers to the ability of the CoAP end-point to use a single transport channel for both request and response messages. Depending on the scenario, having a unidirectional transport layer would mean the CoAP end-point might utilise it only for outgoing data or incoming data. Should both functionalities be needed, 2 unidirectional transport channels would be necessary.

Property 3: 1:N communication support.

This refers to the ability of the transport protocol to support broadcast and multicast communication. CoAP's request/response

behaviour depends on unicast messaging. Group communication in CoAP is bound to using multicasting. Therefore a protocol such as TCP would be ill-suited for group communications using multicast. Anycast support, where a message is sent to a well defined destination address to which several nodes belong, on the other hand, is supported by TCP.

Property 4: Transport-level reliability.

This refers to the ability of the transport protocol to provide a guarantee of reliability against packet loss, ensuring ordered packet delivery and having error control. When CoAP Request and Response messages are delivered over such transports, the CoAP implementations elide certain fields in the packet header. As an example, if the usage of a connection-oriented transport renders it unnecessary to specify the various CoAP message types, the Type field can be elided. For some connection-oriented transports, such as WebSockets, the version of CoAP being used can be negotiated during the opening transfer. Consequently, the Version field in CoAP packets can also be elided.

Property 5: Message encoding.

While parts of the CoAP payload are human readable or are transmitted in XML, JSON or SenML format, CoAP is essentially a low overhead binary protocol. Efficient transmission of such packets would therefore be met with a transport offering binary encoding support, although techniques exist in allowing binary payloads to be transferred over text-based transport protocols such as base-64 encoding. A fuller discussion about performing CoAP message encoding for SMS can be found in Appendix A.5 of [I-D.bormann-coap-misc]

Property 6: Network byte order.

CoAP, as well as transports based on the IP stack use a Big Endian byte order for transmitting packets over the air or wire, while transports based on Bluetooth and Zigbee prefer Little Endian byte ordering for packet fields and transmission. Any CoAP implementation that potentially uses multiple transports has to ensure correct byte ordering for the transport used.

Property 7: MTU correlation with CoAP PDU size.

Section 4.6 of [RFC7252] discusses the avoidance of IP fragmentation by ensuring CoAP message fit into a single UDP datagram. End-points on constrained networks using 6LoWPAN may use blockwise transfers to accommodate even smaller packet sizes to avoid fragmentation. The MTU sizes for Bluetooth Low Energy as well as Classic Bluetooth are

provided in Section 2.4 of [I-D.ietf-6lo-btle]. Transport MTU correlation with CoAP messages helps ensure minimal to no fragmentation at the transport layer. On the other hand, allowing a CoAP message to be delivered using a delay-tolerant transport service such as the Bundle Protocol [RFC5050] would imply that the CoAP message may be fragmented (or reconstituted) along various nodes in the DTN as various sized bundles and bundle fragments.

#### Property 8: Framing

When using CoAP over a streaming transport protocol such as TCP, as opposed to datagram based protocols, care must be observed in preserving message boundaries. Commonly applied techniques at the transport level include the use of delimiting characters for this purpose as well as message framing and length prefixing.

#### Property 9: Transport latency.

A confirmable CoAP request would be retransmitted by a CoAP end-point if a response is not obtained within a certain time. A CoAP end-point registering to a Resource Directory uses a POST message that could include a lifetime value. A sleepy end-point similarly uses a lifetime value to indicate the freshness of the data to a CoAP Mirror Server. Care needs to be exercised to ensure the latency of the transport being used to carry CoAP messages is small enough not to interfere with these values for the proper operation of these functionalities.

#### Property 10: Connection Management.

A CoAP endpoint using a connection-oriented transport should be responsible for proper connection establishment prior to sending a CoAP Request message. Both communicating endpoints may monitor the connection health during the Data Transfer phase. Finally, once data transfer is complete, at least one end point should perform connection teardown gracefully.

### 6. IANA Considerations

This memo includes no request to IANA.

### 7. Security Considerations

While no new security risks are envisaged simply from the introduction of support for alternative transports, end-applications and CoAP implementations should take note if certain transports require privacy trade-offs that may arise if identifiers such as MAC addresses or phone numbers are made public in addition to FQDNs.

## 8. Acknowledgements

Feedback, ideas and ongoing discussions with Klaus Hartke, Martin Thomson, Mark Nottingham, Dave Thaler, Graham Klyne, Carsten Bormann, Markus Becker and Golnaz Karbaschi provided useful insights and ideas for this work.

## 9. References

### 9.1. Normative References

- [I-D.ietf-appsawg-uri-scheme-reg]  
Thaler, D., Hansen, T., Hardie, T., and L. Masinter,  
"Guidelines and Registration Procedures for New URI  
Schemes", draft-ietf-appsawg-uri-scheme-reg-01 (work in  
progress), July 2014.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform  
Resource Identifier (URI): Generic Syntax", STD 66, RFC  
3986, January 2005.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for  
Constrained-Node Networks", RFC 7228, May 2014.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained  
Application Protocol (CoAP)", RFC 7252, June 2014.

### 9.2. Informative References

- [BTCorev4.1]  
BLUETOOTH Special Interest Group, "BLUETOOTH Specification  
Version 4.1", December 2013.
- [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-26 (work in progress),  
December 2013.
- [I-D.bormann-core-coap-tcp]  
Bormann, C., "A TCP transport for CoAP", draft-bormann-  
core-coap-tcp-01 (work in progress), July 2014.

- [I-D.ietf-6lo-btle]  
Nieminen, J., Savolainen, T., Isomaki, M., Patil, B.,  
Shelby, Z., and C. Gomez, "Transmission of IPv6 Packets  
over BLUETOOTH(R) Low Energy", draft-ietf-6lo-btle-02  
(work in progress), June 2014.
- [I-D.ietf-core-groupcomm]  
Rahman, A. and E. Dijk, "Group Communication for CoAP",  
draft-ietf-core-groupcomm-20 (work in progress), July  
2014.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-  
core-observe-14 (work in progress), June 2014.
- [I-D.jimenez-p2psip-coap-reload]  
Jimenez, J., Lopez-Vega, J., Maenpaa, J., and G.  
Camarillo, "A Constrained Application Protocol (CoAP)  
Usage for REsource LOcation And Discovery (RELOAD)",  
draft-jimenez-p2psip-coap-reload-04 (work in progress),  
July 2014.
- [I-D.savolainen-core-coap-websockets]  
Savolainen, T., Hartke, K., and B. Silverajan, "CoAP over  
WebSockets", draft-savolainen-core-coap-websockets-02  
(work in progress), April 2014.
- [I-D.vial-core-mirror-server]  
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-  
server-01 (work in progress), April 2013.
- [OMALWM2M]  
Open Mobile Alliance (OMA), "Lightweight Machine to  
Machine Technical Specification", 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2609] Guttman, E., Perkins, C., and J. Kempf, "Service Templates  
and Service: Schemes", RFC 2609, June 1999.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst,  
R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant  
Networking Architecture", RFC 4838, April 2007.

- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.
- [RFC5092] Melnikov, A. and C. Newman, "IMAP URL Scheme", RFC 5092, November 2007.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.
- [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.
- [RFC6733] Fajardo, V., Arkko, J., Loughney, J., and G. Zorn, "Diameter Base Protocol", RFC 6733, October 2012.
- [WWWArchv1]  
http://www.w3.org/TR/webarch/#uri-aliases, "Architecture of the World Wide Web, Volume One", December 2004.

## 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  
.1.0.0.2.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 2: 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      CoAP Resource
        Prefix

```

Figure 3: 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 [BTCorev4.1].

### 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  
Hermiankatu 12 D  
FI-33720 Tampere  
Finland

Email: teemu.savolainen@nokia.com

CoRE  
Internet Draft  
Intended status: Standards track  
Expires: December 2014

A. Bhattacharyya  
S. Bandyopadhyay  
A. Pal  
Tata Consultancy Services Ltd.  
June 19, 2014

CoAP option for no server-response  
draft-tcs-coap-no-response-option-06

Abstract

There can be typical M2M scenarios where responses from the data sink to the data source against request 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 December 19, 2014.

#### Copyright Notice

Copyright (c) 2014 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 .....	4
3.2. Multicasting actuation command from a handheld device to a group of appliances .....	5
3.2.1. Using granular response suppression .....	6
4. Option Definition .....	6
4.1. Achieving granular suppression .....	7

5. Miscellaneous aspects .....	9
5.1. Re-use interval for message IDs .....	9
5.2. Re-using Tokens.....	9
5.3. Taking care of congestion .....	10
5.4. Duality with the 'Observe' option .....	10
6. Example .....	11
6.1. Request/response Scenario .....	11
6.1.1. Using No-Response with PUT .....	11
6.1.2. Using No-Response with POST .....	12
6.1.2.1. POST updating a target resource .....	12
6.1.2.2. POST performing updates through resource creation .....	13
6.2. An end-to-end system combining No-Response and Observe .....	14
7. IANA Considerations .....	16
8. Security Considerations .....	16
9. Acknowledgments .....	16
10. References .....	16
10.1. Normative References .....	16
10.2. Informative References .....	17

## 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 by effectively reducing the overall traffic
- \* Reduction in server-side loading by relieving the server from responding to each request when not necessary
- \* Reduction in battery consumption at the constrained end-point
- \* Reduction in communication cost
- \* Help satisfy hard real-time requirements since waiting due to closed loop latency is completely avoided

## 3. Exemplary application scenarios

Next sub-section describes some exemplary user stories which may potentially benefit through the use of No-Response option.

### 3.1. Frequent update of geo-location from vehicles to backend

Let us consider an intelligent traffic system (ITS) consisting of vehicles each of which 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.

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 actuation command from a handheld device to a group of appliances

A handheld device (e.g. a smart phone) may be programmed to act as an IP enabled switch to remotely operate on a single or group of IP enabled appliances. For example the smart phone can be programmed to send a multicast request to switch on/ off all the lights of a building. In this case the IP switch application can use No-Response option along with NON to reduce the traffic generated due to simultaneous status responses from hundreds of lights.

Thus No-Response helps in reducing overall communication cost and the probability of network clogging in this case.

### 3.2.1. Using granular response suppression

The IP switch application may optionally use granular response suppression such that the error responses are not suppressed. In that case the lights which could not execute the request would respond back and be readily identified.

## 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. It is unsafe-to-forward since the forward proxy should be aware of the special unidirectional nature of the requests containing No-Response option. If a proxy happens to encounter this option it should not forward.

This option is for requests and presently intended for updates (e.g., PUT) in NON mode and should have no effect if used with a CON request. This option is not applicable and should have no effect for usual GET requests asking for resource representation. However, this option may be used with specialized GET requests for 'cancellation' of an observe session (Section 3.6 of [I-D.ietf-core-observe]). 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	This option does not apply under usual circumstances when the client requests the contents of a resource. However, may be useful for special GET requests. For example, the 'cancellation' procedure for CoAP observe requires a client to issue a GET request which has the Token field set to the token of the observation to be cancelled and includes an Observe Option with the value set to 'deregister' (1). In this case the server response does not contain any payload. Under such situation the client MAY express its disinterest in the response from the server.
PUT	Mostly suitable 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	Deletion is usually a permanent action and the client SHOULD make sure that the deletion actually happened. SHOULD NOT be applicable.

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 'weaves' CON notifications within series of NON notifications to check if the observer is alive).

## 5. Miscellaneous aspects

This section further describes few important implementation aspects worth considering while using No-Response. As mentioned in the previous section, judicious use of this option enables the application developer to enhance the overall system throughput. To keep the flexibility on the application developer part the following discussion does not mandate anything, rather provides suggestive guidelines.

### 5.1. Re-use interval for message IDs

Since No-Response is primarily based on CoAP-NON, 'NON-LIFETIME' (as defined in Section 4.8.2 of [I-D.ietf-core-coap]) is suggested as the time interval over which a message ID can be safely re-used.

### 5.2. Re-using Tokens

Tokens provide a matching criteria between request and response. The life of a token starts when it is assigned to a request and ends when the final matching response is received. Then the token can again be re-used. However, a NON request with NO-response does not have any response path. So, the client has to decide on its own about when it can retire a token which has been used in an earlier request so that the token can be reused in a future request. Since the No-Response option is 'elective' a server which has not implemented this option MAY emanate a response. This leads to the following two scenarios:

The first scenario is when the client is never going to care about any response coming back or about relating the response to the original request. In that case it MAY reuse the token value at liberty.

However, as a second scenario, let us consider that the client sends two requests where the first request is with No-Response and the second request, with same token, is without No-Response. In this case a delayed response to the first one can be interpreted as a

response to the second request (client needs a response in the second case) if the gap between using the same tokens is not enough to allow the response to the first request from the server to reach the client.

So the client implementation should implement an application specific 'patience' time till which it can re-use the token. Appendix-B.4.1 of [I-D.draft-bormann-coap-misc] defines 'patience' option which in effect puts a deadline to the server to respond back. However, 'patience' is not exposed to the protocol level at present. Hence, a reuse time for tokens is suggested with similar expression as in Section 2.5 of [I-D.ietf-core-groupcomm]:

$$\text{TOKEN\_REUSE\_TIME} = \text{NON\_LIFETIME} + \text{MAX\_SERVER\_RESPONSE\_DELAY} + \text{MAX\_LATENCY}.$$

NON\_LIFETIME and MAX\_LATENCY are defined in 4.8.2 of [I-D.ietf-core-coap]. MAX\_SERVER\_RESPONSE\_DELAY has same interpretation as in Section 2.5 of [I-D.ietf-core-groupcomm] for multicast request. But for unicast request MAX\_SERVER\_RESPONSE\_DELAY is simply the expected maximum response delay from the server to which client sent the request. This delay includes the maximum Leisure time period as defined in Section 8.2 of [I-D.ietf-core-coap] and Appendix-B.4.2 of [I-D.draft-bormann-coap-misc] where group size (G) = 1 for unicast request.

### 5.3. Taking care of congestion

The possible communication scenarios taking advantage of 'No-Response' should primarily fall into the class of low-data volume applications as described in Section 3.1.2 of [RFC 5405]. Precisely, this should map to the scenario where the application cannot maintain an RTT estimate. Hence, following [RFC 5405], a 3s interval is suggested as the minimum interval between successive updates. However, an application developer MAY interweave occasional closed-loop exchanges (e.g. CoAP-NON without No-Response or CoAP-CON) to get an RTT estimate between the end-points and adjust time-to-time the interval between updates.

### 5.4. Duality with the 'Observe' option

Scenarios like frequent update of a given resource at server by a client using No-Response leads to an interesting observation. The 'No-Response' option actually complements the 'Observe' option with NON-notifications ([I-D.ietf-core-observe]). In case of the later the update notifications from the server reach the observer client without triggering any response from the observer. However, there is

a difference in the point of interest. In the 'Observe' scenario the interest is expressed by the 'consumer' to get the data. On the contrary, the updates using 'No-Response' applies to the scenario when it is the interest of the 'producer' to update the data. It is up to the application designer to choose between No-Response and NON-observe. For example, the scenario of location update described in Section 3.1 above might also be deployed using NON-observe. In that case the backend server would have to subscribe to each individual sensor gateway at the vehicles. But, the 'book-keeping' exercise required at the server for such an implementation may not be very trivial and deployment with No-Response may be far more straight-forward. However, 'No-Response' and 'Observe' using NON-notification may be combined together, under permitting condition, to achieve high performance gain in an end-to-end producer-consumer application. A typical example is illustrated in Section 6.2.

## 6. 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.

### 6.1. Request/response Scenario

#### 6.1.1. 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.

#### 6.1.2. 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.

##### 6.1.2.1. 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
<pre> +-----&gt;   POST           </pre>	<pre> Header: POST (T=NON, Code=0.02, 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;   POST           </pre>	<pre> Header: PUT (T=NON, Code=0.02, 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 2: Exemplary unreliable update with No-Response option using POST as the update-method.

#### 6.1.2.2. 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
<pre> +-----&gt;   POST           </pre>	<pre> Header: POST (T=NON, Code=0.02, MID=0x7d38) Token: 0x53 Uri-Path: "insertInfo" Uri-Query: "VehID=00" Uri-Query: "RouteID=DN47" Uri-Query: "Lat=22.5658745" Uri-Query: "Long=88.4107966667" Uri-Query: "Time=2013-01-13T11:24:31" No-Response: 0  [No response from the server. Next update in 20 secs.]           </pre>
<pre> +-----&gt;   POST           </pre>	<pre> Header: POST (T=NON, Code=0.02, MID=0x7d39) 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           </pre>

Figure 3: Exemplary unreliable update with No-Response option using POST with a query-string to insert update information to backend database.

## 6.2. An end-to-end system combining No-Response and Observe

This example illustrates the scenario pointed out in Section 5.3 above. The 'No-Response' option can be combined with the 'Observe' option with NON-notifications to create a lightweight end-to-end producer-consumer system. For example, the vehicular updates from a remote vehicle may be observed by a remote observer in a PDA as shown in figure 4.



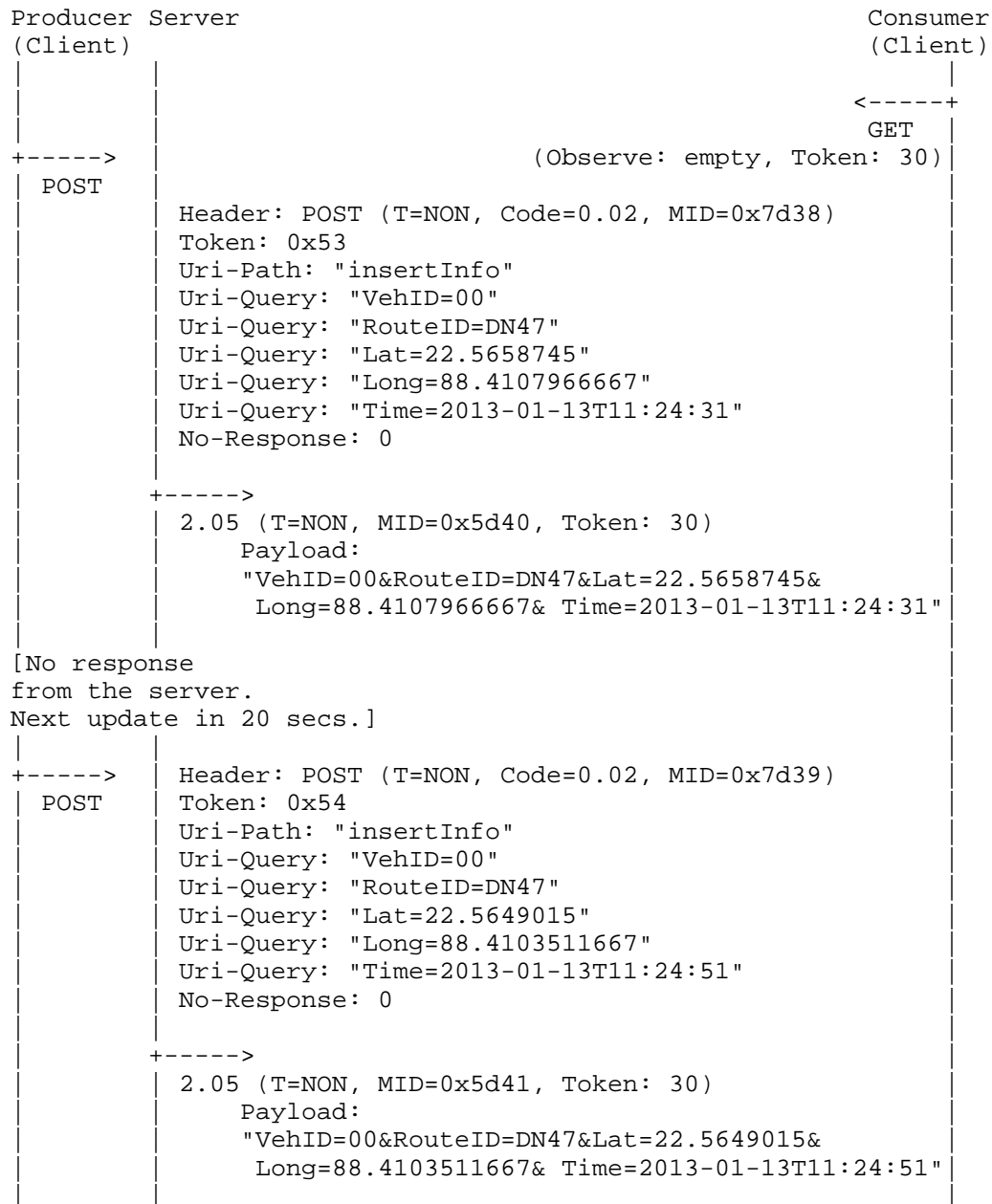


Figure 4: Exemplary end-to-end update and observe scenario using 'No-Response' for NON-updates from 'producer' and observe with NON-notifications by the 'consumer'.

## 7. IANA Considerations

The IANA is requested to add the following option number entries:

Number	Name	Reference
TBD	No-Response	Section 4 of this document

## 8. 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].

## 9. Acknowledgments

Thanks to Carsten Bormann, Esko Dijk, Bert Greevenbosch, Akbar Rahman and Claus Hartke for their valuable inputs.

## 10. References

### 10.1. 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-13, April 10, 2014

[I-D.ietf-core-groupcomm]

Rahman, A. and Dijk, E., "Group Communication for CoAP", draft-ietf-core-groupcomm-19, June 17, 2014

[I-D.draft-bormann-coap-misc]

Bormann, C. and Hartke, K., "Miscellaneous additions to CoAP", draft-bormann-coap-misc-26, December 19, 2013

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

Kovatsch, M., Bergmann, O., Dijk, E., He, X. and Bormann, C., "CoAP Implementation Guidance", draft-kovatsch-lwig-coap-03, February 28, 2014

[RFC 5405]

Eggert, L. and Fairhurst, G., "Unicast UDP Usage Guidelines for Application Designers"

## 10.2. Informative References

[MOBIQUITOUS 2013]

Bhattacharyya, A., Bandyopadhyay, S. and Pal, A., "ITS-light: Adaptive lightweight scheme to resource optimize intelligent transportation tracking system (ITS)-Customizing CoAP for opportunistic optimization", 10th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous 2013), December, 2013.

[Sensys 2013]

Bandyopadhyay, S., Bhattacharyya, A. and Pal, A., "Adapting protocol characteristics of CoAP using sensed indication for vehicular analytics", 11th ACM Conference on Embedded Networked Sensor Systems (Sensys 2013), November, 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  
Internet-Draft  
Intended status: Informational  
Expires: November 9, 2014

P. van der Stok  
consultant  
B. Greevenbosch  
Huawei Technologies  
May 8, 2014

CoAp Management Interfaces  
draft-vanderstok-core-comi-04

Abstract

The draft describes an interface based on CoAP to manage constrained devices via MIBs. The proposed integration of CoAP with SNMP reduces the code- and application development complexity by accessing MIBs via a standard CoAP server. The payload of the MIB request is CBOR based on JSON. The mapping from SMI to CBOR is specified. The introduction motivates the choices of CoMI with respect to utilization of YANG, NETCONF, SMI, CBOR, CoAP, and URI structure.

Note

Discussion and suggestions for improvement are requested, and should be sent to [core@ietf.org](mailto: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 November 9, 2014.

Copyright Notice

Copyright (c) 2014 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. Design considerations . . . . .	4
1.2. Terminology . . . . .	5
2. CoAP Interface . . . . .	5
3. MIB Function Set . . . . .	6
3.1. SNMP/MIB architecture . . . . .	6
3.1.1. SNMP functions . . . . .	7
3.1.2. MIB structure . . . . .	7
3.2. CoMI Function Set . . . . .	9
3.2.1. Single MIB values . . . . .	10
3.2.2. multi MIB values . . . . .	12
3.2.3. Table row . . . . .	14
3.2.4. MIB discovery . . . . .	15
3.2.5. Error returns . . . . .	16
4. Mapping SMI to CoMI payload . . . . .	16
4.1. CBOR format for MIB data . . . . .	16
4.2. Table generation . . . . .	17
4.3. Mapping SMI to CBOR . . . . .	18
4.3.1. General overview . . . . .	18
4.3.2. Conversion from YANG datatypes to CBOR datatypes . . . . .	18
4.3.3. Examples . . . . .	20
4.3.4. 6LoWPAN MIB . . . . .	22
5. Trap functions . . . . .	23
6. MIB access management . . . . .	23
6.1. Notify destinations . . . . .	23
6.2. Conversion table management . . . . .	23
7. Error handling . . . . .	24
8. Security Considerations . . . . .	25
9. IANA Considerations . . . . .	26
10. Acknowledgements . . . . .	26
11. Changelog . . . . .	26
12. References . . . . .	27
12.1. Normative References . . . . .	27
12.2. Informative References . . . . .	28
Appendix A. Notational Convention for CBOR data . . . . .	30
Appendix B. Example conversion table and instance for the LOWPAN	

MIB . . . . .	31
B.1. Generating the convTableId . . . . .	31
B.2. Generating the string numbers . . . . .	31
B.3. Example instance . . . . .	35
Authors' Addresses . . . . .	37

## 1. Introduction

The Constrained RESTful Environments (CoRE) working group aims at Machine to Machine (M2M) applications such as smart energy and building control.

Small M2M devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected to be installed in future installations. The management protocol of choice for Internet is SNMP [RFC3410] as is testified by the large number of Management Information Base (MIB) [RFC3418] specifications currently published [STD0001]. More recently, the NETCONF protocol [RFC6241] was developed with an extended set of messages using XML [XML] as data format. The data syntax is specified with YANG [RFC6020] and a mapping from Yang to XML is specified. In [RFC6643] SMIV2 syntax is expressed in Yang. Contrary to SNMP and also CoAP, NETCONF assumes persistent connections for example provided by SSH. The NETCONF protocol provides operations to retrieve, configure, copy, and delete configuration data-stores. Configuring data-stores distinguishes NETCONF from SNMP which operates on standardized MIBs.

The CoRE Management Interface (CoMI) is intended to work on standardized data-sets in a stateless client-server fashion and is thus closer to SNMP than to NETCONF. Standardized data sets promote interoperability between small devices and applications from different manufacturers. Stateless communication is encouraged to keep communications simple and the amount of state information small in line with the design objectives of 6lowpan [RFC4944] [RFC6775], RPL [RFC6650], and CoAP [I-D.ietf-core-coap].

The draft [I-D.bierman-netconf-restconf] describes a restful interface to NETCONF data stores and approaches the CoMI approach. CoMI uses SMI encoded in CBOR, and CoAP/UDP to access MIBs, whereas RESTCONF uses YANG encoded in JSON and HTTP/TCP to access NETCONF data stores. CoMI is more low resource oriented than RESTCONF is, and only supports the methods GET, PUT, POST and DELETE. RESTCONF also uses the HTTP methods HEAD, OPTIONS and PATCH, which are not available in CoAP.

Given the automatic conversion from SMI to YANG, from YANG to JSON, from YANG to XML and from JSON to CBOR, the transported data format is not strongly related to the chosen management protocol.



Currently, managed devices need to support two protocols: CoAP and SNMP. When the MIB can be accessed with the CoAP protocol, the SNMP protocol can be replaced with the CoAP protocol. This arrangement reduces the code complexity of the stack in the constrained device, and harmonizes applications development.

The objective of CoMI is to provide a CoAP based Function Set that reads and sets values of MIB variables in devices to (1) initialize parameter values at start-up, (2) acquire statistics during operation, and (3) maintain nodes by adjusting parameter values during operation.

The payload of CoMI is encoded in CBOR [RFC7049] which is similar to JSON [JSON], but has a binary format and hence has more coding efficiency. CoMI is intended for small devices. The JSON overhead can be prohibitive. It is therefore chosen to transport CBOR in the payload. CBOR, like BER used for SNMP, transports the data type in the payload.

The end goal of CoMI is to provide information exchange over the CoAP transport protocol in a uniform manner as a first step to the full management functionality as specified in [I-D.ersue-constrained-mgmt].

#### 1.1. Design considerations

COMI supports discovery of resources, accompanied by reading, writing and notification of resource values. As such it is close to the device management of the Open Mobile Alliance described in [OMA]. However, the structure of the MIB does not reflect the structure of the OMA management objects. It is assumed that the structure and semantics of the management data are the most important aspect of a standard like the MIB. The right path forward to integrate OMA management with COMI is the adaptation of the MIB structure by OMA.

COMI supports the conversion of SMIV2 via YANG to CBOR. Assuming that CBOR is used for the transport of NETCONF data, the utilization of the CBOR conversion table to reduce payload size can be envisaged for NETCONF data as well.

COMI uses a simple URI to access the MIB resources. Complexity introduced by module name, context specification, or row selection, is expressed with uri-query attributes. The choice for uri-query attributes makes the uri structure less context dependent.

## 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 [RFC2119].

Readers of this specification are required to be familiar with all the terms and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

Core Management Interface (CoMI) specifies the profile of Function Sets which access MIBs with the purpose of managing the operation of constrained devices in a network.

The following list defines the terms used in this document:

**Managing Entity:** An entity that manages one or more managed devices. Within the CoMI framework, the managing entity acts as a CoAP client for CoMI.

**Managed Device:** An entity that is being managed. The managed device acts as a CoAP server for CoMI.

**NOTE:** It is assumed that the managed device is the most constrained entity. The managing entity might be more capable, however this is not necessarily the case.

The following list contains the abbreviations used in this document.

**OID:** ASN.1 OBJECT-IDENTIFIER, which is used to uniquely identify MIB objects in the managed device.

## 2. CoAP Interface

In CoRE a group of links can constitute a Function Set. The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Function Set. CoMI end-points that implement the CoMI management protocol support at least one discoverable management resource of resource type (rt): core.mg, with path: /mg, where mg is short-hand for management. The mg resource has two sub-resources accessible with the paths:

- o MIB with path /mg/mib and a CBOR content format.
- o conv with path /mg/conv and CBOR content format.

The mib resource provides access to the MIBs as described in Section 3.2. The conv resource provides access to a table that maps

a string to CBOR identifier, as described in Section 4.1. The mib and conv resources are introduced as sub resources to mg to permit later additions to CoMI mg resource.

The profile of the management function set, with IF=core.mg.mib, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

name	path	RT	Data Type
Management	/mg	core.mg	n/a
MIB	/mg/mib	core.mg.mib	application/cbor
conv	/mg/conv	core.mg.conv	application/cbor

### 3. MIB Function Set

The MIB Function Set provides a CoAP interface to perform equivalent functions to the ones provided by SNMP. Section 3.1 explains the structure of SNMP Protocol Data Units (PDU), their transport, and the structure of the MIB modules. An excellent overview of the documents describing the SNMP/MIB architecture is provided in section 7 of [RFC3410].

#### 3.1. SNMP/MIB architecture

The architecture of the Internet Standard management framework consists of:

- o A data definition language that is referred to as Structure of Management Information (SMI)[RFC2578].
- o The Management Information Base (MIB) which contains the information to be managed and is defined for each specific function to be managed [RFC3418].
- o A protocol definition referred to as Simple Network Management Protocol (SNMP) [RFC3416].
- o Security and administration that provides SNMP message based security on the basis of the user-based security model [RFC3414].
- o A management domain definition where a SNMP entity has access to a collection of management information called a "context" [RFC3411].

In addition [RFC4088] describes a URI scheme to refer to a specific MIB instance.

Separation in modules was motivated by the wish to respond to the evolution of Internet. The protocol part (SNMP) and data definition part (MIB) are independent of each other. The separation has enabled the progressive passage from SNMPv1 via SNMPv2 to SNMPv3. This draft leverages this separation to replace the SNMP protocol with a CoAP based protocol.

#### 3.1.1. SNMP functions

The SNMP protocol supports seven types of access supported by as many Protocol Data Unit (PDU) types:

- o Get Request, transmits a list of OBJECT-IDENTIFIERS to be paired with values.
- o GetNext Request, transmits a list of OBJECT-IDENTIFIERS to which lexicographic successors are returned for table traversal.
- o GetBulk Request, transmits a list of OBJECT-IDENTIFIERS and the maximum number of expected paired values.
- o Response, returns an error or the (OBJECT-IDENTIFIER, value) pairs for the OBJECT-IDENTIFIERS specified in Get, GetNext, GetBulk, Set, or Inform Requests.
- o Set Request, transmits a list of (OBJECT-IDENTIFIERS, value) pairs to be set in the specified MIB object.
- o Trap, sends an unconfirmed message with a list of (OBJECT-IDENTIFIERS, value) pairs to a notification requesting end-point.
- o Inform Request, sends a confirmed message with a list of (OBJECT-IDENTIFIERS, value) pairs to a notification requesting end-point.

#### 3.1.2. MIB structure

A MIB module is composed of MIB objects. MIB objects are standardized by the IETF or by other relevant Standards Developing Organizations (SDO).

MIB objects have a descriptor and an identifier: OBJECT-IDENTIFIER (OID). The identifier, following the OSI hierarchy, is an ordered list of non-negative numbers [RFC2578]. OID values are unique. Each number in the list is referred as a sub-identifier. The descriptor is unique within a module. Different modules may contain the same

descriptor. Consequently, a descriptor can be related to several OIDs.

Many instances of an object type exist within a management domain. Each instance can be identified within some scope or "context", where there are multiple such contexts within the management domain. Often, a context is a physical or logical device. A context is always defined as a subset of a single SNMP entity. To identify an individual item of management information within the management domain, its contextName and contextEngineID must be identified in addition to its object type and its instance. A default context is assumed when no context is specified.

A MIB object is usually a scalar object. A MIB object may have a tabular form with rows and columns. Such an object is composed of a sequence of rows, with each row composed of a sequence of typed values. The index is a subset (1-2 items) of the typed values in the row. An index value identifies the row in the table.

In SMI, a table is constructed as a SEQUENCE OF its entries. For example, the IpAddrTable from [RFC4293] has the following definition:

```

ipv6InterfaceTable OBJECT-TYPE
    SYNTAX                SEQUENCE OF Ipv6InterfaceEntry
    MAX-ACCESS             not-accessible
    STATUS                 current
    DESCRIPTION
        "The table containing per-interface IPv6-specific
        information."
    ::= { ip 30 }

ipv6InterfaceEntry OBJECT-TYPE
    SYNTAX                Ipv6InterfaceEntry
    MAX-ACCESS             not-accessible
    STATUS                 current
    DESCRIPTION
        "An entry containing IPv6-specific information for a given
        interface."
    INDEX { ipv6InterfaceIfIndex }
    ::= { ipv6InterfaceTable 1 }

Ipv6InterfaceEntry ::= SEQUENCE {
    ipv6InterfaceIfIndex      InterfaceIndex,
    ipv6InterfaceReasmMaxSize Unsigned32,
    ipv6InterfaceIdentifier   Ipv6AddressIfIdentifierTC,
    ipv6InterfaceEnableStatus INTEGER,
    ipv6InterfaceReachableTime Unsigned32,
    ipv6InterfaceRetransmitTime Unsigned32,
    ipv6InterfaceForwarding   INTEGER
}

```

The descriptor (name) of the MIB table is used for the name of the CoMI variable. However, there is no explicit mention of the names "ipv6InterfaceEntry" and "Ipv6InterfaceEntry". Instead, the value of the main CoMI variable, ipv6InterfaceTable, consists of an array, each element of which contains 7 CoMI variables: one element for "ipv6InterfaceIfIndex", one for "ipv6InterfaceReasmMaxSize" and so on until "ipv6InterfaceForwarding".

### 3.2. CoMI Function Set

Three types of interfaces are supported by CoMI:

Single value: Reading/Writing one MIB variable, specified in the URI with path /mg/mib/descriptor or with path /mg/mib/OID.

Multiple values: Reading/writing arrays or multiple MIB variables, specified in the payload.

Discovery: Discovery of MIB descriptors as specified in [RFC6690].

The examples in this section use a JSON payload with one or more entries describing the pair (descriptor, value), or (OID, value). CoMI transports the CBOR format to transport the equivalent contents. The CBOR syntax of the payloads is specified in Section 4.

### 3.2.1. Single MIB values

A request to read the value of a MIB variable is sent with a confirmable CoAP GET message. The single MIB variable is specified in the URI path with the OID or descriptor suffixing the /mg/mib/ path name. When the descriptor is used to specify the MIB value, the same descriptor may be present in multiple module. To disambiguate the descriptor the "mod" uri-query attribute specifies the enveloping modules. A request to set the value of a MIB variable is sent with a confirmable CoAP PUT message. The Response is piggybacked to the CoAP ACK message corresponding with the Request.

TODO: for multicast send unconfirmed PUT

Using for example the same MIB from [RFC1213] as used in [RFC3416], a request is sent to retrieve the value of sysUpTime specified in module SNMPv2-MIB. The answer to the request returns a (descriptor, value) pair.

As announced, in all examples the payload is expressed in JSON, although the operational payload is specified to be in CBOR.

REQ: GET example.com/mg/mib/sysUpTime?mod=SNMPv2-MIB

RES: 2.05 Content (Content-Format: application/json)  
{  
 "sysUpTime" : 123456  
}

Another way to express the descriptor of the required value is by specifying the pair (descriptor or oid, null value) in the payload of the request message.

```
REQ: GET example.com/mg/mib/(Content-Format: application/json)
{
  "SNMPv2-MIB.sysUpTime" : "null"
}
```

```
RES: 2.05 Content (Content-Format: application/json)
{
  "SNMPv2-MIB.sysUpTime" : 123456
}
```

The module name SNMPv2-MIB can be omitted when there is no possibility of ambiguity. The module.descriptor can of course be replaced with the corresponding oid.

In some cases it is necessary to determine the "context" by specifying a context name and a contextEngine identifier. The context can be specified in the URI with the uri-query attribute "con". Based on the example of figure 3 in section 3.3 of [RFC3411], the context name, bridge1, and the context Engine Identifier, 800002b804616263, separated by an underscore, are specified in the following example:

```
REQ: GET example.com/mg/mib/sysUpTime?con=bridge1_800002b804616263
```

```
RES: 2.05 Content (Content-Format: application/json)
{
  "sysUpTime" : 123456
}
```

The specified object can be a table. The returned payload is composed of all the rows associated with the table. Each row is returned as a set of (column name, value) pairs. For example the GET of the ipNetToMediaTable, sent by the managing entity, results in the following returned payload sent by the managed entity:



REQ: GET example.com/mg/mib/ipNetToMediaTable

RES: 2.05 Content (Content-Format: application/json)

```
{
  "ipNetToMediaTable" : [
    {
      "ipNetToMediaIfIndex" : 1,
      "ipNetToMediaPhysAddress" : "00:00::10:01:23:45",
      "ipNetToMediaNetAddress" : "10.0.0.51",
      "ipNetToMediaType" : "static"
    },
    {
      "ipNetToMediaIfIndex" : 1,
      "ipNetToMediaPhysAddress" : "00:00::10:54:32:10",
      "ipNetToMediaNetAddress" : "9.2.3.4",
      "ipNetToMediaType" : "dynamic"
    },
    {
      "ipNetToMediaIfIndex" : 2,
      "ipNetToMediaPhysAddress" : "00:00::10:98:76:54",
      "ipNetToMediaNetAddress" : "10.0.0.15",
      "ipNetToMediaType" : "dynamic"
    }
  ]
}
```

It is possible that the size of the returned payload is too large to fit in a single message.

CoMI gives the possibility to send the contents of the objects in several fragments with a maximum size. The "sz" link-format attribute [RFC6690] can be used to specify the expected maximum size of the mib resource in (identifier, value) pairs. The returned data MUST terminate with a complete (identifier, value) pair.

In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [I-D.ietf-core-block] is used. Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

### 3.2.2. multi MIB values

A request to read multiple MIB variables is done by expressing the pairs (MIB descriptor, null) in the payload of the GET request message. A request to set multiple MIB variables is done by expressing the pairs (MIB descriptor, value) in the payload of the

PUT request message. The key word `_multiMIB` is used as array name to signal that the payload contains multiple MIB values as separate `_multiMIB` array entries.

The following example shows a request that specifies to return the values of `sysUpTime` and `ipNetToMediaTable`:

```
REQ: GET example.com/mg/mib (Content-Format: application/json)
```

```
{
  "_multiMIB" : [
    { "sysUpTime" : "null"},
    { "ipNetToMediaTable" : "null" }
  ]
}
```

```
RES: 2.05 Content (Content-Format: application/json)
```

```
{
  "_multiMIB" : [
    { "sysUpTime" : 123456},
    { "ipNetToMediaTable" : [
      {
        "ipNetToMediaIfIndex" : 1,
        "ipNetToMediaPhysAddress" : "00:00::10:01:23:45",
        "ipNetToMediaNetAddress" : "10.0.0.51",
        "ipNetToMediaType" : "static"
      },
      {
        "ipNetToMediaIfIndex" : 1,
        "ipNetToMediaPhysAddress" : "00:00::10:54:32:10",
        "ipNetToMediaNetAddress" : "9.2.3.4",
        "ipNetToMediaType" : "dynamic"
      },
      {
        "ipNetToMediaIfIndex" : 2,
        "ipNetToMediaPhysAddress" : "00:00::10:98:76:54",
        "ipNetToMediaNetAddress" : "10.0.0.15",
        "ipNetToMediaType" : "dynamic"
      }
    ]
  ]
}
```

### 3.2.3. Table row

The managing entity MAY be interested only in certain table entries. One way to specify a row is to specify its row number in the URI with the "row" uri-query attribute. The specification of row=1 returns row 1 values of the ipNetToMediaTable in the example:

```
REQ: GET example.com/mg/mib/ipNetToMediaTable?row=1
```

```
RES: 2.05 Content (Content-Format: application/json)
```

```
{ "ipNetToMediaTable" : [  
  {  
    "ipNetToMediaIfIndex" : 1,  
    "ipNetToMediaPhysAddress" : "00:00::10:01:23:45",  
    "ipNetToMediaNetAddress" : "10.0.0.51",  
    "ipNetToMediaType" : "static"  
  }  
]
```

An alternative mode of selection is by specifying the value of the INDEX attributes. Towards this end, the managing entity can include the required entries in the payload of its "GET" request by specifying the values of the index attributes. The key word \_indexMIB is used to specify the index value.

For example, to obtain a table entry from ipNetToMediaTable, the rows are specified by specifying the index attributes: ipNetToMediaIfIndex and ipNetToMediaNetAddress. The managing entity could have sent a GET with the following payload:

REQ: GET example.com/mg/mib/ipNetToMediaTable(Content-Format: application/json)

```
{ "_indexMIB" :  
  {  
    "ipNetToMediaIfIndex" : 1,  
    "ipNetToMediaNetAddress" : "9.2.3.4"  
  }  
}
```

RES: 2.05 Content (Content-Format: application/json)

```
{ "ipNetToMediaTable" : [  
  {  
    "ipNetToMediaIfIndex" : 1,  
    "ipNetToMediaPhysAddress" : "00:00::10:01:23:45",  
    "ipNetToMediaNetAddress" : "9.2.3.4",  
    "ipNetToMediaType" : "static"  
  }  
]
```

Constrained devices MAY support this kind of filtering. However, if they don't support it, they MUST ignore the payload in the GET request and handle the message as if the payload was empty.

It is advised to keep MIBs for constrained entities as simple as possible, and therefore it would be best to avoid extensive tables.

TODO: Describe how the contents of the next lexicographical row can be returned.

#### 3.2.4. MIB discovery

MIB objects are discovered like resources with the standard CoAP resource discovery. Performing a GET on `/.well-known/core` with `rt=core.mg.mib` returns all MIB descriptors and all OIDs which are available on this device. For table objects there is no further possibility to discover the row descriptors. For example, consider there are two MIB objects with descriptors `"sysUpTime"` and `"ipNetToMediaTable"` associated with OID `1.3.6.1.2.1.1.3` and `1.3.6.1.2.1.4.22`

REQ: GET example.com/.well-known/core?rt=core.mg.mib

RES: 2.05 Content (Content-Format: application/text)

```
</mg/mib/sysUpTime>;rt="core.mg.mib";oid="1.3.6.1.2.1.1.3";  
  mod="SNMPv2-MIB"  
</mg/mib/ipNetToMediaTable>;rt="core.mg.mib";oid="1.3.6.1.2.1.4.22";  
  mod="ipMIB"
```

The link format attribute 'oid' is used to associate the name of the MIB resource with its OID. The OID is written as a string in its conventional form.

Notice that a MIB variable normally is associated with a descriptor and an OID. The OID is unique, whereas the descriptor is unique in combination with the module name.

The "mod", "con", and "rt" attributes can be used to filter resource queries as specified in [RFC6690].

### 3.2.5. Error returns

When a variable with the specified name cannot be processed, CoAP Error code 5.01 is returned. In addition, a MIB specific error can be returned in the payload as specified in Section 7.

## 4. Mapping SMI to CoMI payload

The SMI syntax is mapped to CBOR necessary for the transport of MIB data in the CoAP payload. This section first describes an additional data reduction technique by creating a table that maps string values to numbers used in CBOR encoded data.

The section continues by describing the mapping from SMI to CBOR. The mapping is inspired by the mapping from SMI to JSON via YANG [RFC6020], as described in [RFC6643] defining a mapping from SMI to YANG, and [I-D.lhotka-netmod-yang-json] defining a mapping from YANG to JSON.

Notice that such conversion chain MAY be virtual only, as SMI could be converted directly to JSON by combining the rules from the above documents.

### 4.1. CBOR format for MIB data

Because descriptors may be rather long and may occur repeatedly, CoMI allows for association of a given string value with an integer, henceforth called "string number". The association between string value and string number is done through a conversion table, leveraging CBOR encoding. Section 4.2 defines how the conversion table is generated.

Using the notational convention from Appendix A, the CBOR data has the following syntax:

```
cBorMIB      : CBorMIB;

*CBorMIB {
    convTableId : tstr;
    mibData      : map( uint, . );
}
```

The main structure consist of an array of two elements: "convTableId" and "mibData".

The conversion table identifier "convTableId" is constructed from the LAST\_UPDATED attribute and module name as defined in Section 4.2.

The values of the MIB variables are stored in the "mibData" field. This field consist of integer-value pairs. The integers correspond to the string numbers, whereas the values contain the actual value of the associated MIB variable. Section 4.3 elaborates on the mapping from SMI to CBOR.

#### 4.2. Table generation

The conversion table contents MUST be automatically generated from the MIB module as specified in this section. Automatic generation in the managed device removes the need to transport the tables and subsequently store them in the nodes, and avoids a cumbersome management of the conversion tables.

The conversion table identifier "convTableId" is generated from the LAST-UPDATED field in the MODULE IDENTITY macro, and the module name as follows:

```
convTableId = moduleName UNDERSCORE lastUpdateTimestamp
UNDERSCORE  = %x5F
```

where moduleName is the module name, and lastUpdateTimestamp is the value of the related LAST-UPDATED field in the MODULE-IDENTITY macro, using the format with four year digits as defined in [RFC2578].

The conversion table is generated using the following algorithm:

1. Collect all used OIDs/descriptors defined in the MIB object, and as well as those imported using IMPORTS.
2. Sort the collection of OIDs according to the following rules:
  1. x.y.z < x.y.z.k
  2. x.y.z... < x.y.k... if z < k

3. Assign the string numbers to the sorted list of OIDs, in numerical order starting with 1.

Note that variables imported with IMPORTS could be tables, such that the OIDs of the table entries need to be included in the collection of OIDs as well.

#### 4.3. Mapping SMI to CBOR

##### 4.3.1. General overview

Starting from the intermediate conversion from SMI to YANG as defined in [RFC6643], This section defines how to convert the resulting YANG structure to CBOR [RFC7049]. The actual conversion code from SMI to CBOR MAY be direct conversion code from SMI to CBOR or a sequence of existing SMI to YANG conversion code followed by YANG to CBOR conversion code.

##### 4.3.2. Conversion from YANG datatypes to CBOR datatypes

Table 1 defines the mapping between YANG datatypes and CBOR datatypes.

Elements of types not in this table, and of which the type cannot be inferred from a type in this table, are ignored in the CBOR encoding by default. Examples include the "description" and "key" elements. However, conversion rules for some elements to CBOR MAY be defined elsewhere.

YANG type	CBOR type	Specification
int8, int16, int32, int64, uint16, uint32, uint64, decimal64	unsigned int (major type 0) or negative int (major type 1)	The CBOR integer type depends on the sign of the actual value.
boolean	either "true" (major type 7, simple value 21) or "false" (major type 7, simple value 20)	

string	text string (major type 3)	
enumeration	unsigned int (major type 0)	
bits	array of text strings	Each text string contains the name of a bit value that is set.
binary	byte string (major type 2)	
empty	null (major type 7, simple value 22)	TBD: This MAY not be applicable to true MIBs, as SNMP may not support empty variables...
union		Similar ot the JSON transcription from [I-D.lhotka-netmod-yang-json], the elements in a union MUST be determined using the procedure specified in section 9.12 of [RFC6020].
leaf-list	array (major type 4)	The array is encapsulated in the map associated with the descriptor.
list	map (major type 4)	Like the higher level map, the lower level map contains descriptor number - value pairs of the elements in the list.
container	map (major type 5)	The map contains decriptor number - value pairs corresponding to the elements in the container.
smiv2:oid	array of integers	Each integer contains an element of the OID, the first integer in the array corresponds to the most left element in the OID.

Table 1: Conversion of YANG datatypes to CBOR



### 4.3.3. Examples

#### 4.3.3.1. ipNetToMediaTable to JSON/CBOR

The YANG translation of the SMI specifying the ipNetToMediaTable yields:

```

container ipNetToMediaTable {
  list ipNetToMediaEntry {
    leaf ipNetToMediaIfIndex {
      type: int32;
    }
    leaf ipNetToPhysAddress {
      type: phys-address;
    }
    leaf ipNetToMediaNetAddress {
      type: ipv4-address;
    }
    leaf ipNetToMediaType {
      type: int32;
    }
  }
}

```

The coresponding JSON looks like:

```

{
  "ipNetToMediaTable" : {
    "ipNetToMediaEntry" : [
      {
        "ipNetToMediaIfIndex" : 1,
        "ipNetToMediaPhysAddress" : "00:00::10:01:23:45",
        "ipNetToMediaNetAddress" : "10.0.0.51",
        "ipNetToMediaType" : "static"
      },
      {
        "ipNetToMediaIfIndex " : 1,
        "ipNetToMediaPhysAddress " : "00:00::10:54:32:10",
        "ipNetToMediaNetAddress" : "9.2.3.4",
        "ipNetToMediaType " : "dynamic"
      }
    ]
  }
}

```

An example CBOR instance of the MIB can be found in Figure 1. The names "ipNetToMediaTable", "ipNetToMediaEntry", and "ipNetToMediaIfIndex" are represented with the string numbers 00, 01, and 02 as described in Section 4.1.

```

82                                # two element array
 74 49 50 2d 4d 49 42 5f
 32 30 30 36 30 32 30 32
 30 30 30 30 5a                  # "IP-MIB_200602020000Z"
BF                                # indefinite length map
  00                              # descriptor number related to
                                # ipNetToMediaTable
    BF                            # indefinite length map related to
                                # ipNetToMediaTable
      01                          # descriptor number related to
                                # ipNetToMediaEntry
        BF                        # map related to ipNetToMediaEntry
          02                      # descriptor number associated with
                                # ipNetToMediaIfIndex
            1A 00 00 00 01        # associated value as 32-bit integer
            # ...
      FF
    FF
  FF

```

Figure 1: Example CBOR encoding for ifTable

The associated "descriptor string" to "string number" conversion table is given in Figure 2.

```

82                                     # two element array
 74 49 50 2d 4d 49 42 5f
32 30 30 36 30 32 30 32
30 30 30 30 5a                       # "IP-MIB_200602020000Z"
BF                                     # indefinite length map
 00                                     # descriptor number related to
                                     # ipNetToMediaTable
 72 69 70 50 65 74 57
6F 51 65 64 61 57 61
62 6C 65                             # "ipNetToMediaTable"
01                                     # descriptor number related to
                                     # ipNetToMediaEntry
 72 69 70 50 65 74 57
6F 51 65 64 61 45 6E
74 72 78                             # "ipNetToMediaEntry"
02                                     # descriptor number related to
                                     # ipNetToMediaIfIndex
 75 69 70 50 65 74 57
6F 51 65 64 61 61 49
66 49 6E 64 65 77                   # "ipNetToMediaIfIndex"
# ...
FF

```

Figure 2: Conversion table for ifTable

#### 4.3.4. 6LoWPAN MIB

A MIB for 6LoWPAN is defined in [I-D.ietf-6lo-lowpan-mib] with an example JSON representation in its Appendix A. Appendix B.3 shows the associated CBOR representation with string number, and the corresponding string to string number conversion table.

In this example, a GET to /mg/mib/lowpanOutFragFails would give:

```

82                                     # two element array
 78 18 4c 4f 57 50 41 4e
2d 4d 49 42 5f 32 30 31
34 30 34 30 38 30 30 30
30 5a                               # conversion table id:
                                     # "LOWPAN-MIB_201404080000Z"
BF                                     # indefinite length map
 18 1B                               # "lowpanOutFragFails"
 00                                   # 0
FF

```

## 5. Trap functions

A trap can be set through the CoAP Observe [I-D.ietf-core-observe] function. As regular with Observe, the managing entity subscribes to the variable by sending a GET request with an "Observe" option.

TODO: Observe example

In the registration request, the managing entity MAY include a "Response-To-Uri-Host" and optionally "Response-To-Uri-Port" option as defined in [I-D.becker-core-coap-sms-gprs]. In this case, the observations SHOULD be sent to the address and port indicated in these options. This can be useful when the managing entity wants the managed device to send the trap information to a multicast address.

## 6. MIB access management

Two topics are relevant: (1) the definition of the destination of Notify messages, and (2) the creation and maintenance of "string to number" tables.

### 6.1. Notify destinations

The destination of notifications need to be communicated to the applications sending them. Draft [I-D.ietf-core-interfaces] describes the binding of end-points to end-points on remote devices. The object with type "binding table" contains a sequence of bindings. The contents of bindings contains the methods, location, the interval specifications, and the step value as suggested in [I-D.ietf-core-interfaces]. The method "notify" has been added to the binding methods "poll", "obs" and "push", to cater for the binding of notification source to the receiver.

TODO: describe interface for NOTIFY destination definition.

### 6.2. Conversion table management

Since the conversion table is unique for a MIB, it can be generated off-line by a managing entity, and independently generated in the managed device with the same result. Because different versions of a given MIB may be available the managing entity must be able to check the version on the managed device. The convTableId can be acquired with a confirmable GET request:

```
REQ: GET example.com/mg/conv/ident
```

```
RES: 2.05 Content (Content-Format: application/json)
{
  "convTableId" : convTableId
}
```

The conversion table MAY be available on a managed device, and can then be acquired as follows:

```
GET /mg/conv/[convTableId]
```

where "[convTableId]" is replaced by the the value of convTableId from the CBorMIB structure.

In case the managed device provides the table, it must be transmitted in the payload of the GET response using the following format:

```
convTable      : ConvTable;

*ConvTable {
  convTableId : tstr;
  convData    : map( uint, tstr );
}
```

where "convTableId" is the conversion table identifier, as defined in Section 4.2, and "convData" is a CBOR map between the string number and associated variable descriptor.

## 7. Error handling

In case a request is received which cannot be processed properly, the managed entity MUST return an error message. This error message MUST contain a CoAP 4.xx or 5.xx response code, and SHOULD include additional information in the payload.

Such an error message payload is encoded in CBOR, using the following structure:

```
errorMsg      : ErrorMsg;

*ErrorMsg {
  errorCode    : uint;
  ?errorText   : tstr;
}
```

The variable "errorCode" has one of the values from the table below, and the OPTIONAL "errorText" field contains a human readable explanation of the error.

CoMI Error Code	CoAP Error Code	Description
0	4.00	General error
1	4.00	Malformed CBOR data
2	4.00	Incorrect CBOR datatype
3	4.00	Unknown MIB variable
4	4.00	Unknown conversion table
5	4.05	Attempt to write read-only variable
0..2	5.01	Access exceptions
0..18	5.00	SMI error status

The CoAP error code 5.01 is associated with the exceptions defined in [RFC3416] and CoAP error code 5.00 is associated with the error-status defined in [RFC3416].

## 8. Security Considerations

For secure network management, it is important to restrict access to MIB variables only to authorised parties. This requires integrity protection of both requests and responses, and depending on the application encryption.

CoMI re-uses the security mechanisms already available to CoAP as much as possible. This includes DTLS for protected access to resources, as well suitable authentication and authorisation mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [I-D.ietf-core-coap]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorisation may need to be selected.

CoMI avoids defining new security mechanisms as much as possible. However some adaptations may still be required, to cater for CoMI's specific requirements.

## 9. IANA Considerations

'rt="core.mg.mib"' needs registration with IANA.

Content types to be registered:

- o application/comi+json
- o application/comi+cbor

## 10. Acknowledgements

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR. Juergen Schoenwalder has commented on inconsistencies and missing aspects of SNMP in earlier versions of the draft. The draft has benefited from comments (alphabetical order) by Dee Denteneer, Esko Dijk, Michael van Hartskamp, Zach Shelby, Michael Verschoor, and Thomas Watteyne. The CBOR encoding borrows extensively from Ladislav Lhotka's description on conversion from YANG to JSON.

## 11. Changelog

Changes from version 00 to version 01

- o Focus on MIB only
- o Introduced CBOR, JSON, removed BER
- o defined mappings from SMI to xx
- o Introduced the concept of addressable table rows

Changes from version 01 to version 02

- o Focus on CBOR, used JSON for examples, removed XML and EXI
- o added uri-query attributes mod and con to specify modules and contexts

- o Definition of CBOR string conversion tables for data reduction
- o use of Block for multiple fragments
- o Error returns generalized
- o SMI - YANG - CBOR conversion

Changes from version 02 to version 03

- o Added security considerations

Changes from version 03 to version 04

- o Added design considerations section
- o Extended comparison of management protocols in introduction
- o Added automatic generation of CBOR tables
- o Moved lowpan table to Appendix

## 12. References

### 12.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.
- [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.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",  
draft-ietf-core-block-14 (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-13 (work in progress), April 2014.
- [I-D.ietf-json-rfc4627bis]  
Bray, T., "The JSON Data Interchange Format", draft-ietf-json-rfc4627bis-10 (work in progress), December 2013.
- [I-D.lhotka-netmod-yang-json]  
Lhotka, L., "Modeling JSON Text with YANG", draft-lhotka-netmod-yang-json-02 (work in progress), September 2013.

## 12.2. Informative References

- [RFC1213] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets:MIB-II", STD 17, RFC 1213, March 1991.
- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.
- [RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.
- [RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.
- [RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.
- [RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.
- [RFC3416] Presuhn, R., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

- [RFC3418] Presuhn, R., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4088] Black, D., McCloghrie, K., and J. Schoenwaelder, "Uniform Resource Identifier (URI) Scheme for the Simple Network Management Protocol (SNMP)", RFC 4088, June 2005.
- [RFC4113] Fenner, B. and J. Flick, "Management Information Base for the User Datagram Protocol (UDP)", RFC 4113, June 2005.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.
- [RFC4293] Routhier, S., "Management Information Base for the Internet Protocol (IP)", RFC 4293, April 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.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, July 2012.
- [RFC6650] Falk, J. and M. Kucherawy, "Creation and Use of Email Feedback Reports: An Applicability Statement for the Abuse Reporting Format (ARF)", RFC 6650, June 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.

- [I-D.ietf-core-groupcomm]  
Rahman, A. and E. Dijk, "Group Communication for CoAP",  
draft-ietf-core-groupcomm-18 (work in progress), December  
2013.
- [I-D.ietf-core-interfaces]  
Shelby, Z. and M. Vial, "CoRE Interfaces", draft-ietf-  
core-interfaces-01 (work in progress), December 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.ietf-6lo-lowpan-mib]  
Schoenwaelder, J., Sehgal, A., Tsou, T., and C. Zhou,  
"Definition of Managed Objects for IPv6 over Low-Power  
Wireless Personal Area Networks (6LoWPANs)", draft-ietf-  
6lo-lowpan-mib-01 (work in progress), April 2014.
- [I-D.bierman-netconf-restconf]  
Bierman, A., Bjorklund, M., Watsen, K., and R. Fernando,  
"RESTCONF Protocol", draft-bierman-netconf-restconf-04  
(work in progress), February 2014.
- [STD0001] "Official Internet Protocols Standard", Web  
<http://www.rfc-editor.org/rfcxx00.html>, .
- [XML] "Extensible Markup Language (XML)", Web  
<http://www.w3.org/xml>, .
- [JSON] "JavaScript Object Notation (JSON)", Web  
<http://www.json.org>, .
- [OMA] "OMA-TS-LightweightM2M-V1\_0-20131210-C", Web  
[http://technical.openmobilealliance.org/Technical/  
current\\_releases.aspx](http://technical.openmobilealliance.org/Technical/current_releases.aspx), .

#### Appendix A. Notational Convention for CBOR data

To express CBOR structures [RFC7049], this document uses the following conventions:

A declaration of a CBOR variable has the form:

```
name : datatype;
```

where "name" is the name of the variable, and "datatype" its CBOR datatype.

The name of the variable has no encoding in the CBOR data.

"datatype" can be a CBOR primitive such as:

tstr: A text string (major type 3)

uint: An unsigned integer (major type 0)

map(x,y): A map (major type 5), where each first element of a pair is of datatype x, and each second element of datatype y. A '.' character for either x or y means that all datatypes for that element are valid.

A datatype can also be a CBOR structure, in which case the variable's "datatype" field contains the name of the CBOR structure. Such CBOR structure is defined by a character sequence consisting of first its name, then a '{' character, then its subfields and finally a '}' character.

A CBOR structure can be encapsulated in an array, in which case its name in its definition is preceded by a '\*' character. Otherwise the structure is just a grouping of fields, but without actual encoding of such grouping.

The name of an optional field is preceded by a '?' character. This means, that the field may be omitted if not required.

## Appendix B. Example conversion table and instance for the LOWPAN MIB

The conversion table of the 6LoWPAN MIB [I-D.ietf-6lo-lowpan-mib] is generated as described in this section.

### B.1. Generating the convTableId

The module name is "LOWPAN-MIB", and the LAST-UPDATED field in the MODULE-IDENTITY has the value "201404080000Z". Hence the convId has the value "LOWPAN-MIB\_201404080000Z".

### B.2. Generating the string numbers

The following table shows how the string numbers are associated with the related strings and object IDs.

Object ID	Descriptor	String
-----------	------------	--------

		number
1.3.6.1.2.1	mib-2	1
1.3.6.1.2.1.2.2.1.1	ifIndex	2
1.3.6.1.2.1.XXXX	lowpanMib	3
1.3.6.1.2.1.XXXX.0	lowpanNotifications	4
1.3.6.1.2.1.XXXX.1	lowpanObjects	5
1.3.6.1.2.1.XXXX.2	lowpanConformance	6
1.3.6.1.2.1.XXXX.1.1	lowpanStats	7
1.3.6.1.2.1.XXXX.1.1.1	lowpanReasmTimeout	8
1.3.6.1.2.1.XXXX.1.1.2	lowpanInReceives	9
1.3.6.1.2.1.XXXX.1.1.3	lowpanInHdrErrors	10
1.3.6.1.2.1.XXXX.1.1.4	lowpanInMeshReceives	11
1.3.6.1.2.1.XXXX.1.1.5	lowpanInMeshForwds	12
1.3.6.1.2.1.XXXX.1.1.6	lowpanInMeshDelivers	13
1.3.6.1.2.1.XXXX.1.1.7	lowpanInReasmReqds	14
1.3.6.1.2.1.XXXX.1.1.8	lowpanInReasmFails	15
1.3.6.1.2.1.XXXX.1.1.9	lowpanInReasmOKs	16
1.3.6.1.2.1.XXXX.1.1.10	lowpanInCompReqds	17
1.3.6.1.2.1.XXXX.1.1.11	lowpanInCompFails	18
1.3.6.1.2.1.XXXX.1.1.12	lowpanInCompOKs	19
1.3.6.1.2.1.XXXX.1.1.13	lowpanInDiscards	20
1.3.6.1.2.1.XXXX.1.1.14	lowpanInDelivers	21
1.3.6.1.2.1.XXXX.1.1.15	lowpanOutRequests	22
1.3.6.1.2.1.XXXX.1.1.16	lowpanOutCompReqds	23

1.3.6.1.2.1.XXXX.1.1.17	lowpanOutCompFails	24
1.3.6.1.2.1.XXXX.1.1.18	lowpanOutCompOKs	25
1.3.6.1.2.1.XXXX.1.1.19	lowpanOutFragReqds	26
1.3.6.1.2.1.XXXX.1.1.20	lowpanOutFragFails	27
1.3.6.1.2.1.XXXX.1.1.21	lowpanOutFragOKs	28
1.3.6.1.2.1.XXXX.1.1.22	lowpanOutFragCreates	29
1.3.6.1.2.1.XXXX.1.1.23	lowpanOutMeshHopLimitExceeds	30
1.3.6.1.2.1.XXXX.1.1.24	lowpanOutMeshNoRoutes	31
1.3.6.1.2.1.XXXX.1.1.25	lowpanOutMeshRequests	32
1.3.6.1.2.1.XXXX.1.1.26	lowpanOutMeshForwds	33
1.3.6.1.2.1.XXXX.1.1.27	lowpanOutMeshTransmits	34
1.3.6.1.2.1.XXXX.1.1.28	lowpanOutDiscards	35
1.3.6.1.2.1.XXXX.1.1.29	lowpanOutTransmits	36
1.3.6.1.2.1.XXXX.1.2	lowpanIfStatsTable	37
1.3.6.1.2.1.XXXX.1.2.1	lowpanIfStatsEntry	38
1.3.6.1.2.1.XXXX.1.2.1.1	lowpanIfReasmTimeout	39
1.3.6.1.2.1.XXXX.1.2.1.2	lowpanIfInReceives	40
1.3.6.1.2.1.XXXX.1.2.1.3	lowpanIfInHdrErrors	41
1.3.6.1.2.1.XXXX.1.2.1.4	lowpanIfInMeshReceives	42
1.3.6.1.2.1.XXXX.1.2.1.5	lowpanIfInMeshForwds	43
1.3.6.1.2.1.XXXX.1.2.1.6	lowpanIfInMeshDelivers	44
1.3.6.1.2.1.XXXX.1.2.1.7	lowpanIfInReasmReqds	45
1.3.6.1.2.1.XXXX.1.2.1.8	lowpanIfInReasmFails	46
1.3.6.1.2.1.XXXX.1.2.1.9	lowpanIfInReasmOKs	47

1.3.6.1.2.1.XXXX.1.2.1.1 0	lowpanIfInCompReqds	48
1.3.6.1.2.1.XXXX.1.2.1.1 1	lowpanIfInCompFails	49
1.3.6.1.2.1.XXXX.1.2.1.1 2	lowpanIfInCompOKs	50
1.3.6.1.2.1.XXXX.1.2.1.1 3	lowpanIfInDiscards	51
1.3.6.1.2.1.XXXX.1.2.1.1 4	lowpanIfInDelivers	52
1.3.6.1.2.1.XXXX.1.2.1.1 5	lowpanIfOutRequests	53
1.3.6.1.2.1.XXXX.1.2.1.1 6	lowpanIfOutCompReqds	54
1.3.6.1.2.1.XXXX.1.2.1.1 7	lowpanIfOutCompFails	55
1.3.6.1.2.1.XXXX.1.2.1.1 8	lowpanIfOutCompOKs	56
1.3.6.1.2.1.XXXX.1.2.1.1 9	lowpanIfOutFragReqds	57
1.3.6.1.2.1.XXXX.1.2.1.2 0	lowpanIfOutFragFails	58
1.3.6.1.2.1.XXXX.1.2.1.2 1	lowpanIfOutFragOKs	59
1.3.6.1.2.1.XXXX.1.2.1.2 2	lowpanIfOutFragCreates	60
1.3.6.1.2.1.XXXX.1.2.1.2 3	lowpanIfOutMeshHopLimitExceed s	61
1.3.6.1.2.1.XXXX.1.2.1.2 4	lowpanIfOutMeshNoRoutes	62
1.3.6.1.2.1.XXXX.1.2.1.2 5	lowpanIfOutMeshRequests	63

1.3.6.1.2.1.XXXX.1.2.1.2 6	lowpanIfOutMeshForwds	64
1.3.6.1.2.1.XXXX.1.2.1.2 7	lowpanIfOutMeshTransmits	65
1.3.6.1.2.1.XXXX.1.2.1.2 8	lowpanIfOutDiscards	66
1.3.6.1.2.1.XXXX.1.2.1.2 9	lowpanIfOutTransmits	67
1.3.6.1.2.1.XXXX.2.1	lowpanGroups	68
1.3.6.1.2.1.XXXX.2.1.1	lowpanStatsGroup	69
1.3.6.1.2.1.XXXX.2.1.2	lowpanStatsMeshGroup	70
1.3.6.1.2.1.XXXX.2.1.3	lowpanIfStatsGroup	71
1.3.6.1.2.1.XXXX.2.1.4	lowpanIfStatsMeshGroup	72
1.3.6.1.2.1.XXXX.2.2	lowpanCompliances	73
1.3.6.1.2.1.XXXX.2.2.1	lowpanCompliance	74

## Notes:

- o The IMPORTS "MODULE-IDENTITY", "OBJECT-TYPE", "Unsigned32", "Counter32", "OBJECT-GROUP" and "MODULE-COMPLIANCE" do not have associated Object IDs, and hence do not show up in the table.
- o The OBJECT-GROUPS lowpanStatsGroup, lowpanIfStatsGroup, lowpanIfStatsMeshGroup cannot appear in the CBOR instance, as they are lost in the intermediate MIB to YANG conversion step. However, as they have been defined in the original MIB, they still appear in the conversion table.
- o (FOR THE EDITOR:) the value XXXX is defined to be the RFC number once [I-D.ietf-6lo-lowpan-mib] becomes an RFC. It is logically greater than 7000.

## B.3. Example instance

A MIB for 6LoWPAN is defined in [I-D.ietf-6lo-lowpan-mib]. The document also provides an example JSON representation in its



Appendix A. Figure 3 shows the associated CBOR representation with string number, using the string numbers derived in Appendix B.2.

The request would have been as follows:

GET /mg/mib/lowpanIfStatsTable

The resulting table would be as follows:

```

82                                     # two element array
  78 18 4c 4f 57 50 41 4e
  2d 4d 49 42 5f 32 30 31
  34 30 34 30 38 30 30 30
  30 5a                               # conversion table id:
                                     # "LOWPAN-MIB_201404080000Z"
  BF                                  # indefinite length map
    18 25                             # "lowpanIfStatsTable"
    BF                                # indefinite length map related
                                     # to lowpanIfStatsTable
      18 27                           # "lowpanIfReasmTimeout"
      14                              # 20
      18 28                           # "lowpanIfInReceives"
      18 2A                           # 42
      18 29                           # "lowpanIfInHdrErrors"
      00                              # 0
      18 2A                           # "lowpanIfInMeshReceives"
      08                              # 8
      18 2B                           # "lowpanIfInMeshForwds"
      00                              # 0
      18 2C                           # "lowpanIfInMeshDelivers"
      00                              # 0
      18 2D                           # "lowpanIfInReasmReqds"
      16                              # 22
      18 2E                           # "lowpanIfInReasmFails"
      02                              # 02
      18 2F                           # "lowpanIfInReasmOKs"
      14                              # 20
      18 30                           # "lowpanIfInCompReqds"
      10                              # 16
      18 31                           # "lowpanIfInCompFails"
      02                              # 2
      18 32                           # "lowpanIfInCompOKs"
      0E                              # 14
      18 33                           # "lowpanIfInDiscards"
      01                              # 01
      18 34                           # "lowpanIfInDelivers"
      0C                              # 12
      18 35                           # "lowpanIfOutRequests"

```

```
0C          # 12
18 36       # "lowpanIfOutCompReqds"
00          # 0
18 37       # "lowpanIfOutCompFails"
00          # 0
18 38       # "lowpanIfOutCompOKs"
00          # 0
18 39       # "lowpanIfOutFragReqds"
05          # 5
18 3A       # "lowpanIfOutFragFails"
00          # 0
18 3B       # "lowpanIfOutFragOKs"
05          # 5
18 3C       # "lowpanIfOutFragCreates"
08          # 8
18 3D       # "lowpanIfOutMeshHopLimitExceeds"
00          # 0
18 3E       # "lowpanIfOutMeshNoRoutes"
00          # 0
18 3F       # "lowpanIfOutMeshRequests"
00          # 0
18 40       # "lowpanIfOutMeshForwds"
00          # 0
18 41       # "lowpanIfOutMeshTransmits"
00          # 0
18 42       # "lowpanIfOutDiscards"
00          # 0
18 43       # "lowpanIfOutTransmits"
0F          # 15
FF
FF
```

Figure 3: Example CBOR encoding for the 6LoWPAN MIB

## Authors' Addresses

Peter van der Stok  
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)  
Email: [consultancy@vanderstok.org](mailto:consultancy@vanderstok.org)  
URI: [www.vanderstok.org](http://www.vanderstok.org)

Bert Greevenbosch  
Huawei Technologies Co., Ltd.  
Huawei Industrial Base  
Bantian, Longgang District  
Shenzhen 518129  
P.R. China

Email: bert.greevenbosch@huawei.com