CoRE
Internet-Draft
Intended status: Standards Track
Expires: February 9, 2015

M. Becker, Ed.
ComNets, TZI, University Bremen
K. Li
Huawei Technologies
K. Kuladinithi
T. Poetsch
ComNets, TZI, University Bremen

August 8, 2014

# Transport of CoAP over SMS
# draft-becker-core-coap-sms-gprs-05

## Abstract

Short Message Service (SMS) of mobile cellular networks is frequently used in Machine-To-Machine (M2M) communications, such as for telematic devices. The service offers small packet sizes and high delays just as other typical low-power and lossy networks (LLNs), i.e. 6LoWPANs. The design of the Constrained Application Protocol (CoAP) [RFC7252], that took the limitations of LLNs into account, is thus also applicable to other transports. The adaptation of CoAP to SMS transport mechanisms is described in this document.

## Status of This Memo



## Copyright Notice

Table of Contents

1. Introduction . . . . . 2
1.1. Motivation . . . . . 3
2. Terminology . . . . . 3
3. Requirements Language . . . . . 3
4. Scenarios . . . . . 4
4.1. MO-MT Scenarios . . . . . 4
4.2. MT Scenarios . . . . . 4
4.3. MO Scenarios . . . . . 5
5. Message Exchanges . . . . . 6
5.1. Message Exchange for SMS in a Cellular-To-Cellular Mobile-Originated and Mobile-Terminated Scenario . . . . 6
6. Encoding Schemes of CoAP for SMS transport . . . . . 7
7. Message Size Implementation Considerations . . . . . 7
8. Addressing . . . . . 8
9. Options . . . . . 8
9.1. New Options for mixed IP operation. . . . . 8
10. URI Scheme . . . . . 9
11. Transmission Parameters . . . . . 9
12. Multicast . . . . . 9
13. Security Considerations . . . . . 9
14. IANA Considerations . . . . . 9
14.1. CoAP Option Number . . . . . 10
14.2. URI Scheme Registration . . . . . 10
15. Acknowledgements . . . . . 10
16. References . . . . . 10
16.1. Normative References . . . . . 10
16.2. Informative References . . . . . 11
Appendix A. Changelog . . . . . 12
Authors' Addresses . . . . . 13

## 1. Introduction

This specification details the usage of the Constrained Application Protocol on the Short Message Service (SMS) of mobile cellular networks.

### 1.1. Motivation

In some M2M environments, internet connectivity is not supported by the constrained end-points, but a cellular network connection is supported instead. Internet connectivity might also be switched off for power saving reasons or the cellular coverage does not allow for Internet connectivity. In these situations, SMS will be supported, instead of UDP/IP over General Packet Radio Service (GPRS), High Speed Packet Access (HSPA) or Long Term Evolution (LTE) networks.

In 3GPP, SMS is identified as the transport protocol for small data transmissions (See [ts23_888] for Key Issue on Machine Type Communication (MTC) Device Trigger and the proposed solutions in Sections 6.2, 6.42, 6.44, 6.48, 6.52, 6.60, and 6.61). In [ts23_682] 'Architecture Enhancements to facilitate communications with Packet Data Networks and Applications' SMS is at the moment the only Trigger Delivery (Trigger Delivery using T4).

M2M protocols using SMS, e.g. for telematics, are using mostly various diverse proprietary and closed binary protocols with limited publicly available documentation at the moment.

In Open Mobile Alliance (OMA) LightweightM2M technical specification [oma_lightweightm2m_ts], SMS is identified as an alternative transport for CoAP messages.

## 2. Terminology

This document uses the following terminology:

CoAP Server and Client
: The terms CoAP Server and CoAP Client are used synonymously to Server and Client as specified in the terminology section of [RFC7252].

Mobile Station (MS)
: A Mobile Station includes all required user equipment and software that is needed for communication with a mobile network. As defined in [etsi_ts101_748].

## 3. 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].

## 4. Scenarios

Several scenarios are presented first for M2M communications with CoAP over SMS. First Mobile-Originating Mobile-Terminating (MO-MT) scenarios are presented, where both CoAP endpoints are in devices in a cellular network. Next, Mobile-Terminating (MT) scenarios are detailed, where only the CoAP server is in a cellular network. Finally, Mobile-Originating (MO) scenarios where the CoAP client is in the cellular network.

### 4.1. MO-MT Scenarios

Two mobile cellular terminals communicate by exchanging a CoAP Request and Response embedded into short message protocol data units (PDUs) (depicted in Figure 1). Both terminals are connected via a Short Message Service Centre (SMS-C).

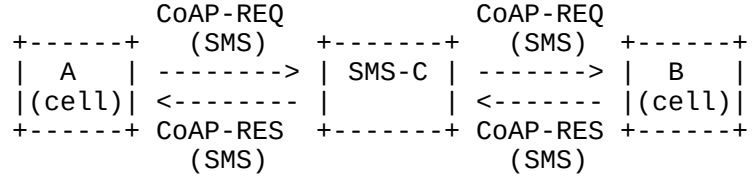



Figure 1: Cellular and Cellular Communication (only SMS-based)

### 4.2. MT Scenarios

An IP host and a mobile cellular terminal communicate by exchanging CoAP Request and Response. The IP host uses protocols offered by the SMS-C (e.g. Short Message Peer-to-Peer (SMPP [smpp]), Computer Interface to Message Distribution (CIMD [cimd]), Universal Computer Protocol/External Machine Interface (UCP/EMI [ucp])) to submit a short message for delivery, which contains the CoAP Request (depicted in Figure 2).

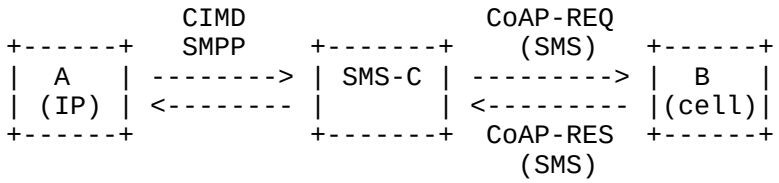



Figure 2: IP and Cellular Communication

There are service providers that offer SMS delivery and notification using an HTTP/REST interface (depicted in Figure 3).

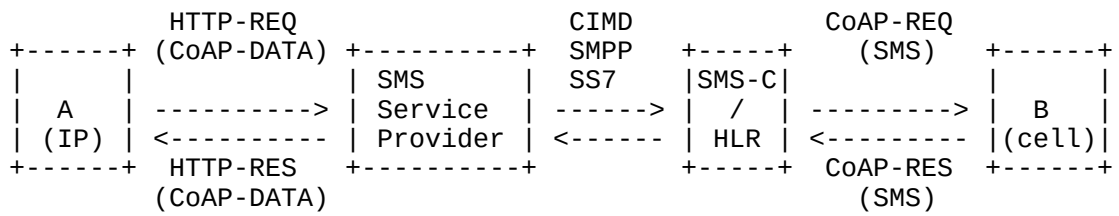



Figure 3: IP and Cellular Communication (using an SMS Service Provider)

## 4.3. MO Scenarios

A mobile cellular terminal and an IP host communicate by exchanging CoAP Request and Response. The mobile cellular terminal sends a CoAP Request in a short message, which is in turn forwarded by the SMS-C (e.g. with Short Message Peer-to-Peer (SMPP [smpp]), Computer Interface to Message Distribution (CIMD [cimd]), Universal Computer Protocol/External Machine Interface (UCP/EMI [ucp])) as depicted in Figure 4). This scenario can be a fall-back for mobile-originating communication, when IP connectivity cannot be setup (e.g. due to missing coverage).

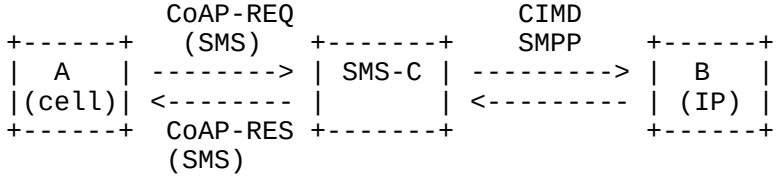



Figure 4: Cellular and IP Communication

There are service providers offering SMS delivery and notification using an HTTP/REST interface (depicted in Figure 5).

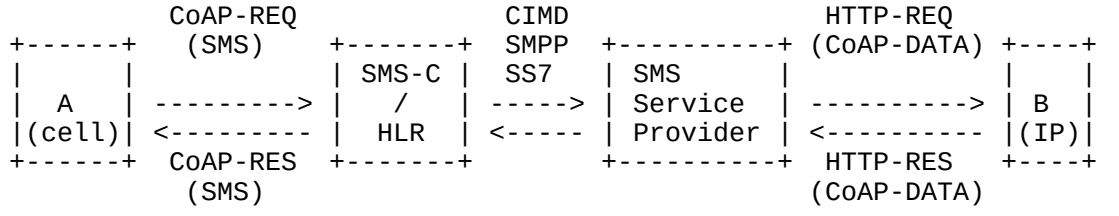



Figure 5: IP and Cellular Communication (using an SMS Service Provider)

## 5. Message Exchanges

### 5.1. Message Exchange for SMS in a Cellular-To-Cellular Mobile-Originated and Mobile-Terminated Scenario

The CoAP Client works as a Mobile Station to send the short message, and the CoAP Server works as another Mobile Station to receive the short message. All short messages are stored and forwarded by the Service Center. The message exchange between the CoAP Client and the CoAP Server is depicted in the figure below:

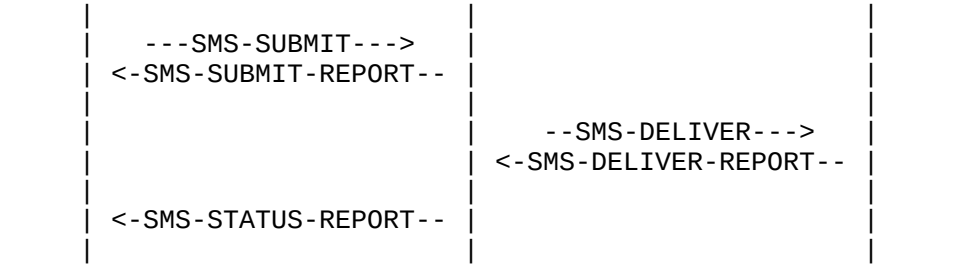



Figure 6: CoAP Messages over SMS

Note that the message exchange is just for one request message from CoAP Client and CoAP Server. It includes the following steps:

Step 1: The CoAP Client sends a CoAP request in a SMS-SUBMIT message to the Service Center. The CoAP Server address is specified as TP-Destination-Address (see [ts23_040]).

Step 2: The Service Center returns a SMS-SUBMIT-REPORT message to the CoAP Client.

Step 3: The Service Center stores the received SMS message and forwards it to the CoAP Server, using an SMS-DELIVER message. The CoAP Client address is specified as a TP Originating Address (see [ts23_040]).

Step 4: The CoAP Server returns an SMS-DELIVER-REPORT message to the Service Center.

Step 5: The Service Center returns the SMS-STATUS-REPORT message to the CoAP Client to indicate the SMS delivery status, if required by the CoAP Client.

Note that the SMS-STATUS-REPORT message just indicates the transport layer SMS delivery status and has no relationship with the confirmable message or non-confirmable message. If the CoAP Client has sent a confirmable message, the CoAP Server MUST use a separate SMS message to transmit the ACK.

## 6. Encoding Schemes of CoAP for SMS transport

Short messages can be encoded by using various alphabets: GSM 7 bit default alphabet ([ts23_038]), 8 bit data alphabet, and 16 bit USC2 data alphabet ([iso_ucs2]). These encodings lead to message sizes of 160, 140, and 70 characters, respectively. Whereas the support of 7 bit encoding is mandatory on a MS, the two other encodings are dependent on the language that needs to be encoded, e.g. USC2 for Arabic, Chinese, Japanese, etc. Furthermore, the supported encoding highly depends on the implementations of the MS itself.

According to [ts23_038], GSM 7 bit encoding shall be supported by all MSs offering SMS services. Since not all MSs support 8 bit short message encoding, the prefered encoding scheme for CoAP messages over SMS is therefore 7 bit, e.g. Base64 ([RFC4648]) or SMS encoding in [I-D.bormann-coap-misc].

More considerations about SMS encoding can be found in [I-D.bormann-coap-misc].

## 7. Message Size Implementation Considerations

By using 7 bit encoding, a maximum length of 160 characters is allowed in one short message [ts23_038]. Consequently, the maximum length for a CoAP message results in 140 bytes. 160 characters = (140 bytes * 8)/7.

Possible options for larger CoAP messages are:

Concatenated short messages
: Most MSs are able to send concatenation short messages in order to tranmsit longer messages. The total length of a concatenated short message can consist of up to 255 single messages and result in total length of 39015 7 bit characters or 34170 bytes. Resulting from this, the maximum length of each individual message reduces to 153 (160 - 7) characters (133 bytes).

CoAP blockwise transfer
: According to [I-D.ietf-core-block], the Block Size (SZX) of blockwise transfer in CoAP is represented as a three-bit unsigned integer. Thus, the possible block sizes are to the power of two. (Block size = 2**(SZX + 4)). Due to the limitations of 160

characters (140 bytes) for one short message, the maximum value of SZX is 3 (Block size = 128 byte).

However, it is RECOMMENDED that SMS is not used to transfer very large resource data using blockwise transfer.

## 8. Addressing

For SMS in cellular networks, the CoAP endpoints have to work with a SIM (Subscriber Identity Module) card and have to be addressed by the MSISDN (Mobile Station ISDN (MSISDN) number).

To allow the CoAP client to detect that the short message contains a CoAP message, the TP-DATA-Coding-Scheme SHOULD be included.

## 9. Options

### 9.1. New Options for mixed IP operation.

In case a CoAP Server has more than one network interface, e.g. SMS and IP, the CoAP Client might want the server to send the response via an alternative transport, i.e. to it's alternative address. However, that implies that the initiating CoAP Client is aware of the presence of the alternative interface. For this reason the new options Response-To-Uri-Host and Response-To-Uri-Port are proposed.

| No. | C | U | N | R | Name | Format | Length | Default |
|---|---|---|---|---|---|---|---|---|
| TBD | | | | | Response-To-Uri-Host | string | 1-270 B | (none) |
| TBD | | | | | Response-To-Uri-Port | uint | 0-2 B | 5683 |

Table 1: New CoAP Option Numbers

If the Response-To-Uri-Host is present in the request, server MUST send the response to the indicated URI address, instead of the client's original request URI.

The options SHOULD NOT be used in the response.

The options MUST NOT occur more than once.

## 10. URI Scheme

The coap:// scheme defines that a CoAP server is reachable over UDP/IP. Hence, a new URI scheme is needed for CoAP servers which are reachable over SMS.

As specified in [I-D.silverajan-core-coap-alternative-transports], the transport information is expressed as part of the URI scheme component. This is performed by minting new schemes for SMS transport using the form "coap+sms", where the name of the transport is clearly and unambiguously described. The endpoint identifier, path and query components together with each scheme name would be used to uniquely identify each resource.

Example of such URI :

o coap+sms://0015105550101/sensors/temperature

In the URI, 0015105550101 is a telephone subscriber number.

## 11. Transmission Parameters

It is RECOMMENDED to configure the RESPONSE_TIMEOUT variable for a higher duration than specified in [RFC7252] for the applications described here. The actual value SHOULD be chosen based on experience with SMS .

## 12. Multicast

Multicast is not possible with SMS transports.

## 13. Security Considerations

It is possible that a malicious CoAP Client sends repeated requests, and it may cost money for the CoAP Server to use SMS to send back associated responses. To avoid this situation, the CoAP Server implementation can authenticate the CoAP Client before responding to the requests. For example, the CoAP Server can maintain an MSISDN white list. Only the MSISDN specified in the white list will be allowed to send requests. The requests from others will be ignored or rejected.

## 14. IANA Considerations

### 14.1. CoAP Option Number

The IANA is requested to add the following option number entries to the CoAP Option Number Registry:

| Number | Name | Reference |
|---|---|---|
| TBD | Response-To-Uri-Host | Section 2 of this document |
| TBD | Response-To-Uri-Port | Section 2 of this document |

### 14.2. URI Scheme Registration

According to [I-D.silverajan-core-coap-alternative-transports] this document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+alt". The registration request complies with [RFC4395].

## 15. Acknowledgements

This document is partly based on research for the research project 'The Intelligent Container' which is supported by the Federal Ministry of Education and Research, Germany, under reference number 01IA10001.

The authors of this draft would like to thank Bert Greevenbosch, Marcus Goetting, Nils Schulte and Klaus Hartke for the discussions on the topic and the reviews of this document.

## 16. References

### 16.1. Normative 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.

[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.silverajan-core-coap-alternative-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-06 (work in progress), July 2014.

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

[RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, December 2004.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

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

[etsi_ts101_748]
ETSI, "Technical Report: Digital cellular telecommunications system; Abbreviations and acronyms (GSM 01.04 version 8.0.0 release 1999)", 2000.

[iso_ucs2]
ISO, "ISO/IEC10646: "Universal Multiple-Octet Coded Character Set (UCS)"; UCS2, 16 bit coding.", 2000.

[ts23_038]
ETSI 3GPP, "Technical Specification: Alphabets and language-specific information (3GPP TS 23.038 version 11.0.0 Release 11)", 2012.

## 16.2. Informative References

[cimd] Nokia, "CIMD Interface Specification (SMSCDOC8000.00, Nokia SMS Center 8.0)", 2005.

[oma_lightweightm2m_ts]
OMA, "Lightweight Machine to Machine Technical Specification", 2013.

[smpp] SMPP Developers Forum, "Short Message Peer to Peer Protocol Specification v3.4 Issue 1.2", 1999.

[ts23_040]
3GPP, "Technical realization of the Short Message Service (SMS)", 3GPP-23.040 a00, Mar 2011.

[ts23_682] ETSI 3GPP, "Technical Specification Group Services and System Aspects; Architecture Enhancements to facilitate communications with Packet Data Networks and Applications; (Release 11)", 2012.

[ts23_888] ETSI 3GPP, "Technical Specification Group Services and System Aspects; System Improvements for Machine-Type Communications; (3GPP TR 23.888 version 1.6.0, Release 11)", 2011.

[ucp] Vodafone, "Short Message Service Centre (SMSC) External Machine Interface (EMI) Description Version 4.3d", 2011.

## Appendix A. Changelog

Changed from draft-04 to draft-05:

- Removed reference to USSD.
- Updated reference to RFC7252 and 3GPP specs.
- Updated Options.
- Adapted URI scheme.

Changed from draft-03 to draft-04:

- Removed USSD and GPRS related parts.
- Removed section 5: Examples
- Removed section 14: Proxying Considerations
- Added more block size considerations.
- Added more concatenated SMS considerations.
- Rewrote encoding scheme section; 7 bit encoding only.

Changed from draft-02 to draft-03:

- Added reference to OMA LightweightM2M Technical Specification in "Motivation" section.
- Chose CoAP option numbers and updated the option number table to meet draft-ietf-core-coap-13.

Changed from draft-01 to draft-02:

- Added security considerations: Transport and Object Security. Section 13
- Reply-To-* changed to Response-To-*. Section 14
- Added URI scheme.
- Added possible CON/NON/ACK interactions.
- Added possible M2M proxy scenarios.
- Added reference to bormann-coap-misc for other SMS encoding. Section 6
- Updated requirements on Uri-Host and Uri-Port for coap+tel://.
- Chose CoAP option numbers and updated the option number table to meet draft-ietf-core-coap-10. />
- Added an IANA registration for the URI scheme. Section 14.2

Authors' Addresses

Markus Becker (editor)
ComNets, TZI, University Bremen
Bibliothekstrasse 1
Bremen 28359
Germany

Phone: +49 421 218 62379
Email: mab@comnets.uni-bremen.de

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

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

Koojana Kuladinithi
ComNets, TZI, University Bremen
Bibliothekstrasse 1
Bremen 28359
Germany

Phone: +49 421 218 62382
Email: koo@comnets.uni-bremen.de

Thomas Poetsch
ComNets, TZI, University Bremen
Bibliothekstrasse 1
Bremen 28359
Germany

Phone: +49 421 218 62379
Email: thp@comnets.uni-bremen.de

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

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:

- 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:

- 0xF0 would indicate that a zero-length (absent) payload follows
- 0xF1 would indicate a single-byte payload
- 0xFD 0x47 would indicate a 84-byte payload
- 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-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-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

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

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:

RTO_overall_ := 0.25 * RTO_weak_ + 0.75 * RTO_overall_ (1)

RTO_overall_ := 0.5 * RTO_strong_ + 0.5 * RTO_overall_ (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 x RTO and ACK_RANDOM_FACTOR x 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

```
RTO_overall_ := 1 s + (0.5 * 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/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/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 Beroset, 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

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:

- 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.

- 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.

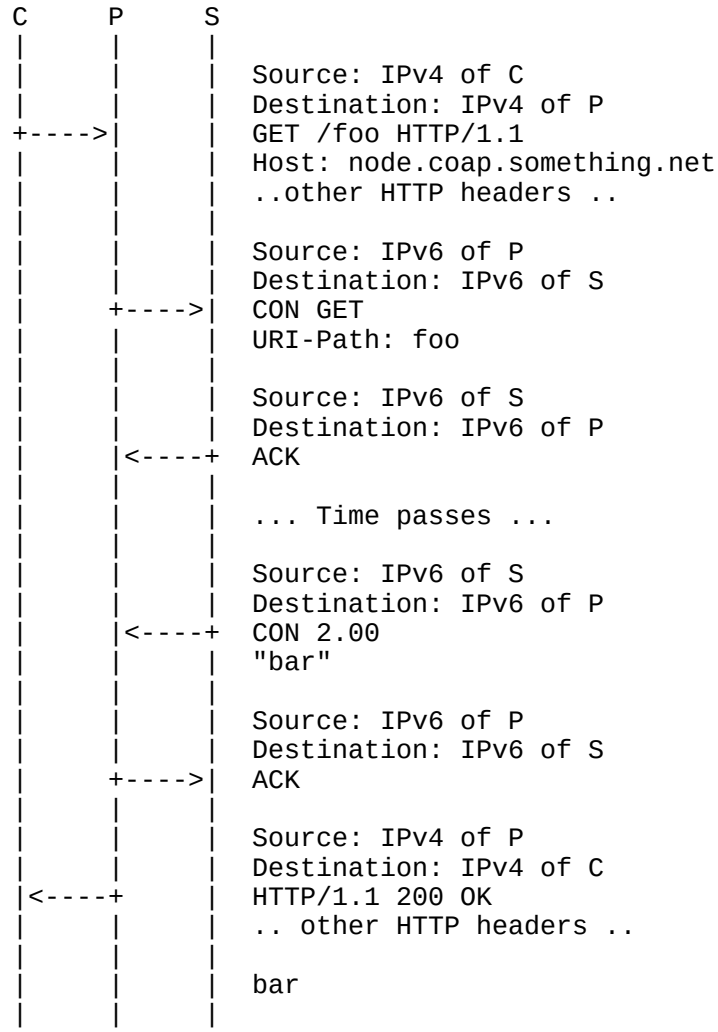



Figure 1: HTTP/IPv4 to CoAP/IPv6 Mapping

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.

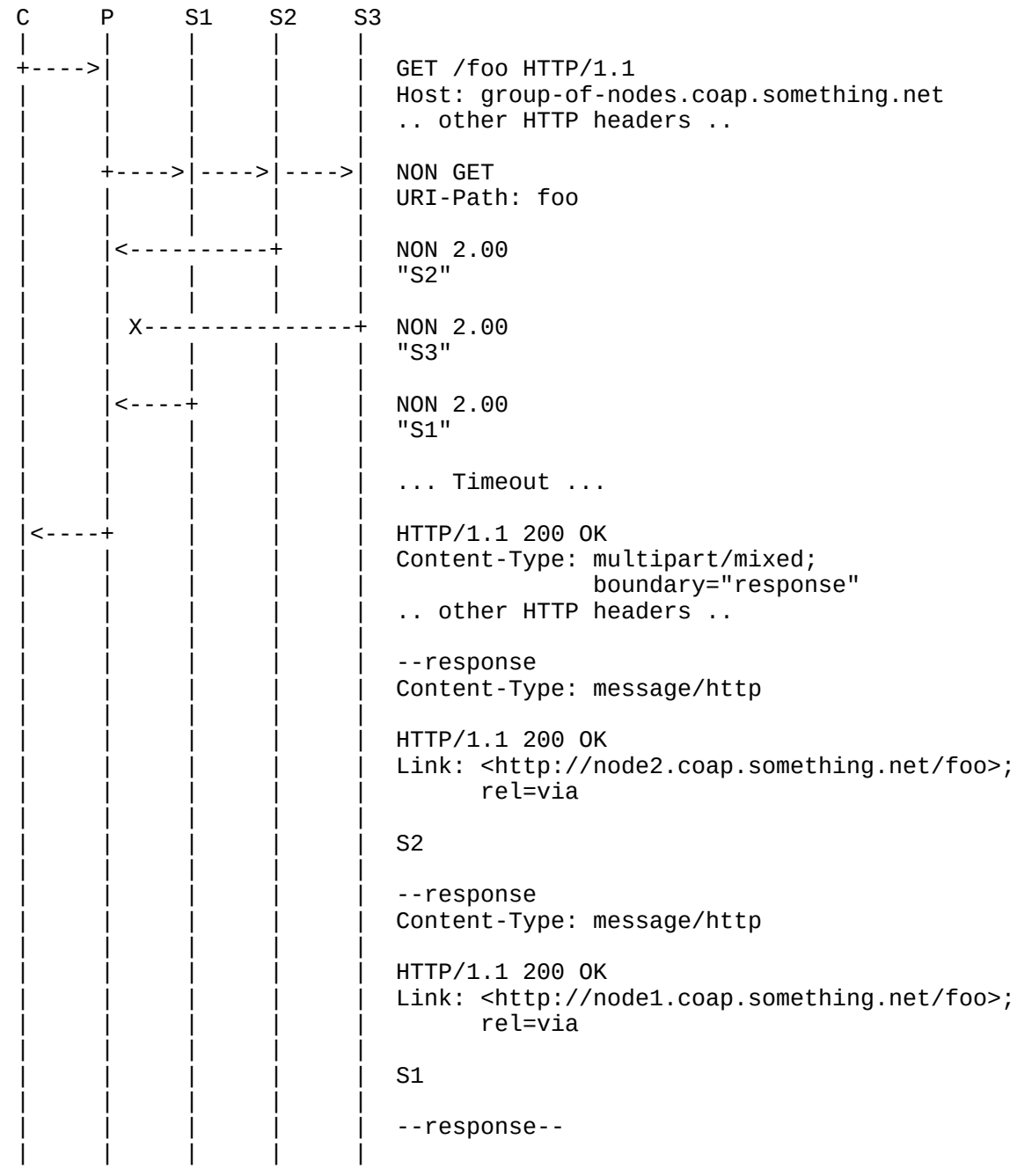



Figure 2: Unicast HTTP to Multicast CoAP Mapping

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]).

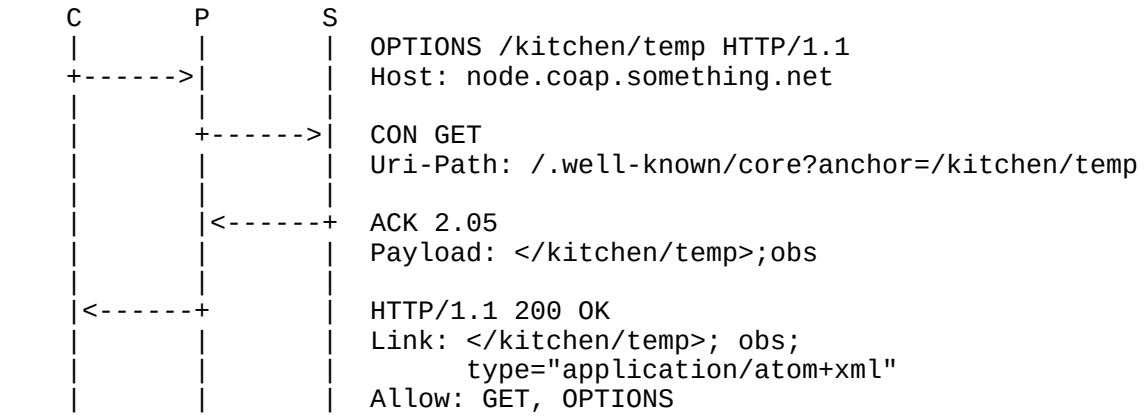



Figure 3: Discover Observability with HTTP OPTIONS

#### 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.

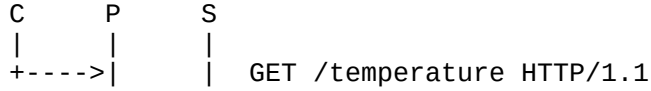
Figure 3 shows a resource supporting ’application/atom+xml’ media-type. In such case clients can listen to update notification by regularly polling the resource via opportunely spaced GETs, i.e. driven by the advertised max-age value.

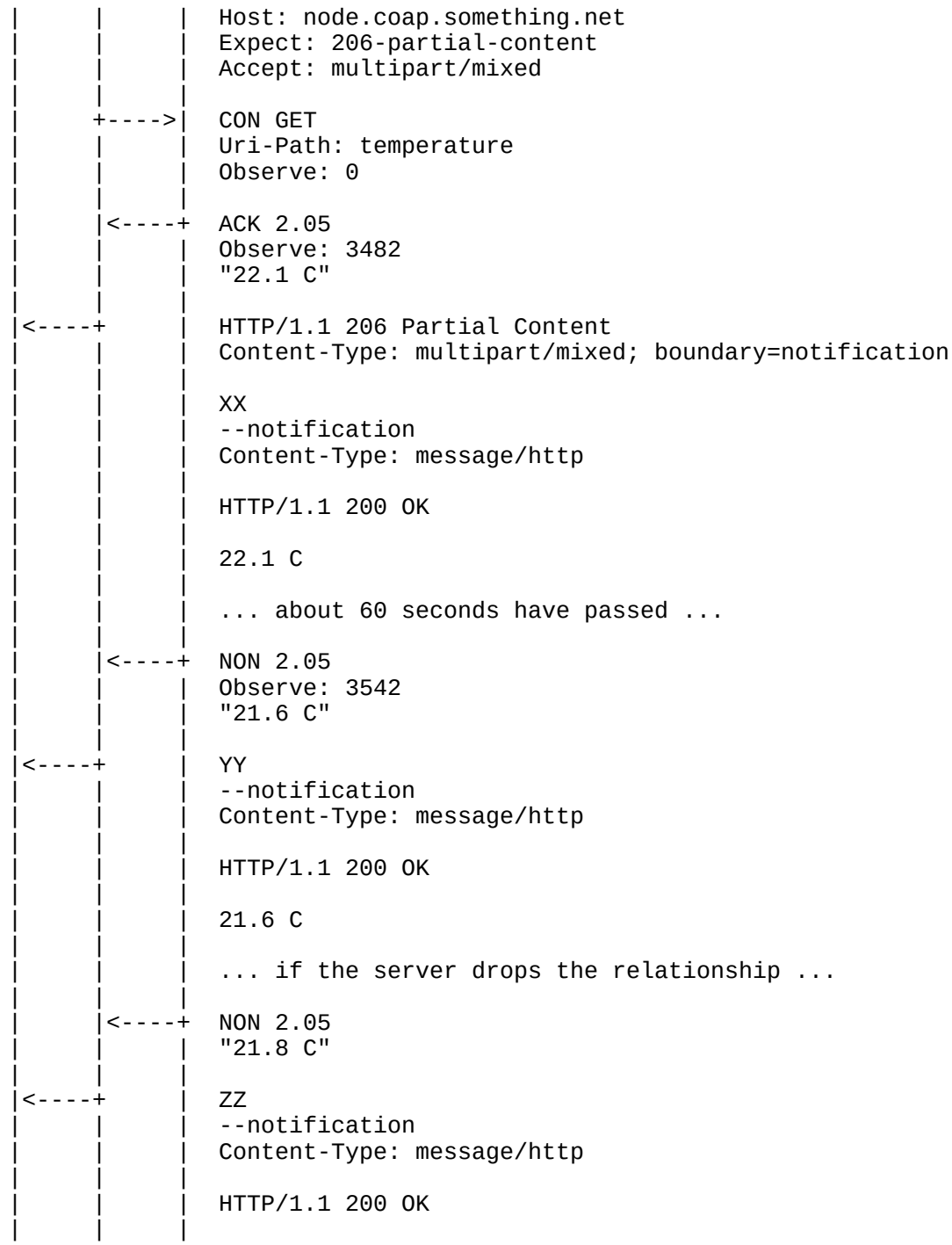
### 5.4.3. Examples

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.




Host: node.coap.something.net
Expect: 206-partial-content
Accept: multipart/mixed
CON GET
Uri-Path: temperature
Observe: 0
ACK 2.05
Observe: 3482
"22.1 C"
HTTP/1.1 206 Partial Content
Content-Type: multipart/mixed; boundary=notification
XX
--notification
Content-Type: message/http
HTTP/1.1 200 OK
22.1 C
... about 60 seconds have passed ...
NON 2.05
Observe: 3542
"21.6 C"
YY
--notification
Content-Type: message/http
HTTP/1.1 200 OK
21.6 C
... if the server drops the relationship ...
NON 2.05
"21.8 C"
ZZ
--notification
Content-Type: message/http
HTTP/1.1 200 OK

```
|     |     |  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.

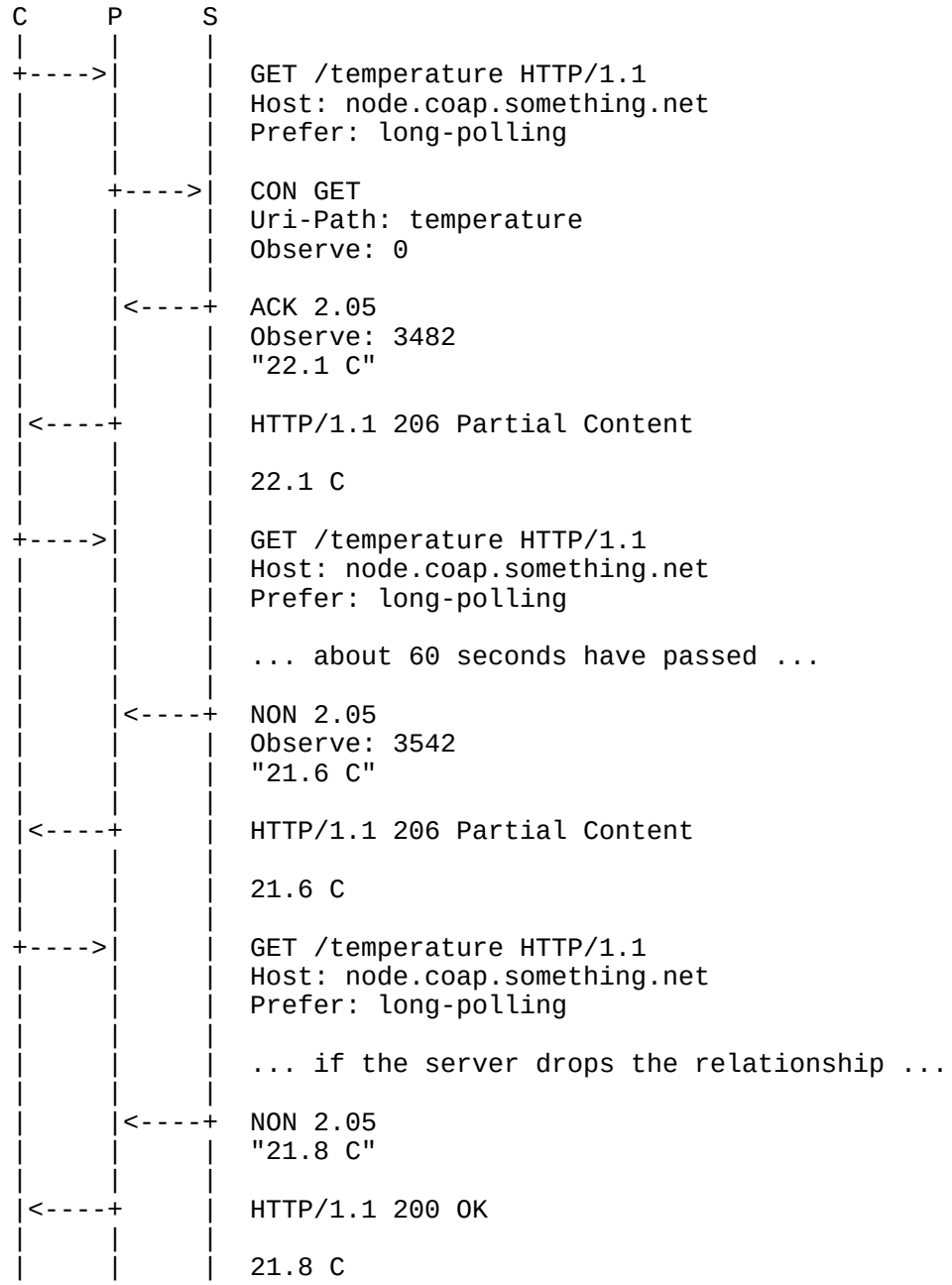



Figure 5: HTTP Long Polling to CoAP Observe

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.

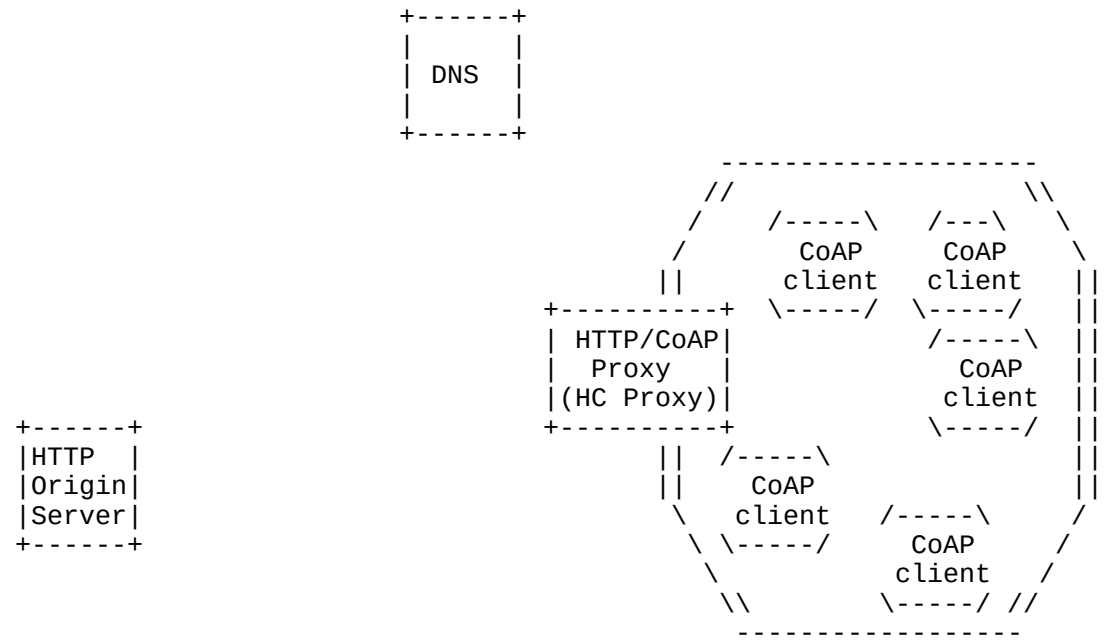



Figure 7: Client-side HC Proxy Deployment Scenario

## 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.

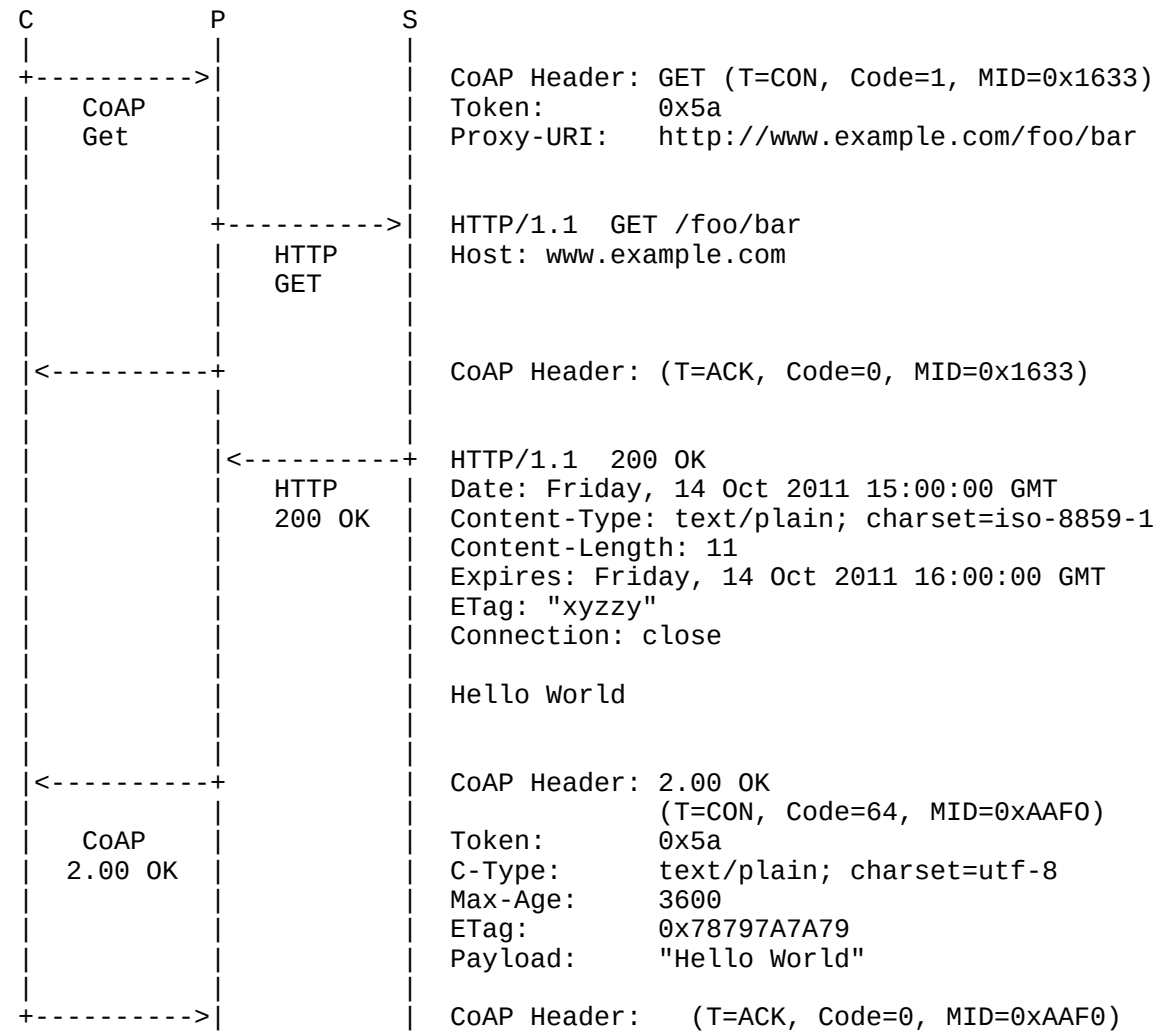



Figure 8: A Basic CoAP-HTTP GET Request

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.

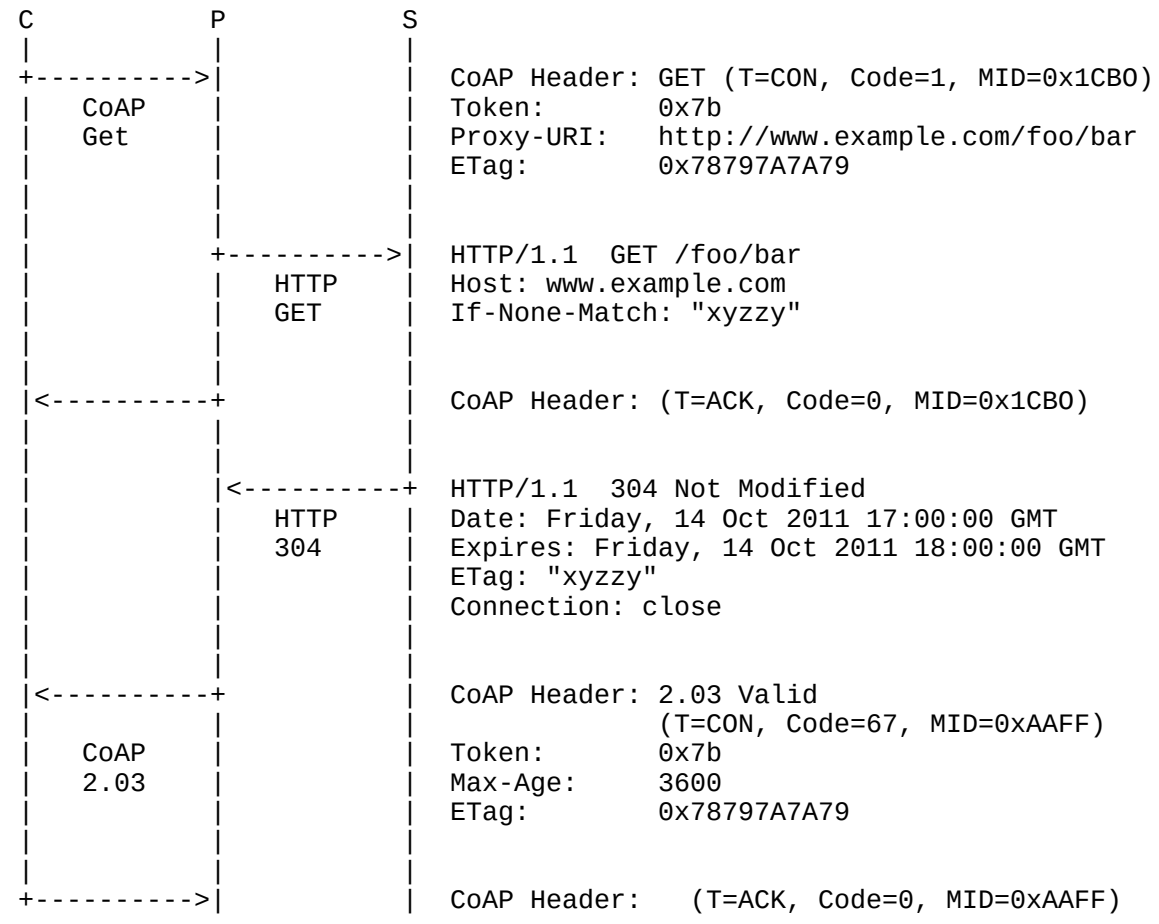



Figure 9: A CoAP-HTTP GET Request with an ETag Option

## 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 todays web components (e.g. Django URL dispatcher, Apache mod_rewrite, etc.), which provides a compact and powerful engine to implement what essentially is an URL rewrite function.

```
INPUT
    * requested URL

OUTPUT
    * target URL

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

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

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

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

### A.2. Security Policy Map Algorithm

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

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

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

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

   EXAMPLE
       <TBD>
```

### A.3. Content-Type Map Algorithm

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

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

   OUTPUT
       * mapped content-type

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

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

Authors' Addresses

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

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

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

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

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

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

Esko Dijk
Philips Research

Email: esko.dijk@philips.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 16, 2015

T. Fossati
Alcatel-Lucent
H. Tschofenig
ARM Ltd.

October 13, 2014

# Datagram Transport Layer Security (DTLS) over Global System for Mobile Communications (GSM) Short Message Service (SMS)
## draft-fossati-dtls-over-gsm-sms-01

## Abstract

This document specifies the use of Datagram Transport Layer Security (DTLS) over the Global System for Mobile Communications (GSM) Short Message Service (SMS).

## Status of This Memo

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

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

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

This Internet-Draft will expire on April 16, 2015.

## Copyright Notice

Table of Contents

1. Introduction . . . . . 2
1.1. Usage of DTLS and SMS in CoAP M2M Environments . . . . . 2
2. Terminology . . . . . 3
3. DTLS over SMS . . . . . 3
3.1. Data Coding Scheme . . . . . 3
3.2. Handshake Overview . . . . . 3
3.2.1. X.509 Certificate-based Authentication Caveats . . . 4
3.3. Message Segmentation and Re-Assembly . . . . . 4
3.4. DTLS State Machine Timers Adjustments . . . . . 5
3.5. Multiplexing Security Associations . . . . . 6
4. New Versions of DTLS . . . . . 6
5. Security Considerations . . . . . 7
6. Acknowledgements . . . . . 7
7. IANA Considerations . . . . . 7
8. References . . . . . 7
8.1. Normative References . . . . . 7
8.2. Informative References . . . . . 8
Authors' Addresses . . . . . 8

## 1. Introduction

This document specifies the use of DTLS [RFC6347] over GSM SMS [GSM-SMS] for securing end-to-end communication between Mobile Stations (i.e. devices implementing the GSM SMS communication standard).

DTLS provides communications privacy for applications that use datagram transport protocols and allows client/server applications to communicate in a way that is designed to prevent eavesdropping and detect tampering or message forgery.

SMS is a generic transport protocol for narrow-band end-to-end communication between devices, and is an integral part of the GSM network technology.

### 1.1. Usage of DTLS and SMS in CoAP M2M Environments

One of the main reasons for defining a DTLS/SMS binding is its envisaged usage in machine-to-machine (M2M) communication.

Specifically, M2M environments based on the CoAP protocol mandate DTLS for securing transactions between endpoints -- as detailed in Section 9 of [RFC7252], and further articulated in [I-D.ietf-dice-profile], while the [OMA-LWM2M] architecture identifies SMS as an alternative transport for CoAP messages.

## 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].

This specification requires readers to be familiar with all the terms and concepts that are described in [GSM-SMS], [WAP-WDP], [RFC5246], and [RFC6347].

## 3. DTLS over SMS

### 3.1. Data Coding Scheme

The remainder of this specification assumes Mobile Stations are capable of producing and consuming 8-bit binary data encoded Transport Protocol Data Units (TPDU).

### 3.2. Handshake Overview

DTLS adds an additional roundtrip to the TLS [RFC5246] handshake to serve as a return-routability test for protection against certain types of DoS attacks. Thus a full blown DTLS handshake comprises up to 6 "flights" (i.e. logical message exchanges), each of which is then mapped on to one or more lower layer PDUs using the segmentation and reassembly (SaR) scheme described in Section 4.2.3 of [RFC6347]. The overhead for said scheme is 6 bytes per Handshake message which, given a realistic 10+ messages handshake, would amount around 60 bytes across the whole handshake sequence.

(Note that the DTLS SaR scheme is defined for handshake messages only. In fact, Record Layer messages are never fragmented and MUST fit within a single transport layer datagram, whatever be the limit imposed by the underlying transport.)

SMS provides an optional segmentation and reassembly scheme as well, known as Concatenated short messages (see Section 9.2.3.24.1 of [GSM-SMS]). However, since the SaR scheme in DTLS can’t be circumvented, the Concatenated short messages mechanism SHOULD NOT be used during handshake to avoid redundant overhead. Before starting the handshake phase (either actively or passively), the DTLS implementation MUST be explicitly configured with the PMTU of the SMS transport in order to correctly instrument its SaR function. The PMTU SHALL be 133 bytes if WDP-based multiplexing is used (see Section 3.5), 140 bytes otherwise.

It is RECOMMENDED to use the established security context over the longest possible period (possibly until a Closure Alert message is

received, or after a very long inactivity timeout) to avoid the expensive re-establishment of the security association.

#### 3.2.1. X.509 Certificate-based Authentication Caveats

X.509 certificate-based authentication (used in Certificate mode CoAP) exacerbates the number of TPDUs -- especially those involved in flight 4 and 5 -- needed to complete the handshake phase.

In such case, given the typical latency of the SMS transport, the time to finalise the handshake could be in the order of 10s of seconds (maybe even minutes).

More importantly, the large number of TPDUs involved increases the likelihood to incur packet loss which DTLS does not handle efficiently. In fact, the DTLS timeout and retransmission logics apply to whole flights, but not to message fragments individually. So, loss or delay of a single fragment may disrupt the current flight, which needs to be entirely retransmitted.

Depending on the delay and packet loss characteristics of the network link, completing a DTLS handshake which involves exchanging X.509 data may prove to be a daunting task [[CREF1: TODO: substantiate with figures]].

For these reasons, it is advisable to carefully consider whether the use of X.509 certificate-based authentication is compatible with the characteristics of the network link between the involved parties.

### 3.3. Message Segmentation and Re-Assembly

[RFC6347] requires that each DTLS message fits within a single transport layer datagram

The content of an SMS message is carried in the TP-UserData field, and its size may be up to 140 bytes. As already mentioned in Section 3.2, longer (i.e. up to 34170 bytes) messages can be sent using a segmentation and reassembly scheme known as Concatenated SMS (see Section 9.2.3.24.1 of [GSM-SMS]).

This scheme consumes 6-7 bytes (depending on whether the short or long segmentation format is used) of the TP-UserData field, thus reducing the space available for the actual content of the SMS message to 133-134 bytes per TPDU.

Though in principle a PMTU value higher than 140 bytes could be used (which may look like an appealing option given its more efficient use of the transport) there is a significant number of disadvantages to

consider (apart from the fixed tax of 7 bytes per TPDU to be paid to the SaR function):

1. high sensitivity to packet loss -- since there is no automatic recovery mechanism in case one TPDU in the chain is lost, and since the SaR function is transparent to the application layer, then a PMTU worth of data may be discarded even if just 1/255th of the data were lost;

2. some networks may support the Concatenated SMS function partially, if at all;

3. TPDU reordering may delay data delivery to the application;

4. high buffering requirement on both ends of the communication path.

For these reasons, the Concatenated short messages mechanism SHOULD NOT be used, and it is RECOMMENDED to leave the same PMTU settings used during the handshake phase (Section 3.2), i.e. 133 bytes if WDP-based multiplexing is enabled (Section 3.5), 140 bytes otherwise.

Note that, after DTLS handshake has completed, any fragmentation and reassembly logics that pertains the application layer - e.g. segmenting CoAP messages into DTLS records and reassembling them after the crypto operations have been successfully performed - needs to be handled by the application that uses the established DTLS tunnel.

### 3.4. DTLS State Machine Timers Adjustments

[RFC6347] recommends an initial timer value of 1 second with exponential back off up to no less then 60 seconds. Given the latency characteristics of typical SMS delivery, the 1 second value can easily lead to spurious retransmissions, which in turn may lead to link congestion.

Choosing an appropriate timer value is a difficult problem due to the wide variance in latency in SMS delivery. This specification RECOMMENDS an initial timer value of 10 seconds with exponential back off up to no less then 60 seconds.

If SMS-STATUS-REPORT messages are enabled, their receipt is not to be interpreted as the signal that the specific handshake message has been acted upon by the receiving party. Therefore, it MUST NOT be taken into account by the DTLS timeout and retransmission function.

Handshake messages MUST carry a validity period (TP-VP parameter in a SMS-SUBMIT TPDU) that is not less than the current value of the retransmission timeout. In order to avoid persisting messages in the network that will be discarded by the receiving party, handshake messages SHOULD carry a validity period that is the same as, or just slightly higher than, the current value of the retransmission timeout.

If an RTT estimator (e.g. [I-D.bormann-core-cocoa]) is already available in the protocol stack of the device, it could be used to dynamically update the setting of the retransmit timeout.

### 3.5. Multiplexing Security Associations

Unlike IPsec, DTLS records do not contain any association identifiers. Applications must arrange to multiplex between associations on the same endpoint which, when using UDP/IP, is usually done with the host/port number.

If the DTLS server allows more than one client to be active at any given time, then the WAP User Datagram Protocol [WAP-WDP] can be used to achieve multiplexing of the different security associations. (The use of WDP provides the additional benefit that upper layer protocols can operate independently of the underlying wireless network, hence achieving application-agnostic transport handover.)

The total overhead cost for encoding the WDP source and destination ports is 7 bytes out of the total available for the SMS content.

The receiving side of the communication gets the source address from the originator address (TP-OA) field of the SMS-DELIVER TPDU. This way an unique 4-tuple identifying the security association can be reconstructed at both ends. (When replying to its DTLS peer, the sender will swaps the TP-OA and TP-DA parameters and the source and destination ports in the WDP.)

## 4. New Versions of DTLS

As DTLS matures, revisions to and updates for [RFC6347] can be expected. DTLS includes mechanisms for identifying the version in use, and presumably future versions will either include backward compatibility modes or at least not allow connections between dissimilar versions. Since DTLS over SMS simply encapsulates the DTLS records transparently, these changes should not affect this document and the methods of this document should apply to future versions of DTLS.

Therefore, in the absence of a revision to this document, this document is assumed to apply to all future versions of DTLS. This document will only be revised if a revision to DTLS or SMS makes a revision to the encapsulation necessary.

## 5. Security Considerations

Security considerations for DTLS as specified in [RFC6347] apply.

In most networks, sending SMS messages is not a free service, therefore DoS attacks tend to be a lot less common than in IP networks. However, it is RECOMMENDED not to disable the cookie exchange protection, unless the involved risks are fully understood, and the chance of a DoS attack is deemed low enough to drop the protection mechanism in order to save one round-trip per handshake.

DTLS lays on top of SMS, and therefore it doesn’t provide any security service to it. The SMS implementation must be able to protect itself from any special SMS message that can be used to alter the device state or configuration in an undesired way (e.g. fiddling with the private key store). Any SMS client must make sure that malicious use of such messages is not possible, for example, by filtering out certain SMS User Data header fields.

The layering of DTLS on top of the SMS transport does not introduce any new security issues. We believe that the recommendations contained in this specification (i.e. initial RTO increase, use of WDP for multiplexing security associations, avoidance of SMS SaR) have no impact on the security of DTLS.

## 6. Acknowledgements

Thanks to Tim Carey, Thierry Garnier, Zhiyuan Hu, Kathleen Moriarty, Eric Rescorla, Padmakumar Subramani, for helpful comments and discussions that have shaped this document.

## 7. IANA Considerations

This specification contains no request to IANA.

## 8. References

### 8.1. Normative References

[GSM-SMS] ETSI, "3GPP TS 23.040 V7.0.1 (2007-03): 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Technical realization of the Short Message Service (SMS) (Release 7)", March 2007.

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

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

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[WAP-WDP] Wireless Application Protocol Forum, "Wireless Datagram Protocol", June 2001.

## 8.2. Informative References

[I-D.bormann-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-bormann-core-cocoa-02 (work in progress), July 2014.

[I-D.ietf-dice-profile]
Tschofenig, H., "A Datagram Transport Layer Security (DTLS) 1.2 Profile for the Internet of Things", draft-ietf-dice-profile-04 (work in progress), August 2014.

[OMA-LWM2M]
OMA, "Lightweight Machine to Machine Technical Specification", 2013.

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

## Authors' Addresses

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

Email: thomas.fossati@alcatel-lucent.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
UK

Email: hannes.tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

ACE
Internet-Draft
Intended status: Informational
Expires: April 30, 2015

B. Greevenbosch
Huawei Technologies

October 27, 2014

# ACE for resource directory
# draft-greevenbosch-ace-resource-directory-00

## Abstract

This document describes how ACE can be used to protect the resource directory and allow update and registration to authorised parties only.

## Status of this Memo

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

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

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

This Internet-Draft will expire on April 30, 2015.

## Copyright Notice

Table of Contents

1. Requirements notation . . . . . 3
2. Introduction . . . . . 4
3. Specifics for the Resource Directory resource . . . . . 5
4. Registration of Resource Server to Resource Directory . . . . 7
5. Querying the RD . . . . . 8
6. Architecture . . . . . 9
7. Solution . . . . . 10
8. Security considerations . . . . . 11
9. IANA considerations . . . . . 12
10. References . . . . . 13
10.1. Normative References . . . . . 13
10.2. Informative References . . . . . 13
Author's Address . . . . . 14

## 1. Requirements notation

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

## 2. Introduction

The document [I-D.ietf-core-resource-directory] defines a resource directory (RD) for Constrained Devices. The RD is used servers to advertise themselves, and by clients to discover those servers.

Thus, information in a resource directory is quite valuable. It should therefore be ensured that only authorised parties can access the RD. Especially, servers that have registered themselves with the RD should only be allowed to update their own registrations, and there could be registrictions on which servers can register themselves.

Similarly, there may be a requirement to only provide information in the RD to authorised clients, for example when the RD is in a private domain, such as in the case of home automation.

This document explores the authorisation and authentication issues associated with resource directories, and describes about how the authentication and authorisation for constrained environments (ACE) work can be applied to protect the RD.

It is the intention to treat the RD as a normal CoAP endpoint as much as possible, thereby providing a testdrive for specifications currently being developed in the ACE working group.

## 3. Specifics for the Resource Directory resource

Figure 1 provides a high level overview of a typical RD deployment. The RD is implemented as a CoAP server, whereas CoAP endpoints that query the RD use their CoAP client functionality.

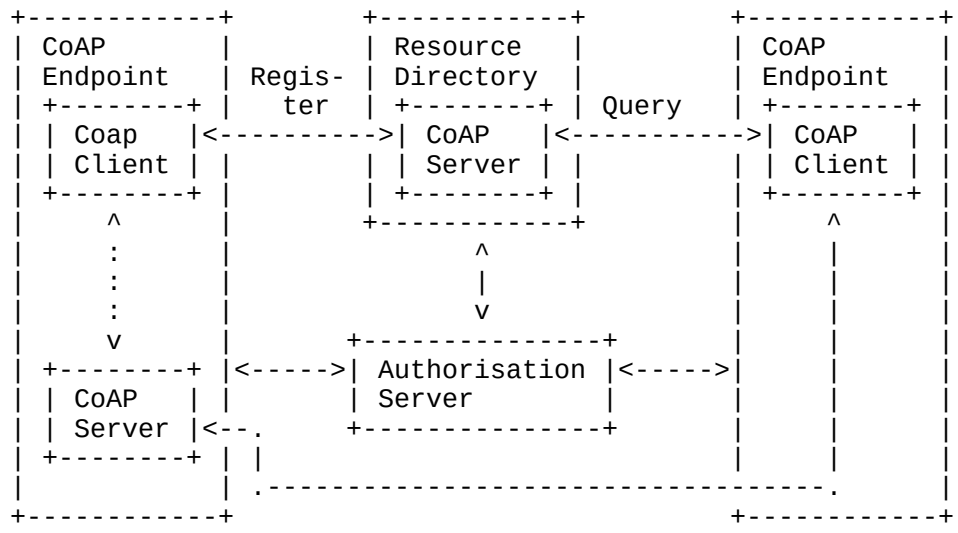



Figure 1: Resource Directory Deployment

In the CoAP architecture, a sensor or actuator is normally implemented as a CoAP server, whereas the endpoint needing to access such sensor or actuator implements a CoAP client. Although not strictly necessary, this could lead to the assumption that in most cases the CoAP server would be the most constrained endpoint. In the RD case, however, the RD is implemented as a CoAP server, but can be expected to be little constrained.

The fact that the RD is implemented as a CoAP server leads to the following, highly similar, requirements:

REQ-1 The RD should be able to perform authentication and authorisation from a CoAP server’s point of view.

REQ-2 The RD should be able to authenticate and verify authorisation of CoAP clients.

To access an RD, a CoAP endpoint should contain CoAP client functionality. This may or may not be feasible for constrained devices. Hence those constrained devices may need to delegate certain tasks to other endpoints, which implement CoAP client

functionality. Those CoAP clients should act on themselves, and may send data (through PUT or POST) to the CoAP server when needed, as well as query the CoAP server's resources (especially .well-known/core) through GET. We will call those clients delegation clients.

The first delegation requirements are as follows:

REQ-3 The RD should be able to authenticate and authorise delegation clients.

REQ-4 CoAP servers should be able to authenticate and authorise delegation clients.

## 4. Registration of Resource Server to Resource Directory

Registration of an endpoint to the RD is done through a POST, as described in [I-D.ietf-core-resource-directory], section 5.2.

During registration, the endpoint provides its identifier, as well as its resources. In addition, it may provide information such as domain, endpoint type, lifetime and context. The context indicates the scheme, ip and port used to access the endpoint. The source of the registration request is considered the default context.

When finishing registration, the RD returns location path information for use by the client to update its registration information.

The endpoint can update its registration with the RD. This is done through a PUT operation, to the path indicated by the location path in the registration response. The client can update the endpoint type, the lifetime and its context.

The endpoint can cancel it registration with the RD through a DELETE operation.

Since a constrained server may make use of a delegation client, it does not make sense to require an endpoint be solely able to handle its own registration.

Registration has the following requirements for authentication and authorisation:

REQ-5 The endpoint should be authenticated, such that it cannot spoof other endpoints,

REQ-6 The endpoint should only be able to provide, change or delete registration information of other endpoints if it is authorised to do so.

REQ-7 If the endpoint is member of a domain, it should be possible to ensure the true membership of that domain.

## 5. Querying the RD

To Query an RD, a CoAP client sends a CoAP GET request to the RD. The URI indicates the specific directory path, and possibly some queries further defining the requisted information.

Since the RD may cater for multiple areas, the directory path can be used to indicate a certain area. For example, an RD for building control may have different areas for handling lights and fire safety equipment. The light may be controlled by the fire safety equipment, whereas the fire safety equipment should not be controlled by the lights. Hence a fire safety device may be provided access to the RD of both the fire safety area and the lighting, whereas a light switch may only be provided access to the RD of the lighting.

This leads to the following requirements:

REQ-8 The RD should be able to grant different access rights to different clients.

REQ-9 If the different areas are implemented through different URIs, it should be possible for the RD to approve or block access to related areas by providing access rights to specific URIs.

REQ-10 (TBD) Do we want to specify specific access rights to URI-queries?

## 6. Architecture

This section will give the architecture of using ACE for the resource directory.

## 7. Solution

This section will go into deeper technical details of using ACE for the resource directory.

Ideally, the Resource Directory should be treated just like any other CoAP server.

## 8. Security considerations

The complete document concerns security considerations.

## 9. IANA considerations

This document does not require any IANA registrations.

# 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.

## 10.2. Informative References

[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.

Author's Address

Bert Greevenbosch
Huawei Technologies Co., Ltd.
Huawei Industrial Base
Bantian, Longgang District
Shenzhen 518129
P.R. China

Email: bert.greevenbosch@huawei.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 30, 2015

Y-G. Hong
Y-H. Choi
ETRI
D-H. Kim
M-S. Khan
W-Q. JIN
Jeju Nat. Univ.

October 27, 2014

# CoAP Endpoint Unit Identification for Multiple Sensor and Actuator in a Node

draft-hong-core-coap-endpoint-unit-id-01

## 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/.

## Copyright Notice



## 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 . . . . . 6
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 . . . . . 13

## 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 IP 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].

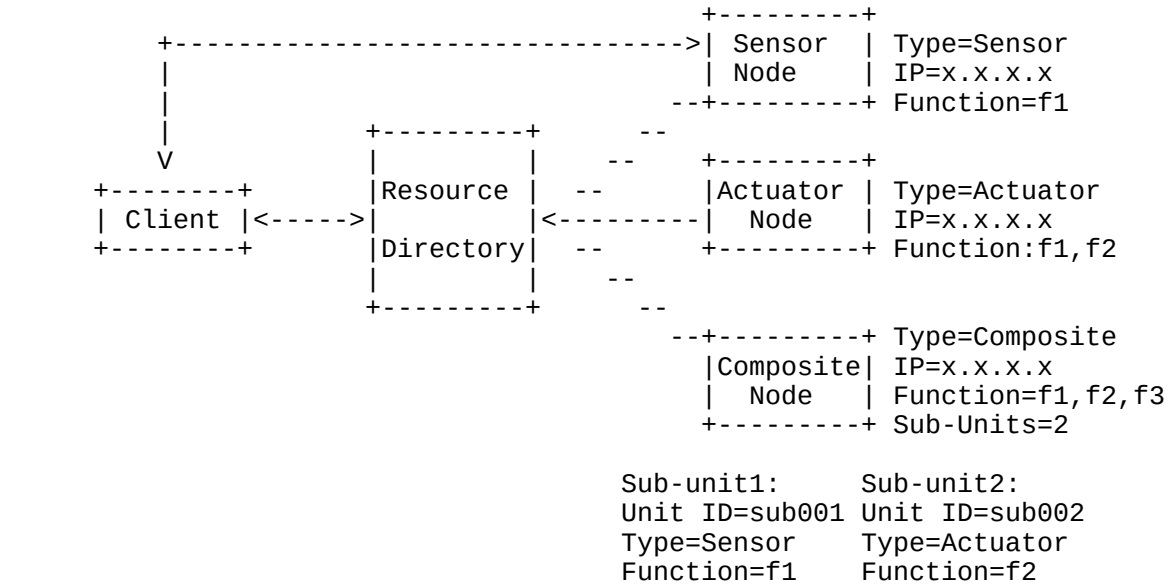



Figure 1: Endpoint Unit ID and Resource Directory

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 Unit-ID(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.

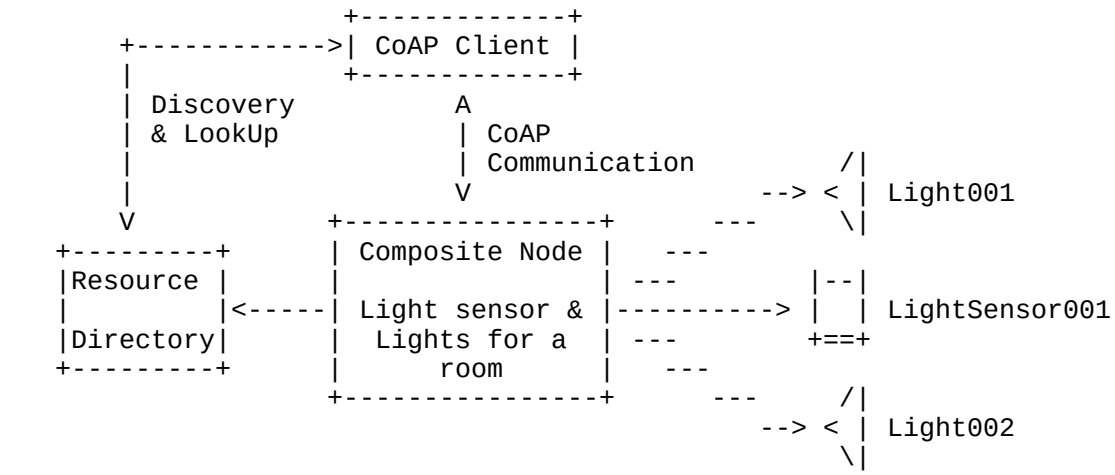



Figure 2: Multiple UnitID based composite CoAP node interaction use case

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

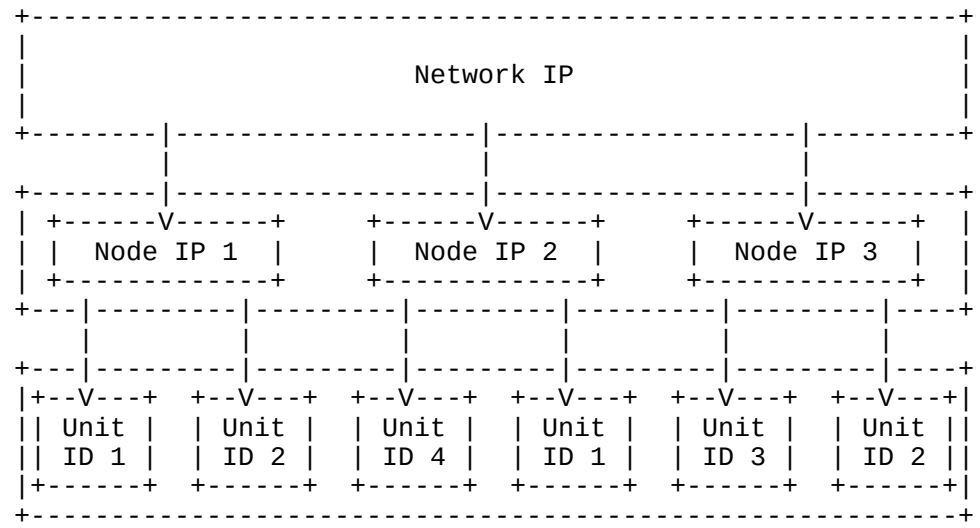



Figure 3: IP address and Endpoint Unit ID mapping architecture

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. We proposed 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.

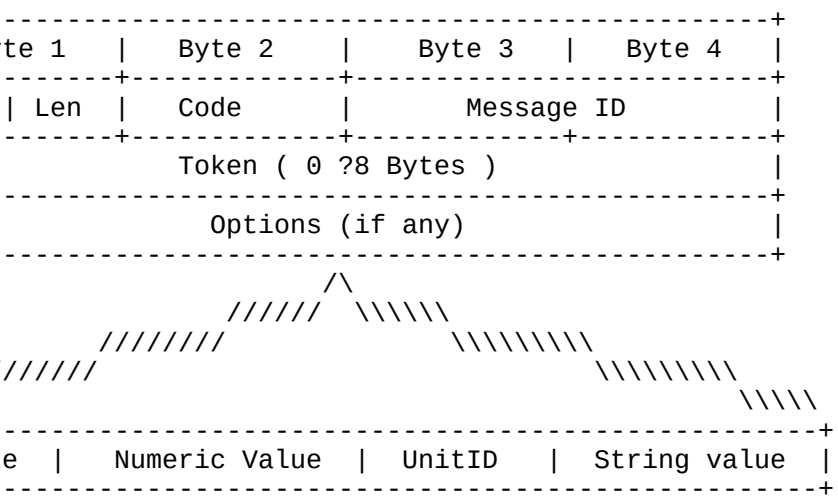



Figure 4: Multi-ID CoAP message format

## 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 IPs 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.

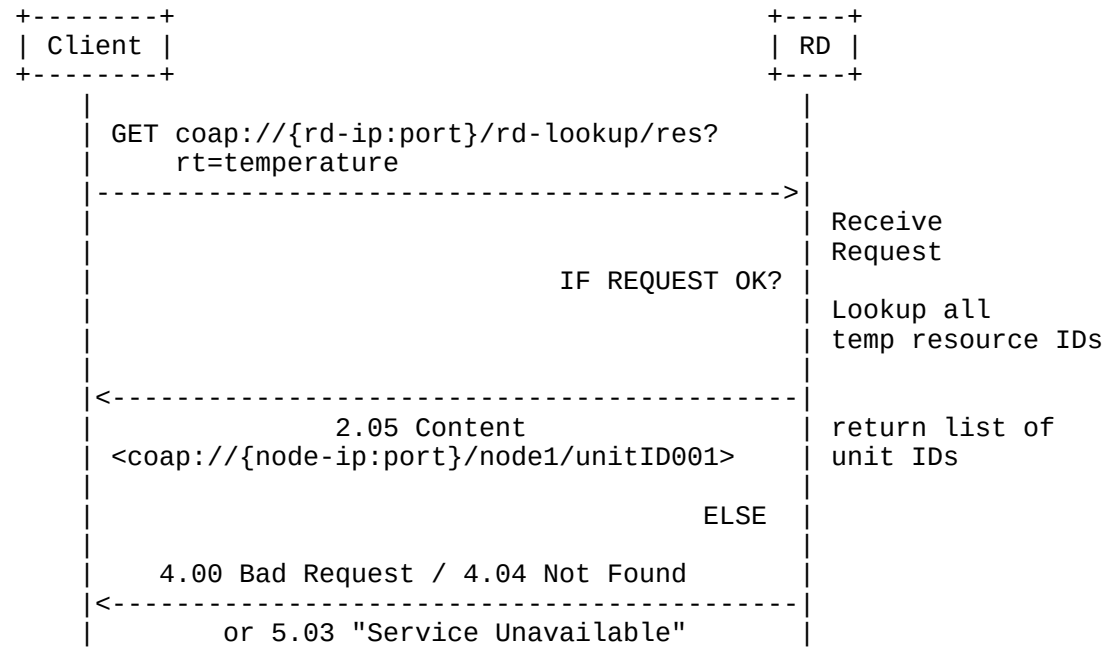



Figure 6: RD based resource lookup

## 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 IP 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 IP address is used to perform the interaction between the CoAP client and server while for a composite node i.e. with multiple integrated resources (multiple unit 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.

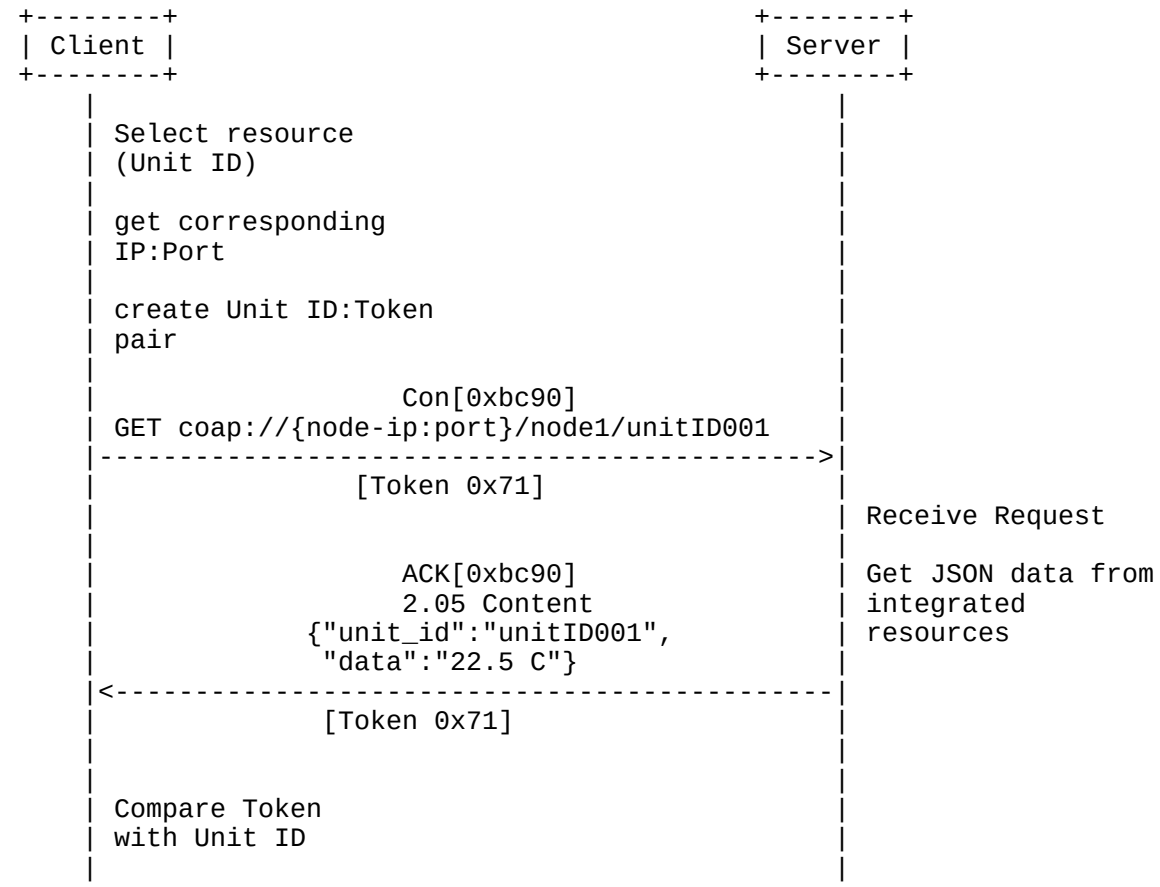



Figure 7: CoAP based client server interaction (single Unit ID)

## 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 (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, 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

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

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

Do-Hyeun 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

Wen-Quan JIN
Jeju Nat. Univ.
Jeju
Korea

Phone: +82 64 754 3658
Email: pluskmk12@live.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

C. Bormann
Universitaet Bremen TZI
Z. Shelby, Ed.
ARM

October 27, 2014

# Blockwise transfers in CoAP
# draft-ietf-core-block-16

## 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

## Copyright Notice



## 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 . . . . . 12
  2.6. Combining Blockwise Transfers with the Observe Option . . 13
  2.7. Combining Block1 and Block2 . . . . . 14
  2.8. Combining Block2 with Multicast . . . . . 14
  2.9. Response Codes . . . . . 14
    2.9.1. 2.31 Continue . . . . . 15
    2.9.2. 4.08 Request Entity Incomplete . . . . . 15
    2.9.3. 4.13 Request Entity Too Large . . . . . 15
  2.10. Caching Considerations . . . . . 15
3. Examples . . . . . 16
  3.1. Block2 Examples . . . . . 16
  3.2. Block1 Examples . . . . . 20
  3.3. Combining Block1 and Block2 . . . . . 21
  3.4. Combining Observe and Block2 . . . . . 23
4. The Size2 and Size1 Options . . . . . 26
5. HTTP Mapping Considerations . . . . . 27
6. IANA Considerations . . . . . 29
7. Security Considerations . . . . . 29

7.1. Mitigating Resource Exhaustion Attacks . . . . . 30
7.2. Mitigating Amplification Attacks . . . . . 31
8. Acknowledgements . . . . . 31
9. References . . . . . 31
9.1. Normative References . . . . . 31
9.2. Informative References . . . . . 32
Authors' Addresses . . . . . 32

## 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:

- reducing implementation complexity in order to minimize code size,
- 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,
- 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:

- Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.
- The transfer of each block is acknowledged, enabling individual retransmission if required.
- Both sides have a say in the block size that actually will be used.
- The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.
- 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:

- by the maximum datagram size (~ 64 KiB for UDP)
- by the desire to avoid IP fragmentation (MTU of 1280 for IPv6)
- 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:

- The size of the block (SZX);
- whether more blocks are following (M);
- 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.)

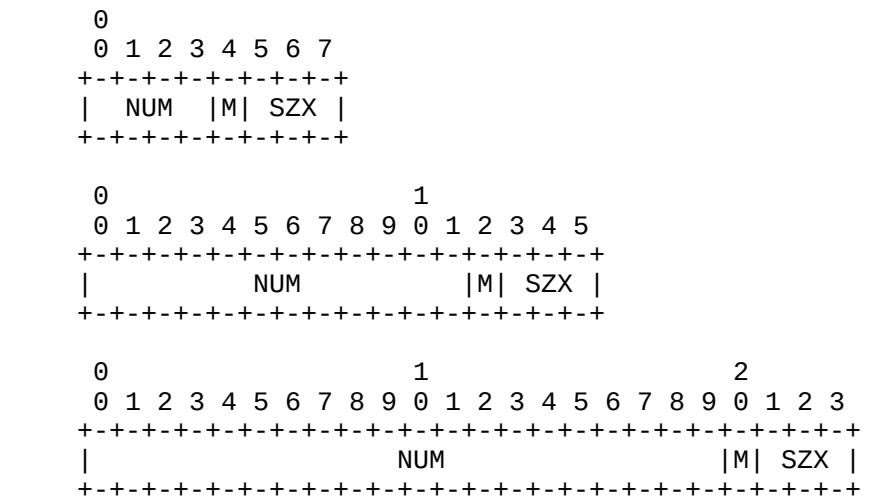



Figure 1: Block option value

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**(SZX + 4)". SZX is transferred in the three least significant bits of the option value (i.e., "val & 7" where "val" is the value of the option).

The fourth least significant bit, the M or "more" bit ("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 "NUM << (SZX + 4)".

Implementation note: As an implementation convenience, "(val & ~0xF) << (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:

- 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.

The Content-Format Option sent with the requests or responses MUST reflect the content-format of the entire body. If blocks of a response body arrive with different content-format options, it is up to the client how to handle this error (it will typically abort any ongoing block-wise transfer). If blocks of a request arrive at a server with mismatching content-format options, the server MUST NOT assemble them into a single request; this usually leads to a 4.08 (Request Entity Incomplete, Section 2.9.2) error response on the mismatching block.

## 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. Clients that want to retrieve all blocks of a resource SHOULD strive to do so without undue delay. Servers can fully expect to be free to discard any cached state after a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) after the last access to the state, however, there is no requirement to always keep the state for as long.

The Block2 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise response payload transfer (e.g., GET) operations to the same resource. This is rarely a requirement, but as a workaround, a client may vary the cache key (e.g., by using one of several URIs accessing resources with the same semantics, or by varying a proxy-safe elective option).

## 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.

One reason that a client might encounter a 4.08 error code is that the server has already timed out and discarded the partial request body being assembled. Clients SHOULD strive to send all blocks of a request without undue delay. Servers can fully expect to be free to discard any partial request body when a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) has elapsed after the most recent block

was transferred; however, there is no requirement on a server to always keep the partial request body for as long.

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). Even more so than for the general case of Block2, clients that want to retrieve all blocks of a resource they have been notified about with a first block SHOULD strive to do so without undue delay.

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 block-wise transfers. First, the Block1 transfer of the request body proceeds as usual. In the exchange of the last slice of this block-wise 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.

### 2.10. Caching Considerations

This specification attempts to leave a variety of implementation strategies open for caches, in particular those in caching proxies. E.g., a cache is free to cache blocks individually, but also could wait to obtain the complete representation before it serves parts of it. Partial caching may be more efficient in a cross-proxy (equivalent to a streaming HTTP proxy). A cached block (partial cached response) can be used in place of a complete response to satisfy a block-wise request that is presented to a cache. Note that different blocks can have different Max-Age values, as they are transferred at different times. A response with a block updates the freshness of the complete representation. Individual blocks can be

validated, and validating a single block validates the complete representation. A response with a Block1 Option in control usage with the M bit set invalidates cached responses for the target URI.

A cache or proxy that combines responses (e.g., to split blocks in a request or increase the block size in a response, or a cross-proxy) may need to combine 2.31 and 2.01/2.04 responses; a stateless server may be responding with 2.01 only on the first Block1 block transferred, which dominates any 2.04 responses for later blocks.

If-None-Match only works correctly on Block1 requests with (NUM=0) and MUST NOT be used on Block1 requests with NUM != 0.

## 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.

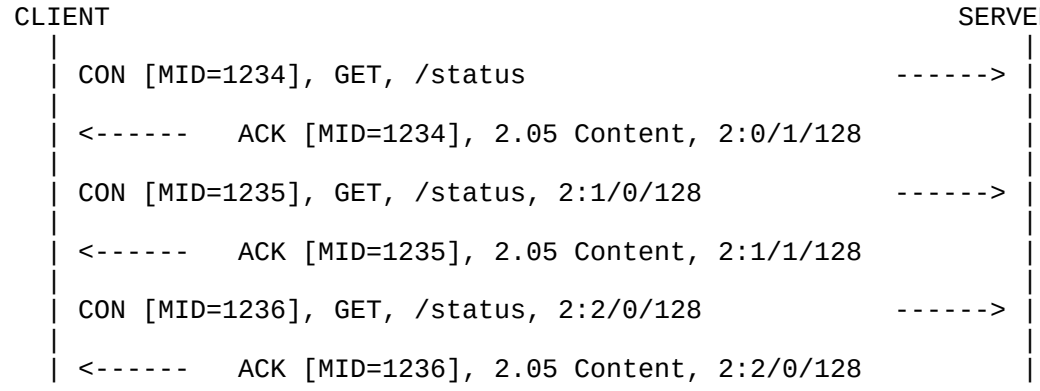



Figure 2: Simple blockwise GET

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.

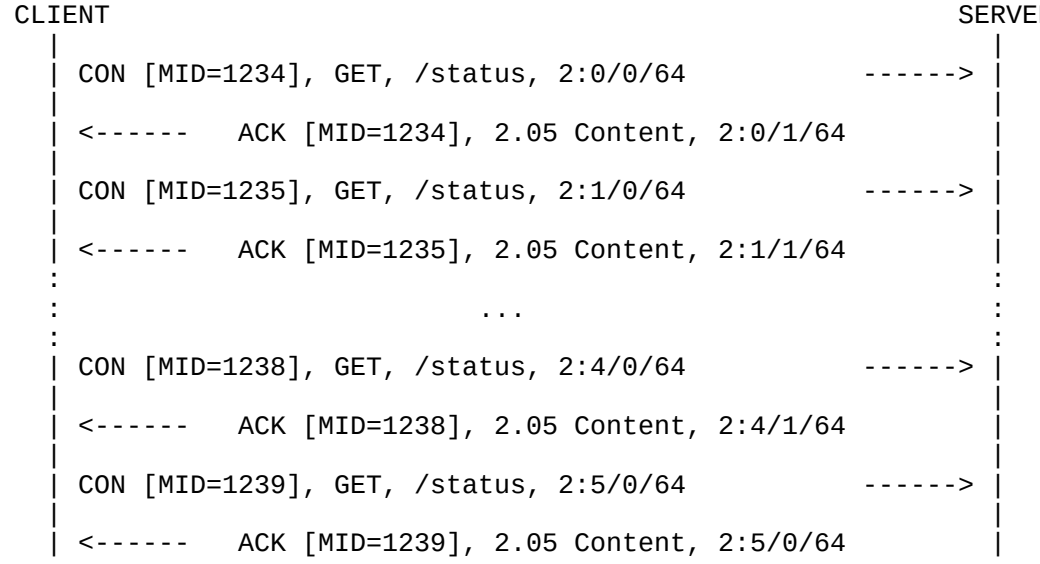



Figure 3: Blockwise GET with early negotiation

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.

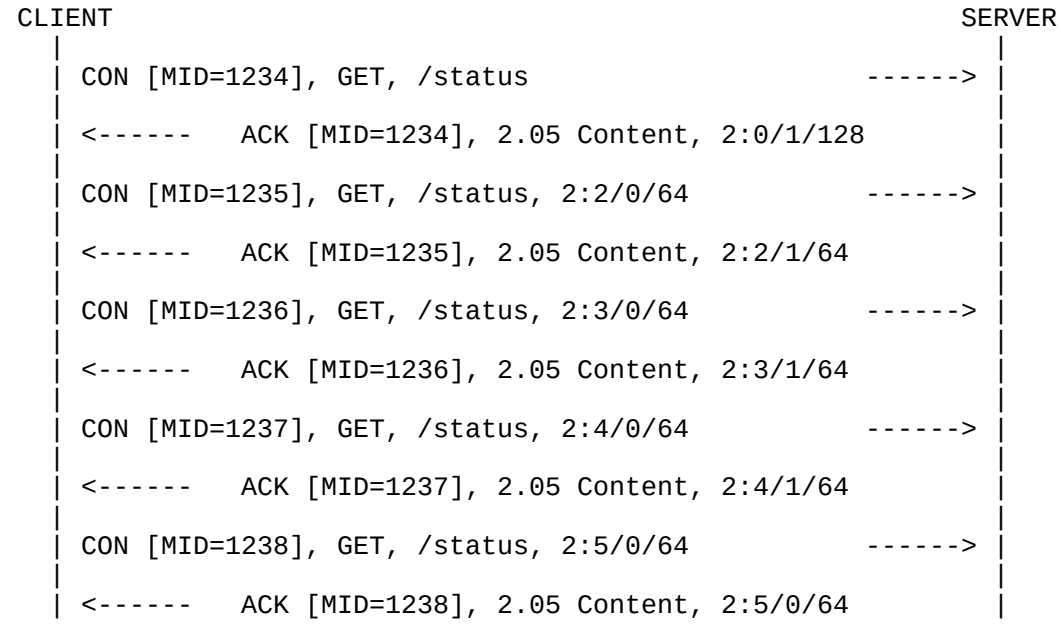



Figure 4: Blockwise GET with late negotiation

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).

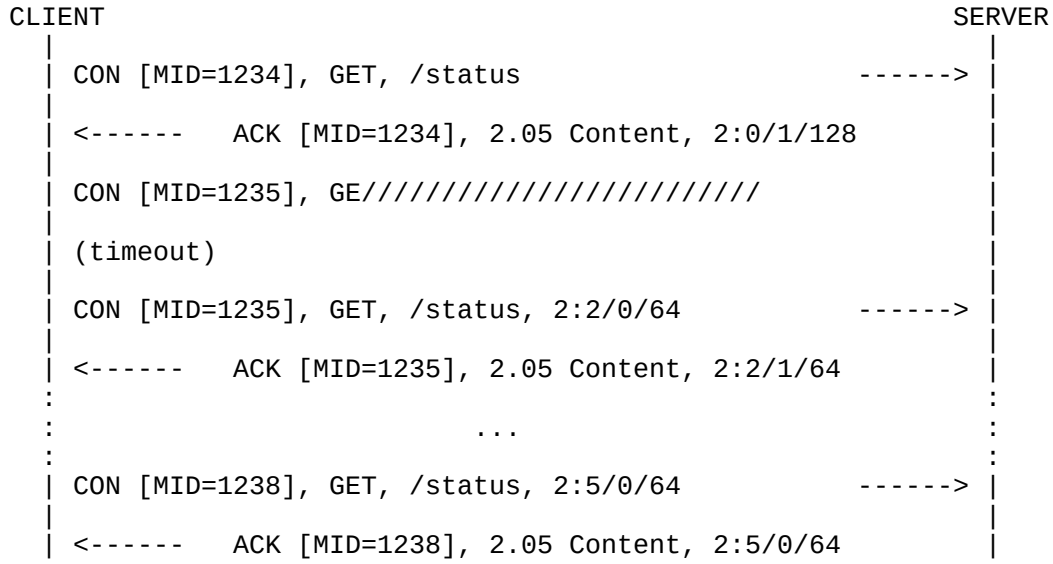



Figure 5: Blockwise GET with late negotiation and lost CON

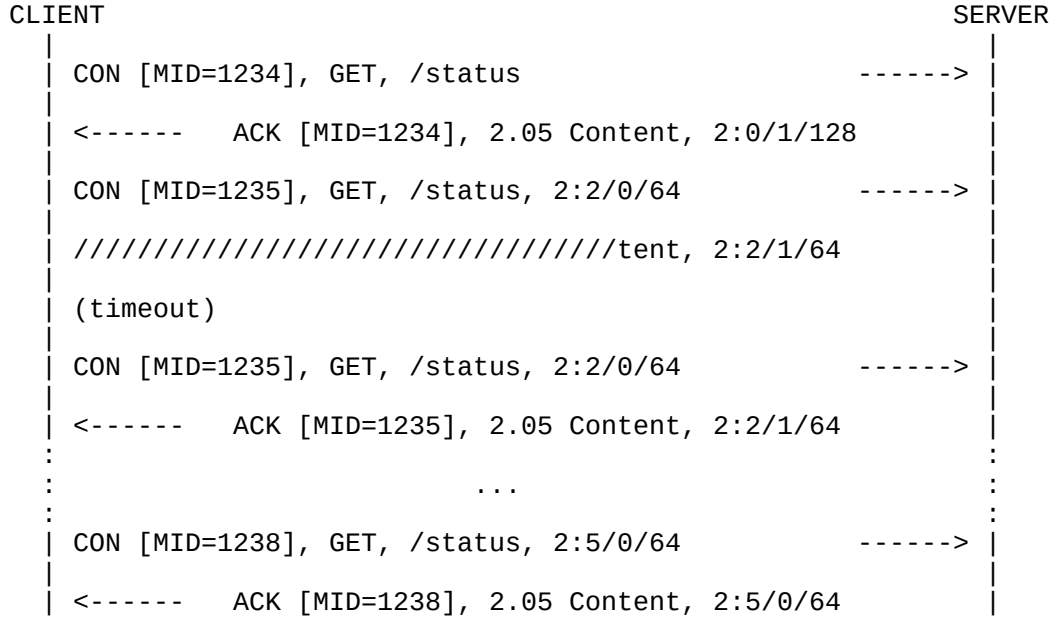



Figure 6: Blockwise GET with late negotiation and lost ACK

### 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.

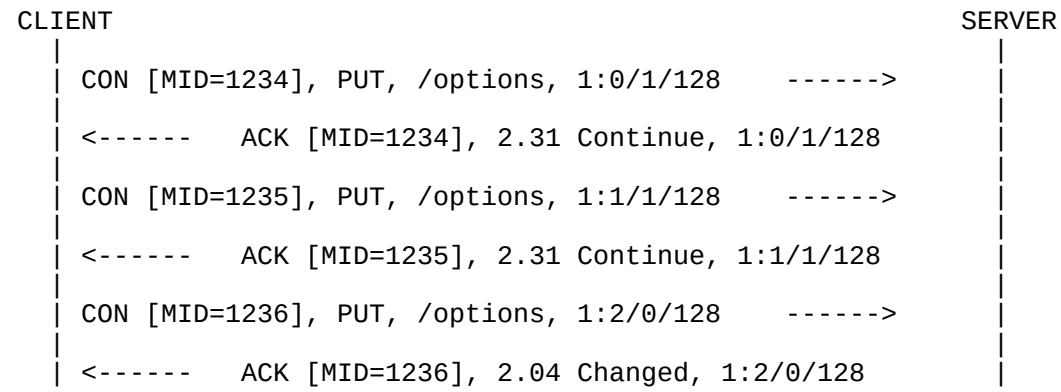



Figure 7: Simple atomic blockwise PUT

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.

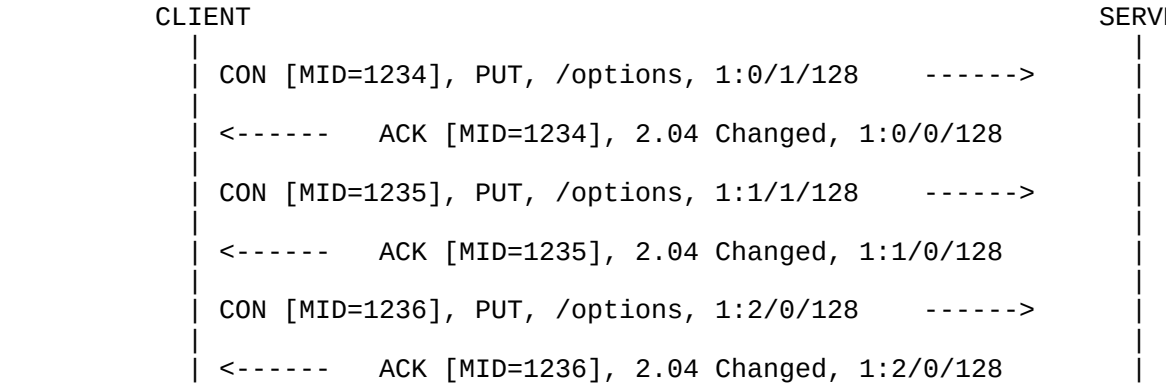



Figure 8: Simple stateless blockwise PUT

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.

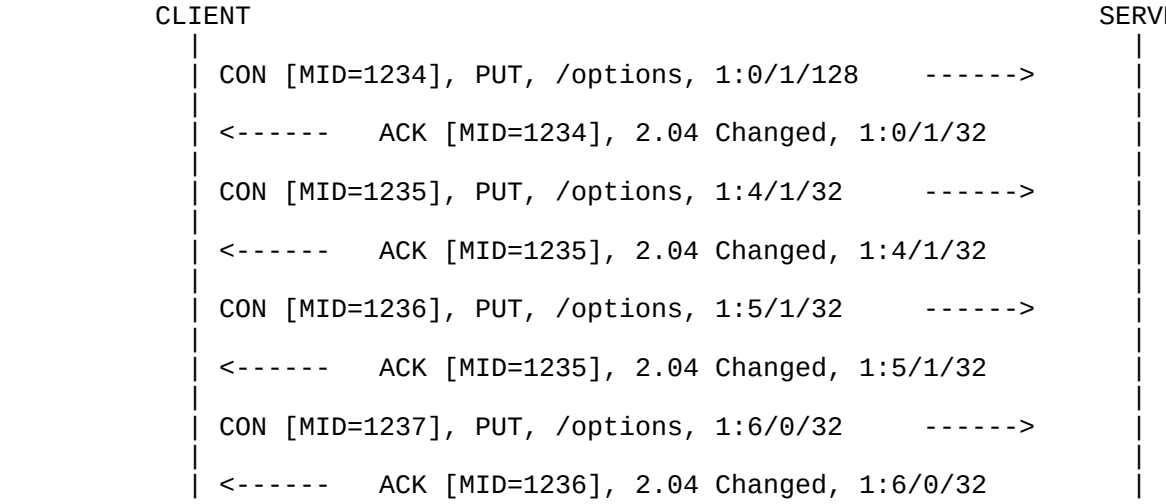



Figure 9: Simple atomic blockwise PUT with negotiation

## 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.

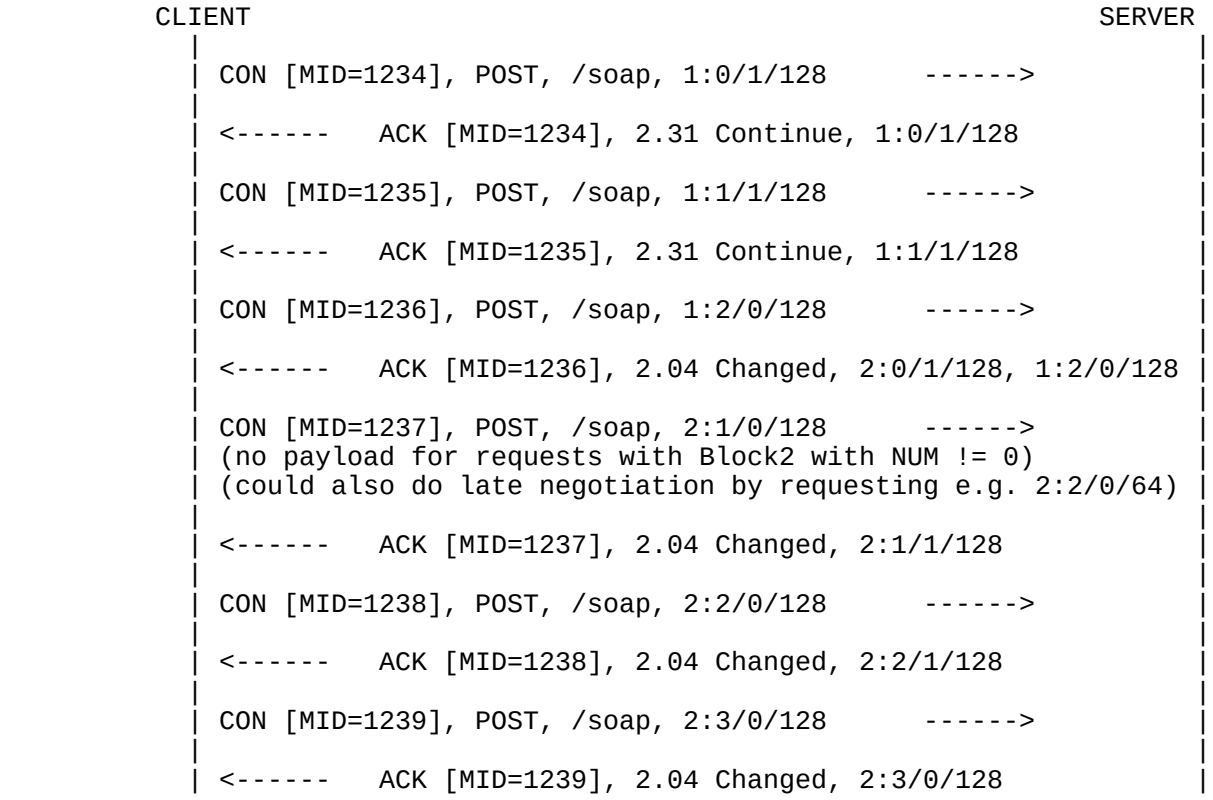



Figure 10: Atomic blockwise POST with blockwise response

This model does provide for early negotiation input to the Block2 blockwise transfer, as shown below.

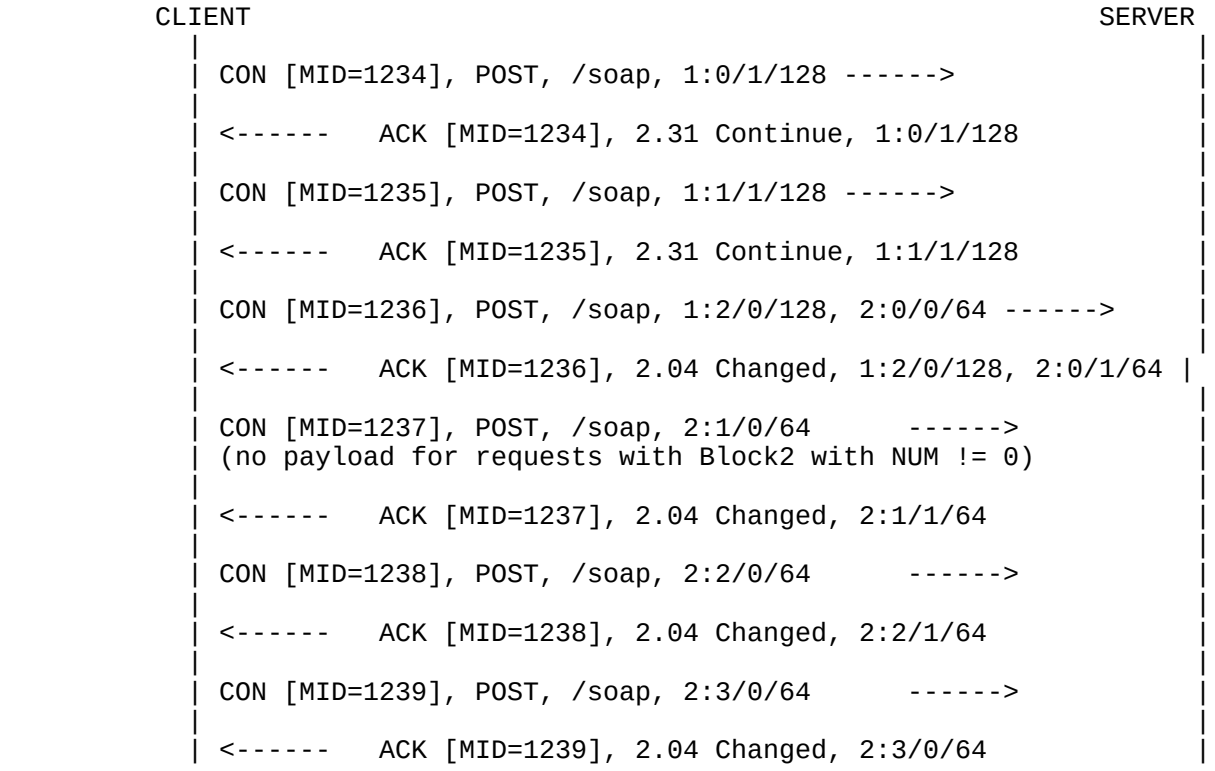
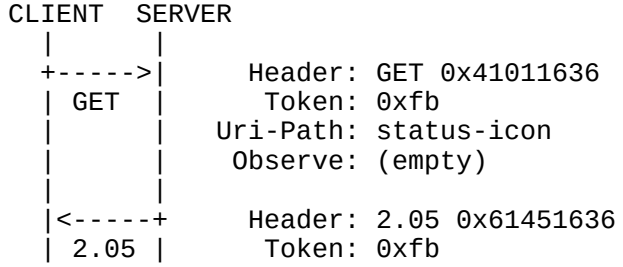



Figure 11: Atomic blockwise POST with blockwise response, early negotiation

## 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.



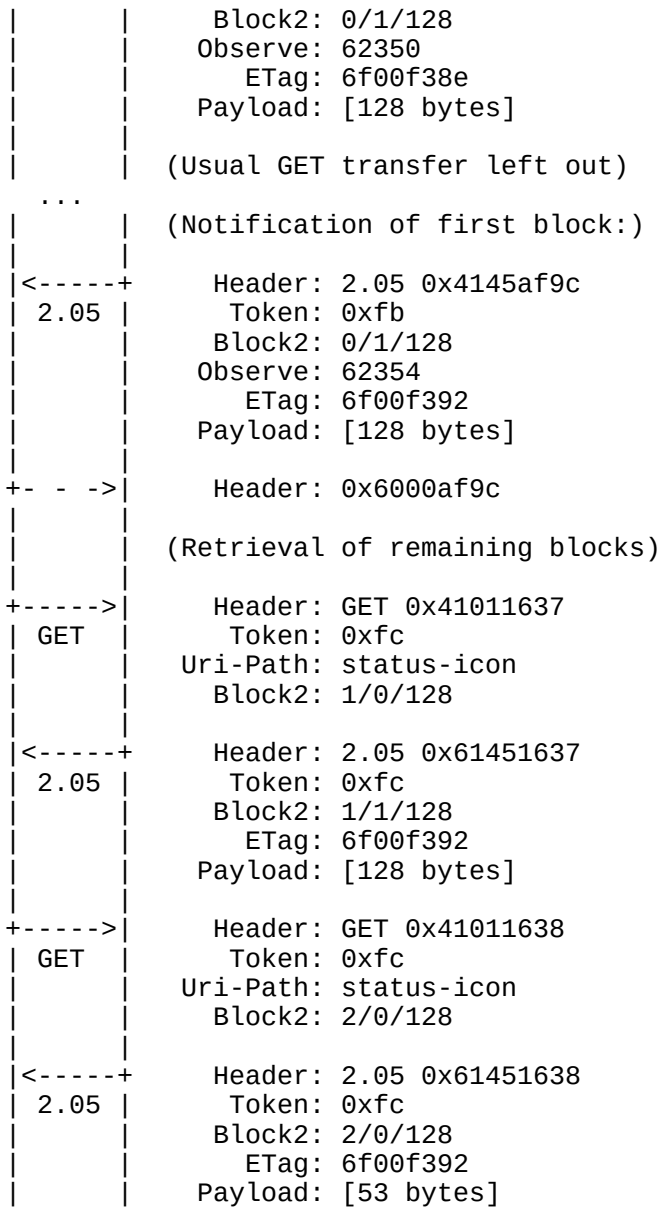
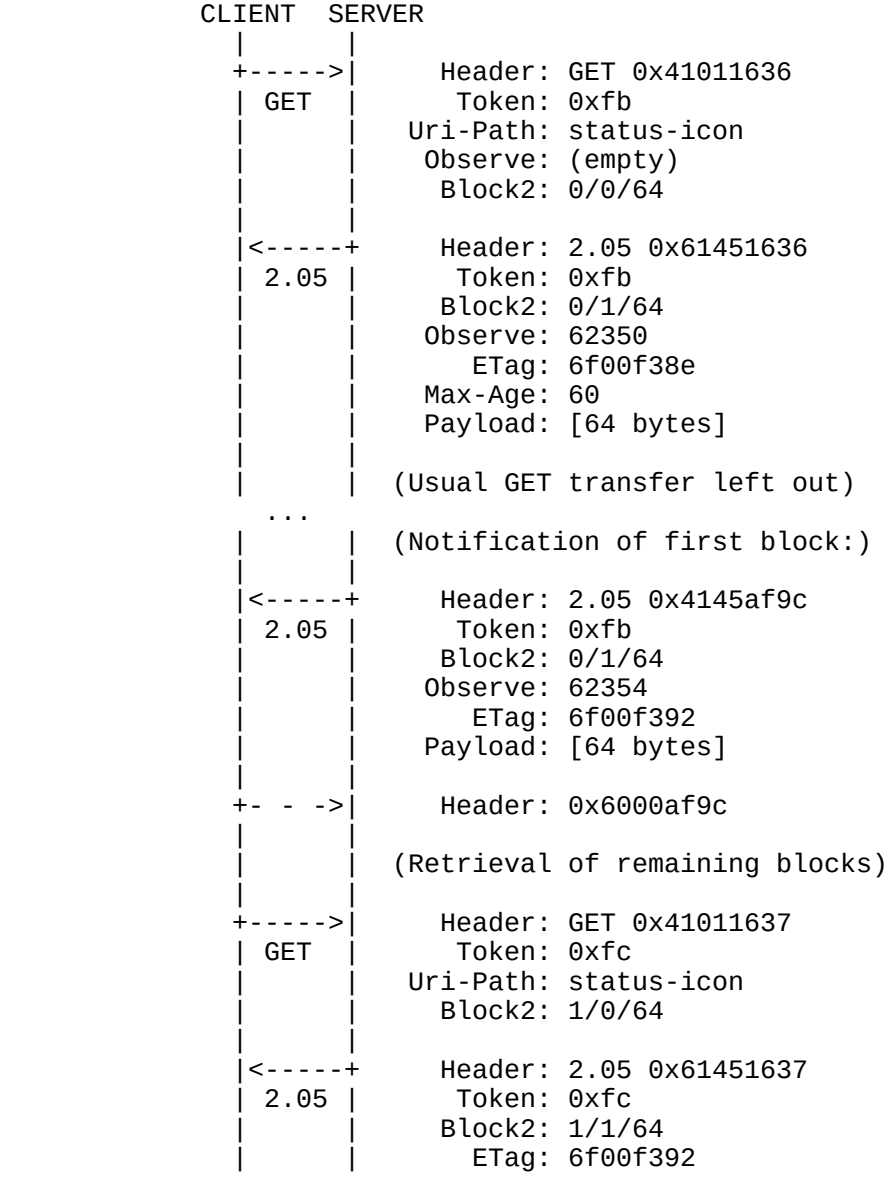



Figure 12: Observe sequence with blockwise response

(Note that the choice of token 0xfc in this examples is arbitrary; tokens are just shown in this example to illustrate that the requests for additional blocks cannot make use of the token of the Observation

relationship. As a general comment on tokens, there is no other mention of tokens in this document, as blockwise transfers handle tokens like any other CoAP exchange. As usual the client is free to choose tokens for each exchange as it likes.)

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



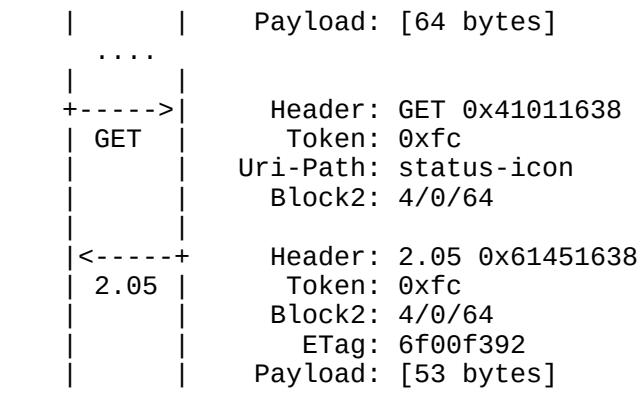



Figure 13: Observe sequence with early negotiation

## 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:

- 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.

- 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:

- 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>.

[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

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: Experimental
Expires: March 16, 2015

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

September 12, 2014

# Group Communication for CoAP
# draft-ietf-core-groupcomm-25

## Abstract

The Constrained Application Protocol (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 specification defines how the CoAP protocol should be used in a group communication context. An approach for using CoAP on top of IP multicast is detailed based on both existing CoAP functionality as well as new features introduced in this specification. 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 March 16, 2015.

## Copyright Notice

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 . . . . . 9
2.5. Request and Response Model . . . . . 9
2.6. Membership Configuration . . . . . 10
2.6.1. Background . . . . . 10
2.6.2. Membership Configuration RESTful Interface . . . . . 11
2.7. Request Acceptance and Response Suppression Rules . . . . 17
2.8. Congestion Control . . . . . 19
2.9. Proxy Operation . . . . . 20
2.10. Exceptions . . . . . 21
3. Use Cases and Corresponding Protocol Flows . . . . . 22
3.1. Introduction . . . . . 22
3.2. Network Configuration . . . . . 22
3.3. Discovery of Resource Directory . . . . . 24
3.4. Lighting Control . . . . . 26
3.5. Lighting Control in MLD Enabled Network . . . . . 30
3.6. Commissioning the Network Based On Resource Directory . . 31
4. Deployment Guidelines . . . . . 32
4.1. Target Network Topologies . . . . . 32
4.2. Networks Using the MLD Protocol . . . . . 33
4.3. Networks Using RPL Multicast Without MLD . . . . . 33
4.4. Networks Using MPL Forwarding Without MLD . . . . . 34
4.5. 6LoWPAN Specific Guidelines for the 6LBR . . . . . 35
5. Security Considerations . . . . . 35
5.1. Security Configuration . . . . . 35
5.2. Threats . . . . . 36
5.3. Threat Mitigation . . . . . 36
5.3.1. WiFi Scenario . . . . . 37
5.3.2. 6LoWPAN Scenario . . . . . 37

5.3.3. Future Evolution . . . . . 37
5.4. Monitoring Considerations . . . . . 38
5.4.1. General Monitoring . . . . . 38
5.4.2. Pervasive Monitoring . . . . . 38
6. IANA Considerations . . . . . 39
6.1. New 'core.gp' Resource Type . . . . . 39
6.2. New 'coap-group+json' Internet Media Type . . . . . 39
7. Acknowledgements . . . . . 41
8. References . . . . . 41
8.1. Normative References . . . . . 41
8.2. Informative References . . . . . 43
Appendix A. Multicast Listener Discovery (MLD) . . . . . 44
Appendix B. Change Log . . . . . 44
Authors' Addresses . . . . . 57

# 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 base protocol aspects of sending CoAP requests on top of IP multicast, and processing the (unicast IP) responses are given in Section 8 of [RFC7252]. To provide a more complete CoAP group communication functionality, this specification introduces new CoAP protocol processing functionality (e.g., new rules for re-use of

Token values, request suppression, and proxy operation) and a new management interface for RESTful group membership configuration.

CoAP group communication will run in Any Source Multicast (ASM) mode [RFC5110] of IP multicast operation. This means that there is no restriction on the source node which sends (originates) the CoAP messages to the IP multicast group. For example, the source node may be part of the IP multicast group or not. Also, there is no restriction on the number of source nodes.

While Section 9.1 of [RFC7252] supports various modes of DTLS-based security for CoAP over unicast IP, it does not specify any security modes for CoAP over IP multicast. That is, [RFC7252] assumes that CoAP over IP multicast is not encrypted, nor authenticated, nor access controlled. This document assumes the same security model (see Section 5.1). However, there are several promising security approaches being developed that should be considered in the future for protecting CoAP group communications (see Section 5.3.3).

### 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] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

Note that this document refers back to other RFCs, and especially [RFC7252], to help explain overall CoAP group communication features. However use of [RFC2119] key words is reserved for new CoAP functionality introduced by this specification.

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. An example of a reliable group communication protocol is [RFC5740].

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 may 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). Also, a multicast forwarding proxy node [RFC4605] may be required.

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 (Section 12.8 of [RFC7252]) 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 (Section 6 of [RFC7252].

Note: A group URI is needed to initiate CoAP group communications. For CoAP client implementations it is recommended to use the URI decomposition method of Section 6.4 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. (This is because DNS infrastructure may not be deployed in many constrained network deployments). 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 trouble shooting 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, usually on the CoAP default UDP port, 5683. If the group uses a specified non-default UDP port, be careful to ensure that all group members are configured to use that same port.

Different ports for the same IP multicast address are preferably not used to specify different CoAP groups. If disjoint groups share the same IP multicast address, then all the devices interested in one group will accept IP traffic also for the other disjoint groups, only to ultimately discard the traffic higher in their IP stack (based on UDP port discrimination).

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.

2. If the client is configured to use service discovery including URI and port discovery, it uses the 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. Regardless of the method of selecting the port number, the same port MUST be used across all CoAP servers in a group and across all CoAP clients performing the group requests.

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. Implementers are free to define the paths as they see fit. However, note that [RFC7320] prescribes that a specification must not constrain, define structure or semantics for any path component. So for this reason, a pre-defined URI path is not specified in this document and also must not be provided in other specifications.

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.

However the URI path is selected, the same path MUST be used across all CoAP servers in a group and all CoAP clients performing the group requests.

## 2.4. RESTful Methods

Group communication most often uses the idempotent CoAP methods GET and PUT. The idempotent method DELETE can also be used. The non-idempotent CoAP method POST may only 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. For example, a client may re-send a multicast POST request for additional reliability. Some servers will receive the request two times while others may receive it only once. For idempotent methods all these servers will be in the same state, while for POST this is not guaranteed; so the resource POST operation must be specifically designed to take message loss into account.

## 2.5. Request and Response Model

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

A server optionally sends 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 2.7.

A CoAP request sent over IP multicast and any unicast response it causes must take into account the congestion control rules defined in Section 2.8.

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 incorrect response/request matching in the client, which is a protocol error. Therefore the time between re-use of Token values used in multicast requests MUST be greater than:

NON_LIFETIME + MAX_LATENCY + MAX_SERVER_RESPONSE_DELAY

where NON_LIFETIME and MAX_LATENCY are defined in Section 4.8 of [RFC7252]. 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 Leisure time period as defined in Section 8.2 of [RFC7252]. The CoAP protocol does not define a time limit for the server response delay. Using the default CoAP protocol parameters, the Token re-use time MUST be greater than 250 seconds plus MAX_SERVER_RESPONSE_DELAY. A preferred solution to meet this requirement is to generate a new unique Token for every multicast request, such that a Token value is never re-used. If a client has to re-use Token values for some reason, and also MAX_SERVER_RESPONSE_DELAY is unknown, then using MAX_SERVER_RESPONSE_DELAY = 250 seconds is a reasonable guideline. The time between Token re-uses is in that case set to a value greater than 500 seconds.

## 2.6. Membership Configuration

### 2.6.1. Background

#### 2.6.1.1. 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]. This discovery interface is not required to invoke CoAP group communications. However, it is a potential complementary interface useful for overall management of CoAP groups. Other methods to discover groups (e.g., proprietary management systems) can also be used. An example of doing some of the RD based lookups is given in Section 3.6.

#### 2.6.1.2. Configuring Members

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].

In the third case, typical in scenarios such as building control, a dynamic commissioning tool determines to which group(s) a sensor or actuator node belongs, and writes this information to the node, which can subsequently join the correct IP multicast group(s) on its network interface. The information written per group may again be an IP multicast address or a hostname.

#### 2.6.2. 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 will use unicast CoAP methods (GET/PUT/POST/DELETE). This interface is a method of configuring group information in individual endpoints.

Also, a form of authorization (preferably making use of unicast DTLS-secured CoAP of Section 9.1 of [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 alternative 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]).

##### 2.6.2.1. 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 JavaScript Object Notation (JSON) based content format (as indicated by the "application/coap-group+json" media type). The resource includes zero or more group membership JSON objects [RFC7159] in a format as defined in Section 2.6.2.4. A group membership JSON object contains one or more key/value pairs as defined below, and represents a single IP multicast group membership for the CoAP endpoint. Each key/value pair is encoded as a member of the JSON object, where the key is the member name and the value is the member's value.

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 (and optionally the port number), 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 3.2.2 of [RFC3986] which are also used in the base CoAP protocol (Section 6 of [RFC7252].

```
group-address = IPv4address [ ":" port ]
                / "[" IPv6address "]" [":" port ]
```

In any group membership object, if the IP address is known when the object is created, it is included in the "a" key/value pair. If the "a" value cannot be provided, the "n" value MUST be included, containing a valid hostname with optional port number that can be translated to an IP multicast address via DNS.

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

If the port number is not provided, then the endpoint will attempt to look up the port number from DNS if it supports a method to do this.

The possible DNS methods include DNS-SRV [RFC2782] or DNS-SD [RFC6763]. If port lookup is not supported or not provided by DNS, the default CoAP port (5683) is assumed.

After any change on a Group configuration resource, the endpoint MUST effect registration/de-registration from the corresponding IP multicast group(s) by making use of APIs such as IPV6_RECVPKTINFO [RFC3542].

#### 2.6.2.2. 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. Index MUST be a string of maximum two (2) alphanumeric AS
CII
    characters (case insensitive). It MUST be locally unique to the endpoint ser
ver.
    It indexes the particular endpoint’s list of group memberships.

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. The "a" key/value pair is always used if it is given. The "n" pair is only used when there is no "a" pair. If only the "n" pair is given, the CoAP endpoint performs DNS resolution to obtain the IP multicast address from the hostname in the "n" pair. If DNS resolution is not successful, then the endpoint does not attempt joining or listening to any multicast group for this case since the IP multicast address is unknown.

After any change on a Group configuration resource, the endpoint MUST effect registration/de-registration from the corresponding IP multicast group(s) by making use of APIs such as IPV6_RECVPKTINFO [RFC3542]. 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.

#### 2.6.2.3. 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
```

#### 2.6.2.4. 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 (member names) are group indices and the values (member values) are the corresponding group membership objects. Each group membership object describes one IP multicast group membership. If no group memberships are configured then an empty JSON object is returned.

Method: GET

URI Template: /{+gp}

URI Template Variables:

gp - see Section 2.6.2.2

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 string will represent the same IPv6 address that was originally submitted in group membership creation, though it might be a different string because of different choices in IPv6 string representation formatting that may be allowed for the same address (see [RFC5952]).

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

Similar to Section 2.6.2.4 but only a single group membership is read. If the requested group index does not exist then a 4.04 Not Found response is returned.

Method: GET

URI Template 1: {+location}

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

URI Template Variables:

location - see Section 2.6.2.3

gp, index - see Section 2.6.2.2

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"}
```

#### 2.6.2.6. 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 MUST 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 Section 2.6.2.2

```
Example: (replacing all existing group memberships with two new group membership
s)
  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 any more present in the new memberships. An API such as IPV6_RECVPKTINFO [RFC3542] should be used for this purpose. 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.

#### 2.6.2.7. 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 - see Section 2.6.2.3

gp, index - see Section 2.6.2.2

```
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 any more present in the new membership. An API such as IPV6_RECVPKTINFO [RFC3542] should be used for this purpose.

## 2.7. Request Acceptance and Response Suppression Rules

CoRE Link Format [RFC6690], and Section 8 of CoAP [RFC7252] define 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 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 behaviors are:

- A server should not accept an IP multicast request that cannot be "authenticated" in some way (i.e, cryptographically or by some multicast boundary limiting the potential sources) (Section 11.3 of [RFC7252]. See Section 5.3 for examples of multicast boundary limiting methods.

- 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.

- 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.)

- Otherwise accept the IP multicast request.

For IP multicast response suppression, the behaviors are:

- A server should not respond to an IP multicast discovery request if the filter specified by the request's query string does not match.

- 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).

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:

- Suppression of all 2.xx success responses;

- Suppression of all 4.xx client errors;

- Suppression of all 5.xx server errors;

- 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:

- CoAP resource discovery: Suppress 2.05 responses with empty payload and all 4.xx and 5.xx errors.

- Lighting control: Suppress all 2.xx responses after a lighting change command.

- 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.

- 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.

- 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).

## 2.8. Congestion Control

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

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

- A server may choose not to respond to an IP multicast request if there’s nothing useful to respond (e.g., error or empty response)(Section 8.2 [RFC7252]). See Section 2.7 for more detailed guidelines on response suppression.

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

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

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

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

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

- 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 (see Section 4 of [RFC4944]) is used.

- 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 [RFC4919] are given in Section 4.5.

## 2.9. Proxy Operation

CoAP (Section 5.7.2 of [RFC7252]) allows a client to request a forward-proxy to process its CoAP request. For this purpose the client 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 (Section 5.7.2 of [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:

- 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.

- 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 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 if all the following conditions are met:

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

- The proxy SHOULD return individual (unicast) CoAP responses to the client (i.e., not aggregated). The exception case here occurs when a (future) standardized aggregation format is being used.

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

## 2.10. Exceptions

CoAP group communication using IP multicast offers improved network efficiency and latency among 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.

CoAP Observe [I-D.ietf-core-observe] is a feature for a client 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). CoAP Observe does not support a group communication mode. CoAP Observe only supports a unicast mode of operation.

# 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:

- 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].

- 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:

- 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, for clarity, only three are shown.

- Light-1 and the Light Switch are connected to a router (Rtr-1).

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

- 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.

- A CoAP RD is connected to the network backbone.

- 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).

- A Controller (CoAP client) is connected to the backbone, which is able to control various building functions including lighting.

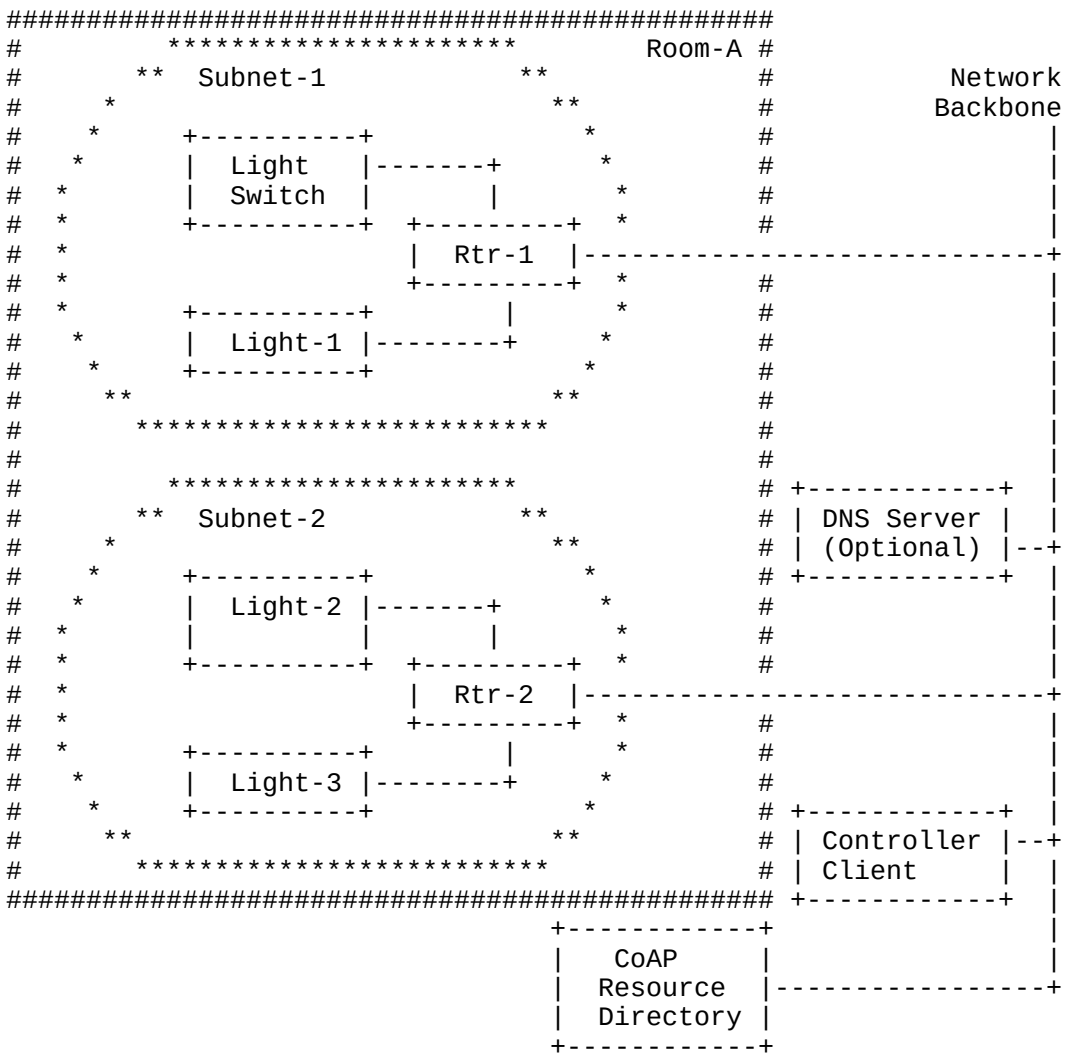



Figure 1: Network Topology of a Large Room (Room-A)

## 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.

- 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).

- 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").

- 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".

- 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.

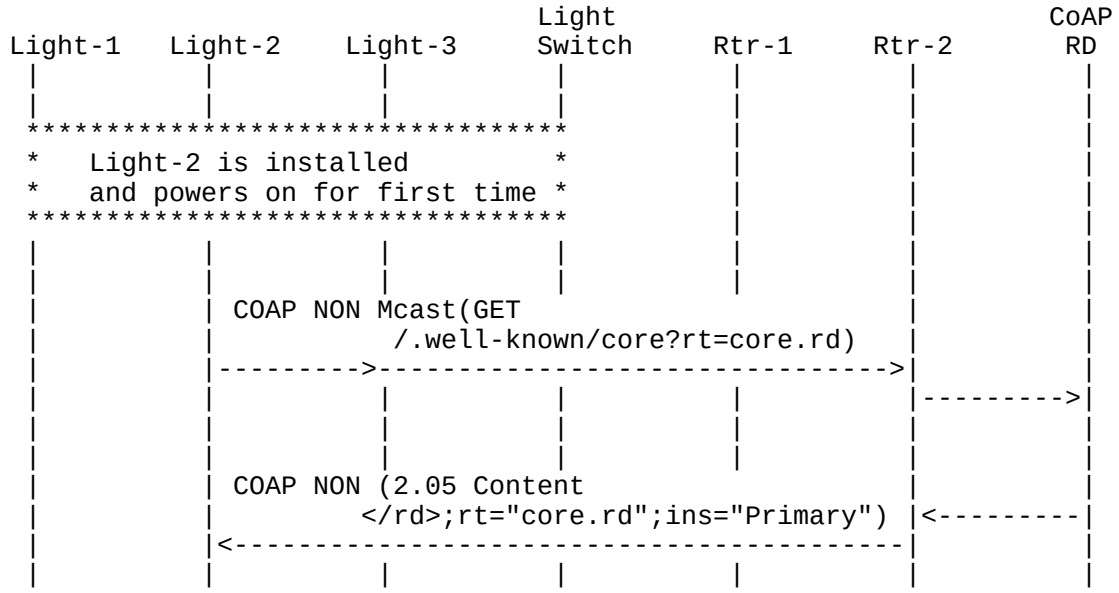



Figure 2: Resource Directory Discovery via Multicast Request

## 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 2.7 for more details on response suppression by a server.)

In addition, Figure 4 shows a protocol flow example for the case that servers do respond to a lighting control 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.

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

3b) As an alternative to the previous step, when a DNS server is available, the Light Switch and/or the Lights have been configured with a group hostname 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.

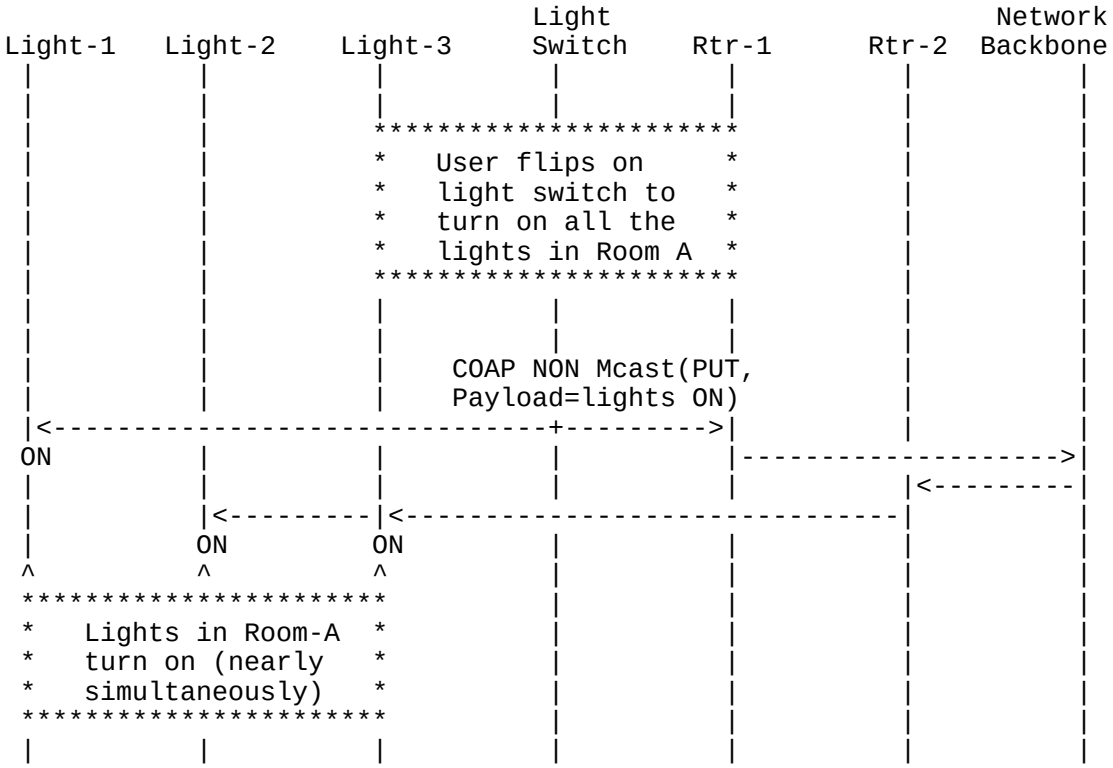



Figure 3: Light Switch Sends Multicast Control Message

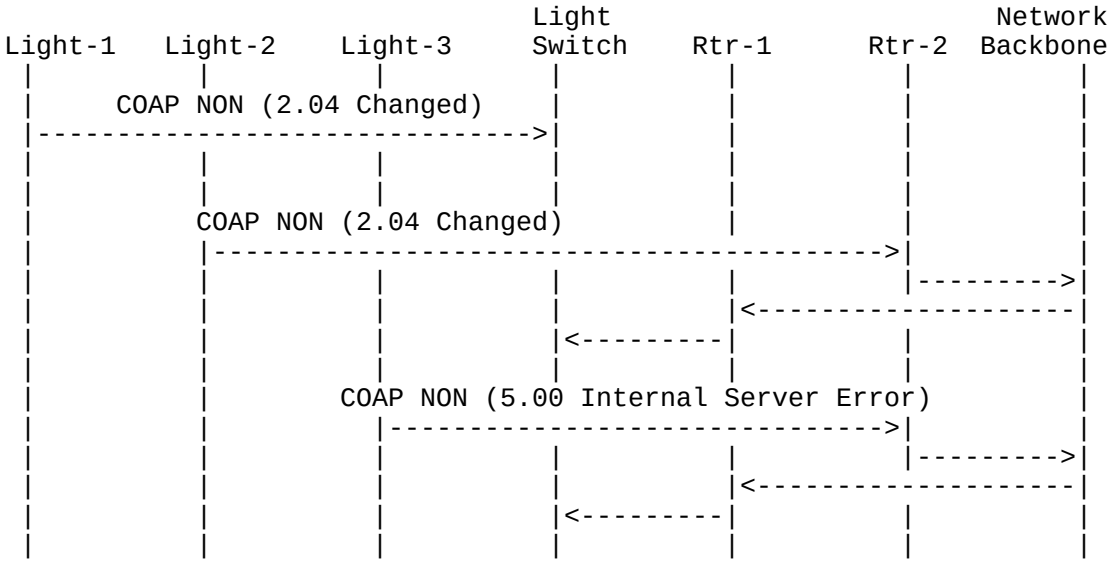



Figure 4: Lights (Optionally) Respond to Multicast CoAP Request

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.

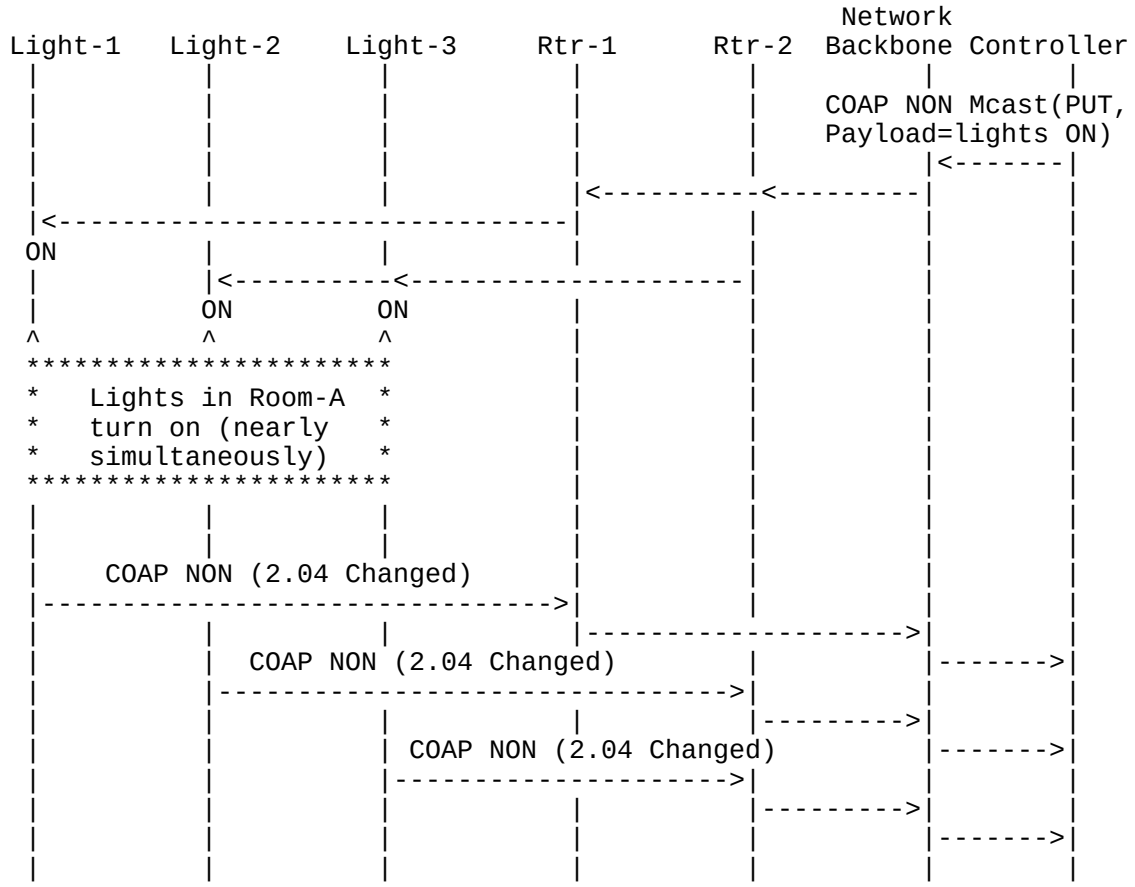



Figure 5: Controller On Backbone Sends Multicast Control Message

## 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.

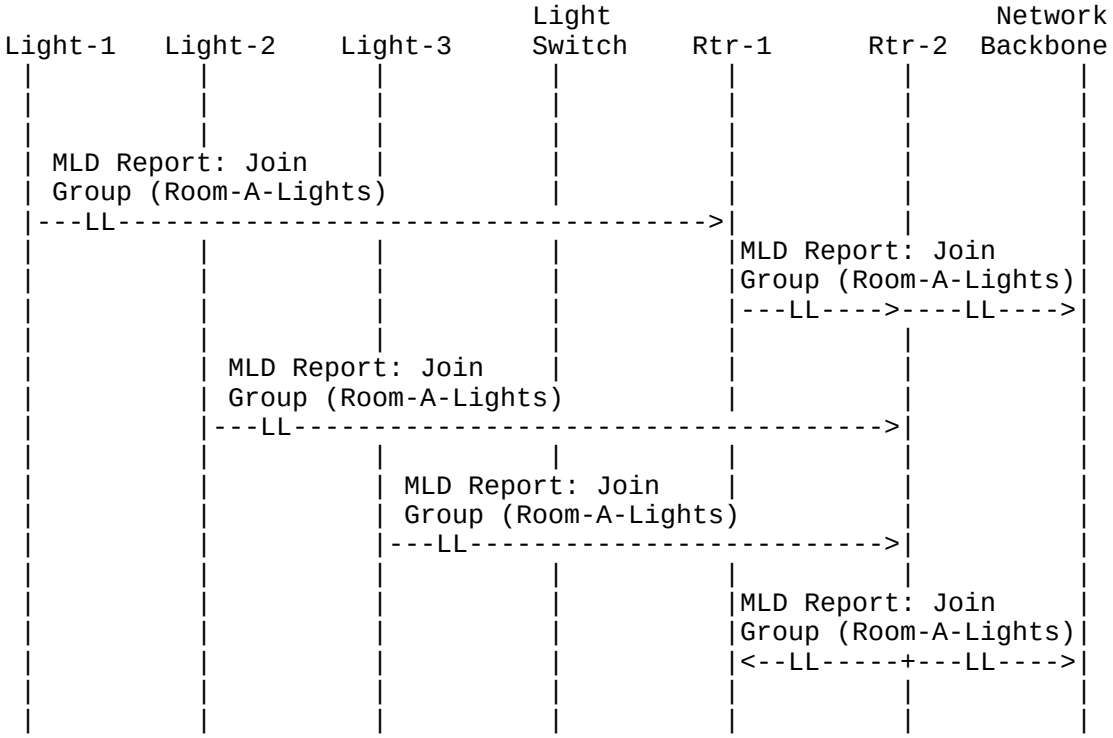



Figure 6: Joining Lighting Groups Using MLD

## 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 unicast 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 2.6.2.2.

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 Destination Advertisement Object (DAO) messages and routing of multicast IPv6 packets based on this. It requires the RPL Mode of Operation 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. In the above example, this could be done through the following three steps: 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 Sections 8.1 and 9.1 of [RFC7252], CoAP group communication based on IP multicast:

- Will operate in CoAP NoSec (No Security) mode, until a future group security solution is developed (see also Section 5.3.3).

- Will use the "coap" scheme. The "coaps" scheme should only be used when a future group security solution is developed (see also Section 5.3.3).

Essentially the above configuration means that there is currently no security at the CoAP layer for group communication. Therefore, for sensitive and mission critical applications (e.g., health monitoring systems, alarm monitoring systems) it is currently recommended to deploy CoAP group communication with an application-layer security mechanism (e.g, data object security) for improved security.

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

Without application-layer security, CoAP group communication should only be currently deployed in non-critical applications (e.g., read-only temperature sensors). Only when security solutions at the CoAP layer are mature enough (see Section 5.3.3) should CoAP group communication without application-layer security be considered for sensitive and mission-critical applications.

## 5.2. Threats

As noted above, 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 Section 11 of [RFC7252] for IP multicast.

## 5.3. Threat Mitigation

Section 11 of [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, security enhancements need to be developed at IETF for group communications. This will allow introduction of a secure mode of CoAP group communication, and use of the "coaps" scheme for that purpose.

At the time of writing of this specification, there are various approaches being considered for security enhancements for group communications. Specifically, a lot of the current effort at IETF is geared towards developing a DTLS-based group communication. This is primarily motivated by the fact that the unicast CoAP security is DTLS-based (Section 9.1 of [RFC7252]. For example, [I-D.keoh-dice-multicast-security] proposes a DTLS-based IP multicast security. However, it is too early to conclude if this is the best approach. Alternatively, [I-D.mglt-dice-ipsec-for-application-payload] proposes an IPSec-based IP multicast security. This approach also needs further investigation and validation.

### 5.4. Monitoring Considerations

#### 5.4.1. General Monitoring

CoAP group communication is meant to be used to control a set of related devices (e.g., simultaneously turn on all the lights in a room). This intrinsically exposes the group to some unique monitoring risks that solitary devices (i.e., devices not in a group) are not as vulnerable to. For example, assume an attacker is able to physically see a set of lights turn on in a room. Then the attacker can correlate a CoAP group communication message to that easily observable coordinated group action even if the contents of the message are encrypted by a future security solution (see Section 5.3.3). This will give the attacker side channel information to plan further attacks (e.g. by determining the members of the group then some network topology information may be deduced).

One mitigation to group communication monitoring risks that should be explored in the future is methods to de-correlate coordinated group actions. For example, if a CoAP group communication GET is sent to all the alarm sensors in a house, then their (unicast) responses should be as de-correlated as possible. This will introduce greater entropy into the system and will make it harder for an attacker to monitor and gather side channel information.

#### 5.4.2. Pervasive Monitoring

A key additional threat consideration for group communication is pointed to by [RFC7258] which warns of the dangers of pervasive monitoring. CoAP group communication solutions which are built on top of IP multicast need to 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 (once future security solutions are developed as in Section 5.3.3) may be 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 nodes for further attacks. For example, the source node (controller) sending out the CoAP group communication messages. 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 2.8.

In the future, even if all the CoAP multicast traffic is encrypted, 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 2.6.2.

## 6.2. New 'coap-group+json' Internet Media Type

This memo registers a new Internet Media Type for CoAP group configuration resource called 'application/coap-group+json'.

(Note to IANA/RFC Editor: This registration follows the guidance from [RFC6838], 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 unicast 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 2.6.2.

Fragment identifier considerations: N/A

Additional Information:

Deprecated alias names for this type: None

Magic number(s): None

File extension(s): *.json

Macintosh file type code(s): TEXT

Person and email address to contact for further information: Esko Dijk ("Esko.Dijk@Philips.com")

Intended usage: COMMON

Restrictions on usage: None

Author: CoRE WG

Change controller: IETF

Provisional registration? (standards tree only): N/A

## 7. Acknowledgements

Thanks to Jari Arkko, Peter Bigot, Anders Brandt, Ben Campbell, Angelo Castellani, Alissa Cooper, Spencer Dawkins, Badis Djamaa, Adrian Farrel, Stephen Farrell, Thomas Fossati, Brian Haberman, Bjoern Hoehrmann, Matthias Kovatsch, Guang Lu, Salvatore Loreto, Kerry Lynn, Andrew McGregor, Kathleen Moriarty, Pete Resnick, Dale Seed, Martin Stiemerling, Zach Shelby, Peter van der Stok, Gengyu Wei, and Juan Carlos Zuniga for their helpful comments and discussions that have helped shape this document.

Special thanks to Carsten Bormann and Barry Leiba for their extensive and thoughtful Chair and AD reviews of the document. Their reviews helped to immeasurably improve the document quality.

## 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.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[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.

[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.

[RFC5110] Savola, P., "Overview of the Internet Multicast Routing Architecture", RFC 5110, January 2008.

[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.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.

[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.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[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.

[RFC4605] Fenner, B., He, H., Haberman, B., and H. Sandick, "Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying")", RFC 4605, August 2006.

[RFC5740] Adamson, B., Bormann, C., Handley, M., and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol", RFC 5740, November 2009.

[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-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.ietf-core-observe] Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 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.droms-6man-multicast-scopes]
Droms, R., "IPv6 Multicast Address Scopes", draft-droms-6man-multicast-scopes-02 (work in progress), July 2013.

[I-D.mglt-dice-ipsec-for-application-payload]
Migault, D. and C. Bormann, "IPsec/ESP for Application Payload", draft-mglt-dice-ipsec-for-application-payload-00 (work in progress), July 2014.

## 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-24 to ietf-25:

- Updated with the remaining agreed minor comments from Ben Campbell's GEN-ART review. Specifically, addressed the two comments on section 2.6.2.1 (which was section 2.7.2.1 in rev-21) as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05604.html".

- Updated with the clarification comment from Badis Djamaa in Section 2.3 as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05612.html".

- Minor editorial updates.

Changes from ietf-23 to ietf-24:

- Clarified in section 2.6.1.2 (Configuring Members) that ABNF rules from Section 3.2.2 of [RFC 3986] should be used for the IP address parsing.

- Minor editorial updates.

Changes from ietf-22 to ietf-23:

- Updated requirements language (RFC 2119) to follow Barry Leiba's suggestions #1, #2b, and #2.1 as per "http://www.ietf.org/mail-archive/web/core/current/msg05593.html".

- Clarified that [RFC 7320] implies that also other specifications cannot pre-define URI structure.

- Added MUST to Token re-use time as it is additional specification to CoAP [RFC 7252].

- Clarified use of multicast POSTing in Section 2.4, in response to Jari Arkko's COMMENTs in "http://www.ietf.org/mail-archive/web/core/current/msg05572.html".

- Added to Section 5.1 (Security Configuration) the possibility to use application-layer (data object) security, which enables to use CoAP group communication also for critical applications, pointed out by Jari Arkko's COMMENTs in "http://www.ietf.org/mail-archive/web/core/current/msg05572.html".

- Fixed subtle error in hex string "c00l" to "c001".

- Clarified in Section 2.11 (Exceptions) that CoAP Observe feature does not support group communication as per Jari's Arkko's comment in "http://www.ietf.org/mail-archive/web/core/current/msg05592.html".

- Moved section 2.6 (Member Discovery) into a new subsection as part of 2.7.1 (Membership Configuration - Background).

- Minor editorial updates.

Changes from ietf-21 to ietf-22:

- Updated with comments from IESG review as follows:

  1. Changed Status from Informational to Experimental.

  2. Addressed Brian Haberman's DISCUSS (to put in reference to ASM) in section 1.2 as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05547.html".

  3. Addressed Brian Haberman's DISCUSS (to put in reference to multicast forwarding proxies) in section 2.1 as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05547.html".

  4. Addressed Brian Haberman's DISCUSS (to put in reference to getting port numbers from URIs) in section 2.3 as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05563.html".

  5. Addressed Brian Haberman's DISCUSS (to put in reference to IGMP/MLD API) in section 2.7.2.1, 2.7.2.2, 2.7.2.6 and 2.7.2.7 as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05547.html".

  6. Addressed Brian Haberman's COMMENT (to put in reference to reliable multicast RFC) in section 1.3 as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05545.html".

  7. Addressed Kathleen Moriarty's DISCUSS (to broaden to cover general monitoring) in section 5.4 as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05566.html".

  8. Addressed Martin Stiemerling's DISCUSS (to clearly indicate that the draft introduces new CoAP protocol functionality) in the Abstract and section 1.2 as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05542.html".

  9. Addressed Martin Stiemerling's DISCUSS (to clarify selected requirements language) in section 2.7.2 as called out in

"http://www.ietf.org/mail-archive/web/core/current/msg05542.html". (Note that the other sections are not impacted as they truly are new requirements and not repetition of the CoAP RFC 7252)

10. Addressed Spencer Dawkins' COMMENT as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05557.html".

11. Addressed Alissa Cooper's COMMENT as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05567.html".

12. Addressed selected Stephen Farrell's COMMENTs as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05576.html".

13. Addressed selected Pete Resnick's COMMENTs as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05568.html".

14. Addressed selected Adrian Farrel's COMMENTs as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05565.html".

15. Addressed selected Jari Arkko's COMMENTs as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05572.html".

- Updated with comments from GEN-ART review as follows:

1. Addressed major issue #1 from Ben Campbell's GEN-ART review (about introducing new functionality beyond CoAP RFC 7252) by changing the status of document to Experimental, and updating Abstract and section 2.1 as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05551.html".

2. Addressed major issue #2 from Ben Campbell's GEN-ART review (about giving a stronger recommendation not to use CoAP group communication in many scenarios until stronger security solutions are available) in section 5.1 and section 5.4 as called out in "http://www.ietf.org/mail-archive/web/core/current/msg05551.html".

3. Addressed selected minor issues and nits from Ben Campbell's GEN-ART review comments from "http://www.ietf.org/mail-archive/web/core/current/msg05535.html".

- Various minor editorial updates.

Changes from ietf-20 to ietf-21:

- Updated with comments from AD review by Barry Leiba. The details of the updates can be seen by looking at the WG mailing list from July 26-31, 2014.

- Various minor editorial updates.

Changes from ietf-19 to ietf-20:

- Replaced obsolete reference [RFC 2616] by [RFC 7230].

- Replaced outdated reference draft-ietf-appsawg-uri-get-off-my-lawn by [RFC 7320] and moved to Normative reference.

- Replaced outdated reference draft-ietf-core-coap by [RFC 7252].

- Moved [RFC 1033] to Informative reference.

- Updated to latest revision numbers for informative draft references by regenerating file through xml2rfc tool.

- Re-ran IETF spell check tool and corrected some minor spelling errors.

- Various minor editorial updates.

Changes from ietf-18 to ietf-19:

- Added guideline on Token value re-use in section 2.5.

- 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.

- Added Pervasive Monitoring considerations in section 5.4.

- Various editorial updates for improved readability.

Changes from ietf-17 to ietf-18:

- Extensive editorial updates based on WGLC comments by Thomas Fossati and Gengyu Wei.

- Addressed ticket #361: Added text for single membership PUT section 2.7.2.7 (Updating a single group membership (PUT)).

- 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)).

- Addressed ticket #359: Fixed requirements text for Section 2.7.2.2 (Creating a new multicast group membership (POST)).

- Addressed ticket #358: Fixed requirements text for Section 2.7.2.1 (CoAP-Group Resource Type and Media Type).

- Addressed ticket #357: Added that "IPv6 addresses of other scopes MAY be enabled" in section 2.2 (Group Definition and Naming).

- Various editorial updates for improved readability.

Changes from ietf-16 to ietf-17:

- Added guidelines on joining of IPv6/IPv4 "All CoAP Nodes" multicast addresses (#356).

- Added MUST support default port in case multicast discovery is available.

- In section 2.1 (IP Multicast Background), clarified that IP multicast is not guaranteed and referenced a definition of Reliable Group Communication (#355).

- Added section 2.5 (Messages and Responses) to clarify how responses are identified and how Token/MID are used in multicast CoAP.

- 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.

- Updated section 2.6.2 (RESTful Interface for Configuring Group Memberships) to allow deletion of individual group memberships (#354).

- Various editorial updates based on comments by Peter van der Stok. Removed reference to expired draft-vanderstok-core-dna at request of its author.

- Various editorial updates for improved readability.

Changes from ietf-15 to ietf-16:

- In section 2.6.2, changed DELETE in group management interface to a PUT with empty JSON array to clear the list (#345).

- 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).

- Addressed some more editorial comments provided by Carsten Bormann in preparation for WGLC.

- Various editorial updates for improved readability.

Changes from ietf-14 to ietf-15:

- In section 2.2, provided guidance on how implementers should parse URIs for group communication (#339).

- 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).

- 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).

- In section 2.6.2.1, specified that notation of IP address in group membership configuration interface should follow standard notation (#342).

- In section 6.2, "coap-group+json" Media Type encoding simplified to just support UTF-8 (and not UTF-16 and UTF-32) (#344).

- Various editorial updates for improved readability.

Changes from ietf-13 to ietf-14:

- 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.

- Changed "ip" field to "a" in Section 2.6 (#337)

Changes from ietf-12 to ietf-13:

- 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.

- 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:

- 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.

- Various editorial updates for improved readability.

Changes from ietf-10 to ietf-11:

- 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).

- Various editorial updates for improved readability.

Changes from ietf-09 to ietf-10:

- Various editorial updates including:

- Added a fourth option in section 3.3 on ways to obtain the URI path for a group request.

- Clarified use of content format in GET/PUT requests for Configuring Group Membership in Endpoints (in section 3.6).

- Changed reference "draft-shelby-core-resource-directory" to "draft-ietf-core-resource-directory".

- Clarified (in section 3.7) that ACKs are never used for a multicast request (from #296).

- Clarified (in section 5.2/5.2.3) that MPL does not support group membership advertisement.

- Adding introductory paragraph to Scope (section 2.2).

- Wrote out fully the URIs in table section 3.2.

- Reworded security text in section 7.2 (New Internet Media Type) to make it consistent with section 3.6 (Configuring Group Membership).

- Fixed formatting of hyperlinks in sections 6.3 and 7.2.

Changes from ietf-08 to ietf-09:

- 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).

- Addressed final comment from Peter van der Stok to define what "IP stack" meant (#296). Following the lead of CoAP-17, we know refer instead to "APIs such as IPV6_RECVPKTINFO [RFC 3542]".

- Changed text in section 3.4 (Group Methods) to allow multicast POST under specific conditions and highlighting the risks with using it (#328).

- Various editorial updates for improved readability.

Changes from ietf-07 to ietf-08:

- 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).

- Addressed various comments from Peter van der Stok (#296).

- Various editorial updates for improved readability including defining all acronyms.

Changes from ietf-06 to ietf-07:

- 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).

- Clarified semantics (in section 3.6) of group management JSON format (#300).

- Added details of IANA request (in section 7.1) for a new CORE Resource Type called ’core.gp’.

- Clarified that DELETE method (in section 3.6) is also a valid group management operation.

- Various editorial updates for improved readability.

Changes from ietf-05 to ietf-06:

- Added a new section on commissioning flow when using discovery services when end devices discover in which multicast group they are allocated (#295).

- 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).

- Added use case of multicasting controller on the backbone (#279).

- 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).

- 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.

- Clarified relationship of this document to draft-ietf-core-coap in section 2.2 (Scope).

- Removed IPSec related requirement, as IPSec is not part of draft-ietf-core-coap anymore.

- 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) .

- Editorial rewording of section 3.7 (Multicast Request Acceptance and Response Suppression) to make the logic easier to comprehend (parse).

- Various editorial updates for improved readability.

Changes from ietf-04 to ietf-05:

- Added a new section 3.9 (Exceptions) that highlights that IP multicast (and hence group communication) is not always available (#187).

- Updated text on the use of [RFC2119] language (#271) in Section 1.

- Included guidelines on when (not) to use CoAP responses to multicast requests and when (not) to accept multicast requests (#273).

- Added guideline on use of core-block for minimizing response size (#275).

- Restructured section 6 (Security Considerations) to more fully describe threats and threat mitigation (#277).

- Clearly indicated that DNS resolution and reverse DNS lookup are optional.

- 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.

- Removed repetitive text about the fact that group communication is not guaranteed.

- 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).

- 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).

- Extended group resource manipulation guidelines with use of pre-configured ports/paths for the multicast group.

- Consolidated all text relating to ports in a new section 3.3 (Port Configuration).

- 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).

- 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:

- 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).
- Added reference to draft-keoh-tls-multicast-security to section 6 (Security Considerations).
- 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).
- Deleted section 8 (Conclusions) as it is redundant (#268).
- Simplified light switch use case (#269) by splitting into basic operations and additional functions (#269).
- Moved section 3.7 (CoAP Multicast and HTTP Unicast Interworking) to draft-dijk-core-groupcomm-misc (#270).
- 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).
- Focus section 3.5 (Configuring Group Membership) on a single proposed solution.
- Scope of section 5.3 (Use of MLD) widened to multicast destination advertisement methods in general.
- Rewrote section 2.2 (Scope) for improved readability.
- Moved use cases that are not addressed to draft-dijk-core-groupcomm-misc.
- Various editorial updates for improved readability.

Changes from ietf-02 to ietf-03:

- Clarified that a group resource manipulation may return back a mixture of successful and unsuccessful responses (section 3.4 and Figure 6) (#251).

- Clarified that security option for group communication must be NoSec mode (section 6) (#250).
- Added mechanism for group membership configuration (#249).
- 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).
- Made the definition of 'group' more specific to group of CoAP endpoints and included text on UDP port selection (#186).
- Added explanatory text in section 3.4 regarding why not to use group communication for non-idempotent messages (i.e., CoAP POST) (#186).
- Changed link-local RD discovery to site-local in RD discovery use case to make it more realistic.
- Fixed lighting control use case CoAP proxying; now returns individual CoAP responses as in coap-12.
- Replaced link format I-D with RFC6690 reference.
- Various editorial updates for improved readability

Changes from ietf-01 to ietf-02:

- Rewrote congestion control section based on latest CoAP text including Leisure concept (#188)
- Updated the CoAP/HTTP interworking section and example use case with more details and use of MLD for multicast group joining
- Key use cases added (#185)
- References to draft-vanderstok-core-dna and draft-castellani-core-advanced-http-mapping added
- Moved background sections on "MLD" and "CoAP-Observe" to Appendices
- Removed requirements section (and moved it to draft-dijk-core-groupcomm-misc)
- Added details for IANA request for group communication multicast addresses

- Clarified text to distinguish between "link local" and general multicast cases
- Moved lengthy background section 5 to draft-dijk-core-groupcomm-misc and replaced with a summary
- Various editorial updates for improved readability
- Change log added

Changes from ietf-00 to ietf-01:

- Moved CoAP-observe solution section to section 2
- Editorial changes
- Moved security requirements into requirements section
- Changed multicast POST to PUT in example use case
- Added CoAP responses in example use case

Changes from rahman-07 to ietf-00:

- Editorial changes
- Use cases section added
- CoRE Resource Directory section added
- Removed section 3.3.5. IP Multicast Transmission Methods
- Removed section 3.4 Overlay Multicast
- Removed section 3.5 CoAP Application Layer Group Management
- Clarified section 4.3.1.3 RPL Routers with Non-RPL Hosts case
- References added and some normative/informative status changes

## Authors' Addresses

Akbar Rahman (editor)
InterDigital Communications, LLC

Email: Akbar.Rahman@InterDigital.com

Esko Dijk (editor)
Philips Research

Email: esko.dijk@philips.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: April 26, 2015

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
Alcatel-Lucent
E. Dijk
Philips Research
October 23, 2014

# Guidelines for HTTP-CoAP Mapping Implementations
# draft-ietf-core-http-mapping-05

## 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 April 26, 2015.

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 . . . . . 21
6.5. Cache Refresh via Observe . . . . . 21
6.6. Use of CoAP Blockwise Transfer . . . . . 22
6.7. Security Translation . . . . . 23
6.8. Other guidelines . . . . . 23
7. IANA Considerations . . . . . 24
8. Security Considerations . . . . . 24
8.1. Traffic overflow . . . . . 24
8.2. Handling Secured Exchanges . . . . . 25
8.3. URI Mapping . . . . . 25
9. Acknowledgements . . . . . 26
10. References . . . . . 26
10.1. Normative References . . . . . 26
10.2. Informative References . . . . . 27
Appendix A. Change Log . . . . . 28
Authors' Addresses . . . . . 29

## 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:

- Section 2 describes terminology to identify proxy types, mapping approaches and proxy deployments;
- Section 3 discusses how URIs refer to resources independent of access protocols;
- Section 4 briefly lists use cases in which HTTP clients need to contact CoAP servers;
- Section 5 introduces a default HTTP-to-CoAP URI mapping syntax;
- Section 6 describes the properties of the HTTP-to-CoAP reverse proxy;
- Section 8 discusses possible security impact related to HTTP-CoAP protocol mapping.

## 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].

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).

## 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: A 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 on the smartphone can provide a friendly 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:

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

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

- the default mechanism to map a CoAP URI into a HTTP URI;
- the URI template format to express a class of CoAP-HTTP URI mapping functions;
- 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 IPv6address, 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 "hct" 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

- 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"
```

- 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:

- 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"
```

- 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"}]
```

- 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"
```

- 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","hct":"{+cu}"},
        {"href":"/hc/secure","rt":"core.hc","hct":"{+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 may 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.)

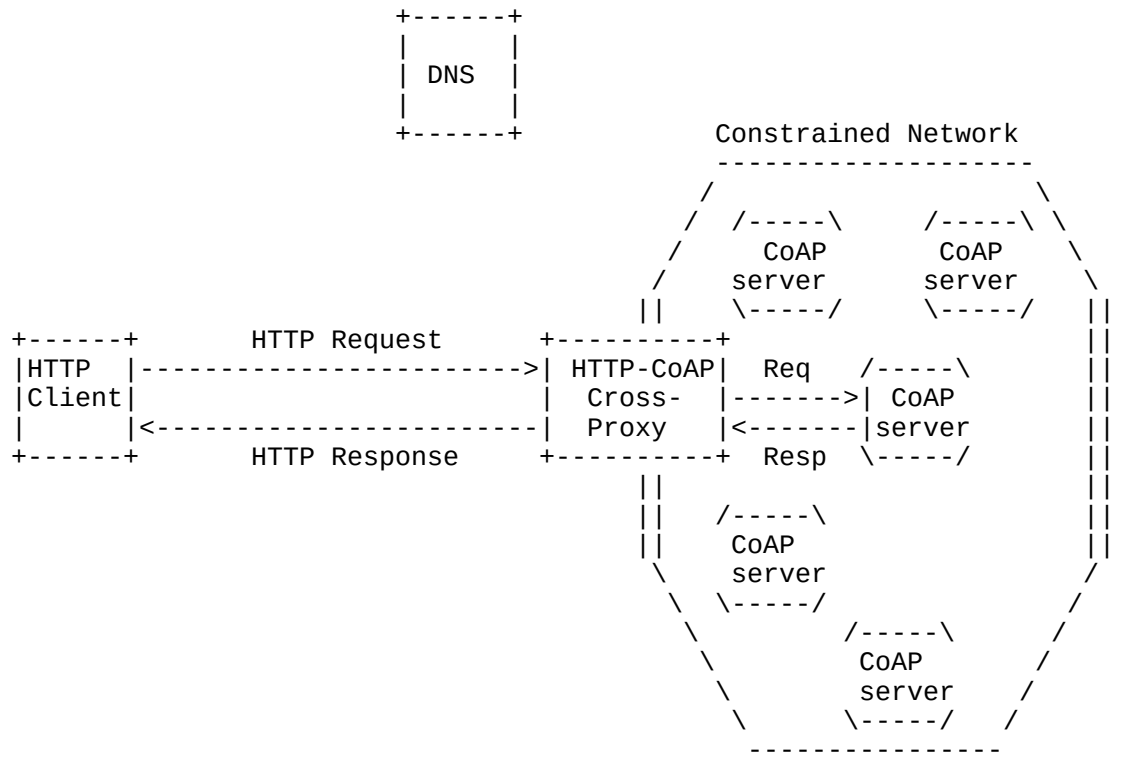



Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

## 6.2. Response Code Translations

Table 1 defines 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 HTTP 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:

- C-T, C-E, and C-F stand for the values of the Content-Type (or Accept) HTTP header, Content-Encoding (or Accept-Encoding) HTTP header, and Content-Format CoAP option respectively.
- If C-E is not given it is assumed to be "identity".
- 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

#### 6.3.3.1. General

Payload content transcoding (e.g. 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.

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 Efficient XML Interchange (EXI) format, or a JSON-encoded resource into CBOR [RFC7049], effectively achieving compression without losing any information.

However, it should be noted that in certain cases, transcoding can lose information in a non-obvious manner. For example, encoding an XML document using schema-informed EXI encoding leads to a loss of information when the destination does not know the exact schema version used by the encoder. So whenever the HC Proxy transcodes an application/XML to application/EXI in-band meta data could be lost. Therefore, the implementer should always carefully verify such lossy payload transformations before triggering the transcoding.

#### 6.3.3.2. CoRE Link Format

The CoRE Link Format [RFC6690] is a set of links (i.e., URIs and their formal relationships) which is carried as content payload in a CoAP response. These links usually include CoAP URIs that might be translated by the HC proxy to the correspondent HTTP URIs using the implemented URI mapping function (see Section 5). Such a process would inspect the forwarded traffic and attempt to re-write the body of resources with an application/link-format media type, mapping the embedded CoAP URIs to their HTTP counterparts. Some potential issues with this approach are:

1. Tampering with payloads is incompatible with resources that are integrity protected (although this is a problem with transcoding in general).

2. The HC proxy needs to fully understand [RFC6690] syntax and semantics, otherwise there is a inherent risk to corrupt the payloads.

Therefore, CoRE Link Format payload should only be transcoded at the risk and discretion of the proxy implementer.

#### 6.3.3.3. Diagnostic Messages

CoAP responses may, in certain error cases, contain a diagnostic message in the payload explaining the error situation, as described in Section 5.5.2 of [RFC7252]. In this scenario, the CoAP response diagnostic payload MUST NOT be returned as the regular HTTP payload (message body). Instead, the CoAP diagnostic payload should be used as the HTTP reason-phrase (of the HTTP status line as defined in Section 3.1.2 of [RFC7230] ) without any alterations.

### 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-15 (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.

[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-25 (work in progress), September 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-04 to ietf-05:

- Addressed Ticket #366 (Mapping of CoRE Link Format payloads to be valid in HTTP Domain?) in section 6.3.3.2 (Content Transcoding - CORE Link Format);
- Addressed Ticket #375 (Add requirement on mapping of CoAP diagnostic payload) in section 6.3.3.3 (Content Transcoding - Diagnostic Messages);
- Addressed comment from Yusuke (http://www.ietf.org/mail-archive/web/core/current/msg05491.html) in section 6.3.3.1 (Content Transcoding - General);
- Various editorial improvements.

Changes from ietf-03 to ietf-04:

- Expanded use case descriptions in Section 4;
- Fixed/enhanced discovery examples in Section 5.4.1;
- 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);
- Updated HTTPBis WG draft references to recently published RFC numbers.
- Various editorial improvements.

Changes from ietf-02 to ietf-03:

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

Changes from ietf-01 to ietf-02:

- Selection of single default URI mapping proposal as proposed to WG mailing list 2013-10-09.

Changes from ietf-00 to ietf-01:

- Added URI mapping proposals to Section 4 as per the Email proposals to WG mailing list from Esko.

## Authors' Addresses

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

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

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

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

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

Email: thomas.fossati@alcatel-lucent.com

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

Email: esko.dijk@philips.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 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

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:

- Canonical mapping
  - lossless round-tripping with [RFC6690]
  - but not trying for bit-preserving (DER-style) round-tripping
- The simplest thing that could possibly work
  - Do not cater for RFC 5988 complications caused by HTTP header character set issues [RFC2047]
- 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:

- the outer collection to an array of links
- 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/t123>;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/
t123","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>.

[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: April 30, 2015

K. Hartke
Universitaet Bremen TZI

October 27, 2014

# Observing Resources in CoAP
draft-ietf-core-observe-15

## Abstract

The Constrained Application Protocol (CoAP) is a RESTful application protocol for constrained nodes and networks. The state of a resource on a CoAP server can change over time. This document specifies a simple protocol extension for CoAP that enables CoAP clients to "observe" resources, i.e., to retrieve a representation of a resource and keep this representation updated by the server over a period of time. The protocol follows a best-effort approach for sending new representations to clients and provides eventual consistency between the state observed by each client and the actual resource state at the server.

## Status of This Memo

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

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

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

This Internet-Draft will expire on April 30, 2015.

## Copyright Notice

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 . . . . . 13
4.3. Caching . . . . . 13
4.4. Reordering . . . . . 14
4.5. Transmission . . . . . 15
5. Intermediaries . . . . . 18
6. Web Linking . . . . . 19
7. Security Considerations . . . . . 19
8. IANA Considerations . . . . . 20
9. Acknowledgements . . . . . 20
10. References . . . . . 21
10.1. Normative References . . . . . 21
10.2. Informative References . . . . . 21
Appendix A. Examples . . . . . 22
A.1. Client/Server Examples . . . . . 22
A.2. Proxy Examples . . . . . 26
Appendix B. Changelog . . . . . 28

# 1. Introduction

## 1.1. Background

The Constrained Application Protocol (CoAP) [RFC7252] 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.

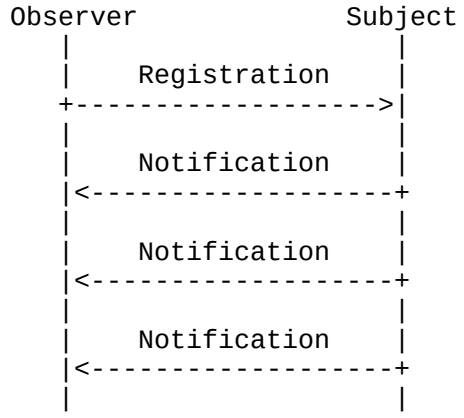



Figure 1: The Observer Design Pattern

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 single extended 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 with the current state upon registration, 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.

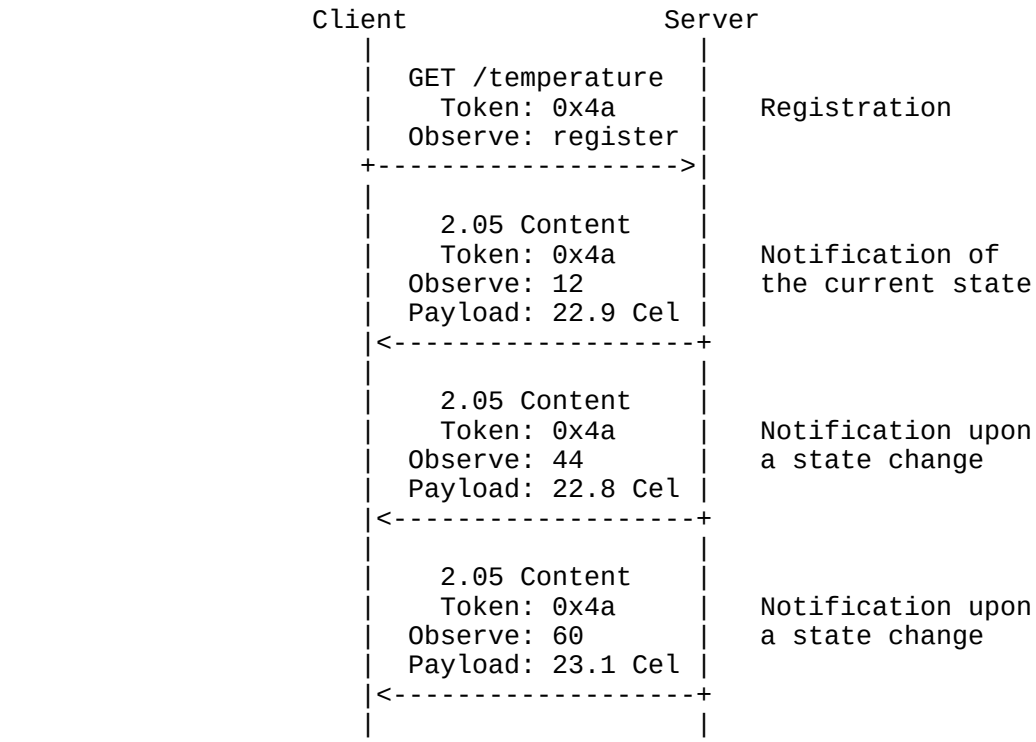



Figure 2: Observing a Resource in CoAP

A client remains on the list of observers as long as the server can determine the client's continued interest in the resource. The server may send a notification in a confirmable CoAP message to request an acknowledgement by the client. 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 by the server from the list of observers.

### 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 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:

- 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. This is limited by congestion control, however, so a client cannot rely on observing every single state that a resource might go through.

- 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.

- 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:

- <coap://server/temperature>, which changes its state every few seconds to a current reading of the temperature sensor;

- <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;

- <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;

- <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, and the response does not include the Observe Option.

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 is identified by all options in the request that are part of the cache-key. This includes, for example, the full request URI and the Accept Option.

### 3.2. Notifications

Notifications are additional responses sent by the server in reply to the single extended GET request that created the registration. Each notification includes the token specified by the client in the request. The only difference between a notification and a normal response is the presence of the Observe Option.

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 GET request (see Section 3.3), notifications can have a 2.03 (Valid) response code rather than a 2.05 (Content) 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 the client's entry 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 reduce collisions 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 consider the notification that was sent most recently as the freshest, regardless of the order of arrival.

To provide an order among notifications for the client, 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 was sent more recently than the freshest notification so far when one of the following conditions is met:

```
(V1 < V2 and V2 - V1 < 2^23) or
(V1 > V2 and V1 - V2 > 2^23) or
(T2 > T1 + 128 seconds)
```

where V1 is the value of the Observe Option in the freshest notification so far, V2 the value of the Observe Option in the incoming notification, T1 a client-local timestamp for the freshest notification so far, and T2 a client-local timestamp for 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 that an incoming notification was sent more recently than the freshest notification it has received so far.

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 of the type used for the request and of 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 process 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 keep this representation updated (as described in Section 1.3) 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 subsequently 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 subsequently 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 keep the resource state that is 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 seconds)/(2^23) = 30.52 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 the CoAP specification [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.)

The protocol follows a best-effort approach for keeping the state observed by a client and the actual resource state at a server in sync. This may have the client and the server become out of sync at times. Depending on the sensitivity of the observed resource, operating on an old state might be a security threat. The client therefore must be careful not to use a representation after its Max-Age expires, and the server must set the Max-Age Option to a sensible value.

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 apply

access controls to 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.

Resources can be observed over DTLS-secured CoAP using any of the security modes described in Section 9 of RFC 7252. The use of DTLS is indicated by the "coaps" URI scheme. All notifications resulting from a GET request with an Observe Option MUST be returned within the same epoch of the same connection as the request.

## 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, Barry Leiba, 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

[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.

[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>.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[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

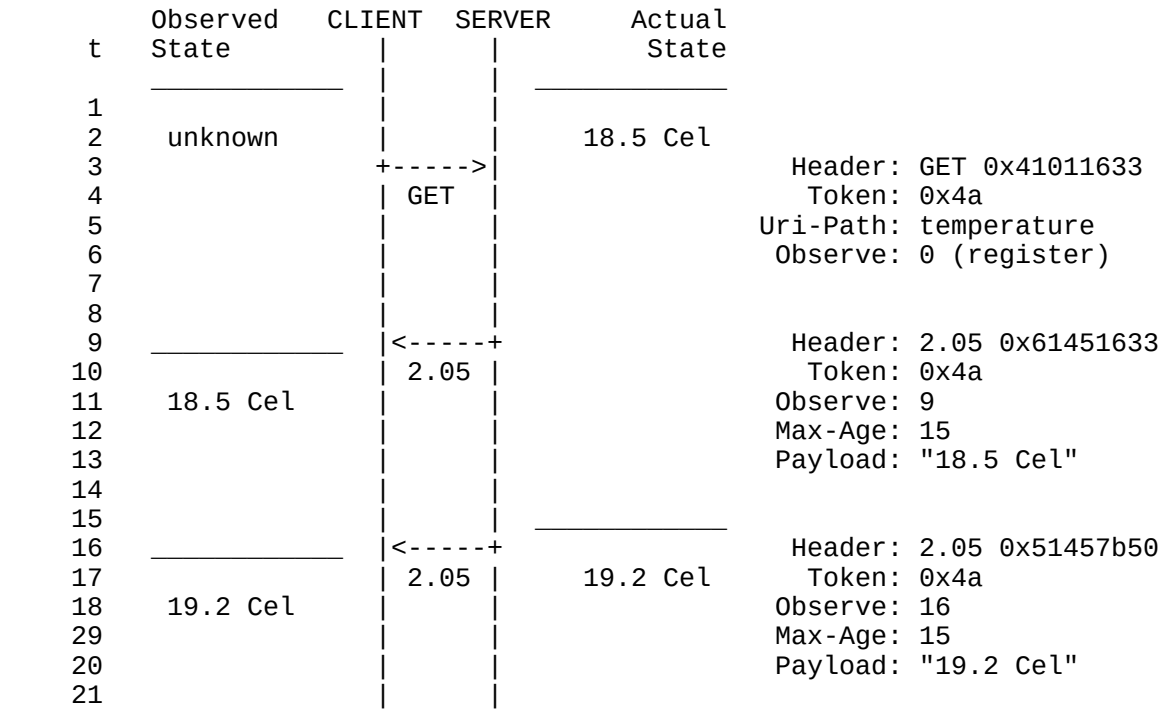



Figure 3: A client registers and receives one notification of the current state and one of a new state upon a state change

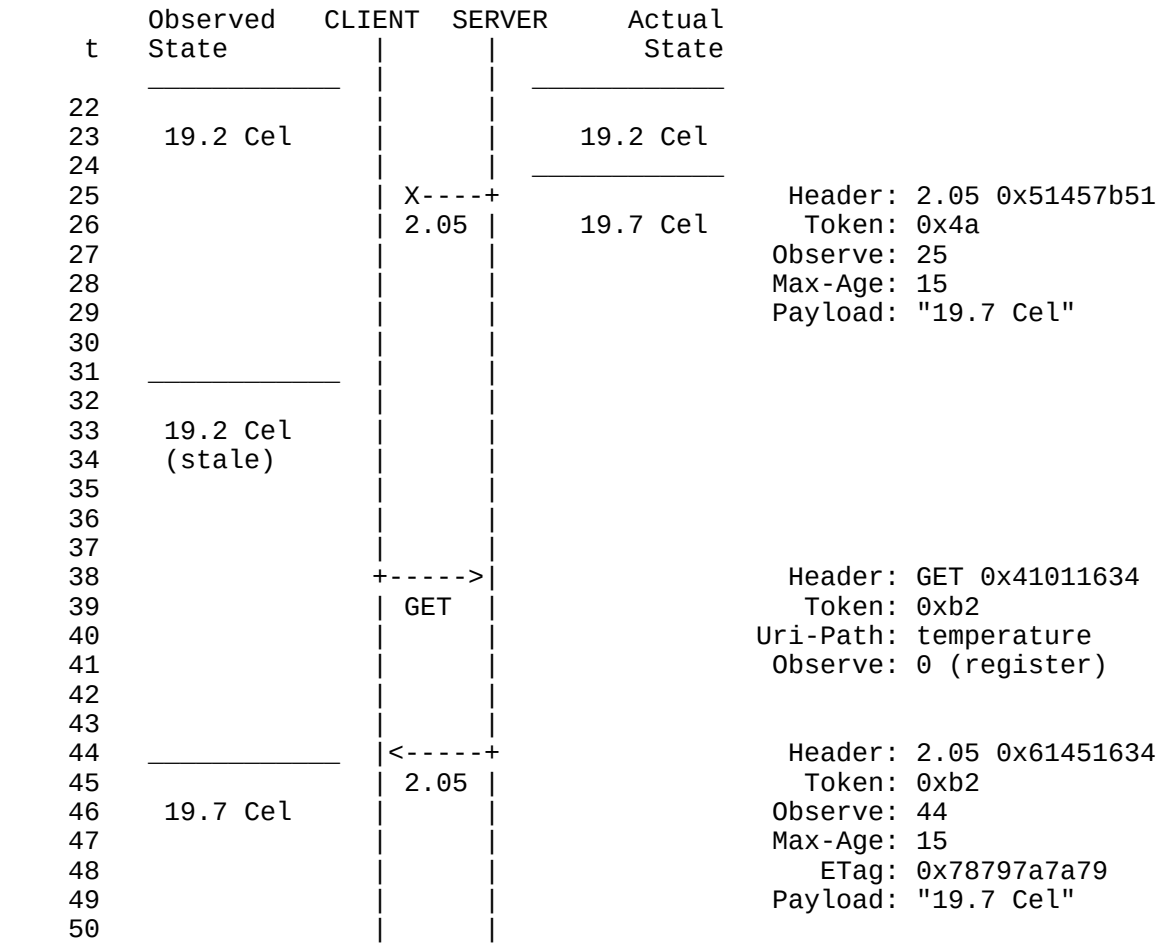



Figure 4: The client re-registers after Max-Age ends

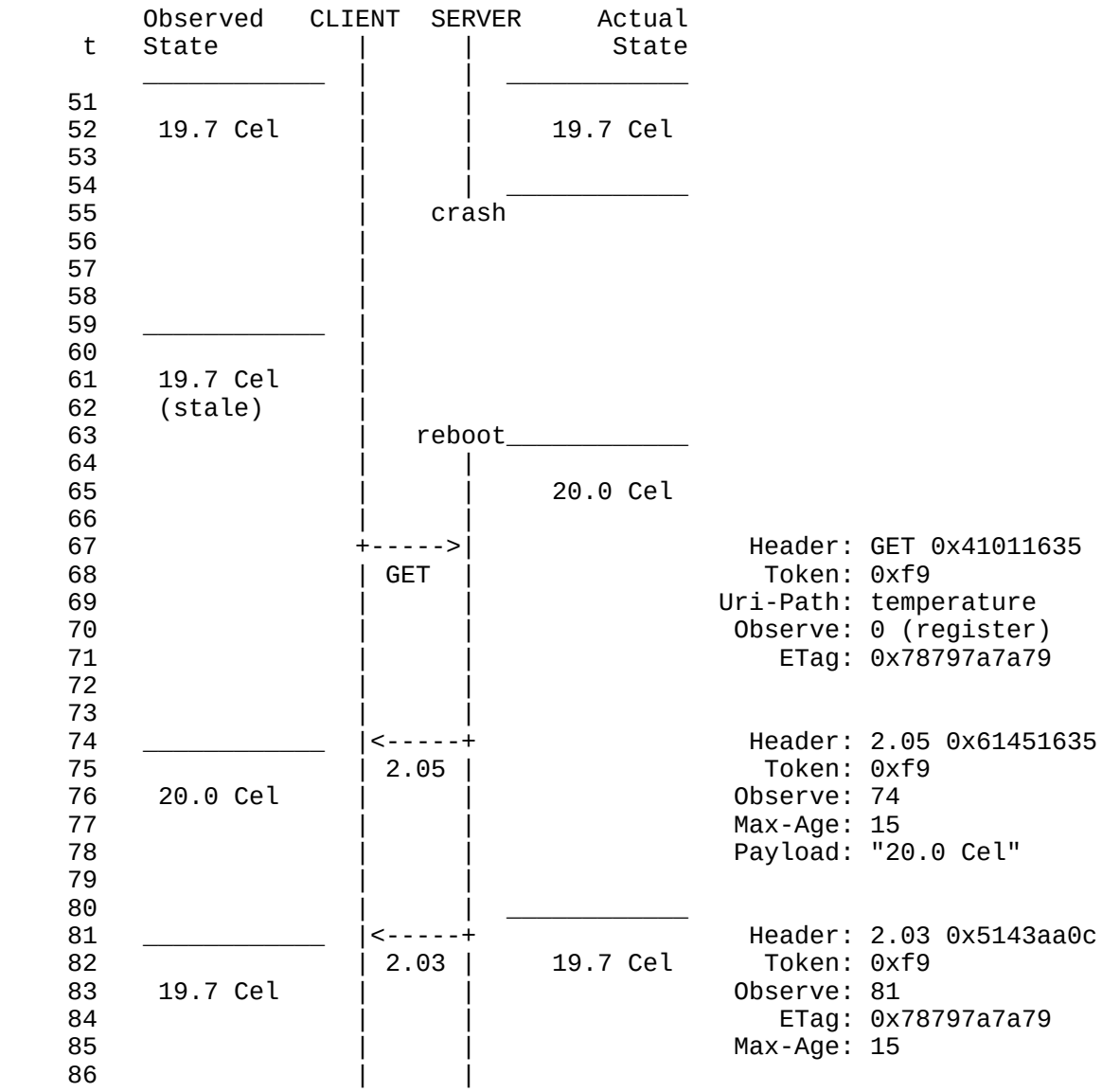



Figure 5: The client re-registers and gives the server the opportunity to select a stored response

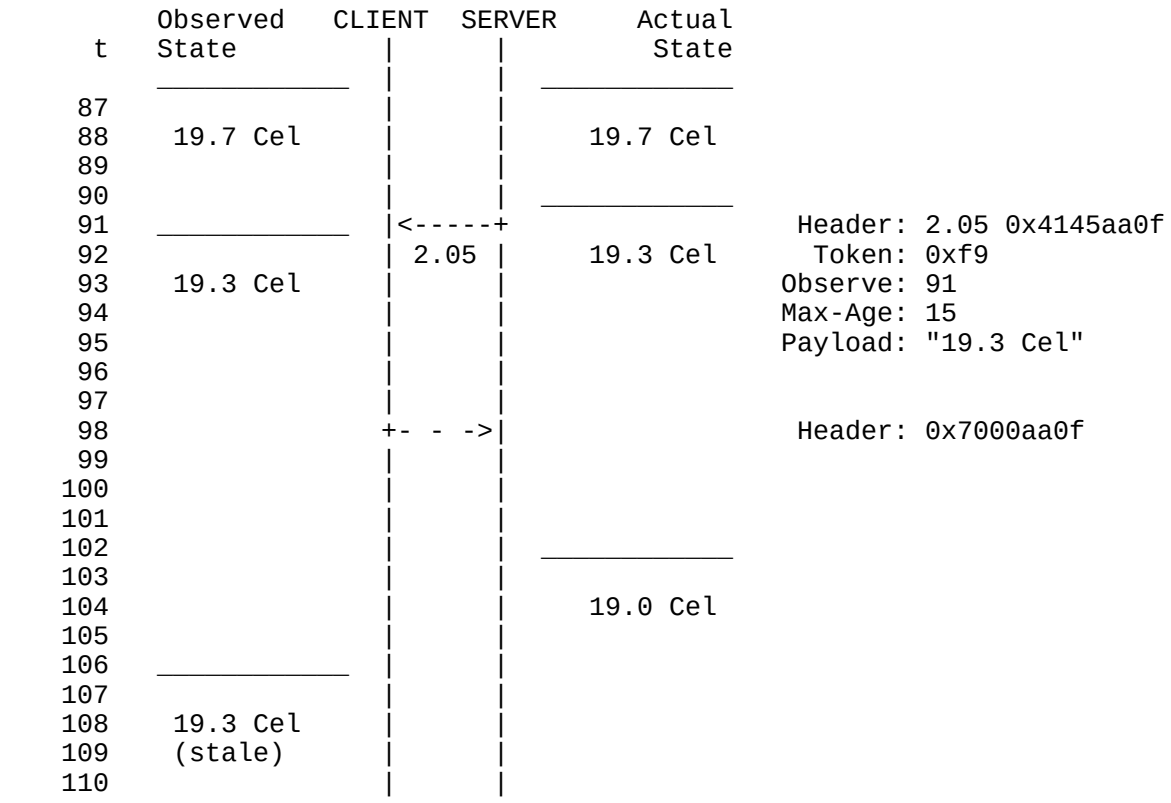



Figure 6: The client rejects a notification and thereby cancels the observation

## A.2. Proxy Examples

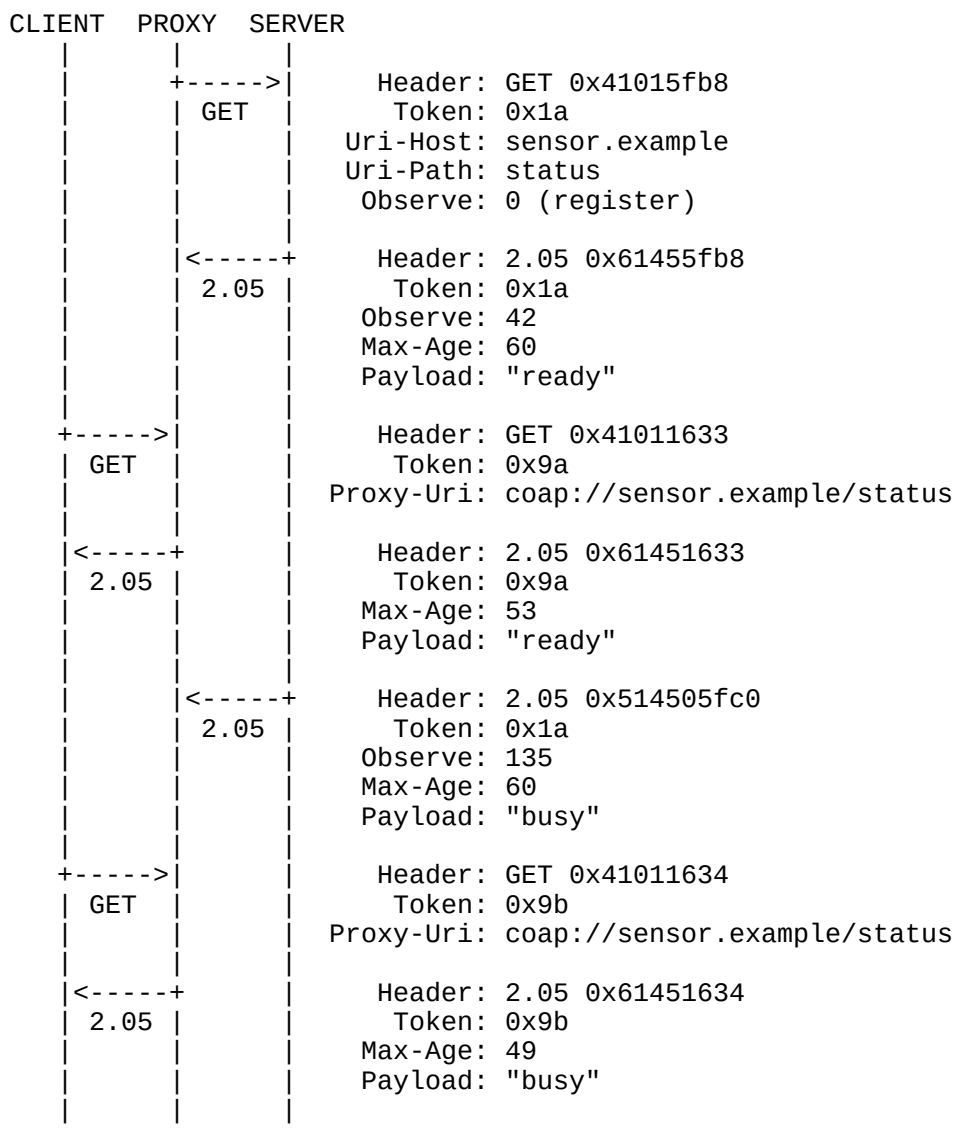



Figure 7: A proxy observes a resource to keep its cache up to date

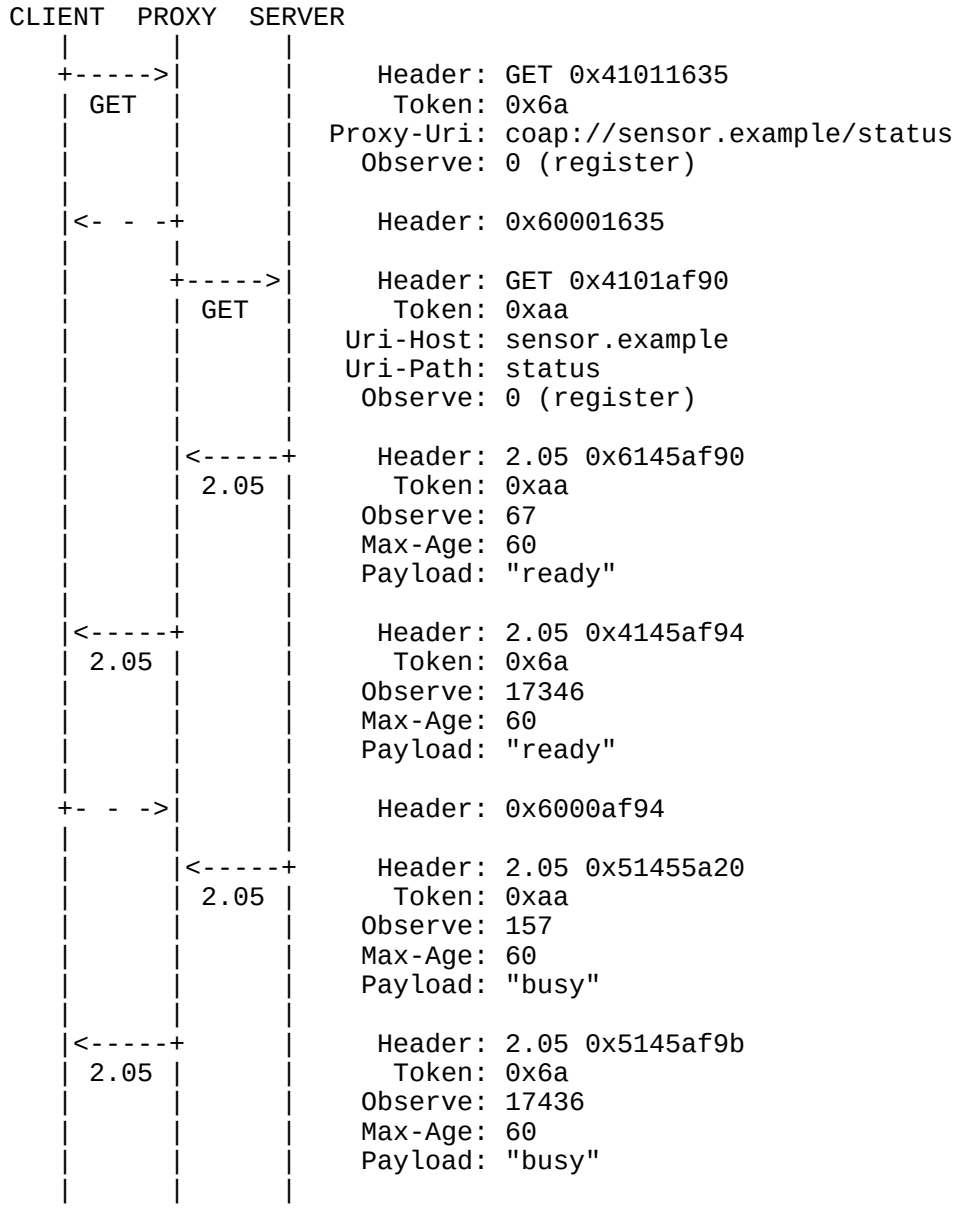



Figure 8: A client observes a resource through a proxy

## Appendix B. Changelog

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

Changes from ietf-14 to ietf-15:

- Clarified several points based on AD, GenART, IESG, and Secdir reviews.

Changes from ietf-13 to ietf-14:

- Updated references.

Changes from ietf-12 to ietf-13:

- 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'.

- Removed the 7.31 Code for cancellation.

Changes from ietf-11 to ietf-12:

- Introduced the 7.31 Code to request the cancellation of a pending request.

- Made the algorithm for superseding an outstanding transmission OPTIONAL.

- 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).

- Simplified the text on cancellation (#352) and the handling of Reset messages (#353).

Changes from ietf-10 to ietf-11:

- 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:

- Required consistent sequence numbers across requests (#333).
- 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.
- 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.
- Improved text on the transmission of non-confirmable notifications to match Section 3.1.2 of RFC 5405 more closely.
- Updated examples to use UCUM units.
- Moved Appendix B into the introduction.

Changes from ietf-08 to ietf-09:

- 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).
- Clarified that a resource (and hence an observation relationship) is identified by the request options that are part of the Cache-Key (#258).
- Clarified that a non-2.xx notification MUST NOT include an Observe Option.
- Moved block-wise transfer of notifications to [I-D.ietf-core-block].

Changes from ietf-07 to ietf-08:

- Expanded text on transmitting a notification while a previous transmission is pending (#242).
- Changed reordering detection to use a fixed time span of 128 seconds instead of EXCHANGE_LIFETIME (#276).

- 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.
- 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.
- 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.
- Added a requirement for sending a confirmable notification at least every 24 hours before continuing with non-confirmable notifications (#221).
- Added congestion control considerations from [I-D.bormann-core-congestion-control-02].
- 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.
- 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.
- Fixed the length of the Observe Option (3 bytes) in the table in Section 2.

- 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.
- Updated examples.

Changes from ietf-06 to ietf-07:

- Moved to 24-bit sequence numbers to allow for up to 15000 notifications per second per client and resource (#217).
- Re-numbered option number to use Unsafe/Safe and Cache-Key compliant numbers (#241).
- Clarified how to react to a Reset message that is sent in reply to a non-confirmable notification (#225).
- Clarified the semantics of the "obs" link target attribute (#236).

Changes from ietf-05 to ietf-06:

- Improved abstract and introduction to say that the protocol is about best effort and eventual consistency (#219).
- Clarified that the value of the Observe Option in a request must have zero length.
- Added requirement that the sequence number must be updated each time a server retransmits a notification.
- Clarified that a server must remove a client from the list of observers when it receives a GET request with an unrecognized critical option.
- Updated the text to use the endpoint concept from [I-D.ietf-core-coap] (#224).
- Improved the reordering text (#223).

Changes from ietf-04 to ietf-05:

- 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).
- Relaxed requirements when sending a Reset message in reply to non-confirmable notifications.

- Added an implementation note about careless GET requests (#184).
- Updated examples.

Changes from ietf-03 to ietf-04:

- Removed the "Max-OFE" Option.
- Allowed a Reset message in reply to non-confirmable notifications.
- Added a section on cancellation.
- Updated examples.

Changes from ietf-02 to ietf-03:

- Separated client-side and server-side requirements.
- 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).
- Simplified the text on message reordering (#129).
- Clarified requirements for intermediaries.
- Clarified the combination of blockwise transfers with notifications (#172).
- Updated examples to show how the state observed by the client becomes eventually consistent with the actual state on the server.
- Added examples for parameterization of observable resource.

Changes from ietf-01 to ietf-02:

- Removed the requirement of periodic refreshing (#126).
- The new "Observe" Option replaces the "Lifetime" Option.
- Introduced a new mechanism to detect message reordering.
- Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:

- Changed terminology from "subscriptions" to "observation relationships" (#33).

- Changed the name of the option to "Lifetime".
- Clarified establishment of observation relationships.
- Clarified that an observation is only identified by the URI of the observed resource and the identity of the client (#66).
- Clarified rules for establishing observation relationships (#68).
- Clarified conditions under which an observation relationship is terminated.
- Added explanation on how clients can terminate an observation relationship before the lifetime ends (#34).
- Clarified that the overriding objective for notifications is eventual consistency of the actual and the observed state (#67).
- Specified how a server needs to deal with clients not acknowledging confirmable messages carrying notifications (#69).
- Added a mechanism to detect message reordering (#35).
- Added an explanation of how notifications can be cached, supporting both the freshness and the validation model (#39, #64).
- Clarified that non-GET requests do not affect observation relationships, and that GET requests without "Lifetime" Option affecting relationships is by design (#65).
- Described interaction with blockwise transfers (#36).
- Added Resource Discovery section (#99).
- Added IANA Considerations.
- Added Security Considerations (#40).
- Added examples (#38).

Author's Address

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
EMail: hartke@tzi.org

CoRE
Internet-Draft
Intended status: Standards Track
Expires: May 13, 2015

Z. Shelby
ARM
C. Bormann
Universitaet Bremen TZI
November 9, 2014

# CoRE Resource Directory
# draft-ietf-core-resource-directory-02

## Abstract

In many M2M applications, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources. This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resources descriptions. Furthermore, new link attributes useful in conjunction with an RD are defined.

## Status of this Memo

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

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

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

This Internet-Draft will expire on May 13, 2015.

Table of Contents

1. Introduction . . . . . 4
2. Terminology . . . . . 4
3. Architecture and Use Cases . . . . . 5
3.1. Use Case: Cellular M2M . . . . . 7
3.2. Use Case: Home and Building Automation . . . . . 7
3.3. Use Case: Link Catalogues . . . . . 8
4. Simple Directory Discovery . . . . . 8
4.1. Finding a Directory Server . . . . . 10
4.2. Third-party registration . . . . . 10
5. Resource Directory Function Set . . . . . 10
5.1. Discovery . . . . . 11
5.2. Registration . . . . . 13
5.3. Update . . . . . 15
5.4. Removal . . . . . 16
6. Group Function Set . . . . . 18
6.1. Register a Group . . . . . 18
6.2. Group Removal . . . . . 20
7. RD Lookup Function Set . . . . . 21
8. New Link-Format Attributes . . . . . 25
8.1. Resource Instance attribute 'ins' . . . . . 25
8.2. Export attribute 'exp' . . . . . 26
9. DNS-SD Mapping . . . . . 26
9.1. DNS-based Service discovery . . . . . 26
9.2. mapping ins to <Instance> . . . . . 27
9.3. Mapping rt to <ServiceType> . . . . . 28
9.4. Domain mapping . . . . . 28
9.5. TXT Record key=value strings . . . . . 28
9.6. Importing resource links into DNS-SD . . . . . 29
10. Security Considerations . . . . . 30
10.1. Endpoint Identification and Authentication . . . . . 30
10.2. Access Control . . . . . 30
10.3. Denial of Service Attacks . . . . . 31
11. IANA Considerations . . . . . 31
11.1. Resource Types . . . . . 31
11.2. Link Extension . . . . . 31
11.3. RD Parameter Registry . . . . . 32
12. Examples . . . . . 32
13. Acknowledgments . . . . . 33
14. Changelog . . . . . 33
15. References . . . . . 35
15.1. Normative References . . . . . 35
15.2. Informative References . . . . . 36
Authors' Addresses . . . . . 36

## 1. Introduction

The work on Constrained RESTful Environments (CoRE) aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC5988]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. This specification however only describes how to discover resources from the web server that hosts them by requesting "/.well-known/core". In many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resource descriptions. Furthermore, new link attributes useful in conjunction with a Resource Directory are defined. Although the examples in this document show the use of these interfaces with CoAP [RFC7252], they can be applied in an equivalent manner to HTTP [RFC7230].

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

Resource Directory
: A web entity that stores information about web resources and implements the REST interfaces defined in this specification for registration and lookup of those resources.

Domain
: In the context of a Resource Directory, a domain is a logical grouping of endpoints. This specification assumes that the list of Domains supported by an RD is pre-configured by that RD. When a domain is exported to DNS, the domain value equates to the DNS domain name.

Group
: In the context of a Resource Directory, a group is a logical grouping of endpoints for the purpose of group communications. All groups within a domain are unique.

Endpoint
: Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the Resource Directory. An endpoint is identified by its endpoint name, which is included during registration, and is unique within the associated domain of the registration.

## 3. Architecture and Use Cases

The resource directory architecture is illustrated in Figure 1. A Resource Directory (RD) is used as a repository for Web Links [RFC5988] about resources hosted on other web servers, which are called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port (called Context), thus a physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain sets of Web Links (called resource directory entries), and for clients to lookup resources from the RD or maintain groups. Endpoints themselves can also act as clients. An RD can be logically segmented by the use of Domains. The domain an endpoint is associated with can be defined by the RD or configured by an outside entity. This information hierarchy is shown in Figure 2.

Endpoints are assumed to proactively register and maintain resource directory entries on the RD, which are soft state and need to be periodically refreshed. An endpoint is provided with interfaces to register, update and remove a resource directory entry. Furthermore, a mechanism to discover an RD using the CoRE Link Format is defined. It is also possible for an RD to proactively discover Web Links from

endpoints and add them as resource directory entries. A lookup interface for discovering any of the Web Links held in the RD is provided using the CoRE Link Format.

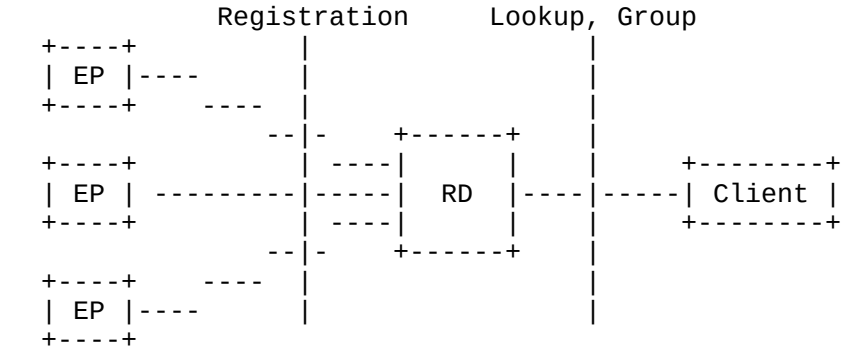



Figure 1: The resource directory architecture.

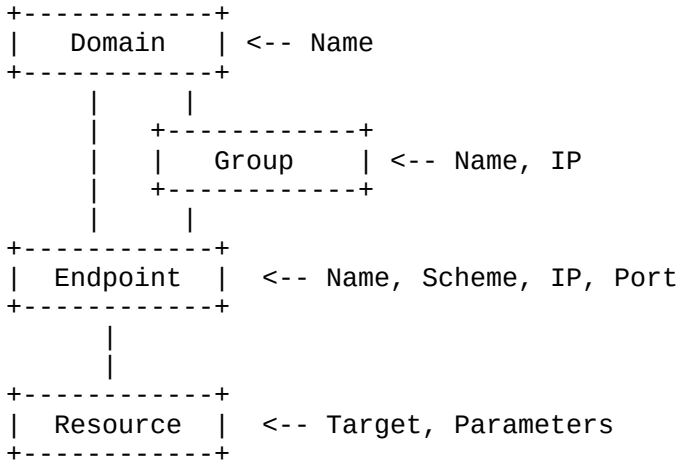



Figure 2: The resource directory information hierarchy.

### 3.1. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded air interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. From the system design point of view, the ambition is to design horizontal solutions that can enable utilization of machines in different applications depending on their current availability and capabilities as well as application requirements, thus avoiding silo like solutions. One of the crucial enablers of such design is the ability to discover resources (machines -- endpoints) capable of providing required information at a given time or acting on instructions from the end users.

In a typical scenario, during a boot-up procedure (and periodically afterwards), the machines (endpoints) register with a Resource Directory (for example EPs installed on vehicles enabling tracking of their position for fleet management purposes and monitoring environment parameters) hosted by the mobile operator or somewhere else in the network, periodically a description of its own capabilities. Due to the usual network configuration of mobile networks, the EPs attached to the mobile network do not have routable addresses. Therefore, a remote server is usually used to provide proxy access to the EPs. The address of each (proxy) endpoint on this server is included in the resource description stored in the RD. The users, for example mobile applications for environment monitoring, contact the RD, look-up the endpoints capable of providing information about the environment using appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server) and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look-up for EPs deployed on the vehicles the application is responsible for.

### 3.2. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of M2M web services. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices.

The exporting of resource information to other discovery systems is also important in these automation applications. In home automation there is a need to interact with other consumer electronics, which may already support DNS-SD, and in building automation larger resource directories or DNS-SD covering multiple buildings.

### 3.3. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge beforehand of who is going to consume the data. Resource Directory can be used to hold links about resources and services hosted anywhere to make them discoverable by a general class of applications.

For example, environmental and weather sensors that generate data for public consumption may provide the data to an intermediary server, or broker. Sensor data are published to the intermediary upon changes or at regular intervals. Descriptions of the sensors that resolve to links to sensor data may be published to a Resource Directory. Applications wishing to consume the data can use the Resource Directory lookup function set to discover and resolve links to the desired resources and endpoints. The Resource Directory service need not be coupled with the data intermediary service. Mapping of Resource Directories to data intermediaries may be many-to-many.

Metadata in link-format or link-format+json representations are supplied by Resource Directories, which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links. External catalogs that are represented in other formats may be converted to link-format or link-format+json for storage and access by Resource Directories. Since it is common practice for these to be URN encoded, simple and lossless structural transforms will generally be sufficient to store external metadata in Resource Directories.

The additional features of Resource Directory allow domains to be defined to enable access to a particular set of resources from particular applications. this provides isolation and protection of sensitive data when needed. Resource groups may defined to allow batched reads from multiple resources.

## 4. Simple Directory Discovery

Not all endpoints hosting resources are expected to know how to implement the Resource Directory Function Set (see Section 5) and thus explicitly register with a Resource Directory (or other such directory server). Instead, simple endpoints can implement the

generic Simple Directory Discovery approach described in this section. An RD implementing this specification MUST implement Simple Directory Discovery. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the endpoint makes available the hosted resources that it wants to be discovered, as links on its "/.well-known/core" interface as specified in [RFC6690].

The endpoint then finds one or more IP addresses of the directory server it wants to know about its resources as described in Section 4.1.

An endpoint that wants to make itself discoverable occasionally sends a POST request to the "/.well-known/core" URI of any candidate directory server that it finds. The body of the POST request is either

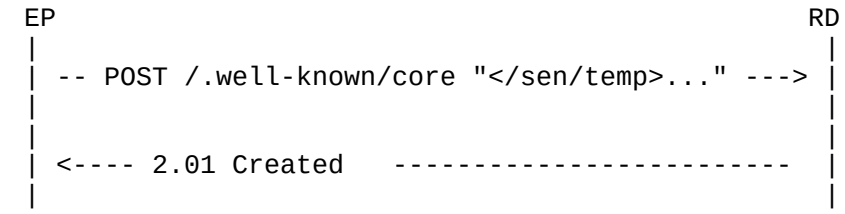
- empty, in which case the directory server is encouraged by this POST request to perform GET requests at the requesting server's default discovery URI.

or

- a non-empty link-format document, which indicates the specific services that the requesting server wants to make known to the directory server.

The directory server integrates the information it received this way into its resource directory. It MAY make the information available to further directories, if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

The following example shows an endpoint using simple resource discovery, by simply sending a POST with its links in the body to a directory.



### 4.1. Finding a Directory Server

Endpoints that want to contact a directory server can obtain candidate IP addresses for such servers in a number of ways.

In a 6LoWPAN, good candidates can be taken from:

- specific static configuration (e.g., anycast addresses), if any,
- the ABRO option of 6LoWPAN-ND [RFC6775],
- other ND options that happen to point to servers (such as RDNSS),
- DHCPv6 options that might be defined later.

In networks with more inexpensive use of multicast, the candidate IP address may be a well-known multicast address, i.e. directory servers are found by simply sending POST requests to that well-known multicast address (details TBD).

As some of these sources are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don’t work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

### 4.2. Third-party registration

For some applications, even Simple Directory Discovery may be too taxing for certain very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the Resource Directory can be filled by a third device, called an installation tool. The installation tool can fill the Resource Directory from a database or other means. For that purpose the scheme, IP address and port of the registered device is indicated in the Context parameter of the registration as well.

## 5. Resource Directory Function Set

This section defines the REST interfaces between an RD and endpoints, which is called the Resource Directory Function Set. Although the examples throughout this section assume the use of CoAP [RFC7252],

these REST interfaces can also be realized using HTTP [RFC7230]. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces defined in this section.

Resource directory entries are designed to be easily exported to other discovery mechanisms such as DNS-SD. For that reason, parameters that would meaningfully be mapped to DNS SHOULD be limited to a maximum length of 63 bytes.

## 5.1. Discovery

Before an endpoint can make use of an RD, it must first know the RD's IP address, port and the path of its RD Function Set. There can be several mechanisms for discovering the RD including assuming a default location (e.g. on an Edge Router in a LoWPAN), by assigning an anycast address to the RD, using DHCP, or by discovering the RD using the CoRE Link Format (see also Section 4.1). This section defines discovery of the RD using the well-known interface of the CoRE Link Format [RFC6690] as the required mechanism. It is however expected that RDs will also be discoverable via other methods depending on the deployment.

Discovery is performed by sending either a multicast or unicast GET request to "/.well-known/core" and including a Resource Type (rt) parameter [RFC6690] with the value "core.rd" in the query string. Likewise, a Resource Type parameter value of "core.rd-lookup" is used to discover the RD Lookup Function Set. Upon success, the response will contain a payload with a link format entry for each RD discovered, with the URL indicating the root resource of the RD. When performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network.

An RD implementation of this specification MUST support query filtering for the rt parameter as defined in [RFC6690].

The discovery request interface is specified as follows:

Interaction: EP -> RD

Method: GET

URI Template: /.well-known/core{?rt}

URI Template Variables:

rt := Resource Type (optional). MAY contain the value "core.rd", "core.rd-lookup", "core.rd-group" or "core.rd*"

Content-Type: application/link-format (if any)

The following response codes are defined for this interface:

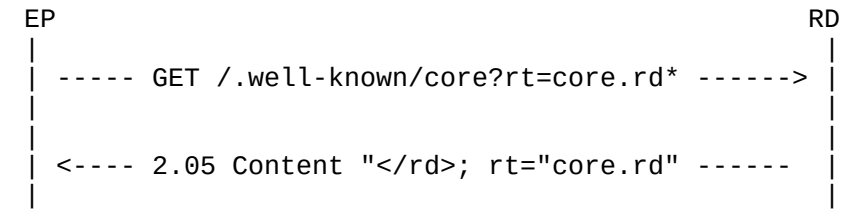
Success: 2.05 "Content" with an application/link-format payload containing one or more matching entries for the RD resource.

Failure: 4.04 "Not Found" is returned in case no matching entry is found for a unicast request.

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

The following example shows an endpoint discovering an RD using this interface, thus learning that the base RD resource is, in this example, at /rd. Note that it is up to the RD to choose its base RD resource, although diagnostics and debugging is facilitated by using the base paths specified here where possible.




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

Res: 2.05 Content
</rd>;rt="core.rd",
</rd-lookup>;rt="core.rd-lookup",
</rd-group>;rt="core.rd-group"

### 5.2. Registration

After discovering the location of an RD Function Set, an endpoint MAY register its resources using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690] or JSON Link Format [I-D.ietf-core-links-json] along with query string parameters indicating the name of the endpoint, its domain and the lifetime of the registration. All parameters except the endpoint name are optional. It is expected that other specifications will define further parameters (see Section 11.3). The RD then creates a new resource or updates an existing resource in the RD and returns its location. An endpoint MUST use that location when refreshing registrations using this interface. Endpoint resources in the RD are kept active for the period indicated by the lifetime parameter. The endpoint is responsible for refreshing the entry within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameter does not create multiple RD entries.

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: /{+rd}{?ep,d,et,lt,con}

URI Template Variables:

rd := RD Function Set path (mandatory). This is the path of the RD Function Set, as obtained from discovery. An RD SHOULD use the value "rd" for this variable whenever possible.

ep := Endpoint (mandatory). The endpoint identifier or name of the registering node, unique within that domain. The maximum length of this parameter is 63 bytes.

d := Domain (optional). The domain to which this endpoint belongs. This parameter SHOULD be less than 63 bytes. Optional. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain. The domain value is needed to export the endpoint to DNS-SD (see Section 9).

et := Endpoint Type (optional). The semantic type of the endpoint. This parameter SHOULD be less than 63 bytes. Optional.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) SHOULD be assumed.

con := Context (optional). This parameter sets the scheme, address and port at which this server is available in the form scheme://host:port. Optional. In the absence of this parameter the scheme of the protocol, source IP address and source port of the register request are assumed. This parameter is mandatory when the directory is filled by a third party such as an installation tool.

Content-Type: application/link-format

Content-Type: application/link-format+json

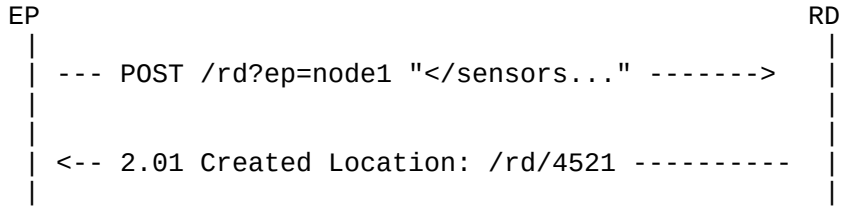
The following response codes are defined for this interface:

Success: 2.01 "Created". The Location header MUST be included with the new resource entry for the endpoint. This Location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration. The resource returned in the Location is only for the purpose of the Update (POST) and Removal (DELETE), and MUST NOT implement GET or PUT methods.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows an endpoint with the name "node1" registering two resources to an RD using this interface. The resulting location /rd/4521 is just an example of an RD generated location.



```
Req: POST coap://rd.example.com/rd?ep=node1
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

Res: 2.01 Created
Location: /rd/4521
```

### 5.3. Update

The update interface is used by an endpoint to refresh or update its registration with an RD. To use the interface, the endpoint sends a POST request to the resource returned in the Location option in the response to the first registration. An update MAY update the lifetime or context parameters if they have changed since the last registration or update. Parameters that have not changed SHOULD NOT be included in an update. Upon receiving an update request, the RD resets the timeout for that endpoint and updates the scheme, IP address and port of the endpoint (using the source address of the update, or the context parameter if present).

An update MAY optionally add or replace links for the endpoint by including those links in the payload of the update as a CoRE Link Format document. Including links in an update message greatly increases the load on an RD and SHOULD be done infrequently. A link is replaced only if both the target URI and relation type match (see Section 10.1).

The update request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: /{+location}{?lt,con}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) SHOULD be assumed.

con := Context (optional). This parameter sets the scheme, address and port at which this server is available in the form scheme://host:port. Optional. In the absence of this parameter the scheme of the protocol, source IP address and source port used to register are assumed. This parameter is compulsory when the directory is filled by a third party such as an installation tool.

Content-Type: application/link-format (optional)

Content-Type: application/link-format+json (optional)

The following response codes are defined for this interface:

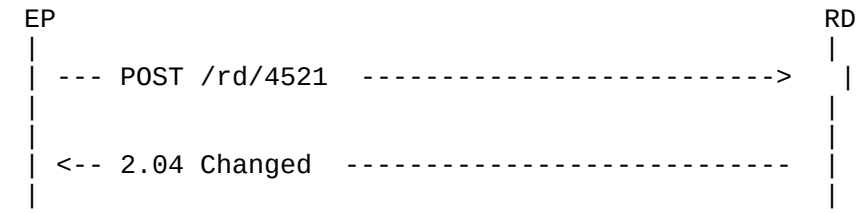
Success: 2.04 "Changed" in the update was successfully processed.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.04 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows an endpoint updating a new set of resources to an RD using this interface.




Req: POST /rd/4521

Res: 2.04 Changed

## 5.4. Removal

Although RD entries have soft state and will eventually timeout after their lifetime, an endpoint SHOULD explicitly remove its entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD

by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful earlier registration.

The following responses codes are defined for this interface:

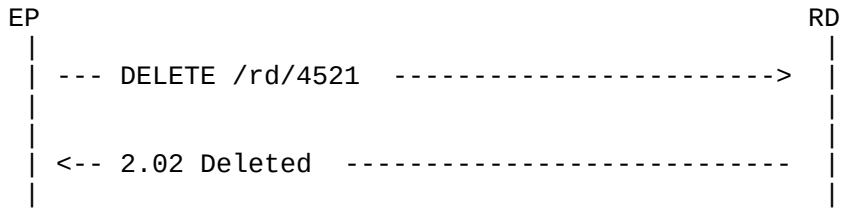
Success: 2.02 "Deleted" upon successful deletion

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.04 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following examples shows successful removal of the endpoint from the RD.




Req: DELETE /rd/4521

Res: 2.02 Deleted

# 6. Group Function Set

This section defines a function set for the creation of groups of endpoints for the purpose of managing and looking up endpoints for group operations. The group function set is similar to the resource directory function set, in that a group may be created or removed. However unlike an endpoint entry, a group entry consists of a list of endpoints and does not have a lifetime associated with it. In order to make use of multicast requests with CoAP, a group MAY have a multicast address associated with it.

## 6.1. Register a Group

In order to create a group, a management entity used to configure groups, makes a request to the RD indicating the name of the group to create (or update), optionally the domain the group belongs to, and optionally the multicast address of the group. The registration message includes the list of endpoints that belong to that group. If an endpoint has already registered with the RD, the RD attempts to use the context of the endpoint from its RD endpoint entry. If the client registering the group knows the endpoint has already registered, then it MAY send a blank target URI for that endpoint link when registering the group. Configuration of the endpoints themselves is out of scope of this specification. Such an interface for managing the group membership of an endpoint has been defined in [I-D.ietf-core-groupcomm].

The registration request interface is specified as follows:

Interaction: Manager -> RD

Method: POST

URI Template: /{+rd-group}{?gp,d,con}

URI Template Variables:

rd-group := RD Group Function Set path (mandatory). This is the path of the RD Group Function Set. An RD SHOULD use the value "rd-group" for this variable whenever possible.

gp := Group Name (mandatory). The name of the group to be created or replaced, unique within that domain. The maximum length of this parameter is 63 bytes.

d := Domain (optional). The domain to which this group belongs. The maximum length of this parameter is 63 bytes. Optional. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain. The domain value is needed to export the endpoint to DNS-SD (see Section 9)

con := Context (optional). This parameter is used to set the IP multicast address at which this server is available in the form scheme://multicast-address:port. Optional. In the absence of this parameter no multicast address is configured. This parameter is compulsory when the directory is filled by an installation tool.

Content-Type: application/link-format

Content-Type: application/link-format+json

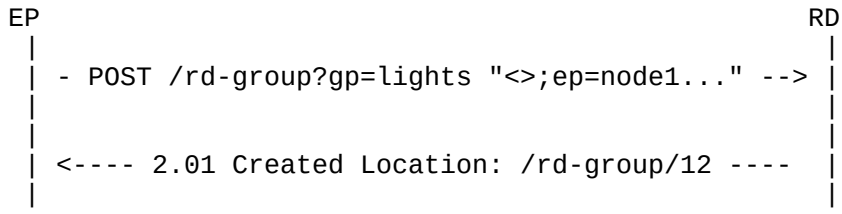
The following response codes are defined for this interface:

Success: 2.01 "Created". The Location header MUST be included with the new group entry. This Location MUST be a stable identifier generated by the RD as it is used for delete operations on this registration.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows a group with the name "lights" registering two endpoints to an RD using this interface. The resulting location /rd-group/12 is just an example of an RD generated group location.



```
Req: POST coap://rd.example.com/rd-group?gp=lights
Payload:
<>;ep="node1",
<>;ep="node2"

Res: 2.01 Created
Location: /rd-group/12
```

### 6.2. Group Removal

A group can be removed simply by sending a removal message to the location returned when registering the group. Removing a group MUST NOT remove the endpoints of the group from the RD.

The removal request interface is specified as follows:

Interaction: Manager -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful group registration.

The following responses codes are defined for this interface:

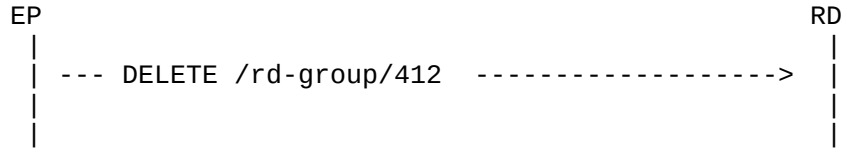
Success: 2.02 "Deleted" upon successful deletion

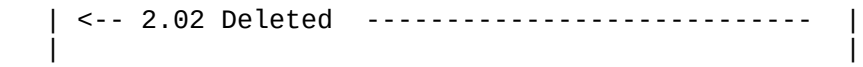
Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.04 "Not Found". Group does not exist.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following examples shows successful removal of the group from the RD.






Req: DELETE /rd-group/12

Res: 2.02 Deleted

## 7. RD Lookup Function Set

In order for an RD to be used for discovering resources registered with it, a lookup interface can be provided using this function set. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. Atom or HTML Link) or using more advanced interfaces (e.g. supporting context or semantic based lookup).

This function set allows lookups for domains, groups, endpoints and resources using attributes defined in the RD Function Set and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Using the Accept Option, the requester can control whether this list is returned in CoRE Link Format ("application/link-format", default) or its JSON form ("application/link-format+json"). The target of these links SHOULD be the actual location of the domain, endpoint or resource, but MAY be an intermediate proxy e.g. in the case of an HTTP lookup interface for CoAP endpoints. Multiple query parameters MAY be included in a lookup, all included parameters MUST match for a resource to be returned. The character '*' MAY be included at the end of a parameter value as a wildcard operator.

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: /{+rd-lookup-base}/
{lookup-type}{?d,ep,gp,et,rt,page,count,resource-param}

Parameters:

rd-lookup-base := RD Lookup Function Set path (mandatory). This is the path of the RD Lookup Function Set. An RD SHOULD use the value "rd-lookup" for this variable whenever possible.

lookup-type := ("d", "ep", "res", "gp") (mandatory) This variable is used to select the kind of lookup to perform (domain, endpoint, resource, or group).

ep := Endpoint (optional). Used for endpoint, group and resource lookups.

d := Domain (optional). Used for domain, group, endpoint and resource lookups.

page := Page (optional). Parameter can not be used without the count parameter. Results are returned from result set in pages that contains 'count' results starting from index (page * count).

count := Count (optional). Number of results is limited to this parameter value. If the parameter is not present, then an RD implementation specific default value SHOULD be used.

rt := Resource type (optional). Used for group, endpoint and resource lookups.

et := Endpoint type (optional). Used for group, endpoint and resource lookups.

resource-param := Link attribute parameters (optional). Any link attribute as defined in Section 4.1 of [RFC6690], used for resource lookups.

The following responses codes are defined for this interface:

Success: 2.05 "Content" with an "application/link-format" or "application/link-format+json" payload containing a matching entries for the lookup.

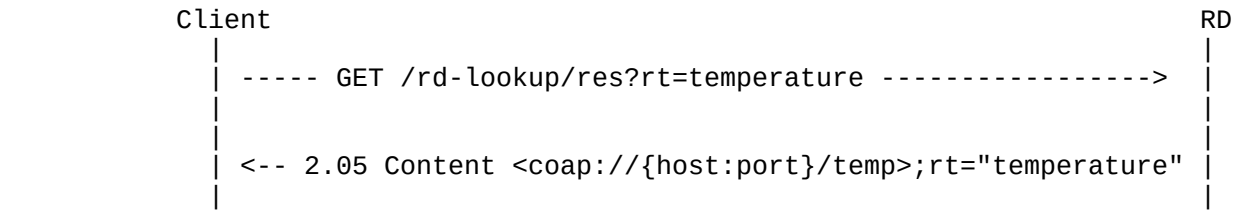
Failure: 4.04 "Not Found" in case no matching entry is found for a unicast request.

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

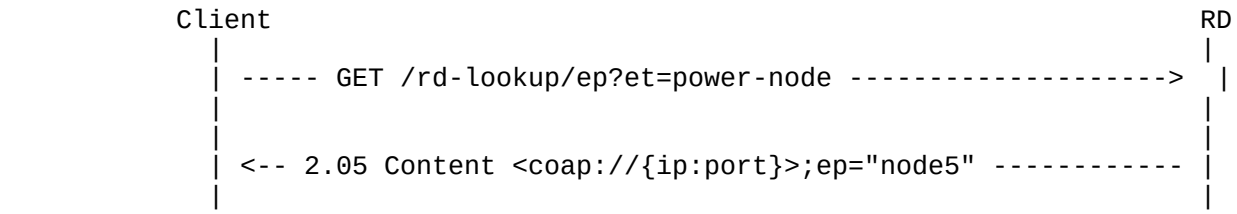
The following example shows a client performing a resource lookup:




Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content
<coap://{host:port}/temp>;rt="temperature"

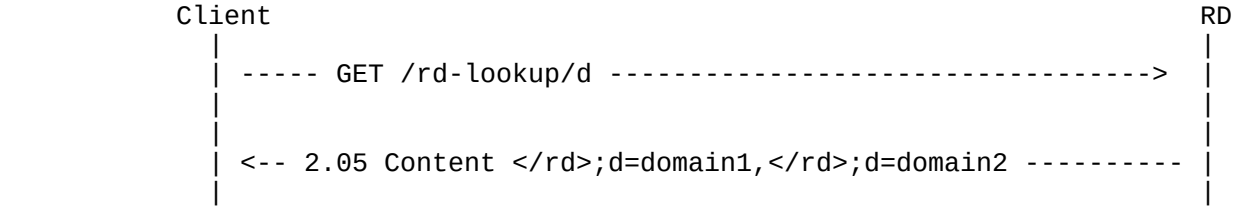
The following example shows a client performing an endpoint lookup:




Req: GET /rd-lookup/ep?et=power-node

Res: 2.05 Content
<coap://{ip:port}>;ep="node5",
<coap://{ip:port}>;ep="node7"

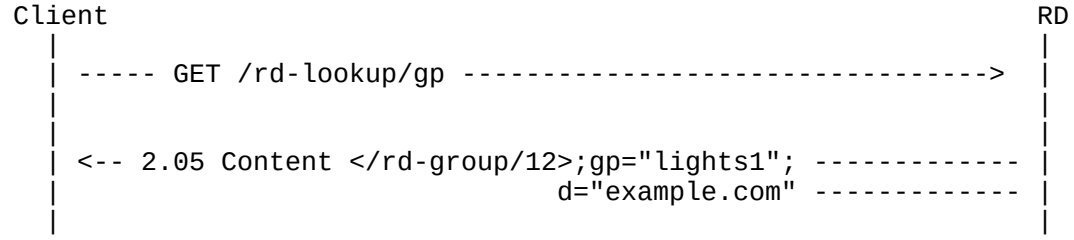
The following example shows a client performing a domain lookup:



Req: GET /rd-lookup/d

Res: 2.05 Content
</rd>;d="domain1",
</rd>;d="domain2"

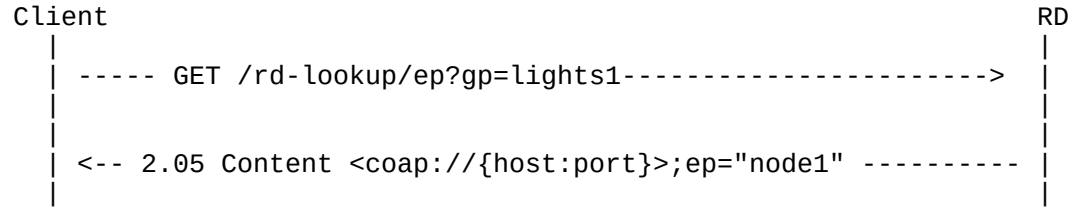
The following example shows a client performing a group lookup for all groups:




Req: GET /rd-lookup/gp

Res: 2.05 Content
</rd-group/12>;gp="lights1";d="example.com"

The following example shows a client performing a lookup for all endpoints in a particular group:

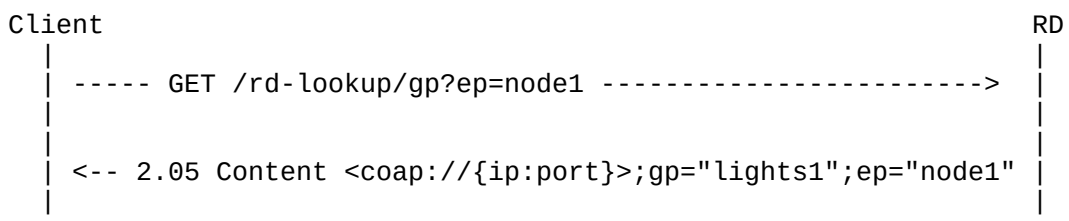



Req: GET /rd-lookup/ep?gp=lights1

Res: 2.05 Content
<coap://{host:port}>;ep="node1",

<coap://{host:port}>;ep="node2",

The following example shows a client performing a lookup for all groups an endpoint belongs to:




Req: GET /rd-lookup/gp?ep=node1

Res: 2.05 Content
<coap://{ip:port}>;gp="lights1";ep="node1",

## 8. New Link-Format Attributes

When using the CoRE Link Format to describe resources being discovered by or posted to a resource directory service, additional information about those resources is useful. This specification defines the following new attributes for use in the CoRE Link Format [RFC6690]:

```
link-extension    = ( "ins" "=" quoted-string ) ; Max 63 bytes
link-extension    = ( "exp" )
```

### 8.1. Resource Instance attribute 'ins'

The Resource Instance "ins" attribute is an identifier for this resource, which makes it possible to distinguish it from other similar resources. This attribute is similar in use to the <Instance> portion of a DNS-SD record (see Section 9.1, and SHOULD be unique across resources with the same Resource Type attribute in the domain it is used. A Resource Instance might be a descriptive string like "Ceiling Light, Room 3", a short ID like "AF39" or a unique UUID or iNumber. This attribute is used by a Resource Directory to distinguish between multiple instances of the same resource type

within the directory.

This attribute MUST be no more than 63 bytes in length. The resource identifier attribute MUST NOT appear more than once in a link description.

### 8.2. Export attribute 'exp'

The Export "exp" attribute is used as a flag to indicate that a link description MAY be exported by a resource directory to external directories.

The CoRE Link Format is used for many purposes between CoAP endpoints. Some are useful mainly locally, for example checking the observability of a resource before accessing it, determining the size of a resource, or traversing dynamic resource structures. However, other links are very useful to be exported to other directories, for example the entry point resource to a functional service.

## 9. DNS-SD Mapping

CoRE Resource Discovery is intended to support fine-grained discovery of hosted resources, their attributes, and possibly other resource relations [RFC6690]. In contrast, service discovery generally refers to a coarse-grained resolution of an endpoint's IP address, port number, and protocol.

Resource and service discovery are complementary in the case of large networks, where the latter can facilitate scaling. This document defines a mapping between CoRE Link Format attributes and DNS-Based Service Discovery [RFC6763] fields that permits discovery of CoAP services by either means.

### 9.1. DNS-based Service discovery

DNS-Based Service Discovery (DNS-SD) defines a conventional method of configuring DNS PTR, SRV, and TXT resource records to facilitate discovery of services (such as CoAP servers in a subdomain) using the existing DNS infrastructure. This section gives a brief overview of DNS-SD; see [RFC6763] for a detailed specification.

DNS-SD service names are limited to 255 octets and are of the form:

Service Name = <Instance>.<ServiceType>.<Domain>.

The service name is the label of SRV/TXT resource records. The SRV RR specifies the host and the port of the endpoint. The TXT RR

provides additional information.

The <Domain> part of the service name is identical to the global (DNS subdomain) part of the authority in URIs that identify servers or groups of servers.

The <ServiceType> part is composed of at least two labels. The first label of the pair is the application protocol name [RFC6335] preceded by an underscore character. The second label indicates the transport and is always "_udp" for UDP-based CoAP services. In cases where narrowing the scope of the search may be useful, these labels may be optionally preceded by a subtype name followed by the "_sub" label. An example of this more specific <ServiceType> is "lamp._sub._dali._udp".

The default <Instance> part of the service name may be set at the factory or during the commissioning process. It SHOULD uniquely identify an instance of <ServiceType> within a <Domain>. Taken together, these three elements comprise a unique name for an SRV/ TXT record pair within the DNS subdomain.

The granularity of a service name MAY be that of a host or group, or it could represent a particular resource within a CoAP server. The SRV record contains the host name (AAAA record name) and port of the service while protocol is part of the service name. In the case where a service name identifies a particular resource, the path part of the URI must be carried in a corresponding TXT record.

A DNS TXT record is in practice limited to a few hundred octets in length, which is indicated in the resource record header in the DNS response message. The data consists of one or more strings comprising a key=value pair. By convention, the first pair is txtver=<number> (to support different versions of a service description).

### 9.2. mapping ins to <Instance>

The Resource Instance "ins" attribute maps to the <Instance> part of a DNS-SD service name. It is stored directly in the DNS as a single DNS label of canonical precomposed UTF-8 [RFC3629] "Net-Unicode" (Unicode Normalization Form C) [RFC5198] text. However, to the extent that the "ins" attribute may be chosen to match the DNS host name of a service, it SHOULD use the syntax defined in Section 3.5 of [RFC1034] and Section 2.1 of [RFC1123].

The <Instance> part of the name of a service being offered on the network SHOULD be configurable by the user setting up the service, so that he or she may give it an informative name. However, the device

or service SHOULD NOT require the user to configure a name before it can be used. A sensible choice of default name can allow the device or service to be accessed in many cases without any manual configuration at all. The default name should be short and descriptive, and MAY include a collision-resistant substring such as the lower bits of the device's MAC address, serial number, fingerprint, or other identifier in an attempt to make the name relatively unique.

DNS labels are currently limited to 63 octets in length and the entire service name may not exceed 255 octets.

### 9.3. Mapping rt to <ServiceType>

The resource type "rt" attribute is mapped into the <ServiceType> part of a DNS-SD service name and SHOULD conform to the reg-rel-type production of the Link Format defined in Section 2 of [RFC6690]. The "rt" attribute MUST be composed of at least a single Net-Unicode text string, without underscore '_' or period '.' and limited to 15 octets in length, which represents the application protocol name. This string is mapped to the DNS-SD <ServiceType> by prepending an underscore and appending a period followed by the "_udp" label. For example, rt="dali" is mapped into "_dali._udp".

The application protocol name may be optionally followed by a period and a service subtype name consisting of a Net-Unicode text string, without underscore or period and limited to 63 octets. This string is mapped to the DNS-SD <ServiceType> by appending a period followed by the "_sub" label and then appending a period followed by the service type label pair derived as in the previous paragraph. For example, rt="dali.light" is mapped into "light._sub._dali._udp".

The resulting string is used to form labels for DNS-SD records which are stored directly in the DNS.

### 9.4. Domain mapping

DNS domains are defined from the "d" attribute.The domain attribute is suffixed to the host name and should be consistent with the domain name attributed to the hosting network segment.

### 9.5. TXT Record key=value strings

A number of [RFC6763] key/value pairs are derived from link-format information, to be exported in the DNS-SD as key=value strings in a TXT record ([RFC6763], Section 6.3).

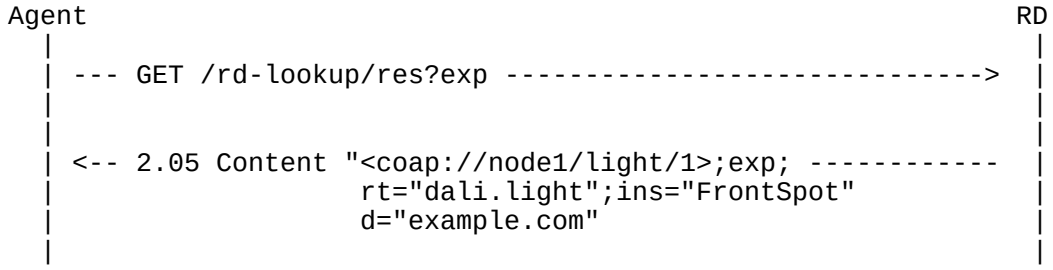
The resource <URI> is exported as key/value pair "path=<URI>".

The Interface Description "if" attribute is exported as key/value pair "if=<Interface Description>".

The DNS TXT record can be further populated by importing any other resource description attributes as they share the same key=value format specified in Section 6 of [RFC6763].

### 9.6. Importing resource links into DNS-SD

Assuming the ability to query a Resource Directory or multicast a GET (?exp) over the local link, CoAP resource discovery may be used to populate the DNS-SD database in an automated fashion. CoAP resource descriptions (links) can be exported to DNS-SD for exposure to service discovery by using the Resource Instance attribute as the basis for a unique service name, composed with the Resource Type as the <ServiceType>, and registered in the correct <Domain>. The agent responsible for exporting records to the DNS zone file SHOULD be authenticated to the DNS server. The following example shows an agent discovering a resource to be exported:




```
Req: GET /rd-lookup/res?exp

Res: 2.05 Content
<coap://[FDFD::1234]:61616/light/1>;
  exp;ct=41;rt="dali.light";ins="FrontSpot";
            d="example.com"
```

The agent subsequently registers the following DNS-SD RRs:

```
node1.example.com.                IN AAAA
                            FDFD::1234
_dali._udp.example.com            IN PTR
                            FrontSpot._dali._udp.example.com
light._sub._dali._udp.example.com IN PTR
                            FrontSpot._dali._udp.example.com
FrontSpot._dali._udp.example.com  IN SRV  0 0 5678
                            node1.example.com.
FrontSpot._dali._udp.example.com  IN TXT
                            txtver=1;path=/light/1
```

In the above figure the Service Name is chosen as FrontSpot._dali._udp.example.com without the light._sub service prefix. An alternative Service Name would be: FrontSpot.light._sub._dali._udp.example.com.

## 10. Security Considerations

The security considerations as described in Section 7 of [RFC5988] and Section 6 of [RFC6690] apply. The "/.well-known/core" resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252]. DTLS or TLS based security SHOULD be used on all resource directory interfaces defined in this document (TODO: Improve the exact DTLS or TLS security requirements and references).

### 10.1. Endpoint Identification and Authentication

An Endpoint is determined to be unique by an RD by the Endpoint identifier parameter included during Registration, and any associated TLS or DTLS security bindings. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint or Client on a resource directory SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security. Endpoints using a Certificate MUST include the Endpoint identifier as the Subject of the Certificate, and this identifier MUST be checked by a resource directory to match the Endpoint identifier included in the Registration message.

### 10.2. Access Control

Access control SHOULD be performed separately for the RD Function Set and the RD Lookup Function Set, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as

fine-grained a level as possible. For example access control for lookups could be performed either at the domain, endpoint or resource level.

### 10.3. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly become part of a DDoS attack as UDP does not require return routability check. Therefore, an attacker can easily spoof the source IP of the target entity and send requests to such a service which would then respond to the target entity. This can be used for large-scale DDoS attacks on the target. Especially, if the service returns a response that is order of magnitudes larger than the request, the situation becomes even worse as now the attack can be amplified. DNS servers have been widely used for DDoS amplification attacks. Recently, it has been observed that NTP Servers, that also run on unprotected UDP have been used for DDoS attacks (http://tools.cisco.com/security/center/content/CiscoSecurityNotice/CVE-2013-5211) [TODO: Ref, and cut down the verbiage, as this is already discussed in RFC 7252] since there is no return routability check and can have a large amplification factor. The responses from the NTP server were found to be 19 times larger than the request. A Resource Directory (RD) which responds to wild-card lookups is potentially vulnerable if run with CoAP over UDP. Since there is no return routability check and the responses can be significantly larger than requests, RDs can unknowingly become part of a DDoS amplification attack. Therefore, it is RECOMMENDED that implementations ensure return routability. This can be done, for example by responding to wild card lookups only over DTLS or TLS or TCP.

## 11. IANA Considerations

### 11.1. Resource Types

"core.rd", "core.rd-group" and "core.rd-lookup" resource types need to be registered with the resource type registry defined by [RFC6690].

### 11.2. Link Extension

The "exp" attribute needs to be registered when a future Web Linking link-extension registry is created (e.g. in RFC5988bis).

### 11.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include the human readable name of the parameter, the query parameter, validity requirements if any and a description. The query parameter MUST be a valid URI query key [RFC3986].

Initial entries in this sub-registry are as follows:

| Name | Query | Validity | Description |
|---|---|---|---|
| Endpoint Name | ep | | Name of the endpoint |
| Lifetime | lt | 60-4294967295 | Lifetime of the registration in seconds |
| Domain | d | | Domain to which this endpoint belongs |
| Endpoint Type | et | | Semantic name of the endpoint |
| Context | con | URI | The scheme, address and port at which this server is available |
| Endpoint Name Group | ep | | Name of the endpoint, max 63 bytes |
| Name | gp | | Name of a group in the RD |
| Page | page | Integer | Used for pagination |
| Count | count | Integer | Used for pagination |

Table 1: RD Parameters

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC5226].

## 12. Examples

## 13. Acknowledgments

Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Peter van der Stok, Anders Brandt, Matthieu Vial, Michael Koster, Mohit Sethi, Sampo Ukkola and Linyi Tian have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his collagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

## 14. Changelog

Changes from -01 to -02:

- Added a catalogue use case.
- Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.
- Additional examples section added for more complex use cases.
- New DNS-SD mapping section.
- Added text on endpoint identification and authentication.
- Error code 4.04 added to Registration Update and Delete requests.
- Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

- Removed the ETag validation feature.
- Place holder for the DNS-SD mapping section.
- Explicitly disabled GET or POST on returned Location.
- New registry for RD parameters.
- Added support for the JSON Link Format.
- Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

- o Updated the version and date.

Changes from -04 to -05:

- o Restricted Update to parameter updates.
- o Added pagination support for the Lookup interface.
- o Minor editing, bug fixes and reference updates.
- o Added group support.
- o Changed rt to et for the registration and update interface.

Changes from -03 to -04:

- o Added the ins= parameter back for the DNS-SD mapping.
- o Integrated the Simple Directory Discovery from Carsten.
- o Editorial improvements.
- o Fixed the use of ETags.

Changes from -02 to -03:

- o Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- o Updated REST interface descriptions to use RFC6570 URI Template format.
- o Introduced an improved RD Lookup design as its own function set.
- o Improved the security considerations section.
- o Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:

- o Added a terminology section.
- o Changed the inclusion of an ETag in registration or update to a MAY.
- o Added the concept of an RD Domain and a registration parameter for it.

o Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.

o Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

## 15. References

### 15.1. Normative References

[I-D.ietf-core-links-json] Bormann, C., "Representing CoRE Link Collections in JSON", draft-ietf-core-links-json-02 (work in progress), July 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.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

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

[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.

[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.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.

## 15.2. Informative References

[I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-25 (work in progress), September 2014.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.

[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.

[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.

## Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
FINLAND

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

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

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

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: March 16, 2015

O. Kleine
University of Luebeck, ITM

September 12, 2014

# CoAP Endpoint Identification
# draft-kleine-core-coap-endpoint-id-01.txt

## Abstract

The Constrained Application Protocol (CoAP) (see [RFC7252]), 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 seperate responses. This holds, e.g. for CON requests that are confirmed by the server with an empty ACK and answered later with a seperate response. According to the CoAP specification the request/response matching is realized using a unique pair of the servers IP address and a token defined by the 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 seperate 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 inRFC 2119 [RFC2119].

## Status of This Memo

## Copyright Notice



## 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 obervations . . . . . 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 . . . . . 9
  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 . . . . . 11
  8.1. Normative References . . . . . 11
  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 one 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 conversion, 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 single 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).

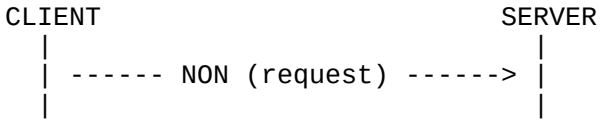



Figure 1: NON request without any response

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

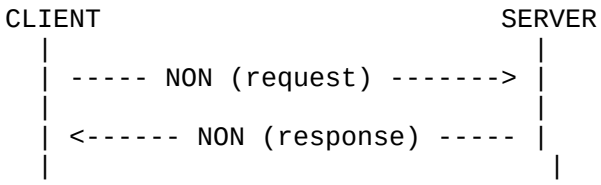



Figure 2: NON requests and NON response

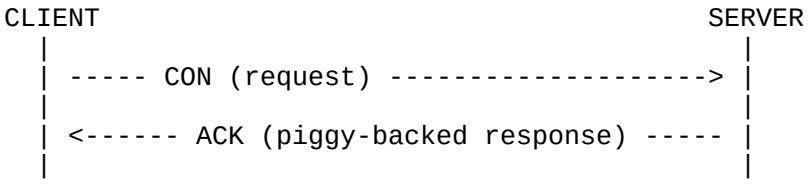



Figure 3: CON request and ACK response (piggy-backed)

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.

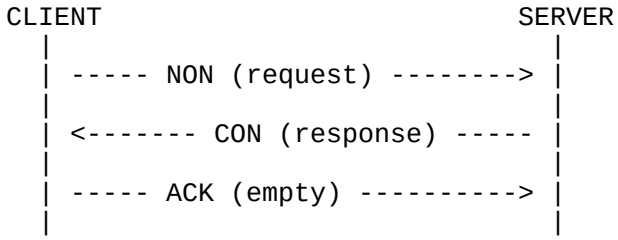



Figure 4: NON requests and CON response

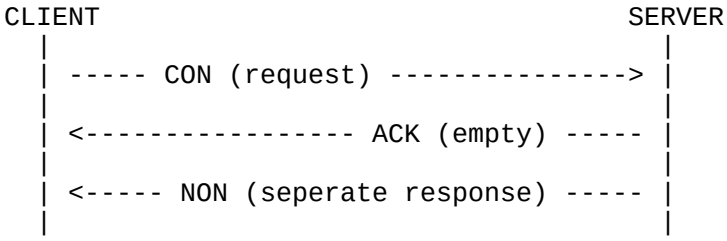



Figure 5: CON request, empty ACK and NON response (seperate)

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

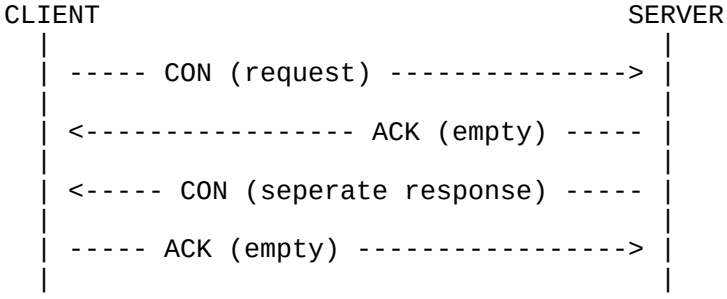



Figure 6: CON request, empty ACK, CON response, empty ACK

## 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 SHOULD 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. Thus, the ENDPOINT_ID_1 option is defined to be elective.

### 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. Consequently, the ENDPOINT_ID_2 option is defined to be critical.

### 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

The maximum length of 4 bytes is arbitrarily chosen.

### 3.4. Endpoint IDs for obervations

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 the message ID, whereas EID_x refers to the value of the ENDPOINT_ID_x option.

### 4.1. NON request and NON response

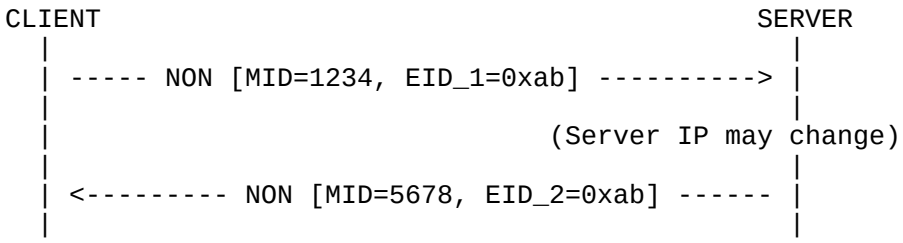



Figure 7: NON requests and NON response

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

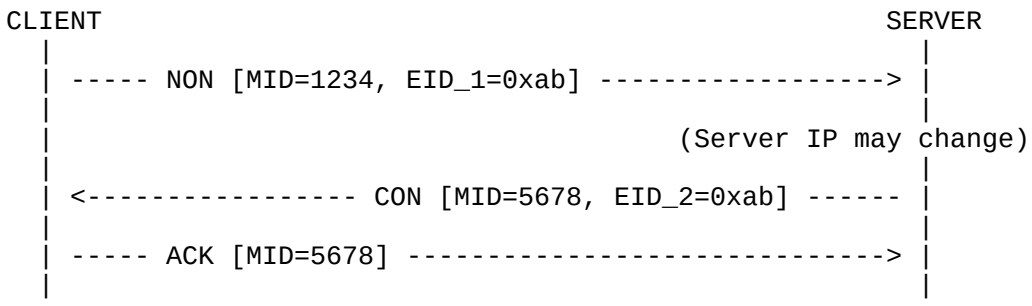



Figure 8: NON requests and NON response

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

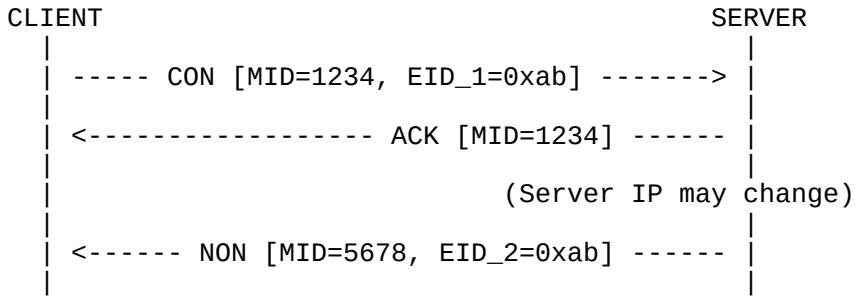



Figure 9: NON requests and NON response

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

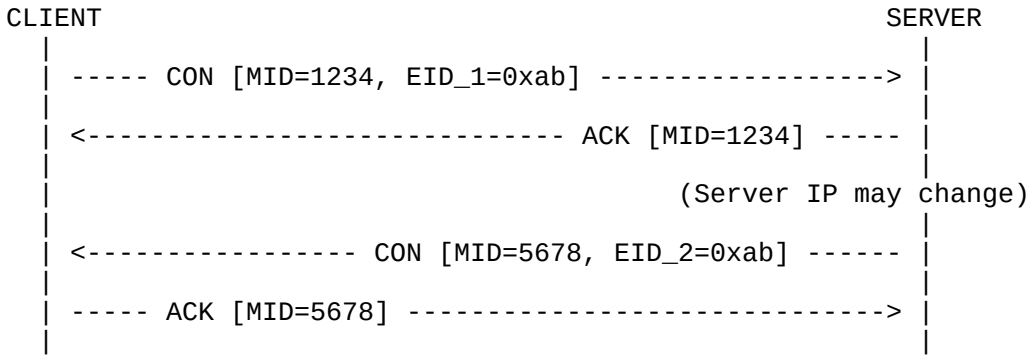



Figure 10: CON request, empty ACK, CON response, and empty ACK

### 4.5. Server IP address changes during observation

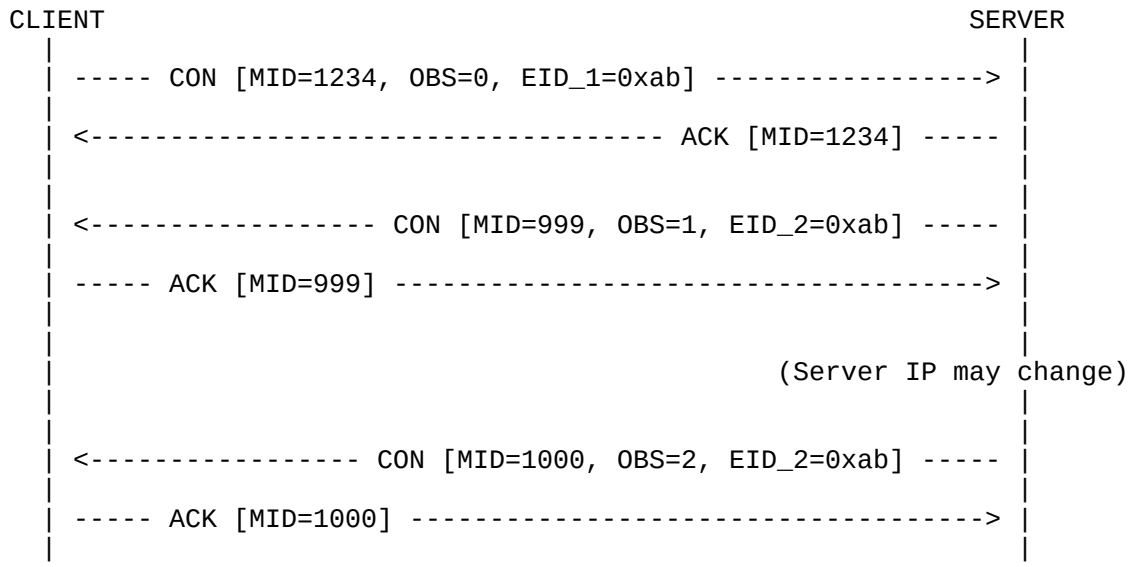



Figure 11: Server IP address changes during observation

## 4.6. Client IP address changes during observation

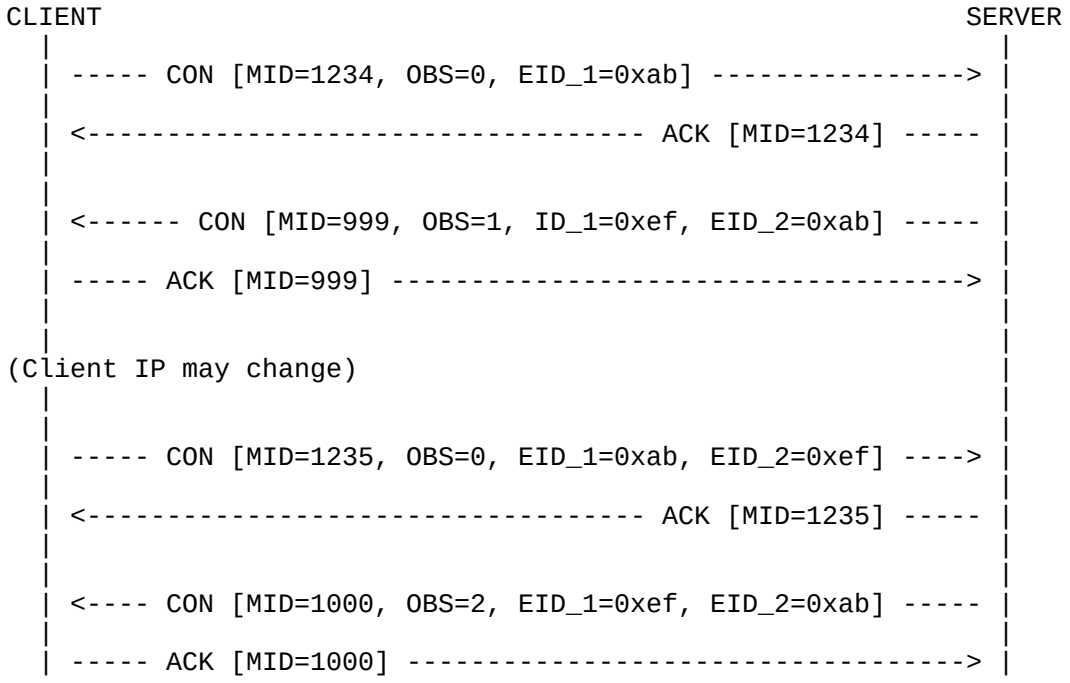



Figure 12: Client IP address changes during observation

## 5. Acknowledgements

No acknowledgements, yet...

## 6. IANA Considerations

This memo includes no request to IANA.

## 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>.

[RFC7252] Shelby, et.al., , "The Contrained Application Protocol (CoAP)", RFC 7252, June 2014, <http://tools.ietf.org/html/rfc7252>.

## 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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

M. Koster
ARM Limited
A. Keranen
J. Jimenez
Ericsson

October 27, 2014

# Publish-Subscribe in the Constrained Application Protocol (CoAP)
# draft-koster-core-coap-pubsub-00

## 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 and 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 April 30, 2015.

## Copyright Notice

Table of Contents

1. Introduction . . . . . 2
2. Terminology . . . . . 3
3. Architecture . . . . . 4
3.1. RD Server with associated CoAP-PubSub Broker . . . . . 4
3.2. Client Endpoint . . . . . 5
3.3. Server Endpoint . . . . . 5
3.4. Publish-Subscribe Topics . . . . . 5
4. CoAP-PubSub Registration and discovery . . . . . 5
4.1. Register CoAP-PubSub Endpoint . . . . . 6
4.2. Unregister Endpoint . . . . . 6
5. CoAP-PubSub Functions and Interactions . . . . . 7
5.1. Client Role Endpoint Functions . . . . . 7
5.1.1. Client Endpoint PUBLISH to CoAP-PubSub broker . . . . 7
5.1.2. Client Endpoint SUBSCRIBE, Broker PUBLISH . . . . . 8
5.1.3. Client Endpoint GET from CoAP-PubSub Broker . . . . . 9
5.2. Server Role Endpoint Functions . . . . . 9
5.2.1. CoAP-PubSub broker SUBSCRIBES to Server Role EP . . . 9
5.2.2. CoAP-PubSub Broker Publishes to Server Role Endpoint 10
5.2.3. CoAP-PubSub Broker GET from Server Role Endpoint . . 10
6. Enabling Multiple Publishers . . . . . 11
6.1. Creating a Topic . . . . . 11
6.2. Publishing a Topic from Multiple Publishers . . . . . 11
6.3. Subscribing to a topic with multiple publishers . . . . . 12
7. Sleep-Wakeup Operation and Message Queueing . . . . . 12
8. Security Considerations . . . . . 12
9. IANA Considerations . . . . . 13
9.1. Resource Type value 'core.pubsub.client' . . . . . 14
9.2. Resource Type value 'core.pubsub.server' . . . . . 14
10. Acknowledgements . . . . . 14
11. References . . . . . 14
11.1. Normative References . . . . . 14
11.2. Informative References . . . . . 15
Authors' Addresses . . . . . 15

## 1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] 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 [I-D.ietf-core-resource-directory]. The extensions enable publish-subscribe communication using a 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 [RFC2119].

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252] and [I-D.ietf-core-resource-directory]. The URI template format, see [RFC6570], is used to describe the REST interfaces defined in this specification.

This specification makes use of the following additional terminology:

CoAP Publish-Subscribe (CoAP-PubSub) 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-PubSub 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-PubSub function set: A group of well-known REST resources that together provide the CoAP-PubSub service.

CoAP-PubSub Endpoint An endpoint that implements the CoAP-PubSub function set. A CoAP-PubSub endpoint has two potential modes, CoAP-PubSub Client and CoAP-PubSub 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-PubSub Broker

Figure 1 shows an example architecture of a CoAP-PubSub capable service. A Resource Directory (RD) 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-PubSub broker. Web clients subscribe to the state of the endpoint from the CoAP-PubSub broker, and publish updates to the endpoint state through the CoAP-PubSub broker. The CoAP-PubSub broker performs a store-and-forward function between web clients and the CoAP-PubSub capable endpoints. The CoAP-PubSub 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.

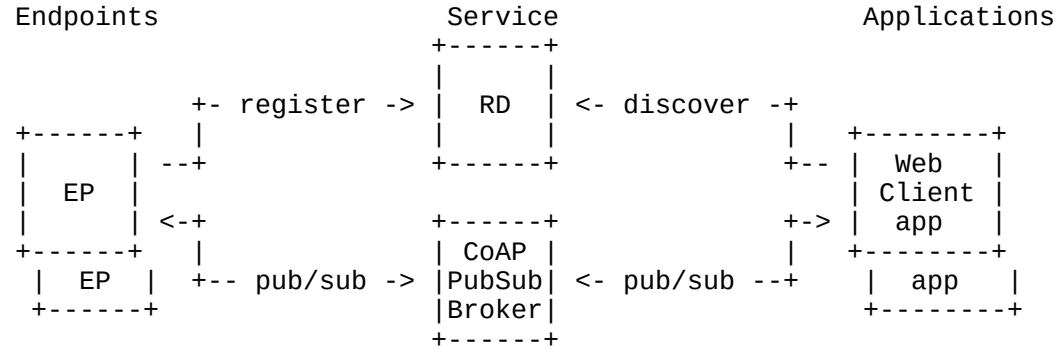



Figure 1: CoAP-PubSub Architecture

### 3.2. Client Endpoint

Client endpoints initiate all interactions with the RD and CoAP-PubSub broker. If the endpoint is an actuator it will need to either use CoAP Observe [I-D.ietf-core-observe] or periodically poll the PubSub broker to check for updates. A CoAP-PubSub client endpoint MUST use CoAP PUT operations to update its state on the PubSub 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-PubSub broker or RD server.

### 3.3. Server Endpoint

Server endpoint interactions require the CoAP-PubSub broker to perform the client role, initiating interaction with the server endpoint. The CoAP-PubSub broker MAY then use PUT operations to update state at the server endpoint, and MAY use GET or GET and Observe to subscribe to resources at the endpoint. Server mode endpoints are required to be reachable from the CoAP-PubSub 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-PubSub capabilities and using GET and 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-PubSub Registration and discovery

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

A server that implements a CoAP-PubSub broker MAY advertize this capability by registering the rt="core.pubsub" with an associated Resource Directory. If a server advertizes as a CoAP-PubSub Broker, it MUST support the transactions described in section 5 of this

document. As server that implements the CoAP-PubSub Broker MAY also implement sleeping endpoint and message queueing support referred to in Section 6 of this document.

### 4.1. Register CoAP-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-PubSub, it discovers the rt="core.pubsub" attribute at the RD service associated with the CoAP-PubSub broker and registers its CoAP-PubSub 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 "/pubsub/sensors/weather/barometer". An implementation MAY mix CoAP-PubSub resources and CoAP REST resources on the same endpoint. Endpoint registration proceeds as per normal RD registration.

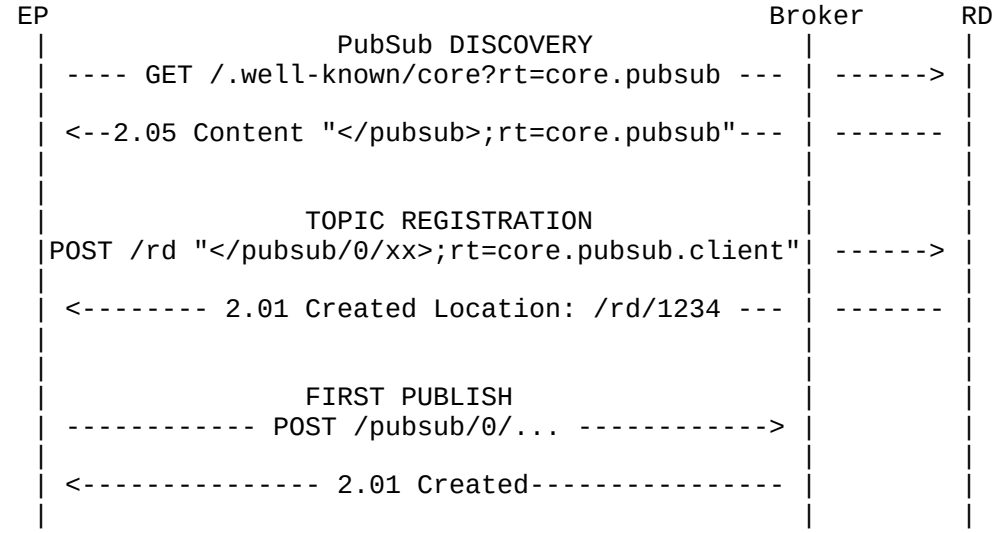



Figure 2: Discovery and Registration

### 4.2. Unregister Endpoint

CoAP-PubSub 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.

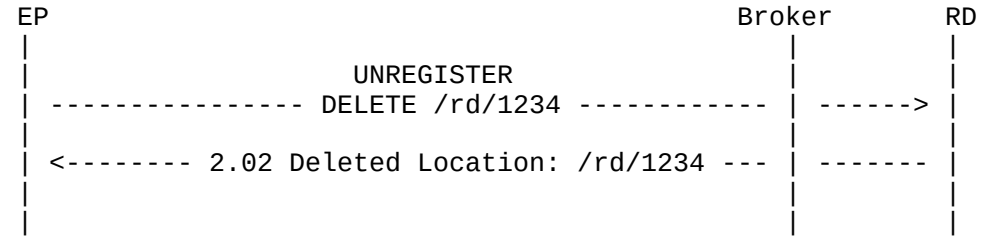



Figure 3: Unregister Endpoint

## 5. CoAP-PubSub Functions and Interactions

This section describes the transaction flows and interactions between CoAP-PubSub endpoints and CoAP-PubSub 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-PubSub 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-PubSub endpoints and CoAP-PubSub 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-PubSub broker

Client endpoint PUBLISHes updates to CoAP-PubSub broker. A CoAP-PubSub client endpoint MAY use PUT to publish state updates to the CoAP-PubSub broker.

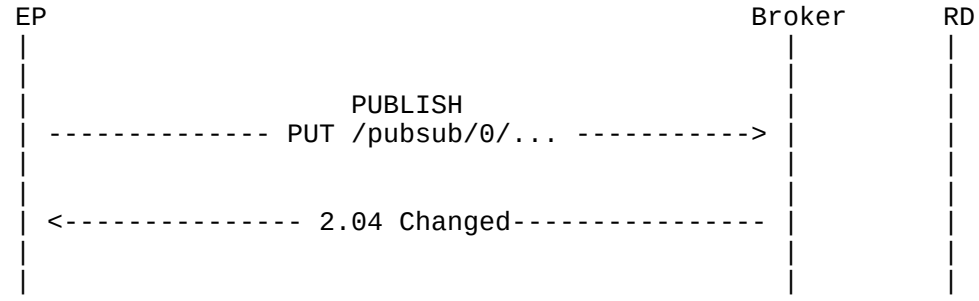



Figure 4: Client Role PUBLISH from EP to Broker

### 5.1.2. Client Endpoint SUBSCRIBE, Broker PUBLISH

Client mode endpoint subscribes to the topic at the CoAP-PubSub broker using GET and Observe. Published updates to the CoAP-PubSub 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-PubSub 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-PubSub Broker

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

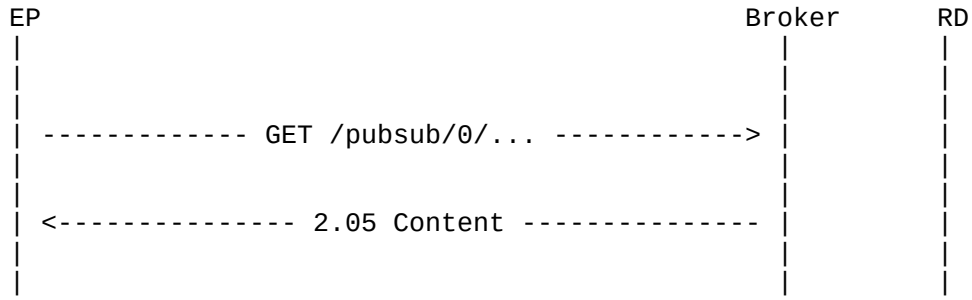



Figure 6: Client EP GET from CoAP-PubSub Broker

### 5.2. Server Role Endpoint Functions

This section describes the transaction flows and interactions between CoAP-PubSub endpoints and CoAP-PubSub 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-PubSub broker SUBSCRIBES to Server Role EP

The server mode endpoint requires the CoAP-PubSub broker to act as a client and subscribe to a resource on the endpoint using GET + Observe. A CoAP-PubSub broker MAY subscribe to topics registered by a server role endpoint at any time. A CoAP-PubSub broker MUST subscribe to a topic registered by a server role endpoint upon receiving a subscription on the associated topic. A CoAP-PubSub 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-PubSub Broker subscribing to a server role endpoint.


Figure 7: Broker SUBSCRIBE to Server Role EP

5.2.2. CoAP-PubSub Broker Publishes to Server Role Endpoint

CoAP-PubSub 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.

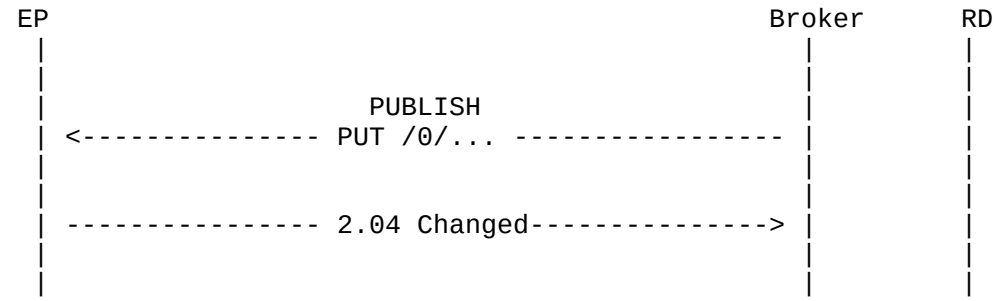



Figure 8: Broker PUBLISH to Server Role EP

5.2.3. CoAP-PubSub Broker GET from Server Role Endpoint

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

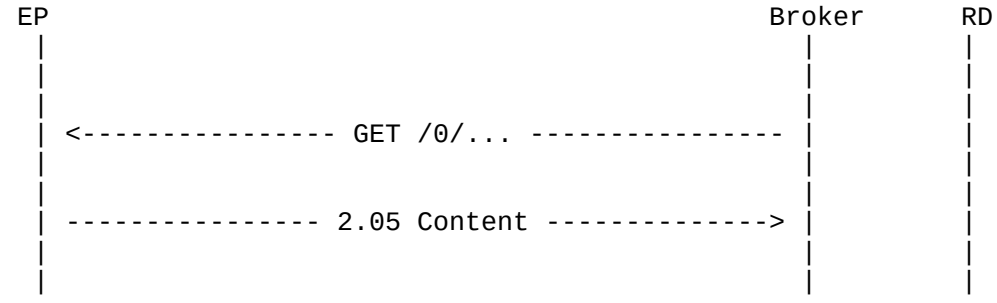



Figure 9: Broker GET from Server Role Endpoint

# 6. Enabling Multiple Publishers

## 6.1. Creating a Topic

After registration of the EP in the RD and discovering the CoAP-PubSub 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 PubSub function by doing a first publication as shown in Figure 2.

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

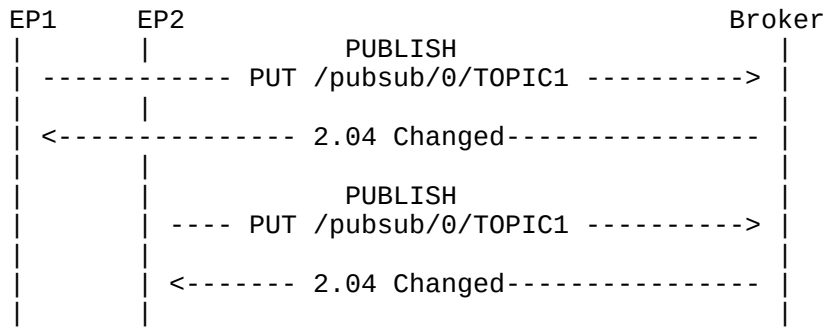



Figure 10: Multiple CoAP-PubSub EPs PUBLISH to Broker

## 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.

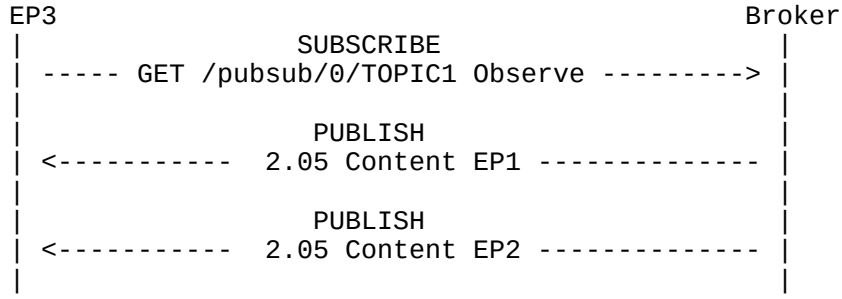



Figure 11: CoAP-PubSub Endpoint SUBSCRIBE to Broker

## 7. Sleep-Wakeup Operation and Message Queueing

A CoAP-PubSub broker MAY implement support for sleeping endpoints and queueing of messages as provided for in [OMALightweightM2M]

## 8. Security Considerations

CoAP-PubSub re-uses CoAP [RFC7252], CoRE Resource Directory [I-D.ietf-core-resource-directory], and Web Linking [RFC5988] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP-PubSub 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-PubSub 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 confidentially 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-PubSub broker, the use of end-to-end encryption may also be envisioned. The CoAP-PubSub 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 [RFC6690].

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

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

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

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

## 10. Acknowledgements

The authors would like to thank Hannes Tschofenig, Zach Shelby, Mohit Sethi, and Anders Eriksson for their contributions and reviews

## 11. References

### 11.1. Normative References

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 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.

[OMALightweightM2M]
Open Mobile Alliance, "OMA LightweightM2M v1.0", http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0, 12 2013.

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

[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.

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

### 11.2. Informative References

[RFC5988] 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

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.

## 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

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-previsioned 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.

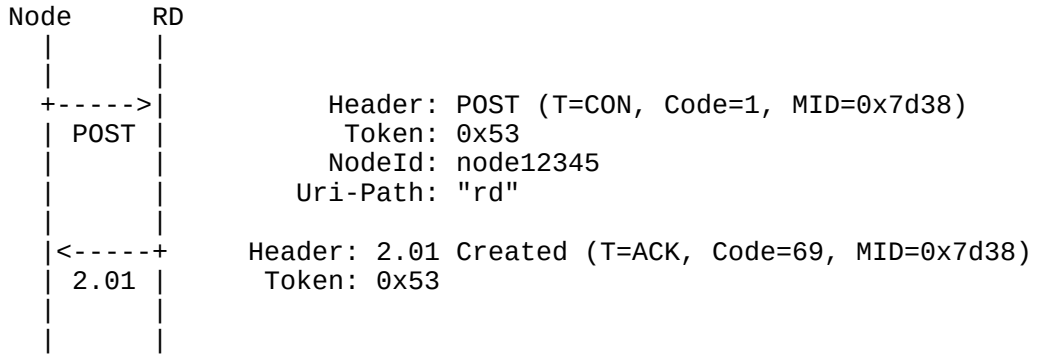



Figure 1: NodeId Option in a registration request

## 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.

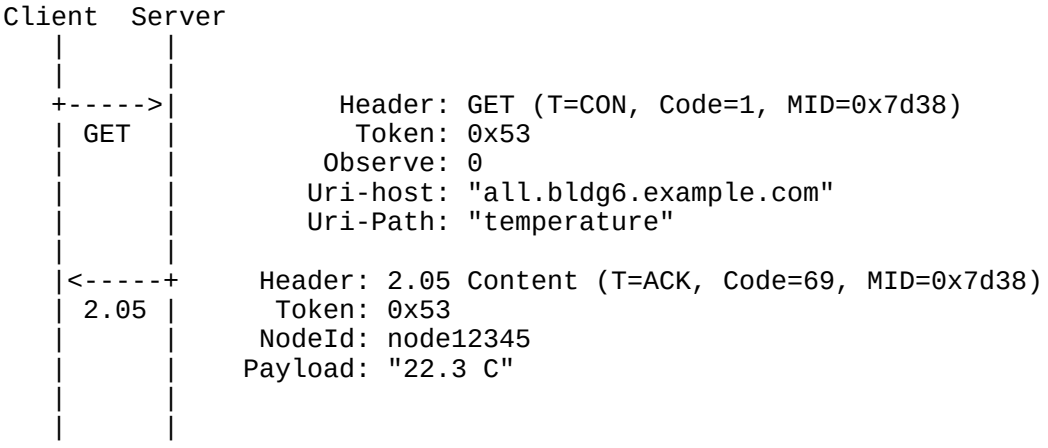



Figure 2: NodeId Option in an observe notification

## 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: April 27, 2015

K. Li
B. Greevenbosch
Huawei Technologies
E. Dijk
Philips Research
S. Loreto
Ericsson
October 24, 2014

# CoAP Option Extension: Patience
# draft-li-core-coap-patience-option-05

## 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 is used by a CoAP client to indicate the maximum time a client is prepared to wait for a response. The CoAP server should try to return the response within the specified time frame.

## Note

Discussion and suggestions for improvement are requested, and should be sent to core@ietf.org.

## Status of This Memo

Table of Contents

1. Introduction . . . . . 2
1.1. Justification . . . . . 2
1.2. Terminology . . . . . 3
2. Patience Option Extension . . . . . 3
2.1. Patience Option Definition . . . . . 3
2.2. Using the Patience Option . . . . . 3
3. Example . . . . . 4
4. Security Considerations . . . . . 5
5. IANA Considerations . . . . . 5
6. Acknowledgements . . . . . 5
7. Normative References . . . . . 5
Authors' Addresses . . . . . 6

## 1. Introduction

This specification adds a new Patience option to CoAP [RFC7252]. The main purpose is for the client to inform the server of the preferred time frame for a response. It is used in a request to indicate the client patience in waiting for a response. It then indicates "a response is most useful within the specified time frame".

### 1.1. Justification

It can be useful for the client to indicate that the response is required to be returned within a certain amount of time. For example, the client 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 client 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 server wastes

resources by sending a response which already exceeds the set patience timeout of the client.

If the Patience option is combined with Observe option in a request, 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 | integer | 1-2 B | none |

The value of the Patience option is measured in seconds. The range is from 1 second to 2^16 seconds, that is, 65535 seconds, around 18 hours. There is no default value for the Patience option.

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

### 2.2. Using the Patience Option

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

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

The Patience option applies to both a piggy-backed response and a separate response. For a separate response, the patience applies to the actual response, not to the ACK. The ACK should be sent immediately upon receipt of the CON message.

TBD: In case a client 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 client 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 client SHOULD give up the request.

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

The server MAY apply a lower value to the patience timeout based on local policy. A server MAY choose to take longer to produce a response, at the risk that the client 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 client does not receive a response within the indicated response time, the client SHOULD consider the request as failed. If the server can’t provide a response within the required time, the server SHOULD discard the request.

## 3. 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 client wants to get a response within 60 seconds.

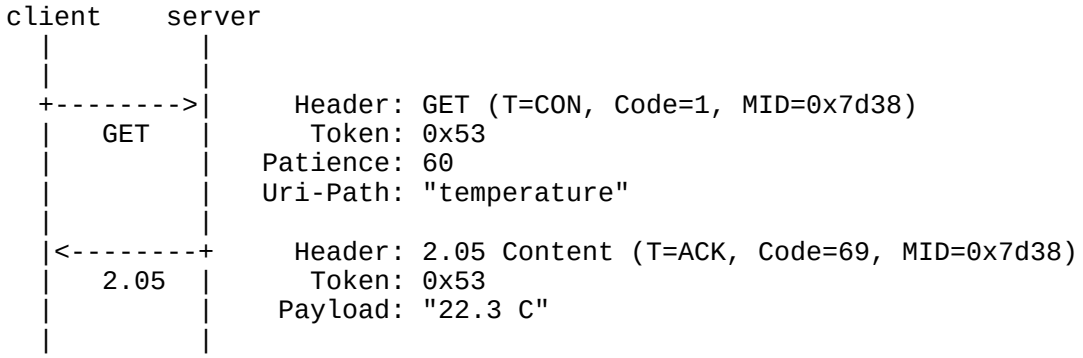



Figure 1: Patience Option in a unicast request

## 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) | 1-2 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. Normative References

[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.

Authors' Addresses

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

Email: likepeng@huawei.com

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

Email: bert.greevenbosch@huawei.com

Esko Dijk
Philips Research
High Tech Campus 34
Eindhoven
The Netherlands

Email: esko.dijk@philips.com

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: 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.

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.

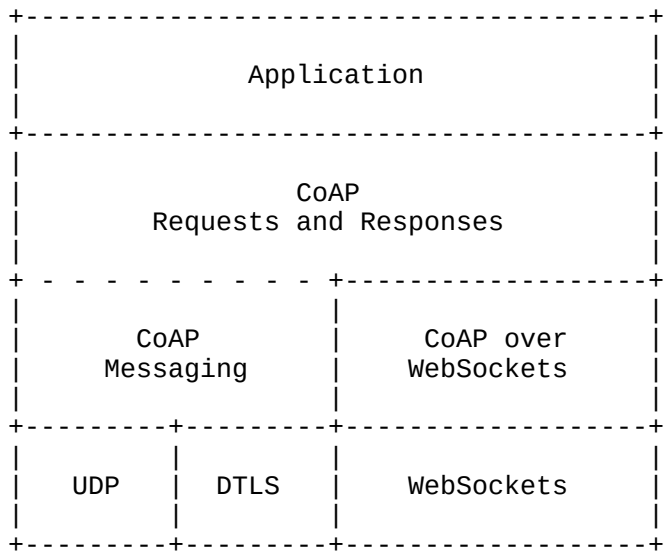



Figure 1: Abstract layering of CoAP extended by WebSockets

### 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.

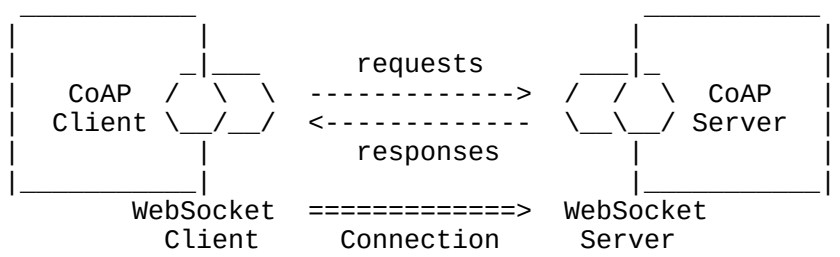



Figure 2: CoAP client (WebSocket client) accesses CoAP server (WebSocket server)

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:

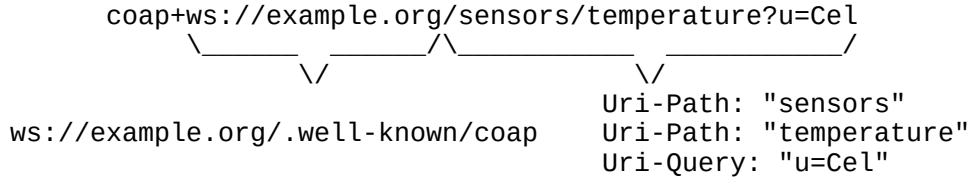



Figure 3: The "coap+ws" URI Scheme

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.

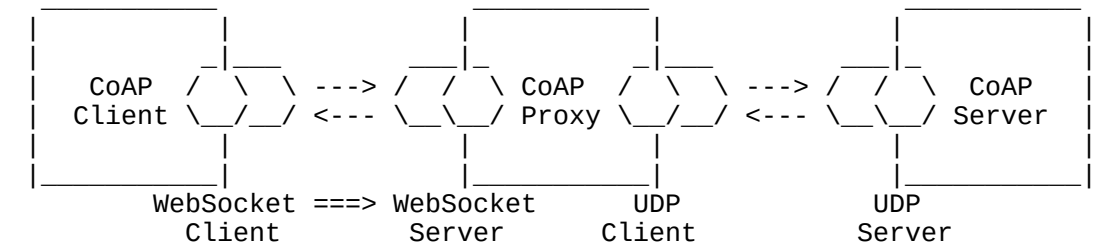



Figure 4: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

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].

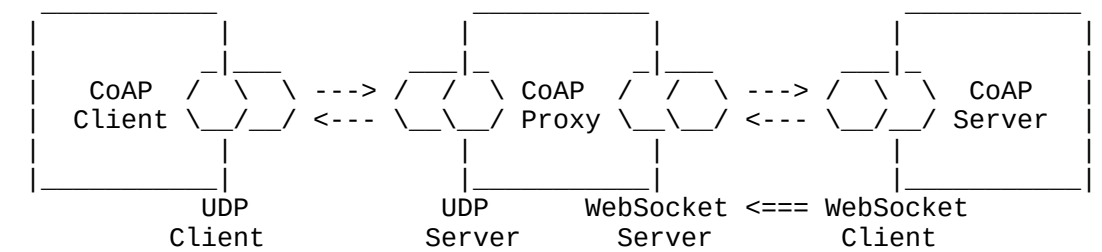



Figure 5: CoAP Client (UDP client) accesses sleepy CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

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:

- 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.

- 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].

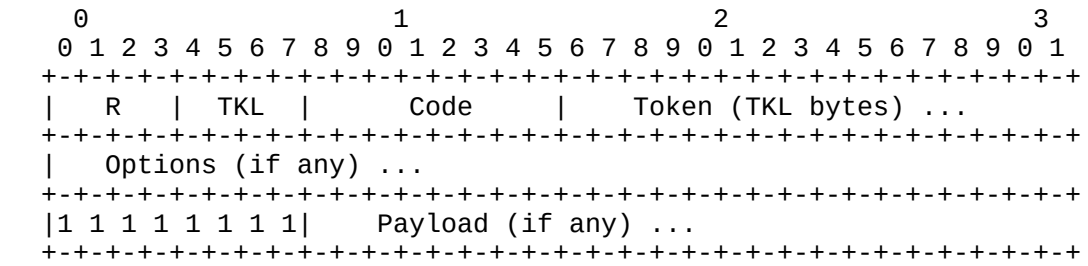



Figure 7: CoAP Message Format over WebSockets

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 URIs 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.

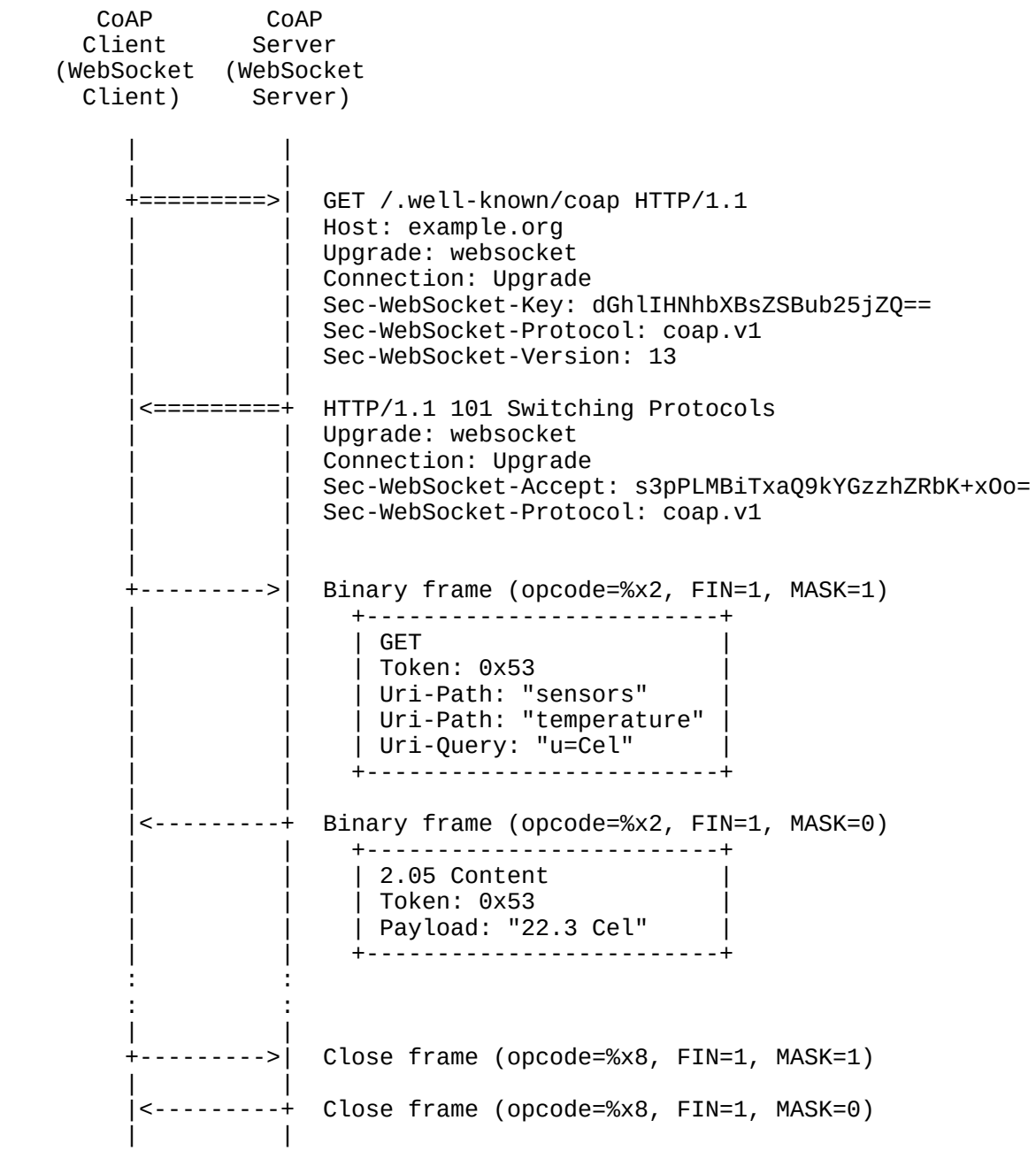



Figure 8: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

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.

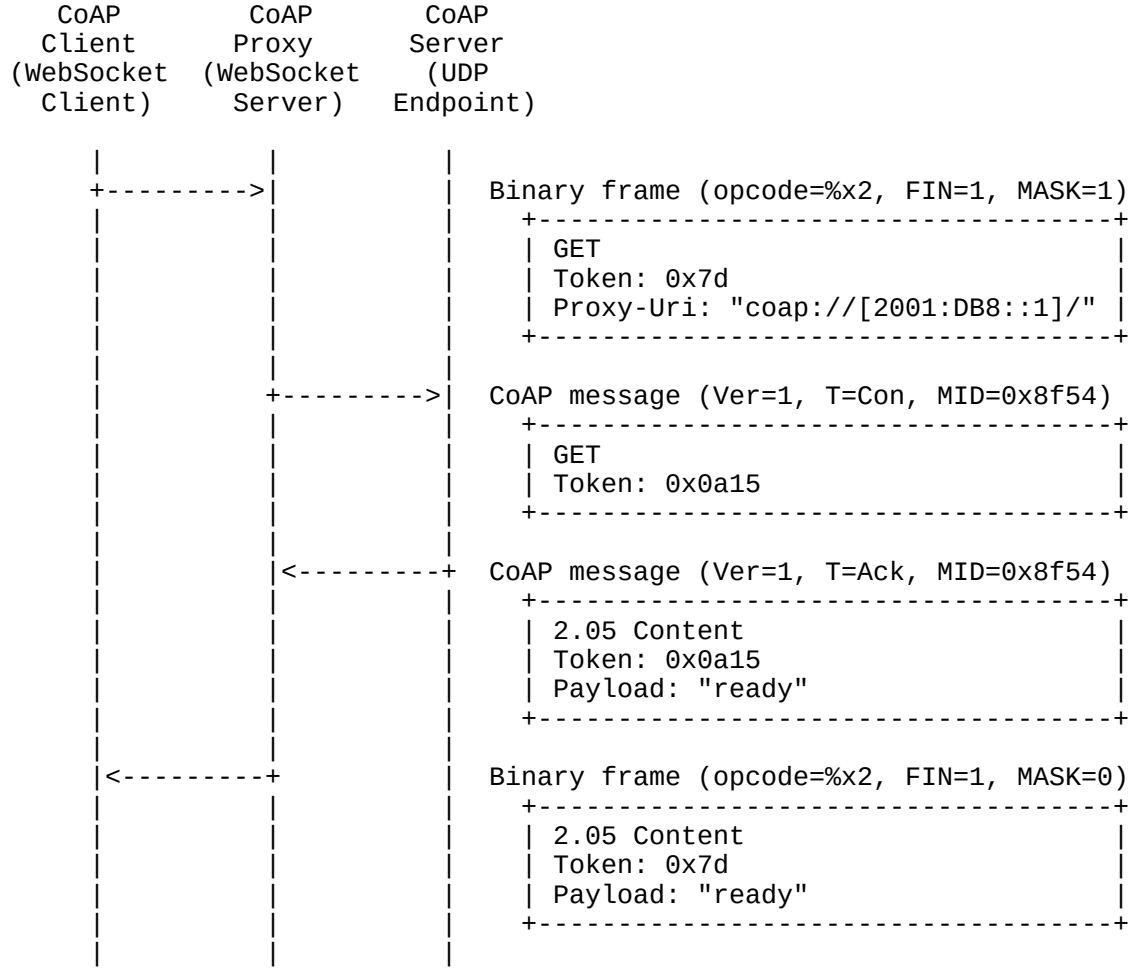



Figure 9: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSockets-enabled CoAP proxy

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.

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] specifices 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:

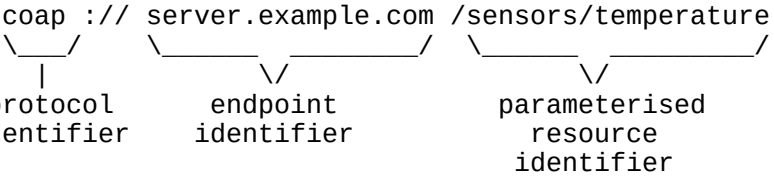



Figure 1: The CoAP URI format

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:

- coap+tcp://[2001:db8::1]:5683/sensors/temperature for using CoAP over TCP

- coap+sms://0015105550101/sensors/temperature for using CoAP over SMS or USSD with the endpoint identifier being a telephone subscriber number

- 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:

- coap-at://tcp-server.example.com/sensors/temperature

- 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-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.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:

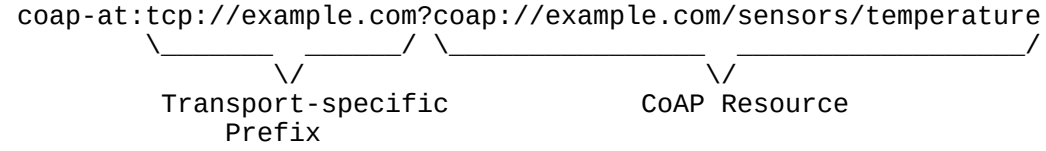



Figure 2: Prefixing a CoAP URI with TCP transport

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:

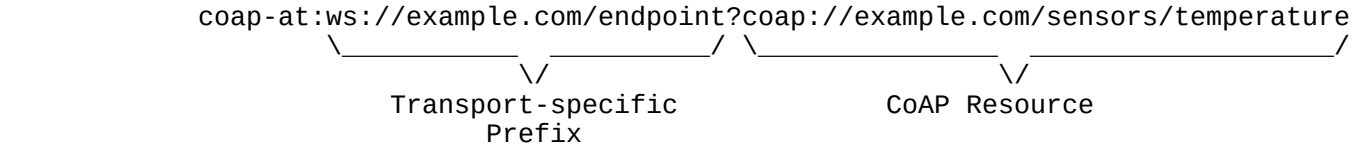



Figure 3: Prefixing a CoAP URI with WebSocket transport

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: February 2015

A. Bhattacharyya
S. Bandyopadhyay
A. Pal
Tata Consultancy Services Ltd.
August 5, 2014

# CoAP option for no server-response
# draft-tcs-coap-no-response-option-07

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.

This Internet-Draft will expire on February 5, 2015.



Table of Contents

1. Introduction.....3
   1.1. Granular suppression of responses.....3
   1.2. Terminology.....3
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.....5
4. Option Definition.....6
   4.1. Granularity in response 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) [RFC7252]. 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 usefulness 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 that 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 [RFC7252].

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 MAY be completely avoided.

## 3. Exemplary application scenarios

Next sub-sections describe some exemplary user stories which may potentially benefit by using 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. The update rate in case of ITS 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 uses 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. This is a request option and primarily intended to be used with non-confirmable update requests (e.g., PUT) 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 special GET request for 'cancellation' of an observe session (Section 3.6 of [I-D.ietf-core-observe]). This option contains values to optionally 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-reference' 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 to GET under usual circumstances when the client requests the contents of a resource. However, this option MAY be useful for special GET requests. At present only one such application is identified which is the 'cancellation' procedure for 'Observe'. Observe-cancellation requires a client to issue a GET request which has the same token as the token of the original observe request 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 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 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 6.1.2.2). |
| 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. Granularity in response 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. For example, the sender may strategically insert requests without No-Response after N numbers of requests with No-Response.

## 5. Miscellaneous aspects

This section further describes few important implementation aspects worth considering while using No-Response. The following discussion does not mandate anything, rather provides suggestive guidelines for the application developer.

### 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 [RFC7252]) 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 a request and the corresponding 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. This creates a problem in the request-response semantics.

The most ideal solution would be to always use a unique token for requests with No-Response. But if a client wants to reuse a token then in most practical cases 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] refers to the 'patience' option defined in [I-D.draft-li-coap-patience]. 'Patience' option effectively puts a deadline to the server to respond back. However, 'patience' is not exposed to the protocol level at present. This draft suggests a reuse time for tokens with similar expression as in Section 2.5 of [I-D.ietf-core-groupcomm]:

TOKEN_REUSE_TIME = NON_LIFETIME + MAX_SERVER_RESPONSE_DELAY + MAX_LATENCY.

NON_LIFETIME and MAX_LATENCY are defined in 4.8.2 of [RFC7252]. 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 [RFC7252] and Appendix-B.4.2 of [I-D.draft-bormann-coap-misc]where group size (G) = 1 for unicast request.

If it is not possible for the client to get a reasonable estimate of the MAX_SERVER_RESPONSE_DELAY then the client SHOULD use a unique token for the request with No-Response to be safe.

### 5.3. Taking care of congestion

The possible communication scenarios leveraging the benefits of 'No-Response' should primarily fall into the class of low-data volume applications as described in Section 3.1.2 of [RFC5405]. Precisely, they should map to the scenario where the application cannot maintain an RTT estimate. Hence, following [RFC5405], 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

Unlike the multicast actuation scenarios (Section 3.2), scenarios like frequent update using No-Response leads to an interesting observation. The 'No-Response' option in a sense 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 'observe' with notifications in NON mode. For example, the scenario of location update described in Section 3.1 above might also be deployed using observe with NON-notifications. In that case the backend infrastructure would have to subscribe to each individual sensor gateway at the vehicles. But, the 'book-keeping' exercise required at the backend 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 respond back.

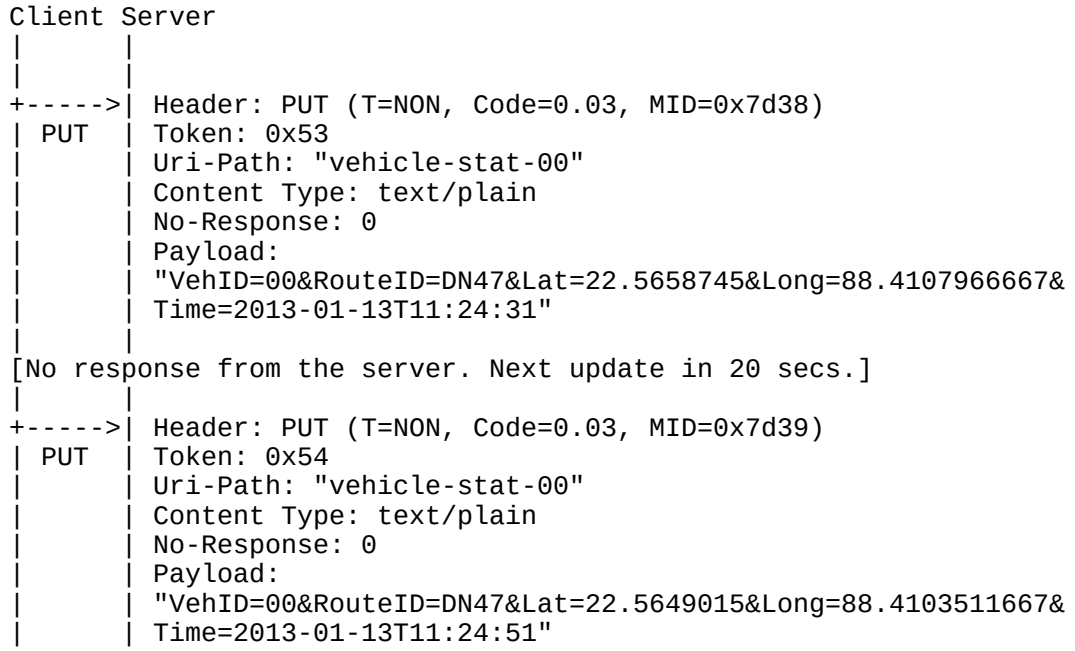



Figure 1: Exemplary unreliable update with No-Response option using PUT.

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
|      |
|      |
+----->| Header: POST (T=NON, Code=0.02, MID=0x7d38)
| POST | Token: 0x53
|      | Uri-Path: "vehicle-stat-00"
|      | Content Type: text/plain
|      | No-Response: 0
|      | Payload:
|      | "VehID=00&RouteID=DN47&Lat=22.5658745&Long=88.4107966667&
|      | Time=2013-01-13T11:24:31"
|      |
[No response from the server. Next update in 20 secs.]
|      |
+----->| Header: PUT (T=NON, Code=0.02, MID=0x7d39)
| POST | Token: 0x54
|      | Uri-Path: "vehicle-stat-00"
|      | Content Type: text/plain
|      | No-Response: 0
|      | Payload:
|      | "VehID=00&RouteID=DN47&Lat=22.5649015&Long=88.4103511667&
|      | Time=2013-01-13T11:24:51"
```

Figure 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 infrastructure as described in 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.

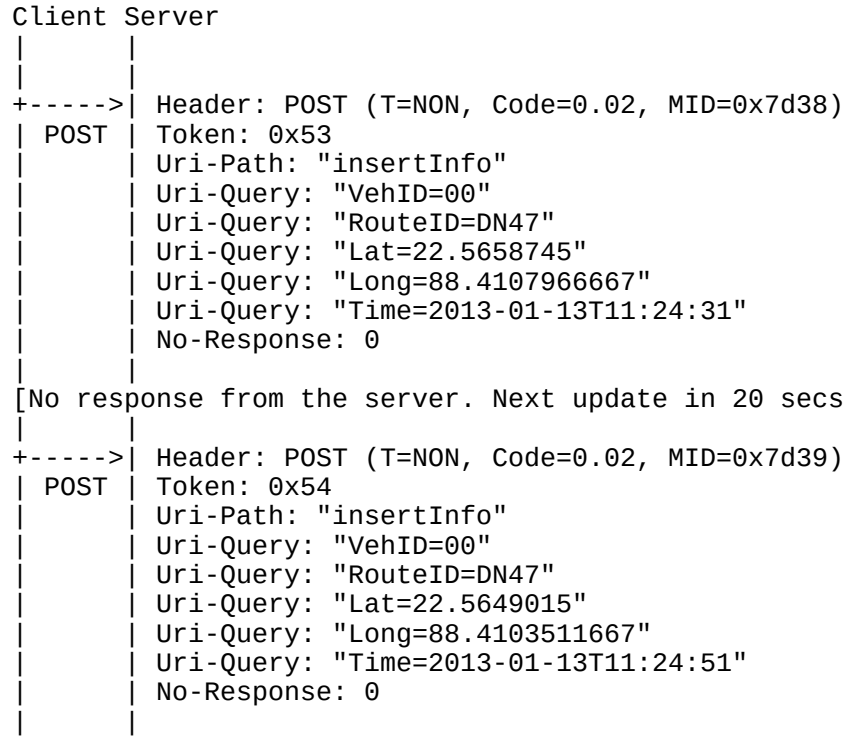



Figure 3: Exemplary unreliable update with No-Response option using POST with a query-string to insert update information to backend database.

## 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.

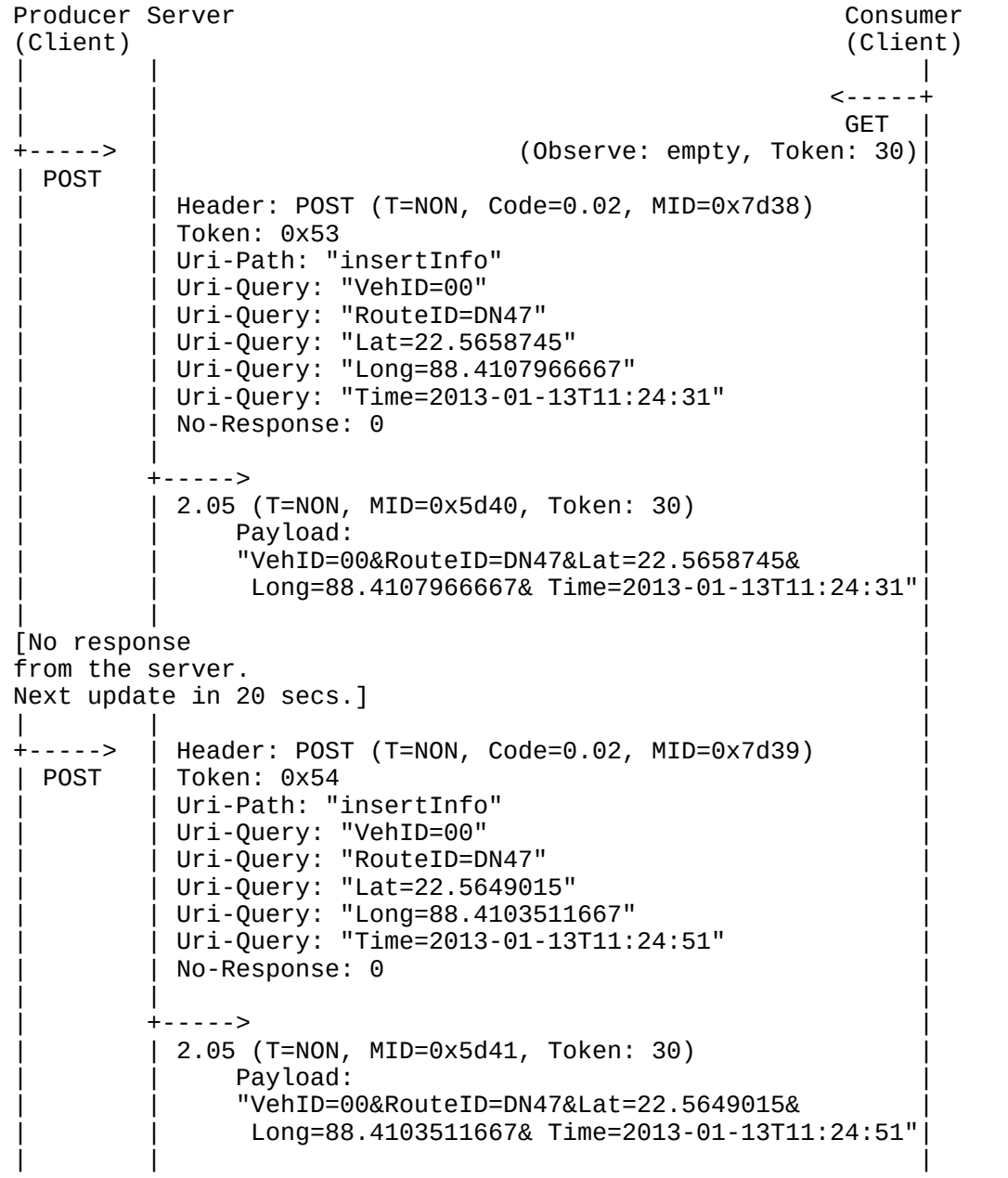



Figure 4: Exemplary end-to-end update and observe scenario using 'No-Response' for NON-updates from 'producer' and observe with NON-notifications by the 'consumer'.

## 7. IANA Considerations

The IANA is requested to add the following option number entries:

| Number | Name | Reference |
|---|---|---|
| 92 | 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 [RFC7252].

## 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

[RFC7252]

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

[I-D.ietf-core-observe]

Hartke, K.,"Observing Resources in CoAP", draft-ietf-core-observe-14, June 30, 2014

[I-D.ietf-core-groupcomm]

Rahman, A. and Dijk, E.,"Group Communication for CoAP", draft-ietf core-groupcomm-21, July 31, 2014

[I-D.draft-bormann-coap-misc]

Bormann, C. and Hartke, K., "Miscelleneous 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

[RFC5405]

Eggert, L. and Fairhurst, G.," Unicast UDP Usage Guidelines for Application Designers", RFC 5405, November, 2008

[I-D.draft-li-coap-patience]

Li, K., Greevenbosch, B., Dijk, E. and Loreto, S.," CoAP Option Extension: Patience", draft-li-core-coap-patience-option-04, July 04, 2014

## 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: Standards Track
Expires: March 8, 2015

S. Lemay
V. Solorzano Barboza
Zebra Technologies
H. Tschofenig, Ed.
ARM Ltd.

September 4, 2014

# A TCP and TLS Transport for the Constrained Application Protocol (CoAP)
draft-tschofenig-core-coap-tcp-tls-01.txt

## Abstract

The Hypertext Transfer Protocol (HTTP) has been designed with TCP as an underlying transport protocol. The Constrained Application Protocol (CoAP), which has been inspired by HTTP, has on the other hand been defined to make use of UDP. Reliable delivery, a simple congestion control mechanism, and flow control had been added to the CoAP protocol. UDP is a good choice for networks that do not perform any form of filtering and firewalling. There are, however, many deployment environments where UDP is either firewalled or subject to deep packet inspection. These environments make the use of CoAP brittle.

This document defines the use of CoAP over TCP as well as CoAP over TLS.

## Status of This Memo

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

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

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

This Internet-Draft will expire on March 8, 2015.

Table of Contents

1. Introduction . . . . . 2
2. Terminology . . . . . 3
3. Shim Header . . . . . 3
4. Developer Considerations . . . . . 3
5. Security Considerations . . . . . 4
6. IANA Considerations . . . . . 4
7. Acknowledgements . . . . . 4
8. Normative References . . . . . 4
Authors' Addresses . . . . . 5

## 1. Introduction

The Internet protocol stack is organized in layers, namely data link layer, network layer, transport layer, and the application layer.

IP emerged as the waist of the hour glass and supports a variety of link layers and new link layer technologies can be added in the future, without affecting IP.

Combined with the end-to-end principle the hour glass indicates the level of protocol understanding intermediaries need to have in order to exchange forward IP packets between a sender and a receiver (absent any specific application layer entities, like proxies or caches). Having IP as the waist meant that anyone could extend the layers above the network layer in the way they wanted to communicate end-to-end, including defining new transport layer protocols (as it was done with SCTP, and DCCP).

Unfortunately, deployments departed from this ideal architecture. When the Constrained Application Protocol (CoAP) [RFC7252] was designed it was assumed that many Internet of Things deployments

would be clean-slate. Today, we know that some deployments have to integrate well with existing enterprise infrastructure, where the use of UDP-based protocols is not well-received and firewalling use is very common.

To make IoT devices work smoothly in these demanding environments CoAP has to make use of a different transport protocol, namely TCP [RFC0793] and in some situations even TLS [RFC5246]. This document describes a shim header that conveys length information about the included payload. Modifications to CoAP are intentially avoided (e.g, to introduce optimizations).

## 2. Terminology

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

## 3. Shim Header

This specification defines a simple layer necessary to convey length information about the exchange payloads in a 32-bit length field indicating the number of bytes in the payload following that header.

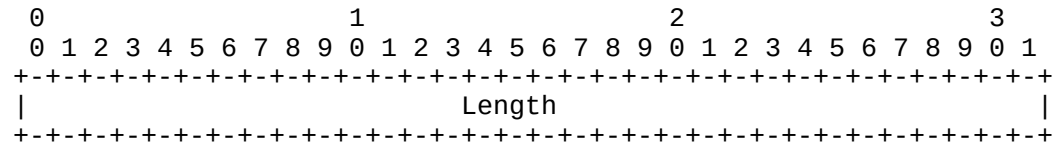



Figure 1: Shim Header.

## 4. Developer Considerations

The use of CoAP over a transport protocol offering reliable transmission already offers functionality that could be offered by CoAP itself. While a developer can re-use an already existing CoAP protocol stack, the use of TCP makes some CoAP features redundant.

Section 4.2 of [RFC7252] discusses the ability to convey messages in CoAP reliably as a "confirmable message", which always generates a response. It can be used without harm but does not add any value since all messages would be transmitted reliably already thanks to the features offered by TCP. A developer writing an application that runs CoAP over TCP or CoAP over TLS needs to be mindful about the changed semantic of CoAP. For example, the marking the message as

non-confirmable (see Section 4.3 of [RFC7252]) does not make the transmission unreliable but it instead saves the transmission of one CoAP message.

## 5. Security Considerations

This document defines how to convey CoAP over TCP and TLS. It does not introduce new vulnerabilities beyond those described already in the CoAP specification.

When CoAP is exchanged over TLS port 443 then the "TLS Application Layer Protocol Negotiation Extension" [I-D.ietf-tls-applayerprotoneg] MUST be used to allow demultiplexing at the server-side unless out-of-band information ensures that the client only interacts with a server that is able to demultiplex CoAP messages over port 443. This would, for example, be true for many Internet of Things deployments where clients are pre-configured to only ever talk with specific servers.

When CoAP over TLS is used then the use of the shim header that includes the length information is redundant since the TLS protocol headers already include length information. As such, the length header MUST be omitted when CoAP is exchanged over TLS.

## 6. IANA Considerations

This document requests a value from the "Application Layer Protocol Negotiation (ALPN) Protocol IDs" created by [I-D.ietf-tls-applayerprotoneg]:

Protocol: CoAP

Identification Sequence: 0x63 0x6f 0x61 0x70 ("coap")

Specification: This document.

## 7. Acknowledgements

We would like to thank Michael Koster, Zach Shelby, and Szymon Sasin for their feedback.

## 8. Normative References

[I-D.ietf-tls-applayerprotoneg]
Friedl, S., Popov, A., Langley, A., and S. Emile, "Transport Layer Security (TLS) Application Layer Protocol Negotiation Extension", draft-ietf-tls-applayerprotoneg-05 (work in progress), March 2014.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

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

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

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

## Authors' Addresses

Simon Lemay
Zebra Technologies
820 W. Jackson Blvd.suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: slemay@zebra.com

Valik Solorzano Barboza
Zebra Technologies
820 W. Jackson Blvd. suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: vsolorzanobarboza@zebra.com

Hannes Tschofenig (editor)
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
Great Britain

Email: Hannes.tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

core
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

P. van der Stok
consultant
B. Greevenbosch
Huawei Technologies
A. Bierman
YumaWorks
J. Schoenwaelder
A. Sehgal
Jacobs University
October 27, 2014

# CoAP Management Interface
# draft-vanderstok-core-comi-05

## Abstract

This document describes a network management interface for constrained devices, called CoMI. CoMI is an adaptation of the RESTCONF protocol for use in constrained devices and networks. It is designed to reduce the message sizes, server code size, and application development complexity. The Constrained Application Protocol (CoAP) is used to access management data resources specified in YANG, or SMIv2 converted to YANG. The payload of the CoMI message is encoded in Concise Binary Object Representation (CBOR).

## Note

Discussion and suggestions for improvement are requested, and should be sent to core@ietf.org.

## Status of This Memo

Table of Contents

1. Introduction . . . . . 3
1.1. Design considerations . . . . . 4
1.2. Terminology . . . . . 5
1.2.1. Tree Diagrams . . . . . 6
2. CoMI Architecture . . . . . 6
2.1. RESTCONF/YANG Architecture . . . . . 10
3. CoAP Interface . . . . . 11
4. MG Function Set . . . . . 12
4.1. Data Retrieval . . . . . 13
4.1.1. GET . . . . . 13
4.1.2. Mapping of the 'select' Parameter . . . . . 13
4.1.3. Retrieval Examples . . . . . 13
4.2. Data Editing . . . . . 16
4.2.1. POST . . . . . 16
4.2.2. PUT . . . . . 17
4.2.3. DELETE . . . . . 17
4.3. Module Discovery . . . . . 17
4.4. Error Return Codes . . . . . 18
5. Mapping YANG to CoMI payload . . . . . 19
5.1. YANG Hash Generation . . . . . 20
5.2. Re-Hash Procedure . . . . . 20
5.3. ietf-yang-hash YANG Module . . . . . 21
5.3.1. YANG Re-Hash Example . . . . . 23
5.4. The 'keys' Query Parameter . . . . . 24
6. Mapping YANG to CBOR . . . . . 25
6.1. Conversion from YANG datatypes to CBOR datatypes . . . . 25
7. Examples . . . . . 27
8. Trap functions . . . . . 27
9. Object access management . . . . . 27
9.1. Notify destinations . . . . . 27
10. Error Handling . . . . . 27

11. Security Considerations . . . . . 29
12. IANA Considerations . . . . . 29
13. Acknowledgements . . . . . 29
14. Changelog . . . . . 30
15. References . . . . . 31
15.1. Normative References . . . . . 31
15.2. Informative References . . . . . 32
Appendix A. Payload and Server sizes . . . . . 35
Appendix B. Notational Convention for CBOR data . . . . . 36
Authors' Addresses . . . . . 37

## 1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. The messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

The draft [I-D.ietf-netconf-restconf] describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG [RFC6020]. RESTCONF allows access to data resources contained in NETCONF [RFC6241] datastores. RESTCONF messages can be encoded in XML [XML] or JSON. The GET method is used to retrieve data resources and the POST, PUT, PATCH, and DELETE methods are used to create, replace, merge, and delete data resources.

A large amount of Management Information Base (MIB) [RFC3418] specifications already exist for monitoring purposes. This data can be accessed in RESTCONF if the server converts the SMIv2 modules to YANG, using the mapping rules defined in [RFC6643].

The CoRE Management Interface (CoMI) is intended to work on standardized data-sets in a stateless client-server fashion. The RESTCONF protocol is adapted and optimized for use in constrained environments, using CoAP instead of HTTP. 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 [RFC7252].

RESTCONF uses the HTTP methods HEAD, OPTIONS, and PATCH, which are not available in CoAP. The use of TCP is also a problem in HTTP. The transport protocols available fo CoAP are much better suited to constrained networks.

CoMI is low resource oriented, uses CoAP, and only supports the methods GET, PUT, POST and DELETE. CoMI uses CBOR as the data format. To support small payloads, CoMI use an additional data identifier string to number conversion to minimise CBOR payloads. It is assumed that the managed device is the most constrained entity. The client might be more capable, however this is not necessarily the case.

Currently, small managed devices need to support at least 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. Although the SNMP server size is not huge (see Appendix A), the code for the security aspects of SMIv3 is not negligible. Using CoAP to access secured management objects 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 managed objects 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 automatically generated from JSON [JSON]. CBOR has a binary format and hence has more coding efficiency than JSON.

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. CoMI supports MIB modules which have been translated from SMIv2 to YANG, using [RFC6643]. This mapping is read-only so writable SMIv2 objects need to be converted to YANG using an implementation-specific mapping.

CoMI uses a simple URI to access the management object 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 should be familiar with all the terms and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

The following terms are defined in the NETCONF protocol [RFC6241]:

- client
- configuration data
- datastore
- server

The following terms are defined in the YANG data modeling language [RFC6020]:

- container
- data node
- key
- key leaf
- leaf
- leaf-list
- list

The following terms are defined in RESTCONF protocol [I-D.ietf-netconf-restconf]:

- data resource
- datastore resource
- edit operation
- query parameter
- target resource

unified datastore

The following terms are defined in this document:

YANG hash: CoMI object identifier, which is a 32-bit numeric hash of the YANG object identifier string for the object. When a YANG hash value is printed in a request target URI, error-path or other string, then the lowercase hexadecimal representation is used. Leading zeros are used so the value uses 8 hex characters.

The following list contains the abbreviations used in this document.

XXXX: TODO, and others to follow.

### 1.2.1. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

Brackets "[" and "]" enclose list keys.

Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).

Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

Ellipsis ("...") stands for contents of subtrees that are not shown.

## 2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for the reading and modifying of instrumentation variables used for the management of the instrumented node.

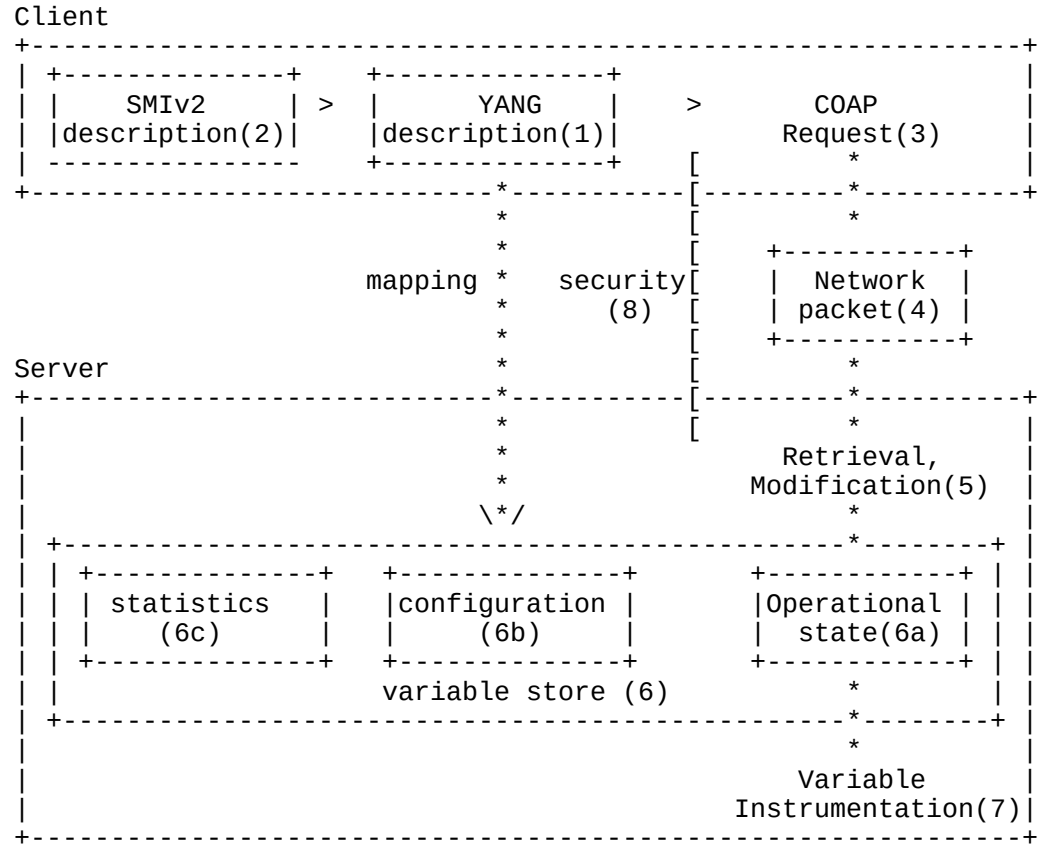



Figure 1 is a high level representation of the main elements of the CoAP management architecture. A client sends requests as payload in packets over the network to a managed constrained node.

Objectives are:

- Equip a constrained node with a management server that provides information about the operational characteristics of the code running in the constrained node.
- The server provides this information in a variable store that contains values describing the performance characteristics and the code parameter values.
- The client receives the performance characteristics on a regular basis or on request.

- The client sets the parameter values in the server at bootstrap and intermittently when operational conditions change.

- The constrained network requires the payload to be as small as possible, and the constrained server memory requirements should be as small as possible.

The components in Figure 1 have the following high-level functionality.

- The instrumentation variables and parameters are described in the YANG or SMIv2 language.

- Descriptions in the SMIv2 language are translated into the YANG language.

- The YANG description serves as input to the writers of application and instrumentation code and the humans analysing the returned values (arrow from YANG description to Variable store). The description is used to check the correctness of the CoAP request and do the CBOR encoding.

- The CoAP request packs the request to read or set variables in the packet payload, which is transmitted over the network using IP. The CoAP request also receives values returned by the server in the payload of a packet. On arrival of the packet in the server, the payload is retrieved and decoded.

- Values are stored in the appropriate variables in the Variable store, and or values are returned from the Variable store into the payload of the packet. The Variable instrumentation code stores the values of the parameters into the appropriate places in the operational code. The variable instrumentation code reads current execution values from the operational code and stores them in the Variable store.

- The network communication must be secured.

For interoperability it is required that in addition to using the Internet Protocol for data transport:

- The names, type, and semantics of the instrumentation variables are standardized.

- The instrumentation variables are described in a standard language.

- The signature of the CoAP request in the server is standardized.

- The format of the packet payload is standardized.

- The notification from server to client is standardized.

The different numbered components of Figure 1 are discussed according to component number.

(1) YANG description: A description contains a set of named and versioned modules. A module contains a hierarchy of named and typed resources. A resource is uniquely identified by a sequence of its name and the names of the enveloping resources following the hierarchy order.

(2) SMIv2 description: A named module contains a set of variables and "conceptual tables". Named variables have simple types. Conceptual tables are composed of typed named columns. The variable name and module name identify the variable uniquely. There is an algorithm to translate SMIv2 specifications to YANG specifications.

(3) CoAP request: The CoAP request needs a Universal Resource Identifier (URI) and the payload of the packet to send a request. The URI is composed of the schema, server, path and query and looks like coap://entry.example.com/<path>?<query>. Fragments are not supported. Allowed operations are PUT, GET, DELETE, and POST. New variables can be created with POST when they exist in the YANG specification. The Observe option can be used to return variable values regularly or on event occurrence (notification).

(3.1) CoAP <path>: The path identifies the variable in the form "/mg/data/<identifier>.

(3.2) CoAP <query>: The query parameter is used to specify additional (optional) aspects like the module name, the smi context, and others. The idea is to keep the path simple and put variations on variable specification in the query.

(3.3) CoAP discovery: Discovery of the variables is done with standard CoAP resource discovery using /.well-known/core with ?rt=/core/mg.

(4) Network packet: The payload contains the CBOR encoding of a single JSON object. This object corresponds to the converted RESTCONF message payload.

(5) Retrieval, modification: The server needs to parse the CBOR encoded message and identify the corresponding entries in the

Variable store. In addition, this component includes the code for CoAP Observe and block options.

(6) Variable store: The store is composed of three parts: Operational state, Configuration datastore, and Statistics (see Section 2.1.

(7) Variable instrumentation: This code depends on implementation of drivers and other node specific aspects.

(8) Security: The server MUST prevent unauthorized users from reading or writing any data resources, however a standard access control model for CoMI should be specified in a separate document. DTLS is specified to secure CoAP communication.

## 2.1. RESTCONF/YANG Architecture

CoMI adapts the RESTCONF architecture so data exchange and implementation requirements are optimized for constrained devices.

The RESTCONF protocol uses a unified datastore to edit conceptual data structures supported by the server. The details of transaction preparation and non-volatile storage of the data are hidden from the RESTCONF client. CoMI also uses a unified datastore, to allow stateless editing of any configuration variables.

The child schema nodes of the unified datastore include all the top-level YANG data nodes in all the YANG modules supported by the server. The YANG data structures represent a hierarchy of data resources. The client discovers the list of YANG modules, and important conformance information such as the module revision dates, YANG features supported, and YANG deviations required. The individual data nodes are discovered indirectly by parsing the YANG modules supported by the server.

The YANG data definition statements contain a lot of information that can help automation tools, developers, and operators use the data model correctly and efficiently. The YANG definitions and server YANG module capability advertisements provide an "API contract" that allow a client to determine the detailed server management capabilities very quickly. CoMI allows access to the same data resources as a RESTCONF server, except the messages are optimized to reduce identifier and payload size.

RESTCONF uses a simple algorithmic mapping from YANG to URI syntax to identify the target resource of a retrieval or edit operation. A client can construct operations or scripts using a predictable syntax, based on the YANG data definitions. The target resource URI

can reference a data resource instance, or the datastore itself (to retrieve the entire datastore or create a top-level data resource instance). CoMI uses a 32-bit YANG hash value (based on the YANG data node path identifier strings) to identify schema nodes in the target resource URI.

Any message payload data is relative to the node specified in the target resource URI in a request message. Each message is a well-formed XML instance document or JSON object, corresponding to the YANG schema definition specified by the target resource node. CoMI uses the hash value for the data node identifier in the message payloads, instead of the module-name prefixed identifier name like RESTCONF. CoMI message payloads are based on the JSON encoding of a RESTCONF message payload. The JSON identifier names are first converted to their 32-bit YANG hash values and then the payload is converted to CBOR.

## 3. 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 three sub-resources accessible with the paths:

/mg/data: YANG-based data with path "/mg/data" and using CBOR content encoding format. This path represents a datastore resource which contains YANG data resources as its descendant nodes. All identifiers referring to YANG data nodes within this path are encoded as YANG hash values.

/mg/moduri: URI indicating the location of the server module information, with path "/mg/moduri" and CBOR content format. This YANG data is encoded with plain identifier strings, not YANG hash values.

/mg/yang-hash: URI indicating the location of the server YANG hash information if any objects needed to be re-hashed by the server. It has path "/mg/yang-hash" and is encoded in CBOR format. The "ietf-yang-hash" module in Section 5.3 is used to define the syntax and semantics of this data structure. This YANG data is encoded with plain identifier strings, not YANG hash values. The server will only have this resource if there are any objects that needed to be re-hashed due to a hash collision.

The "/mg/data" resource provides access to the YANG data resources as described in Section 4. The "/mg/moduri" resource provides access to a URI that can be used to retrieve an instance of the "ietf-yang-library" module, defined in the RESTCONF draft. This module lists the name, revision, and conformance information for each YANG module, which a client needs to determine the YANG data model objects that are supported by a server. This URI can be reference a local data resource within the server, or reference a remote data resource, such as a shared file server. The data resource identifiers are YANG hash values, as described in Section 5.1.

The profile of the management function set, with IF=core.mg, 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 |
| Data | /mg/data | core.mg.data | application/cbor |
| Module Set URI | /mg/moduri | core.mg.moduri | application/cbor |
| YANG Hash Info | /mg/yang-hash | core.mg.yang-hash | application/cbor |

## 4. MG Function Set

The MG Function Set provides a CoAP interface to perform a subset of the functions provided by RESTCONF.

A subset of the operations defined in RESTCONF are used in CoMI:

| Operation | Description |
|---|---|
| GET | Retrieve the datastore resource or a data resource |
| POST | Create a data resource |
| PUT | Create or replace a data resource |
| DELETE | Delete a data resource |

### 4.1. Data Retrieval

#### 4.1.1. GET

Data resources are retrieved by the client with the GET method. The RESTCONF GET operation is supported in CoMI. The same constraints apply as defined in section 3.3 of [I-D.ietf-netconf-restconf]. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252].

It is possible that the size of the payload is too large to fit in a single message. 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.

There are optional query parameters for the GET method. A CoMI server MAY implement these query parameters in order to allow common data retrieval filtering functionality.

| Query Parameter | Description |
|---|---|
| content | Select config and/or non-config data resources |
| depth | Request limited sub-tree depth in the reply content |
| select | Request selected sub-trees from the target resource |

#### 4.1.2. Mapping of the 'select' Parameter

TODO: discuss how only a limited subset of the select parameter is used, and how the node names are changed to YANG hashes.

#### 4.1.3. Retrieval Examples

The examples in this section use a JSON payload with one or more entries describing the pair (objectID, value). CoMI transports the CBOR format to transport the equivalent contents. The CBOR syntax of the payloads is specified in Section 5.

#### 4.1.3.1. Single object values

A request to read the value of a management object or the leaf of an object is sent with a confirmable CoAP GET message. A single object is specified in the URI path prefixed with /mg/data.

A request to set the value of an object/leaf is sent with a confirmable CoAP PUT message. The Response is piggybacked to the CoAP ACK message corresponding with the Request.

Using for example the clock container from [RFC7317], a request is sent to retrieve the value of clock/current-datetime specified in module system-state. The answer to the request returns a (ObjectID, value) pair.

In all examples: (1) the payload is expressed in JSON, although the operational payload is specified to be in CBOR, and (2) the path is expressed in readable names although the transported path is expressed in numbers.

```
REQ: GET example.com/mg/data/system-state/clock/current-datetime

RES: 2.05 Content (Content-Format: application/cbor)
{
    "current-datetime" : "2014-10-26T12:16:31Z"
}
```

TODO: convert the example above so it uses YANG hash values instead of the string "current-datetime". Convert the target resource URI string "system-state/clock/current-datetime" to a YANG hash value.

The specified object can be an entire object. Accordingly, the returned payload is composed of all the leaves associated with the object. Each leaf is returned as a (YANG hash, value) pair. For example, the GET of the clock object, sent by the client, results in the following returned payload sent by the managed entity:

```
REQ: GET example.com/mg/data/system-state/clock
   (Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)
{
   "clock" : {
      "current-datetime" : "2014-10-26T12:16:51Z",
      "boot-datetime" : "2014-10-21T03:00:00Z"
      "timezone" : {
         "timezone-location" : "Europe/Stockholm",
         "timezone-utc-offset" : -60
      }
   }
}
```

TODO: convert the example above so it uses YANG hash values instead of the strings "clock", "current-datetime", "boot-datetime", "timezone", and "timezone-utc-offset". Convert the target resource URI string "system-state/clock" to a YANG hash value.

The specified object can be a list. Accordingly, the returned payload is composed of all the entries associated with the list. Each entry is returned as a (entry name, value) pair. For example, the GET of the ietf-ip/ipv6/neighbor list, sent by the client, results in the following returned payload sent by the managed entity:

```
REQ: GET example.com/mg/data/ietf-ip/ipv6/neighbor
   (Content-Format: application/cbor)


RES: 2.05 Content (Content-Format: application/cbor)
{
   "neighbor" : [
     {
    "ip" : "fe80::200:f8ff:fe21:67cf",
    "link-layer-address" : "00:00::10:01:23:45"
    },
     {
    "ip" : "fe80::200:f8ff:fe21:6708",
    "link-layer-address" : "00:00::10:54:32:10"
    },
     {
    "ip" : "fe80::200:f8ff:fe21:88ee",
    "link-layer-address" : "00:00::10:98:76:54"
    }
    ]
}
```

TODO: convert the example above so it uses YANG hash values instead of the strings "neighbor", "ip", and "link-layer-address". Convert the target resource URI string "ietf-ip/ipv6/neighbor" to a YANG hash value.

TODO: show examples using the "keys" parameter to select a specific instance of a list entry.

### 4.2. Data Editing

CoMI allows datastore contents to be created, modified and deleted using CoAP methods.

TODO: Should this be an optional feature? A server can choose to only support YANG modules with read-only objects. MIN-ACCESS conformance does not exist in YANG.

#### 4.2.1. POST

Data resource instances are created with the POST method. The RESTCONF POST operation is supported in CoMI, however it is only allowed for creation of data resources. The same constraints apply as defined in section 3.4.1 of [I-D.ietf-netconf-restconf]. The operation is mapped to the POST method defined in section 5.8.2 of [RFC7252].

There are no query parameters for the POST method.

TODO: how to support user-ordered lists in YANG? This is done with the 'insert' and 'point' parameters in RESTCONF. In CoMI, a client will have to replace the all list or leaf-list entries (e.g. PUT parent container) to change the order or insert anywhere but last.

#### 4.2.2. PUT

Data resource instances are created or replaced with the PUT method. The PUT operation is supported in CoMI. The same constraints apply as defined in section 3.5 of [I-D.ietf-netconf-restconf]. The operation is mapped to the PUT method defined in section 5.8.3 of [RFC7252].

There are no query parameters for the PUT method.

TODO: how to support user-ordered lists in YANG? Same issue as for POST.

#### 4.2.3. DELETE

Data resource instances are deleted with the DELETE method. The RESTCONF DELETE operation is supported in CoMI. The same constraints apply as defined in section 3.7 of [I-D.ietf-netconf-restconf]. The operation is mapped to the DELETE method defined in section 5.8.4 of [RFC7252].

There are no optional query parameters for the PUT method.

### 4.3. Module Discovery

Management objects are discovered in a manner similar to the RESTCONF protocol, not with the standard CoAP resource discovery. Only the YANG module information needs to be retrieved by the client. The YANG modules contain all the data resource schema and naming information.

The "rt" attribute is used to filter resource queries as specified in [RFC6690].

The resource "/mg/moduri" is used to retrieve the location of the YANG module library information for the server. This data structure is defined in the "ietf-yang-library" module in the RESTCONF draft.

Since many constrained servers within a deployment are likely to be similar, the module list can be stored locally on each server, or remotely on a different server.

TODO: Example:

Local:

REQ: GET example.com/mg/moduri

RES: 2.05 Content (Content-Format: application/cbor)

```
{
  "moduri" : "example.com/mg/data/modules"
}
```

Remote:

REQ: GET example.com/mg/moduri

RES: 2.05 Content (Content-Format: application/cbor)

```
{
  "moduri" : "example-remote-server.com/mg/data/group17/modules"
}
```

### 4.4. Error Return Codes

The RESTCONF return status codes defined in section 6 of the RESTCONF draft are used in CoMI error responses, except they are converted to CoAP error codes.

TODO: complete RESTCONF to CoAP error code mappings

| RESTCONF Status Line | CoAP Status Code |
|---|---|
| 100 Continue | none? |
| 200 OK | 2.05 |
| 201 Created | 2.01 |
| 202 Accepted | none? |
| 204 No Content | ? |
| 304 Not Modified | 2.03 |
| 400 Bad Request | 4.00 |
| 403 Forbidden | 4.03 |
| 404 Not Found | 4.04 |
| 405 Method Not Allowed | 4.05 |
| 409 Conflict | none? |
| 412 Precondition Failed | 4.12 |
| 413 Request Entity Too Large | 4.13 |
| 414 Request-URI Too Large | 4.00 |
| 415 Unsupported Media Type | 4.15 |
| 500 Internal Server Error | 5.00 |
| 501 Not Implemented | 5.01 |
| 503 Service Unavailable | 5.03 |

## 5. Mapping YANG to CoMI payload

A mapping for the encoding of YANG data in CBOR is necessary for the efficient transport of management data in the CoAP payload. Since object names may be rather long and may occur repeatedly, CoMI allows for association of a given object path identifier string value with an integer, called a "YANG hash".

### 5.1. YANG Hash Generation

The association between string value and string number is done through a hash algorithm. The "murmur3" 32-bit hash algorithm is used. This hash algorithm is described online at http://en.wikipedia.org/wiki/MurmurHash. Implementation are available online, including at https://code.google.com/p/smhasher/wiki/MurmurHash.

The hash is generated for the string representing the object path identifier. A canonical representation of the path identifier is used.

Prefix values are used on every node.

The prefix values defined in the YANG module containing the data object are used for the path expression. For external modules, this is the value of the 'prefix' sub-statement in the 'import' statement for each external module.

Path expressions for objects which augment data nodes in external modules are calculated in the augmenting module, using the prefix values in the augmenting module.

Choice and case node names are not included in the path expression. Only 'container', 'list', 'leaf', 'leaf-list', and 'anyxml' nodes are listed in the path expression.

The "murmur3_32" hash function is executed for the entire path string. The value '42' is used as the seed for the hash function.

The resulting 32-bit number is used by the server, unless the value is already being used for a different object by the server. In this case, the re-hash procedure in the following section is executed.

### 5.2. Re-Hash Procedure

A hash collision occurs if two different path identifier strings have the same hash value. If the server has over 77,000 objects in its YANG modules, then the probability of a collision is fairly high. If a hash collision occurs on the server, then the object that is causing the conflict has to be altered, such that the new hash value does not conflict with any value already in use by the server.

In most cases, the hash function is expected to produce unique values for all the objects supported by a constrained device. Given a known set of YANG modules, both server and client can calculate the YANG hashes independently, and offline.

Even though collisions are expected to happen rather rarely, they needs to be considered. Collisions can be detected before deployment, if the vendor knows which modules are supported by the server, and hence all YANG hashes can be calculated. Collisions are only an issue when they occur at the same server. The client needs to discover any re-hash mappings on a per server basis.

If the server needs to re-hash any object identifiers, then it MUST create a "rehash-map" entry for the altered identifier, as described in the following YANG module.

### 5.3. ietf-yang-hash YANG Module

The "ietf-yang-hash" YANG module is used by the server to report any objects that have been mapped to produce a new hash value that does not conflict with any other YANG hash values used by the server.

YANG tree diagram for "ietf-yang-hash" module:

```
   +--ro yang-hash
      +--ro rehash* [hash]
         +--ro hash      uint32
         +--ro path?     string
         +--ro append?   string
```

```
module ietf-yang-hash {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-hash";
  prefix "yh";

  organization
    "IETF CORE (Constrained RESTful Environments) Working Group";

  contact
    "WG Web:   <http://tools.ietf.org/wg/core/>
     WG List:  <mailto:core@ietf.org>

     WG Chair: Carsten Bormann
               <mailto:cabo@tzi.org>

     WG Chair: Andrew McGregor
               <mailto:andrewmcgr@google.com>

     Editor:   Peter van der Stok
               <mailto:consultancy@vanderstok.org>
```

```
        Editor:   Bert Greevenbosch
                  <mailto:andy@bert.greevenbosch@huawei.com>

        Editor:   Andy Bierman
                  <mailto:andy@yumaworks.com>

        Editor:   Juergen Schoenwaelder
                  <mailto:j.schoenwaelder@jacobs-university.de>

        Editor:   Anuj Sehgal
                  <mailto:s.anuj@jacobs-university.de>";

    description
      "This module contains re-hash information for the CoMI protocol.

       Copyright (c) 2014 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject
       to the license terms contained in, the Simplified BSD License
       set forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (http://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC XXXX; see
       the RFC itself for full legal notices.";

    // RFC Ed.: replace XXXX with actual RFC number and remove this
    // note.

    // RFC Ed.: remove this note
    // Note: extracted from draft-vanderstok-core-comi-05.txt

    // RFC Ed.: update the date below with the date of RFC publication
    // and remove this note.
    revision 2014-10-27 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: CoMI Protocol.";
    }

    container yang-hash {
      config false;
      description
        "Contains information on the YANG Hash values used by
         the server.";
```

```
       list rehash {
         key hash;
         description
           "Each entry describes an re-hash mapping in use by
            the server.";

         leaf hash {
           type uint32;
           description "The hash value that has a collision";
         }
         leaf path {
           type string;
           description
             "The YANG identifier path expression that caused the
              collision and is being remapped";
         }
         leaf append {
           type string;
           description
             "The string that the server appended to the path
              expression contained in the 'path' leaf to produce
              a new path expression and therefore new hash value.
              The YANG hash value for the new string (identified
              by 'path' + 'append') is used to identify the
              'path' object.";
         }
       }
     }

   }
```

#### 5.3.1. YANG Re-Hash Example

In this example the server has an object that is already registered when the "/foo:A/foo:B/foo:col1" object is processed. This object path string hashes to value "a9abdcca". The server has appended the string "_" to the path to produce a new hash ("ea7a2044") which does not collide with any other objects.

The server would return the following information if the client retrieved the "/mg/yang-hash" resource.

```
REQ: GET example.com/mg/yang-hash

RES: 2.05 Content (Content-Format: application/cbor)
{
   "ietf-yang-hash:yang-hash" : {
      "rehash" : [
         {
            "hash" : 3933872196,
            "path" :"/foo:A/foo:B/foo:col1",
            "append" : "_"
         }
      ]
   }
}
```

### 5.4. The 'keys' Query Parameter

There is a mandatory query parameter that MUST be supported by servers called "keys". This parameter is used to specify the key values for an instance of an object identified by a YANG hash value. Any key leaf values for the object are passed in order. The first key leaf in the top-most list is the first key encoded in the 'keys' parameter.

Example: In this example the following YANG module is used:

```
module foo-mod {
  namespace foo-mod-ns;
  prefix foo;

  list A {
    key "key1 key2";
    leaf key1 { type string; }
    leaf key2 { type int32; }
    list B {
      key "key3";
      leaf key3 { type string; }
      leaf col1 { type uint32; }
    }
  }
}
```

The path identifier for the leaf "col1" is the following string:

```
/foo:A/foo:B/foo:col1
```

The YANG has value for this identifier string "2846612682" (hex "a9abdcca").

The following string represents the RESTCONF target resource URI expression for the "col1" leaf for the key values "top", 17, and "group1":

```
/restconf/data/foo-mod:A=top,17/B=group1/col1
```

The following string represents the CoMI target resource identifier for the same instance of the "col1" leaf:

```
/mg/data/a9abdcca?keys=top,17,group1
```

## 6. Mapping YANG to CBOR

### 6.1. 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 (mayor 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.ietf-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 5) | 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 descriptor 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

## 7. Examples

TODO: add examples back with YANG hash values and a YANG module.

## 8. Trap functions

A trap can be set through the CoAP Observe [I-D.ietf-core-observe] function. As regular with Observe, the client subscribes to the variable by sending a GET request with an "Observe" option.

TODO: Observe example

In the registration request, the client 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 client wants the managed device to send the trap information to a multicast address.

## 9. Object 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.

### 9.1. 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.

## 10. 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:

TODO: Adapt RESTCONF <errors> data structure for use in CoMI. Need to select the most important fields like <error-path>.

```
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].

## 11. 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 [RFC7252]). 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.

## 12. IANA Considerations

'rt="core.mg.data"' needs registration with IANA.

'rt="core.mg.moduri"' needs registration with IANA.

'rt="core.mg.yang-hash"' needs registration with IANA.

Content types to be registered:

- application/comi+cbor

## 13. 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. 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.

## 14. Changelog

Changes from version 00 to version 01

- Focus on MIB only
- Introduced CBOR, JSON, removed BER
- defined mappings from SMI to xx
- Introduced the concept of addressable table rows

Changes from version 01 to version 02

- Focus on CBOR, used JSON for examples, removed XML and EXI
- added uri-query attributes mod and con to specify modules and contexts
- Definition of CBOR string conversion tables for data reduction
- use of Block for multiple fragments
- Error returns generalized
- SMI - YANG - CBOR conversion

Changes from version 02 to version 03

- Added security considerations

Changes from version 03 to version 04

- Added design considerations section
- Extended comparison of management protocols in introduction
- Added automatic generation of CBOR tables
- Moved lowpan table to Appendix

Changes from version 04 to version 05

- Merged SNMP access with RESTCONF access to management objects in small devices
- Added CoMI architecture section

- Added RESTCONf NETMOD description
- Rewrote section 5 with YANG examples
- Added server and payload size appendix
- Removed Appendix C for now. It will be replaced with a YANG example.

## 15. References

### 15.1. 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.

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

[I-D.becker-core-coap-sms-gprs] Becker, M., Li, K., Kuladinithi, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-05 (work in progress), August 2014.

[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-core-observe] Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 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.ietf-netmod-yang-json] Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-01 (work in progress), October 2014.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-02 (work in progress), October 2014.

## 15.2. 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.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, August 2014.

[I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-25 (work in progress), September 2014.

[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-04 (work in progress), September 2014.

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

[DTLS-size]
Hummen, R., Shafagh, H., Raza, S., Voigt, T., and K. Wehrle, "Delegation-based Authentication and Authorization for the IP-based Internet of Things", Web http://www.vs.inf.ethz.ch/publ/papers/mshafagh_secon14.pdf, .

[dcaf] Bormann, C., Bergmann, O., and S. Gerdes, "Delegated Authenticated Authorization for Constrained Environments", Private Information , .

[openwsn] Watteijne, T., "Coap size in Openwsn", Web http://builder.openwsn.org/, .

[Erbium] Kovatsch, M., "Erbium Memory footprint for coap-18", Private Communication , .

[management]
Schoenwalder, J. and A. Sehgal, "Management of the Internet of Things", Web http://cnds.eecs.jacobs-university.de/slides/2013-im-iot-management.pdf, 2013.

## Appendix A. Payload and Server sizes

This section provides information on code sizes and payload sizes for a set of management servers. Approximate code sizes are:

| Code | processor | Text | Data | reference |
|---|---|---|---|---|
| Observe agent | erbium | 800 | n/a | [Erbium] |
| CoAP server | MSP430 | 1K | 6 | [openwsn] |
| SNMP server | ATmega128 | 9K | 700 | [management] |
| Secure SNMP | ATmega128 | 30K | 1.5K | [management] |
| DTLS server | ATmega128 | 37K | 2K | [management] |
| NETCONF | ATmega128 | 23K | 627 | [management] |
| JSON parser | CC2538 | 4.6K | 8 | [dcaf] |
| CBOR parser | CC2538 | 1.5K | 2.6K | [dcaf] |
| DTLS server | ARM7 | 15K | 4 | [I-D.ietf-lwig-coap] |
| DTLS server | MSP430 | 15K | 4 | [DTLS-size] |
| Certificate | MSP430 | 23K | | [DTLS-size] |
| Crypto | MSP430 | 2-8K | | [DTLS-size] |

Thomas says that the size of the CoAP server is rather arbitrary, as its size depends mostly on the implementation of the underlying library modules and interfaces.

Payload sizes are compared for the following request payloads, where each attribute value is null (N.B. these sizes are educated guesses, will be replaced with generated data). The identifier are assumed to be a string representation of the OID. Sizes for SysUpTime differ due to preambles of payload. "CBOR opt" stands for CBOR payload where the strings are replaced by table numbers.

| Request | BERR SNMP | JSON | CBOR | CBOR opt |
|---|---|---|---|---|
| IPnetTOMediaTable | 205 | 327 | ~327 | ~51 |
| lowpanIfStatsTable | | 710 | 614 | 121 |
| sysUpTime | 29 | 13 | ~13 | 20 |
| RESTconf example | | | | |

## Appendix B. 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.

## Authors' Addresses

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: 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

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Juergen Schoenwaelder
Jacobs University
Campus Ring 1
Bremen 28759
Germany

Email: j.schoenwaelder@jacobs-university.de

Anuj Sehgal
Jacobs University
Campus Ring 1
Bremen 28759
Germany

Email: s.anuj@jacobs-university.de