

CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: August 9, 2013

M. Becker, Ed.  
ComNets, TZI, University Bremen  
K. Li  
Huawei Technologies  
K. Kuladinithi  
T. Poetsch  
ComNets, TZI, University Bremen  
February 5, 2013

Transport of CoAP over SMS, USSD and GPRS  
draft-becker-core-coap-sms-gprs-03

Abstract

The Short Message Service (SMS) and Unstructured Supplementary Service Data (USSD) 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), that took the limitations of LLNs into account, is thus also applicable to telematic M2M devices. The adaptation of CoAP to the SMS or USSD transport mechanisms and the combination with IP transported over cellular networks is described in this document.

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 August 9, 2013.

Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	4
1.1. Motivation . . . . .	4
2. Terminology . . . . .	5
3. Requirements Language . . . . .	5
4. Scenarios . . . . .	5
4.1. MO-MT Scenarios . . . . .	5
4.2. MT Scenarios . . . . .	6
4.3. MO Scenarios . . . . .	7
5. Examples . . . . .	8
6. Message Exchanges . . . . .	13
6.1. Message Exchange for SMS in a Cellular-To-Cellular Mobile-Originated and Mobile-Terminated Scenario . . . . .	13
6.2. Message Exchange for USSD . . . . .	14
7. Encoding of CoAP for non-IP transports . . . . .	15
7.1. Encoding of CoAP for SMS transport . . . . .	15
7.2. Encoding of CoAP for USSD transport . . . . .	16
8. Message Size Implementation Considerations . . . . .	16
9. Addressing . . . . .	16
10. Options . . . . .	16
10.1. New Options for mixed IP/non-IP operation. . . . .	17
11. URI Scheme . . . . .	17
11.1. URI Scheme . . . . .	17
11.1.1. Formal Definition . . . . .	18
11.1.2. Example . . . . .	18
12. Transmission Parameters . . . . .	18
13. Multicast . . . . .	18
14. Proxying Considerations . . . . .	18
14.1. Mobile Telematic Server . . . . .	19
14.2. Mobile Telematic Client . . . . .	20
14.3. Mobile Server . . . . .	21
14.4. Mobile Client . . . . .	22
15. Security Considerations . . . . .	24
16. IANA Considerations . . . . .	24
16.1. CoAP Option Number . . . . .	24
16.2. URI Scheme Registration . . . . .	25
17. Acknowledgements . . . . .	26
18. References . . . . .	26
18.1. Normative References . . . . .	26
18.2. Informative References . . . . .	27
Appendix A. Changelog . . . . .	28
Authors' Addresses . . . . .	28

## 1. Introduction

This specification details the usage of the Constrained Application Protocol on the Short Message Service (SMS) or Unstructured Supplementary Service Data (USSD) 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 and USSD 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 Open Mobile Alliance (OMA) LightweightM2M technical specification [oma\_lightweightm2m\_ts], SMS is identified as an alternative transport for CoAP messages.

In 3GPP, SMS is identified as the transport protocol for small data transmissions (See [3gpp\_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 [3gpp\_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). USSD does seem to be in standardisation as a solution for MTC Device Trigger.

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.

USSD is a very basic service in mobile networks which uses fewer network components to provide a service similar to SMS. This makes USSD very cheap for mobile network operators and chipset manufacturers as they do not have to provide additional infrastructure. This is why USSD is from a technical point of view supported by all handsets and other mobile devices in all networks.

Where short messages are normally stored in the SMS Center (SMS-C) before the actual delivery takes place, USSD messages are not stored but delivered immediately. If it is impossible to deliver a USSD message within the first attempt, delivery fails. This could be a problem, but could also be seen as an advantage as long as delivery problems are covered by higher level protocols, such as CoAP.

Without store-and-forward mechanisms the delivery is absolutely deterministic. There is only "success" or "failure" and no "wait a minute".

## 2. Terminology

The terms CoAP Server and CoAP Client are used synonymously to Server and Client as specified in the terminology section of [I-D.ietf-core-coap].

## 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. 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

Figure 1 to Figure 5 show various applicable usage scenarios of CoAP in M2M communications. Two mobile cellular terminals communicate by exchanging CoAP Request and Response embedded into short message protocol data units (PDUs) (depicted in Figure 1).

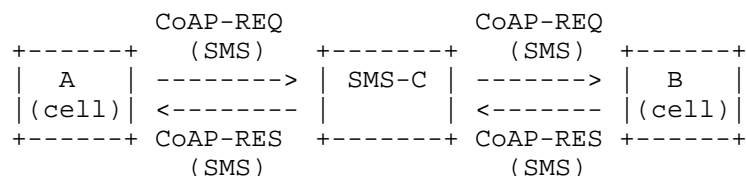


Figure 1: Cellular and Cellular Communication (only SMS-based)

Two mobile cellular terminals communicate by exchanging the CoAP Request in a short message PDU and the CoAP Response using GPRS transport. (depicted in Figure 2).

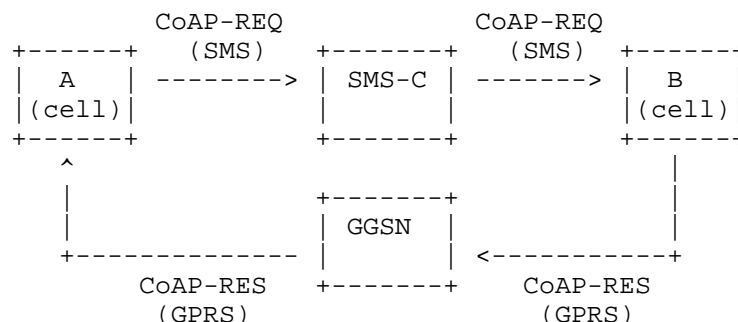


Figure 2: Cellular and Cellular Communication (SMS/GPRS-based)

The support for GPRS for the CoAP responses might be useful, so as to use SMS only for the request and as a wake-up signal for the device hosting the CoAP server. That device could then initiate a packet data protocol (PDP) context with the cellular network in order to bring up Internet connectivity. After having setup Internet connectivity, further message exchange can fully rely on IP. Network initiated PDP contexts could partly obsolete this mechanism.

#### 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 [ucpl])) to submit a short message for delivery, which contains the CoAP Request (depicted in Figure 3).

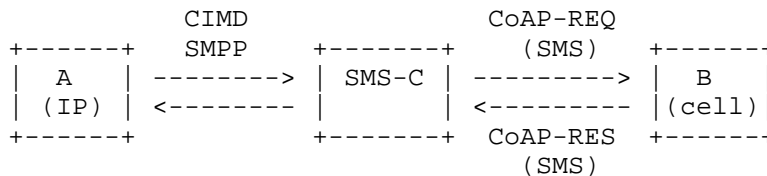


Figure 3: IP and Cellular Communication (only SMS-based)

Again, the return path for the CoAP Response might be GPRS (depicted in Figure 4).

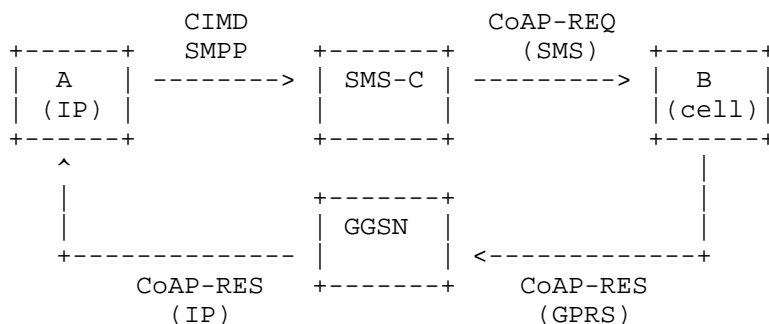


Figure 4: IP and Cellular Communication (SMS and GPRS-based)

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

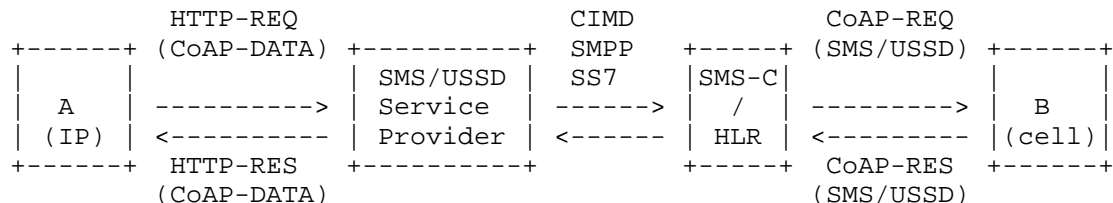


Figure 5: IP and Cellular Communication (only SMS/USSD-based, using an SMS/USSD service provider)

#### 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 [ucpl])) as depicted in Figure 6). This scenario can be a fall-back for mobile-originating communication, when IP connectivity cannot be setup (e.g. due to missing coverage).

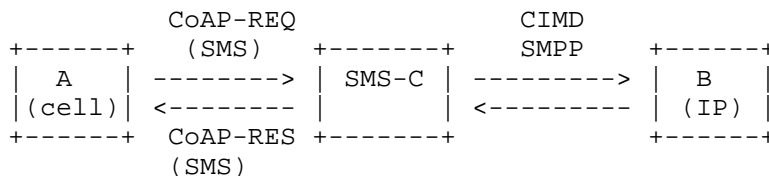


Figure 6: Cellular and IP Communication (only SMS-based)

There are service providers offering SMS and/or USSD delivery and notification using an HTTP/REST interface (depicted in Figure 7).

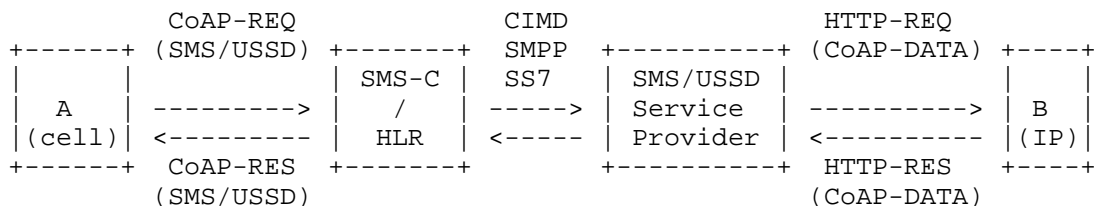


Figure 7: IP and Cellular Communication (only SMS/USSD-based, using an SMS/USSD service provider)

If IP connectivity can be setup by the mobile cellular device, the complete communication can be handled using UDP/IP by employing regular CoAP [I-D.ietf-core-coap] (depicted in Figure 8).



Figure 8: Cellular and IP Communication (GPRS-based)

## 5. Examples

Two mobile cellular terminals communicate by exchanging the CoAP Request in an SMS PDU and the CoAP Response using GPRS transport. (depicted in Figure 9).

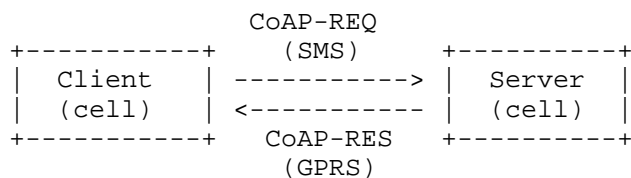


Figure 9

In the examples below, Client (Client Address A) sends GET request to Server (Server Address A) through SMS, and uses Response-To-URI-Host and Response-To-URI-Port to indicate the IP address and port (Client Address B). Then the Server (Server Address B) sends back the response to the Client through GPRS to Client Address B. The tel:



addresses in the examples are to be interpreted as described in RFC 3966 [RFC3966].

Client Address A (CA): tel:+1-201-555-0123  
 Client Address B (CB): 10.1.1.1  
 Server Address A (SA): tel:+1-201-555-0124  
 Server Address B (SB): 10.1.1.2

Figure 10

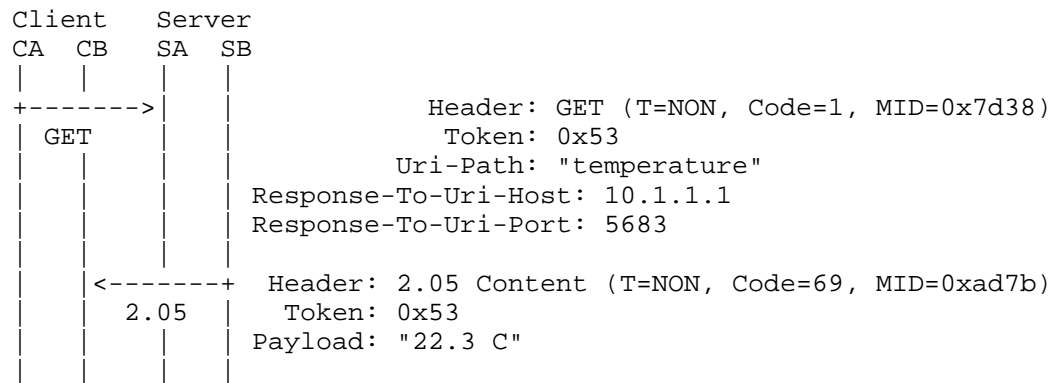


Figure 11: NON A, NON B

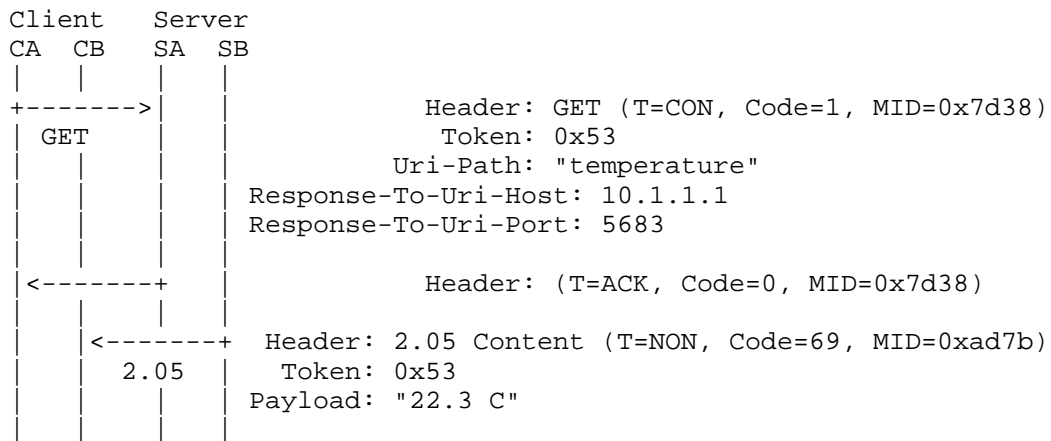


Figure 12: CON A, ACK A, NON B (separate)

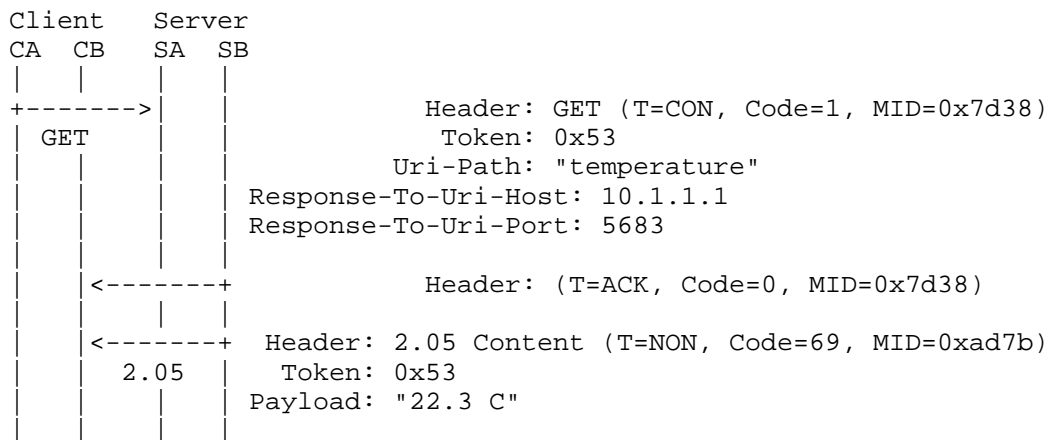


Figure 13: CON A, ACK B, NON B (separate)

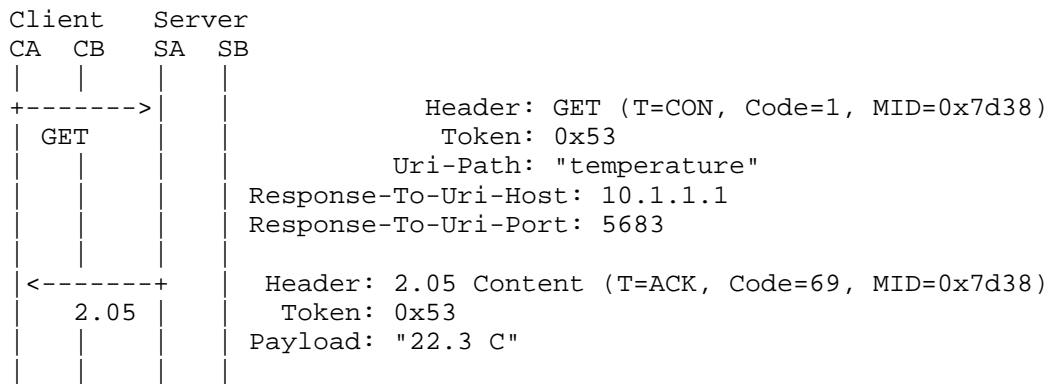


Figure 14: CON A, ACK A

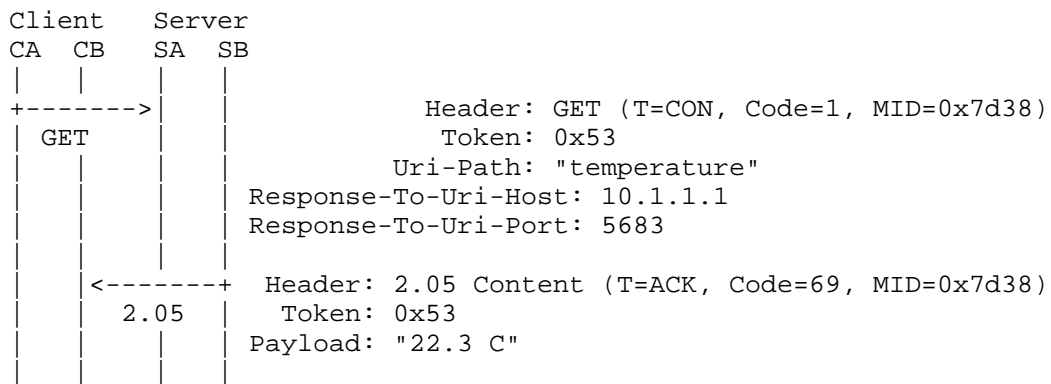


Figure 15: CON A, ACK B

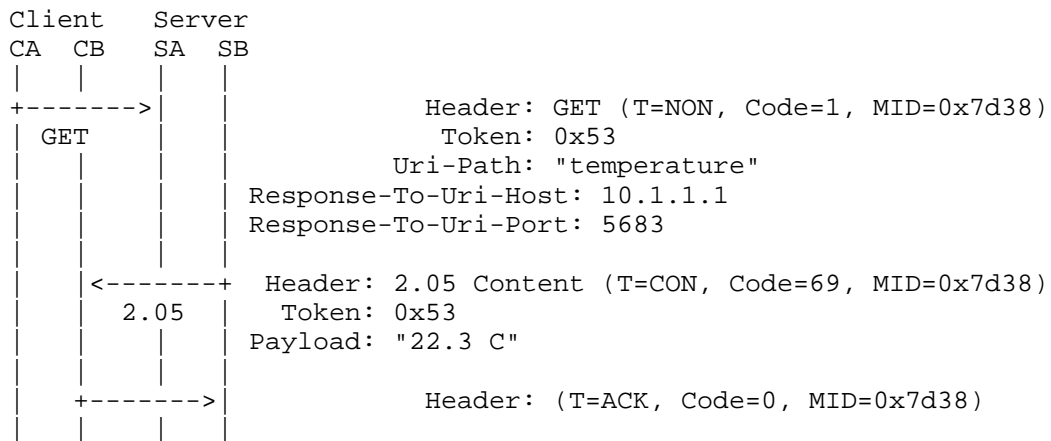


Figure 16: NON A, CON B, ACK B

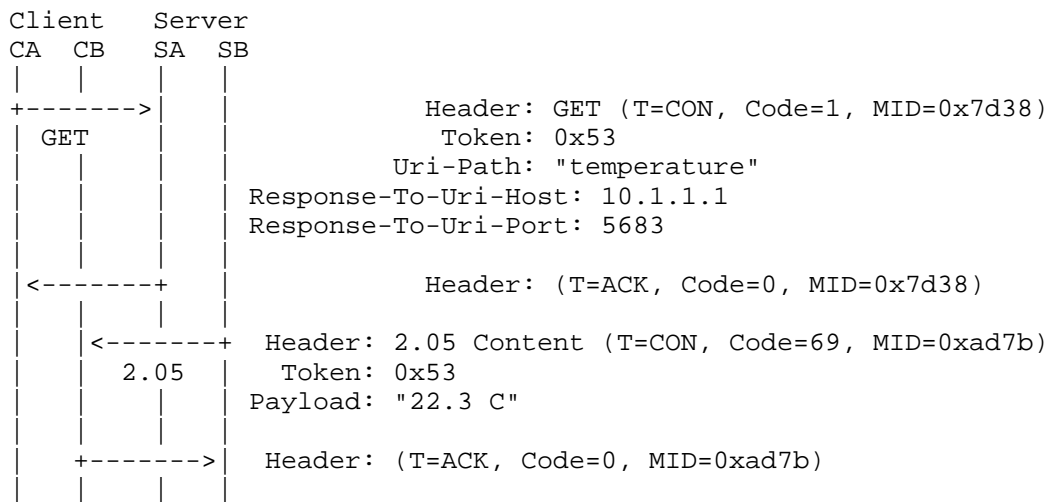


Figure 17: CON A, ACK A, CON B, ACK B (separate)

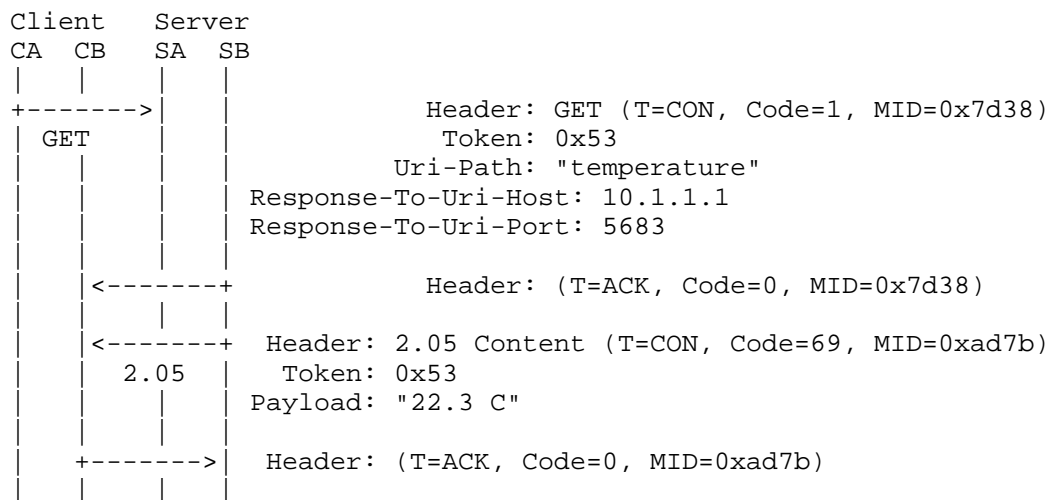


Figure 18: CON A, ACK B, CON B, ACK B (separate)

## 6. Message Exchanges

### 6.1. 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 the 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:

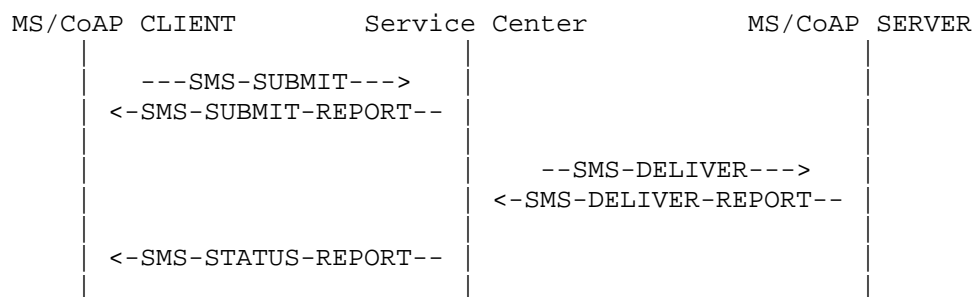


Figure 19: CoAP Messages over SMS

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 [3gpp\_ts\_23\_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 [3gpp\_ts\_23\_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.

## 6.2. Message Exchange for USSD

The message exchange for USSD is shown simplified in Figure 20 and Figure 21. The communication between MS, MSC, VLR, HLR, and USSD-GW is based on SS7 signalling and the communication between USSD-GW is based on IP. Messages ending with \_RPC are Remote Procedure Calls (e.g. REST); messages without \_RPC are SS7 signalling.

Message Sequence Charts with more details can be found in [3gpp\_ts23\_090].

In Figure 20 the message sequence chart for the USSD transport (Mobile Originated) is shown.

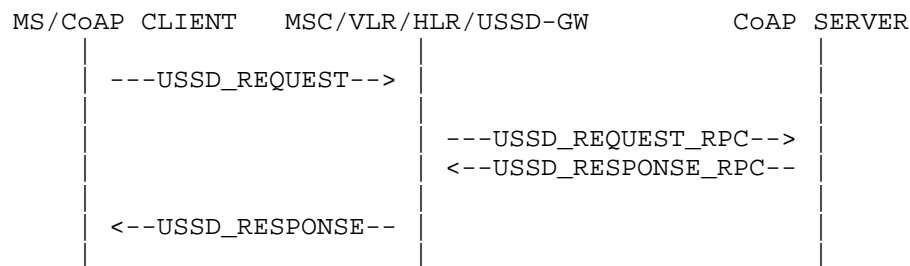


Figure 20: CoAP Messages over USSD (Mobile Originated)

In Figure 21 the message sequence chart for the USSD transport (Mobile Terminated) is shown.

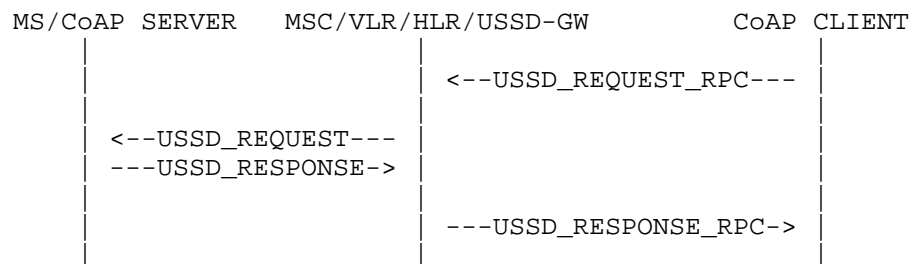


Figure 21: CoAP Messages over USSD (Mobile Terminated)

## 7. Encoding of CoAP for non-IP transports

### 7.1. Encoding of CoAP for SMS transport

The content of a short message can be coded in 7, 8 or 16 bit characters [3gpp\_ts23\_038]. The advantages and disadvantages are:

- a. 7 bit encoding: Sending 7 bit encoded short message possible with almost all devices. CoAP binary data needs to be re-encoded, possibly with Base64 RFC 4648 [RFC4648].
- b. 8 bit encoding: CoAP binary data does not need to be re-encoded. Not all telematic devices support 8 bit short message encoding.
- c. 16 bit encoding: CoAP binary data needs to be re-encoded. Not all telematic devices support 16 bit short message encoding.

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

## 7.2. Encoding of CoAP for USSD transport

The encoding of USSD data is identical to the encoding of short messages.

## 8. Message Size Implementation Considerations

Using 7 bit encoding 160 characters are allowed in 1 short message, while using 8 bit encoding 140 characters are allowed.  
[3gpp\_ts23\_038]

Possible options for larger CoAP messages are:

- a. Multiple short message concatenation
- b. CoAP Block [I-D.ietf-core-block]

It is RECOMMENDED that SMS is not used to transfer very large resource data using Blocks.

There is no possibility to concatenate messages with USSD, thus the only option would be CoAP Block is necessary.

## 9. 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.

For mobile-originated USSD the addressing is done by a so called application numbers.

## 10. Options

Uri-Host: Contrary to the default value of the Uri-Host Option being the IP literal as given in [I-D.ietf-core-coap], the default value when using CoAP with the coap+tel:// scheme is the telephone-subscriber as defined in RFC3966. If Uri-Host is not the default value, the value is an IP literal as in [I-D.ietf-core-coap]

Uri-Port: The default value of the Uri-Port is not useful in combination with the coap+tel:// scheme. Therefore Uri-Port MUST NOT



be included, if the Uri-Host is the default value or is not included in the message.

End-points receiving CoAP messages over SMS with such options MUST behave as specified in [I-D.ietf-core-coap].

#### 10.1. New Options for mixed IP/non-IP operation.

When CoAP should be used in mixed IP and non-IP mode (e.g. SMS/USSD and GPRS as in Figure 2 and Figure 4) the server needs to be informed about the client's alternative address that should be used for the CoAP Response. 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
34					Response-To-Uri-Host	string	1-270 B	(none)
38					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.

#### 11. 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/USSD. The URI scheme for CoAP over SMS/USSD is derived from the CoAP scheme in [I-D.ietf-core-coap]. As there is no host and port available for the SMS/USSD transport, those parts are replaced with the "telephone-subscriber" from [RFC3966].

##### 11.1. URI Scheme

The syntax of the "coap+tel" URI scheme is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of

"path-abempty", "query", are adopted from [RFC3986]. The definition of "telephone-subscriber" is adopted from [RFC3966].

#### 11.1.1. Formal Definition

```
coap-sms-URI = "coap+tel:" "/" telephone-subscriber
               path-abempty [ "?" query ]

telephone-subscriber = <defined in RFC3966>"
path-abempty         = <defined in RFC3986>"
query                = <defined in RFC3986>"
```

#### 11.1.2. Example

```
coap+tel://+15105550101/.well-known/core
```

### 12. Transmission Parameters

It is RECOMMENDED to configure the RESPONSE\_TIMEOUT variable for a higher duration than specified in [I-D.ietf-core-coap] for the applications described here. The actual value SHOULD be chosen based on experience with SMS, USSD and GPRS.

### 13. Multicast

Multicast is not possible with SMS and USSD transports.

### 14. Proxying Considerations

In case of non-IP transport, several use cases might arise for proxies:

- o For a CoAP IP Client and a Mobile Terminated CoAP Server: An HTTP-CoAP Proxy at the mobile network / IP network border.
- o For a Mobile Originated CoAP Client and (CoAP/HTTP) IP Server: A CoAP-CoAP or CoAP-HTTP Proxy at the mobile network and IP network border or in the server network.
- o If an LLN is attached to the Mobile: A CoAP-CoAP Proxy into the LLN.

In Figure 22 a typical M2M scenario is shown where a User (U) is connected by an IP network to an M2M service provider (M). Over a cellular network the M2M telematic device (T) is attached. Possibly

a constrained network is attached to the telematic device.

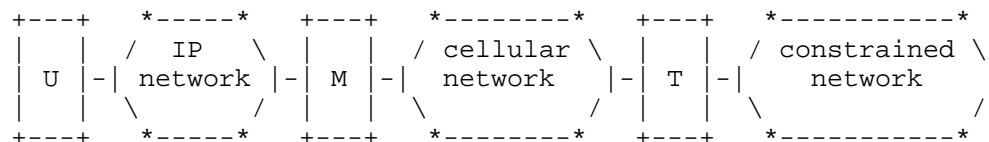


Figure 22: M2M architecture

In sections Section 14.1 to Section 14.4 several combinations of CoAP and HTTP clients, servers and proxies are shown. The various cases are not distinct but can be mixed to meet the M2M requirements.

#### 14.1. Mobile Telematic Server

C-C: CoAP Client

C-S: CoAP Server

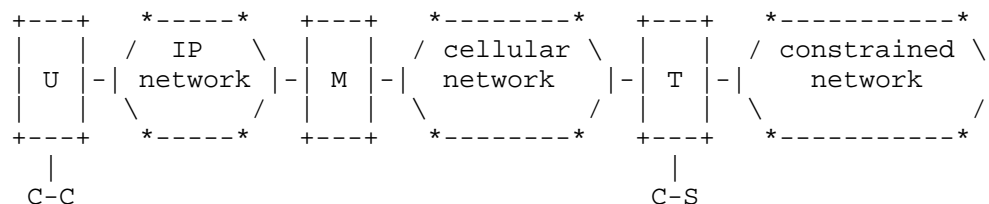


Figure 23: M2M architecture (Mobile Telematic Server) A

C-C: CoAP Client

C-C-P: CoAP-CoAP Proxy (Forward Proxy)

C-S: CoAP Server

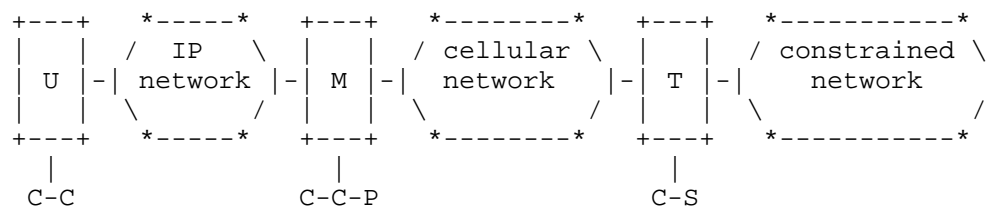


Figure 24: M2M architecture (Mobile Telematic Server) B

H-C: HTTP Client  
H-C-P: HTTP-CoAP Proxy (Forward Proxy)  
C-S: CoAP Server

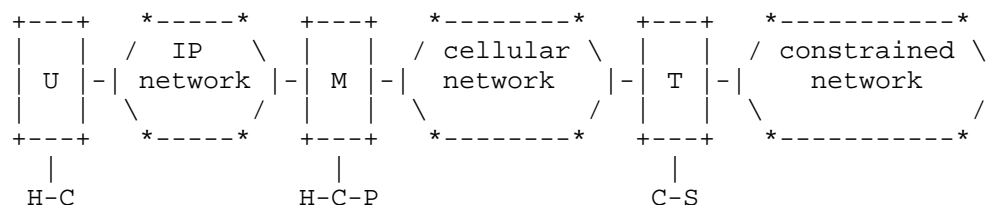


Figure 25: M2M architecture (Mobile Telematic Server) C

#### 14.2. Mobile Telematic Client

C-C: CoAP Client  
C-S: CoAP Server

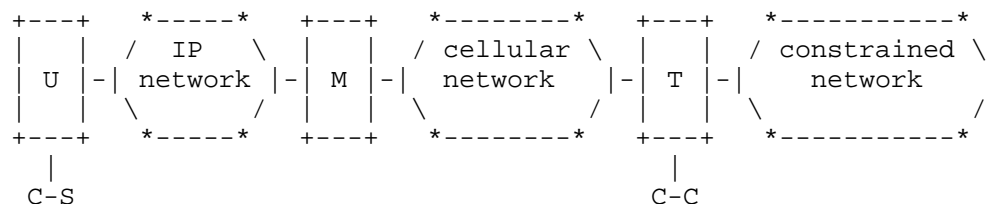


Figure 26: M2M architecture (Mobile Telematic Client) A

C-C: CoAP Client  
C-C-P: CoAP-CoAP Proxy (Forward Proxy)  
C-S: CoAP Server

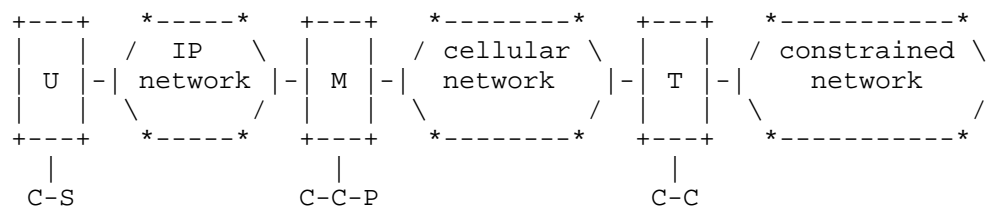


Figure 27: M2M architecture (Mobile Telematic Client) B

C-C: CoAP Client  
H-C-P: HTTP-CoAP Proxy (Forward Proxy)  
H-S: HTTP Server

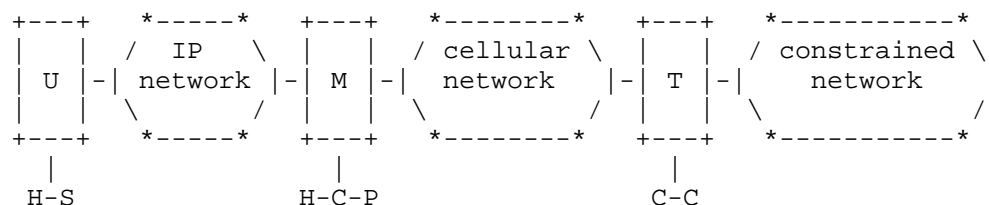


Figure 28: M2M architecture (Mobile Telematic Client) C

#### 14.3. Mobile Server

C-C: CoAP Client  
C-S: CoAP Server

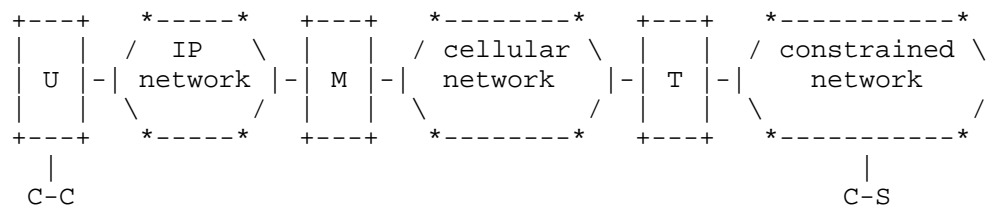


Figure 29: M2M architecture (Mobile Server) A

C-C: CoAP Client  
C-C-P: CoAP-CoAP Proxy (Forward Proxy)  
C-S: CoAP Server

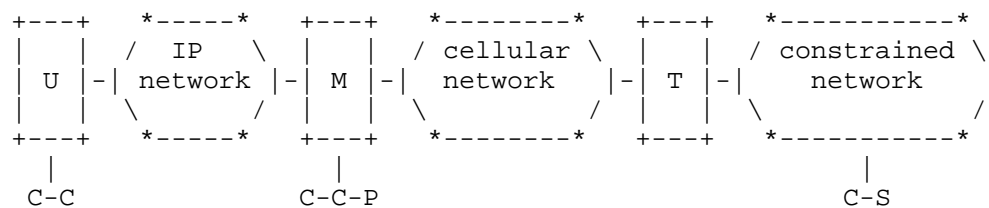


Figure 30: M2M architecture (Mobile Server) B

H-C: HTTP Client  
H-C-P: HTTP-CoAP Proxy (Cross-Protocol Forward Proxy)  
C-S: CoAP Server

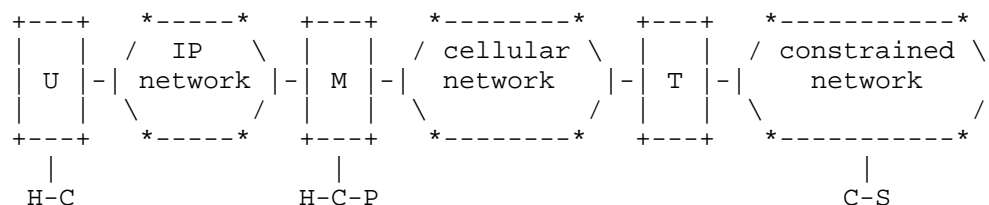


Figure 31: M2M architecture (Mobile Server) C

C-C: CoAP Client  
C-C-P1: CoAP-CoAP Proxy (Forward Proxy)  
C-C-P2: CoAP-CoAP Proxy (Mirror Proxy)  
C-S: CoAP Server (actually Clients which PUT to C-C-P2)

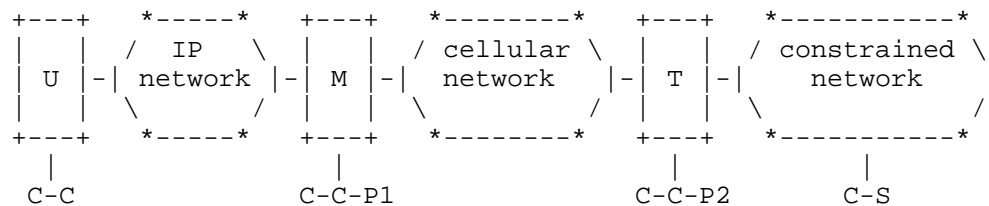


Figure 32: M2M architecture (Mobile Server) D

#### 14.4. Mobile Client

C-C: CoAP Client  
C-S: CoAP Server

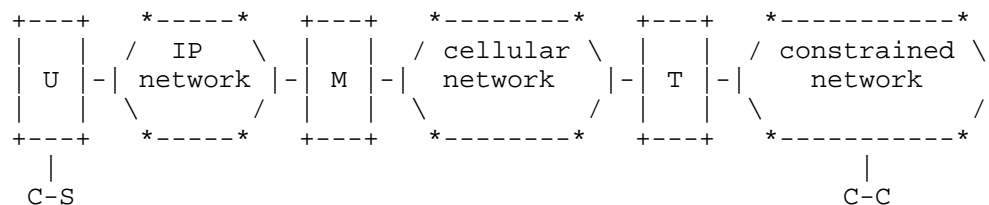


Figure 33: M2M architecture (Mobile Client) A

C-C: CoAP Client  
 C-C-P: CoAP-CoAP Proxy (Reverse Proxy)  
 C-S: CoAP Server

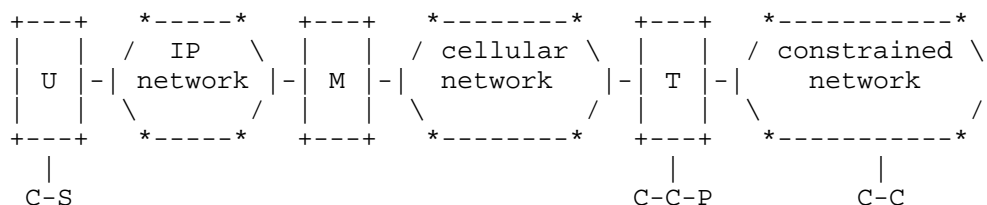


Figure 34: M2M architecture (Mobile Client) B

C-C: CoAP Client  
 C-C-P1: CoAP-CoAP Proxy (Forward Proxy)  
 C-C-P2: CoAP-CoAP Proxy (Mirror Proxy)  
 C-S: CoAP Server (actually Clients which PUT to C-C-P2)

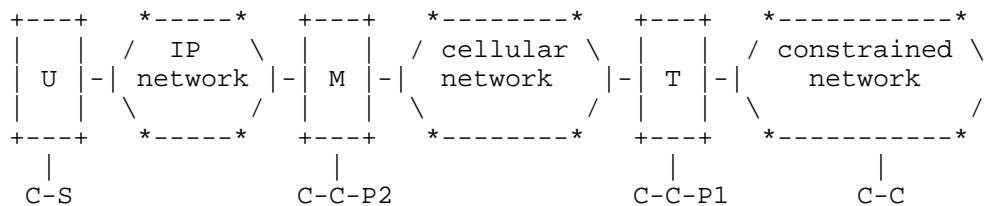


Figure 35: M2M architecture (Mobile Client) C

C-C: CoAP Client  
 C-C-P: CoAP-CoAP Proxy (Forward Proxy)  
 H-C-P: HTTP-CoAP Proxy (Mirror Proxy)  
 H-S: HTTP Server

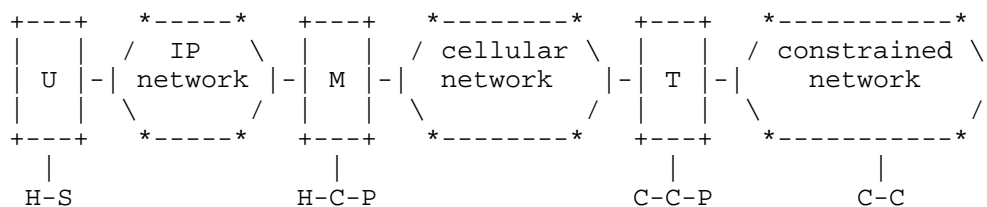


Figure 36: M2M architecture (Mobile Client) D

## 15. 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.

As this option is used to redirect the response to another address, it may be used by a malicious party to send it to an address other than its own. For example, A can use his mobile phone to send an SMS/CoAP GET, with B's IP address as Response-To-Uri-Host. In this way, B will GET data that he never requested.

To avoid this, server implementations need to verify if the requesting client is a trusted client, and also verify if the redirected address is a trusted address.

Security in the cellular network operator network at transport layer by using dedicated Access Points Names (APNs) for the GPRS M2M data. Security for the access to the cellular network operator network (for GPRS/IP as well as short message submission) can be provided at transport layer as well, e.g. by Transport Layer Security (TLS) or Virtual Private Networks (VPNs). Security mechanisms defined in [3gpp\_ts23\_888] are used to guarantee transport security. The CoAP Payload can be secured using Object Security. If the digital signature does not match pre-shared certificates or decryption fails with a pre-shared key, the server SHOULD ignore the message.

## 16. IANA Considerations

### 16.1. CoAP Option Number

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

Number	Name	Reference
34	Response-To-Uri-Host	Section 2 of this document
38	Response-To-Uri-Port	Section 2 of this document



## 16.2. URI Scheme Registration

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+tel". The registration request complies with [RFC4395].

URI scheme name.

coap+tel

Status.

Permanent.

URI scheme syntax.

Defined in Section 11 of [RFCXXXX].

URI scheme semantics.

TBD

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 over non-IP transports, i.e. cellular networks.

Interoperability considerations.

None.

Security considerations.

See Section 15 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

## References.

[RFCXXXX]

## 17. 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.

## 18. References

## 18.1. Normative References

[3gpp\_ts23\_038]

ETSI 3GPP, "Technical Specification: Alphabets and language-specific information (3GPP TS 23.038 version 10.0.0 Release 10)", 2011.

[3gpp\_ts23\_090]

ETSI 3GPP, "Technical Specification: Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Unstructured Supplementary Service Data (USSD); Stage 2 (3GPP TS 23.090 version 10.0.0 Release 10)", 2011.

[I-D.bormann-coap-misc]

Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-22 (work in progress), December 2012.

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-10 (work in progress), October 2012.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.

[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.

## 18.2. Informative References

- [3gpp\_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.
- [3gpp\_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.
- [3gpp\_ts\_23\_040] 3GPP, "Technical realization of the Short Message Service (SMS)", 3GPP-23.040 a00, Mar 2011.
- [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.
- [ucp] Vodafone, "Short Message Service Centre (SMSC) External Machine Interface (EMI) Description Version 4.3d", 2011.

## Appendix A. Changelog

Changed from draft-02 to draft-03:

- o Added reference to OMA LightweightM2M Technical Specification in "Motivation" section.
- o Chose CoAP option numbers and updated the option number table to meet draft-ietf-core-coap-13. Table 1

Changed from draft-01 to draft-02:

- o Added security considerations: Transport and Object Security. Section 15
- o Reply-To-\* changed to Response-To-\*. Section 16 and Section 10.1
- o Added URI scheme. Section 11
- o Added possible CON/NON/ACK interactions. Section 5
- o Added possible M2M proxy scenarios. Section 14
- o Added reference to bormann-coap-misc for other SMS encoding. Section 7.1
- o Updated requirements on Uri-Host and Uri-Port for coap+tel://. Section 10
- o Chose CoAP option numbers and updated the option number table to meet draft-ietf-core-coap-10. Table 1
- o Added an IANA registration for the URI scheme. Section 16.2

## 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 Working Group  
Internet-Draft  
Intended status: Informational  
Expires: February 14, 2013

C. Bormann  
Universitaet Bremen TZI  
August 13, 2012

CoAP Simple Congestion Control/Advanced  
draft-bormann-core-cocoa-00

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 -00, it is mainly a straw-man document to gauge the implementation effort required, with a view towards simplifying it further.

Status of this Memo

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

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

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

This Internet-Draft will expire on February 14, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

## Table of Contents

1. Introduction . . . . .	3
1.1. Terminology . . . . .	3
2. Context . . . . .	4
3. Advanced CoAP Congestion Control: RTO Estimation . . . . .	5
3.1. Blind RTO Estimate . . . . .	5
3.2. Measured RTO Estimate . . . . .	5
3.2.1. Modifications to the algorithm of RFC 6298 . . . . .	6
3.2.2. Discussion . . . . .	6
3.3. Lifetime, Aging . . . . .	6
4. Advanced CoAP Congestion Control: Non-Confirmables . . . . .	7
4.1. Discussion . . . . .	7
5. IANA Considerations . . . . .	8
6. Security Considerations . . . . .	9
7. Acknowledgements . . . . .	10
8. References . . . . .	11
8.1. Normative References . . . . .	11
8.2. Informative References . . . . .	11
Author's Address . . . . .	12



## 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

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.

This specification assumes approximately the following text in the [I-D.ietf-core-coap] base specification:

1. Change SHOULD in second paragraph of [I-D.ietf-core-coap] 4.7 to MUST; define protocol parameter NSTART as 1.
2. Add text that permits advanced congestion control mechanisms and allows them to change protocol parameters, including NSTART and the binary exponential backoff mechanism.
3. Specify that, outside of exchanges, non-confirmable messages cannot be used without an advanced congestion control mechanism (this is mainly relevant for -observe).
4. Add reference to (and/or cite) [RFC5405] guideline about combining congestion control state for a destination; clarify its meaning for CoAP using the definition of an endpoint.

Additional changes have been made to limit the leeway that implementations have in changing the CoRE protocol parameters; these changes are already gathered in Section 4.8 of [I-D.ietf-core-coap] and will not be repeated here.

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

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

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

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. Servers will only trigger 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 RTT estimate much shorter than the 2 to 3 s used as a default SHOULD therefore not expend all of its retransmissions in the shorter estimated timescale.

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.

Up to four (4) exchanges to an endpoint can be started in parallel. If only the initial RTO estimate is available, the RTO estimate for each exchange started in parallel to other exchanges is set to the highest binary multiple of the parallel exchanges, e.g., if another exchange is already running and is into its second retransmission, the RTO estimate for the additional exchange is 8 seconds.

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

#### 3.2. 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 (the "weak estimator"). For the latter, there is some ambiguity whether a response is based on the initial transmission or any retransmission. For the purposes of the weak estimator, the time from the initial transmission counts.

The overall RTO estimate is a 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 or to the strong estimator, from the estimator that made the most recent contribution:

$$\text{RTO\_overall\_} := 0.5 * \text{RTO\_recent\_} + 0.5 * \text{RTO\_overall\_}$$

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

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

### 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 a time that is more than 16 times its current value, its value is doubled.

(It is allowed to implement this cumulatively at the time it is used next, possibly approximating multiple doublings by replacing the value with 1/8th of the time that has elapsed since the last update. Alternatively, simple estimators can be simply updated to 1 s after being without update for a time that is more than 16 times its value, or, even simpler, be clamped at 1 s or above.)

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

A CoAP endpoint can send non-confirmables to another CoAP endpoint only at a rate as defined by this document.

Independent of any congestion control mechanisms, a CoAP endpoint can always send non-confirmables if a rate of 1 B/s is not exceeded.

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 above.
3. The packet rate of non-confirmable messages cannot exceed  $1/\text{RTO}$ , where RTO is the overall RTO estimator value at the time the non-confirmable packet is sent.

##### 4.1. Discussion

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

## 5. IANA Considerations

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

## 6. Security Considerations

(TBD. The security considerations of, e.g., [RFC5681], [RFC2914], and [RFC5405] apply. Some issues are already discussed in the security considerations of [I-D.ietf-core-coap].)

## 7. Acknowledgements

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

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



## 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.

### 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-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-11 (work in progress), July 2012.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-05 (work in progress), March 2012.
- [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.

Author's Address

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

Phone: +49-421-218-63921

Email: [cabo@tzi.org](mailto:cabo@tzi.org)



CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: June 9, 2013

C. Bormann  
Universitaet Bremen TZI  
December 06, 2012

A convention for URIs operating a HTTP-CoAP reverse proxy  
draft-bormann-core-cross-reverse-convention-00

## Abstract

CoAP is a RESTful transfer protocol for constrained nodes and networks. In many applications, CoAP will be used via cross-protocol proxies from HTTP clients. HTTP client libraries may make it hard to operate an HTTP-CoAP forward proxy by not providing a way to put a CoAP URI on the HTTP Request-Line; reverse-proxying may therefore lead to wider applicability of a proxy. This specification will define a convention for URIs operating such a HTTP-CoAP reverse proxy.

The current version of this specification is a placeholder only. It is meant to pick up <http://trac.tools.ietf.org/wg/core/trac/ticket/259> and provide a home for its considerations. It might be merged with other documents later.

## 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 June 9, 2013.

## Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	3
2. Convention . . . . .	4
3. Examples . . . . .	5
4. IANA Considerations . . . . .	6
5. Security Considerations . . . . .	7
6. Acknowledgements . . . . .	8
7. References . . . . .	9
7.1. Normative References . . . . .	9
7.2. Informative References . . . . .	9
Author's Address . . . . .	10

## 1. Introduction

(see abstract for now)

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119] and indicate requirement levels for compliant CoAP implementations.

In this document, the term "byte" is used in its now customary sense as a synonym for "octet".

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C, except that the operator "\*\*\*" stands for exponentiation.

## 2. Convention

(This is a placeholder document. This section will provide a convention.)

### 3. Examples

In his original contribution, Cullen Jennings proposed translating

```
http://www.proxy.com/.wellknown/core-translate/1.2.3.4_4567/foo/  
bar?a=3
```

to

```
coap://1.2.3.4:4567/foo/bar?a=3
```



#### 4. IANA Considerations

(none foreseen.)

## 5. Security Considerations

TBD.

## 6. Acknowledgements

The original point that this document might be needed was brought up by Cullen Jennings.

## 7. References

### 7.1. Normative References

- [I-D.ietf-core-coap]     Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
                         "Constrained Application Protocol (CoAP)",  
                         draft-ietf-core-coap-12 (work in progress), October 2012.
- [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.

### 7.2. Informative References

- [REST]     Fielding, R., "Architectural Styles and the Design of  
                         Network-based Software Architectures", Ph.D. Dissertation,  
                         University of California, Irvine, 2000, <[http://  
www.ics.uci.edu/~fielding/pubs/dissertation/  
fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.

Author's Address

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

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



CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: June 9, 2013

C. Bormann  
Universitaet Bremen TZI  
December 06, 2012

Using CoAP with IPsec  
draft-bormann-core-ipsec-for-coap-00

Abstract

CoAP is a RESTful transfer protocol for constrained nodes and networks. Security for the protocol can be supplied in a number of ways. The mandatory-to-implement security mode for CoAP makes use of DTLS. Other applications may want to use IPsec.

This document will discuss considerations for the use of IPsec with CoAP. It will be advanced on a timescale separate from the main CoAP specification, as most experience in securing CoAP so far has been made with DTLS.

The current version of this specification is a placeholder, built out of text extracted from draft-ietf-core-coap-12. It is meant to pick up <http://trac.tools.ietf.org/wg/core/trac/ticket/262> and provide a home for its considerations. It might be merged with other documents later.

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 June 9, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	3
2. Using CoAP with IPsec . . . . .	4
3. IANA Considerations . . . . .	5
4. Security Considerations . . . . .	6
5. Acknowledgements . . . . .	7
6. References . . . . .	8
6.1. Normative References . . . . .	8
6.2. Informative References . . . . .	8
Author's Address . . . . .	9



## 1. Introduction

(see abstract for now)

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119] and indicate requirement levels for compliant CoAP implementations.

In this document, the term "byte" is used in its now customary sense as a synonym for "octet".

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C, except that the operator "\*\*\*" stands for exponentiation.

## 2. Using CoAP with IPsec

One mechanism to secure CoAP [I-D.ietf-core-coap] in constrained environments is the IPsec Encapsulating Security Payload (ESP) [RFC4303] when CoAP is used without DTLS in NoSec Mode. Using IPsec ESP with the appropriate configuration, it is possible for many constrained devices to support encryption with built-in link-layer encryption hardware. For example, some IEEE 802.15.4 radio chips are compatible with AES-CBC (with 128-bit keys) [RFC3602] as defined for use with IPsec in [RFC4835]. Alternatively, particularly on more common IEEE 802.15.4 hardware that supports AES encryption but not decryption, and to avoid the need for padding, nodes could directly use the more widely supported AES-CCM as defined for use with IPsec in [RFC4309], if the security considerations in Section 9 of that specification can be fulfilled.

Necessarily for AES-CCM, but much preferably also for AES-CBC, static keying should be avoided and the initial keying material be derived into transient session keys, e.g. using a low-overhead mode of IKEv2 [RFC5996] as described in [I-D.kivinen-ipsecme-ikev2-minimal]; such a protocol for managing keys and sequence numbers is also the only way to achieve anti-replay capabilities. However, no recommendation can be made at this point on how to manage group keys (i.e., for multicast) in a constrained environment. Once any initial setup is completed, IPsec ESP adds a limited overhead of approximately 10 bytes per packet, not including initialization vectors, integrity check values and padding required by the cipher suite.

When using IPsec to secure CoAP, both authentication and confidentiality SHOULD be applied as recommended in [RFC4303]. The use of IPsec between CoAP endpoints is transparent to the application layer and does not require special consideration for a CoAP implementation.

IPsec may not be appropriate for all environments. For example, IPsec support is not available for many embedded IP stacks and even in full PC operating systems or on back-end web servers, application developers may not have sufficient access to configure or enable IPsec or to add a security gateway to the infrastructure. Problems with firewalls and NATs may furthermore limit the use of IPsec.

### 3. IANA Considerations

(none foreseen.)

#### 4. Security Considerations

TBD.

## 5. Acknowledgements

This text was extracted from draft-ietf-core-coap-12.txt and probably mostly was written by Zach Shelby.

## 6. References

### 6.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-12 (work in progress), October 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3602] Frankel, S., Glenn, R., and S. Kelly, "The AES-CBC Cipher  
Algorithm and Its Use with IPsec", RFC 3602,  
September 2003.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)",  
RFC 4303, December 2005.
- [RFC4309] Housley, R., "Using Advanced Encryption Standard (AES) CCM  
Mode with IPsec Encapsulating Security Payload (ESP)",  
RFC 4309, December 2005.
- [RFC4835] Manral, V., "Cryptographic Algorithm Implementation  
Requirements for Encapsulating Security Payload (ESP) and  
Authentication Header (AH)", RFC 4835, April 2007.
- [RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen,  
"Internet Key Exchange Protocol Version 2 (IKEv2)",  
RFC 5996, September 2010.

### 6.2. Informative References

- [I-D.kivinen-ipsecme-ikev2-minimal]  
Kivinen, T., "Minimal IKEv2",  
draft-kivinen-ipsecme-ikev2-minimal-01 (work in progress),  
October 2012.

Author's Address

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

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





CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 29, 2013

C. Bormann  
Universitaet Bremen TZI  
February 25, 2013

Representing CoRE Link Collections in JSON  
draft-bormann-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 (RFC4627).

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 August 29, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

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

## 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.

Outside of constrained environments, it may also be useful to represent the same collections of Web links in the widely used JSON format [RFC4627]. 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

(TBD: Convert the shopping list into plaintext)

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

## 1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

## 2. Web Links in JSON

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

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

We straightforwardly map:

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

In the object representing a "link-value", each target attribute or other parameter ("link-param") is represented by a JSON name/value pair (member). The name is a string representation of the parameter or attribute name (as in "paramname"), 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 [RFC4627]).

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

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

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

## 2.1. Examples

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

Figure 1: Example from page 15 of [RFC6690]

becomes

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

(More examples to be added.)

### 3. IANA Considerations

(TBD. All the Media Type boilerplate, too, for:)

application/link-format+json

### 4. Security Considerations

(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.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

#### 6.2. Informative References

- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.pbryan-zyp-json-ref] Bryan, P. and K. Zyp, "JSON Reference", draft-pbryan-zyp-json-ref-03 (work in progress), September 2012.
- [MNOT11] Nottingham, M., "Linking in JSON", November 2011, <[http://www.mnot.net/blog/2011/11/25/linking\\_in\\_json](http://www.mnot.net/blog/2011/11/25/linking_in_json)>.
- [RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

## 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: Informational  
Expires: July 7, 2013

A. Castellani  
University of Padova  
S. Loreto  
Ericsson  
A. Rahman  
InterDigital Communications, LLC  
T. Fossati  
KoanLogic  
E. Dijk  
Philips Research  
January 3, 2013

Best Practices for HTTP-CoAP Mapping Implementation  
draft-castellani-core-advanced-http-mapping-01

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 July 7, 2013.

Copyright Notice

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

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

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

## Table of Contents

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



## 1. Terminology and Conventions

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

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

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

At least two different kinds of HC proxies exist:

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

## 2. Introduction

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

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

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

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

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

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

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

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

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

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

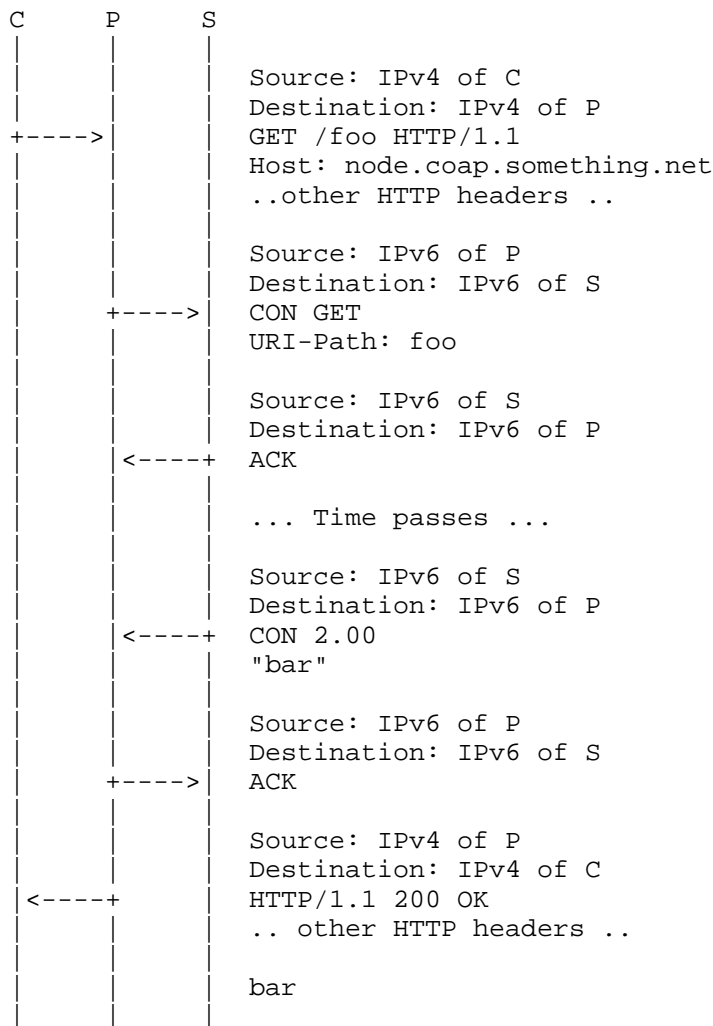


Figure 1: HTTP/IPv4 to CoAP/IPv6 Mapping

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

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

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

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

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

#### 4. Multiple Message Exchanges Mapping

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

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

##### 4.1. Relevant Features of Existing Standards

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

###### 4.1.1. Multipart Messages

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

###### 4.1.2. Immediate Message Delivery

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

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

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

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

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

#### 4.1.3. Detailing Source Information

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

#### 4.2. Multicast Mapping

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

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

##### 4.2.1. URI Identification and Mapping

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

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

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

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

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

#### 4.2.2. Request Handling

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

Multicast request handling consists of the following steps:

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

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

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

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

#### 4.2.3. Examples

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

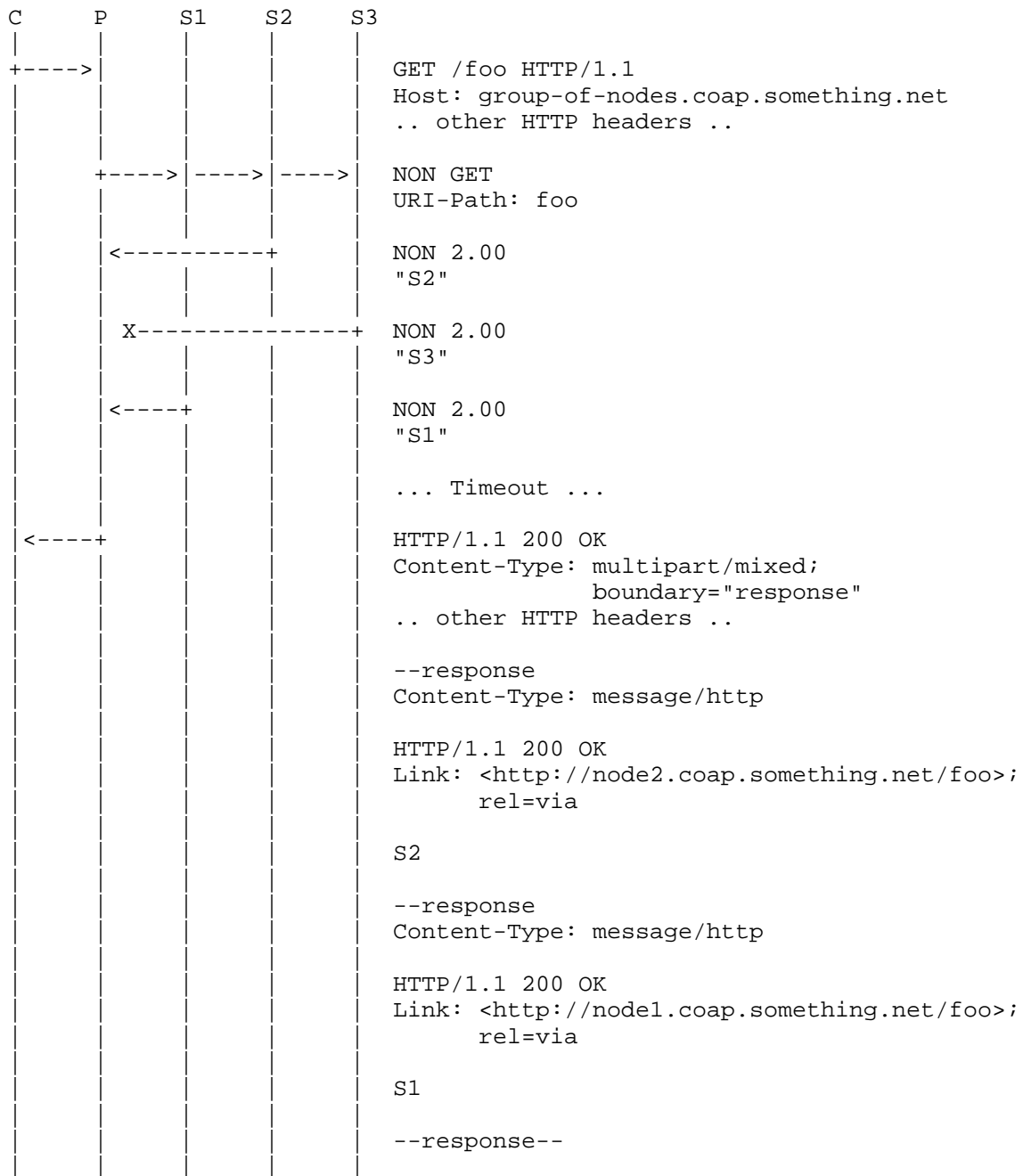


Figure 2: Unicast HTTP to Multicast CoAP Mapping

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

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

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

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

#### 4.3. Multicast Response Caching

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

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

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



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

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

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

#### 4.4. Observe Mapping

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

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

##### 4.4.1. Identification

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

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

##### 4.4.1.1. Observable URI Mapping

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

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

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

#### 4.4.1.1.1. Discovery

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

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

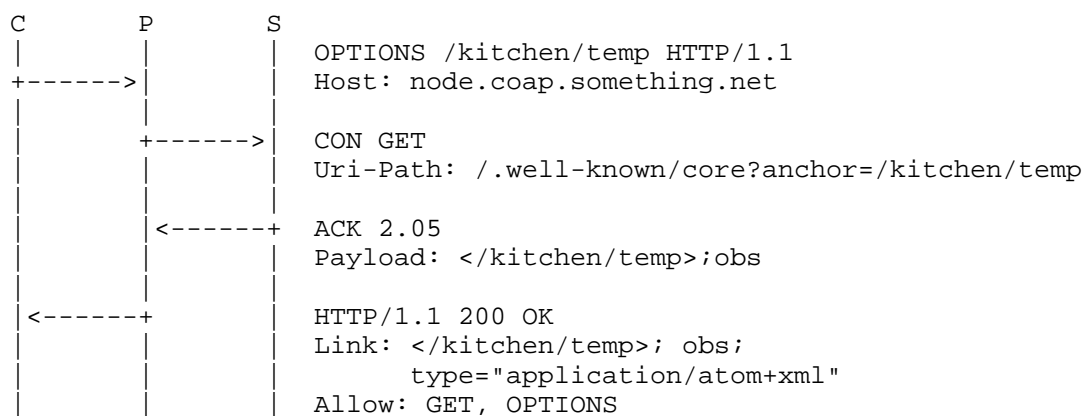


Figure 3: Discover Observability with HTTP OPTIONS

#### 4.4.1.2. Differentiation Using HTTP Header

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

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

#### 4.4.1.2.1. Expect Header

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

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

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

#### 4.4.1.2.2. Prefer Header

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

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

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

#### 4.4.2. Notification(s) Mapping

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

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

#### 4.4.2.1. Multipart Messaging

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

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

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

#### 4.4.2.2. Using ATOM Feeds

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

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

#### 4.4.3. Examples

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

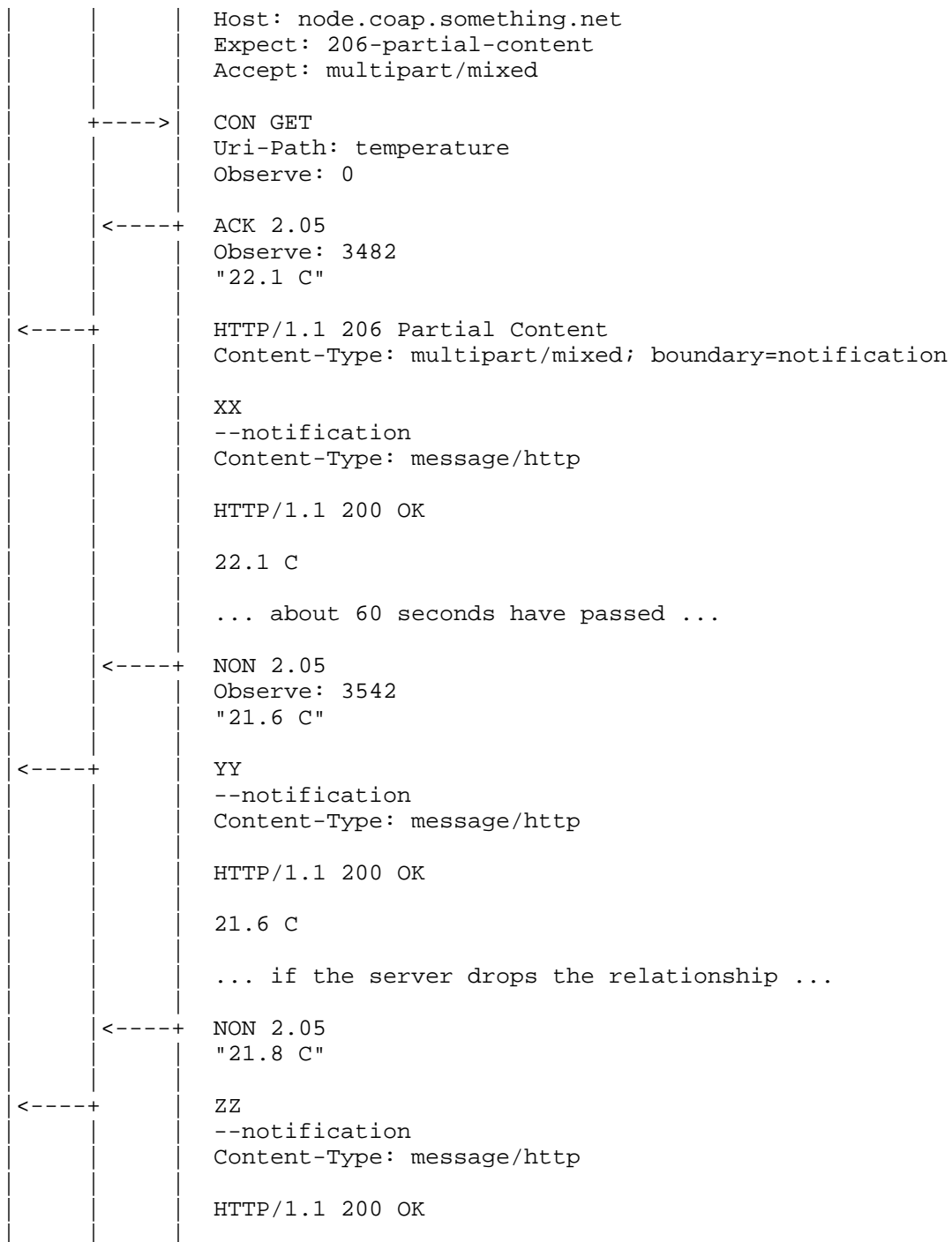
C manifests its intention to observe T by including the Expect Header in the request; if P or S do not support this interaction, the request MUST fail with "417 Expectation Failed" return code. In the presented example, both P and C support this interaction, and the subscription is successful, as stated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

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

```

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

```



			21.8 C
			--notification--
			0

Figure 4: HTTP Streaming to CoAP Observe

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

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

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

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

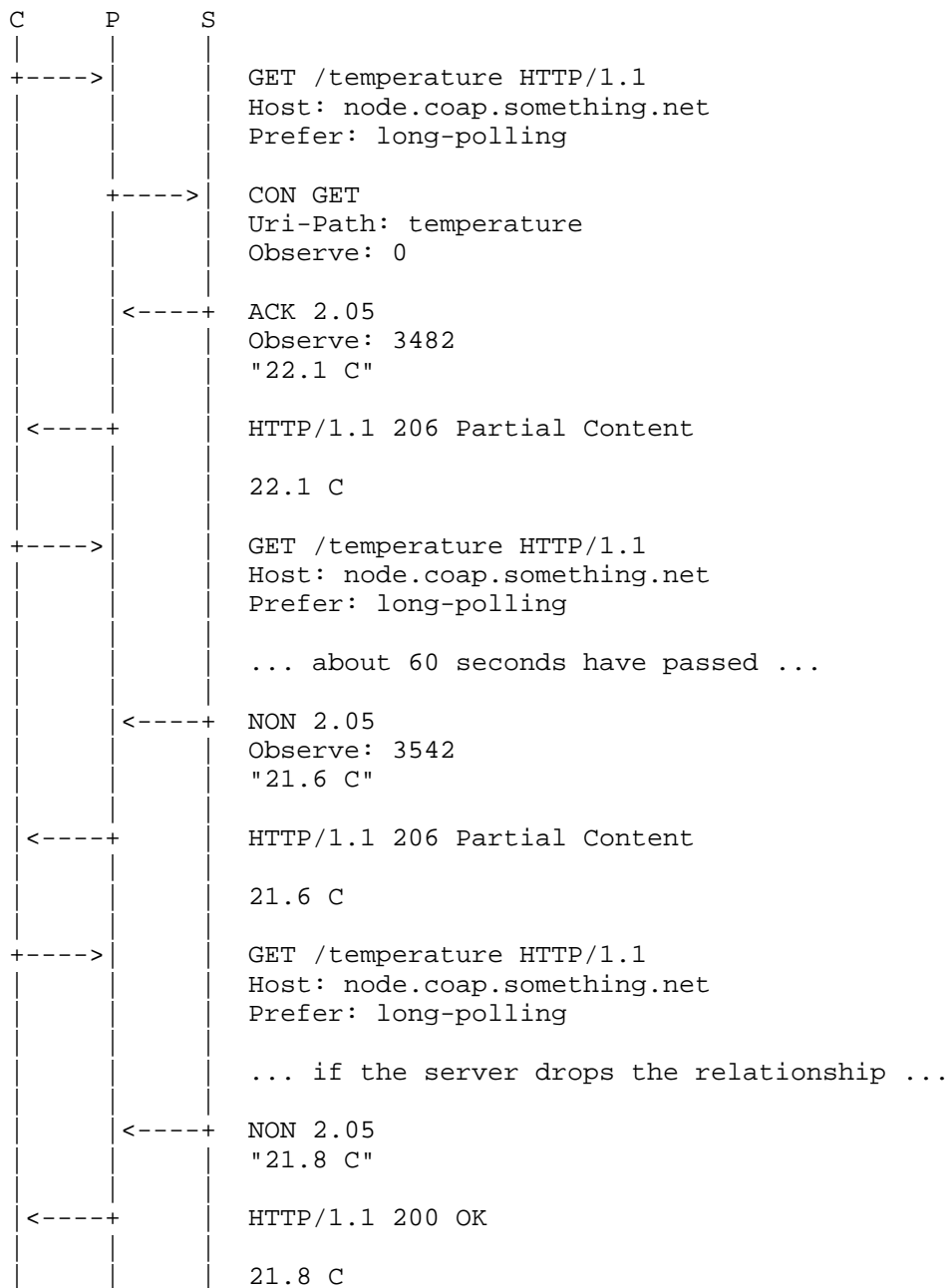


Figure 5: HTTP Long Polling to CoAP Observe

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

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



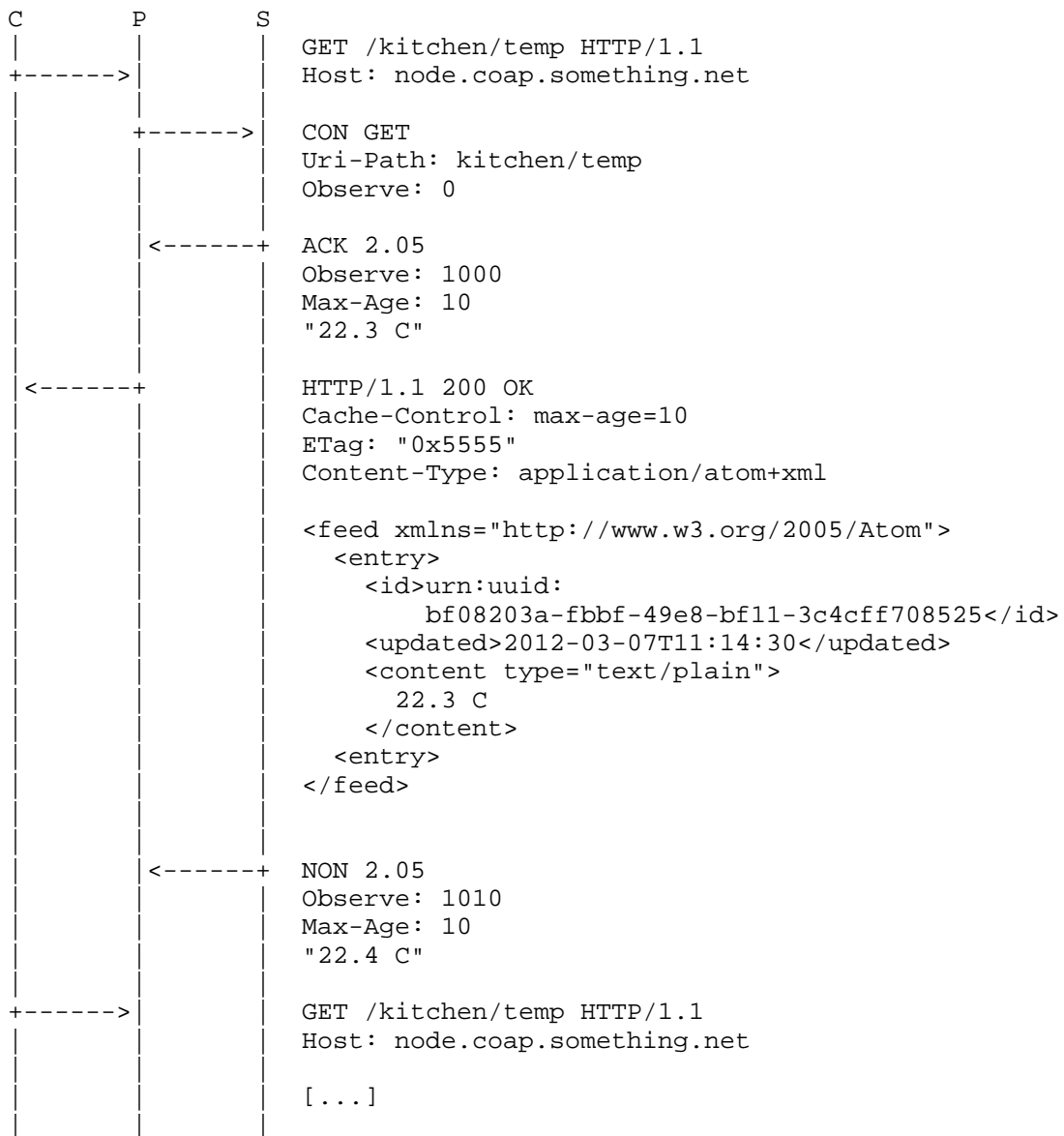


Figure 6: Observation via Atom feeds

## 5. HTML5 Scheme Handler Registration

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

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

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

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

## 6. Placement and Deployment

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

The arguments supporting CS placement are as follows:

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

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

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

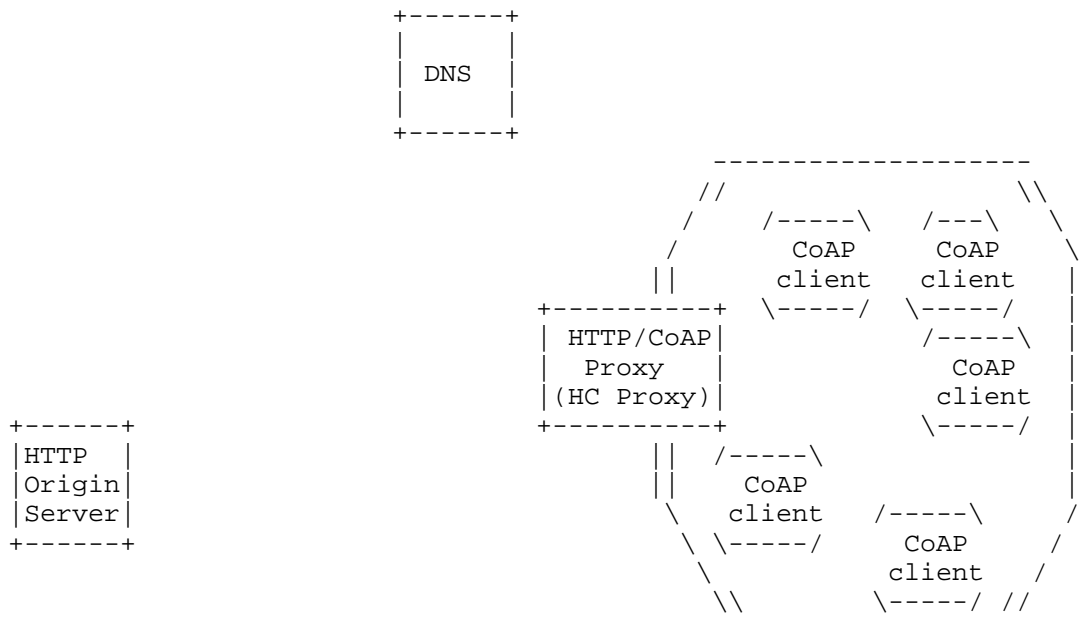


Figure 7: Client-side HC Proxy Deployment Scenario

## 7. Examples

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

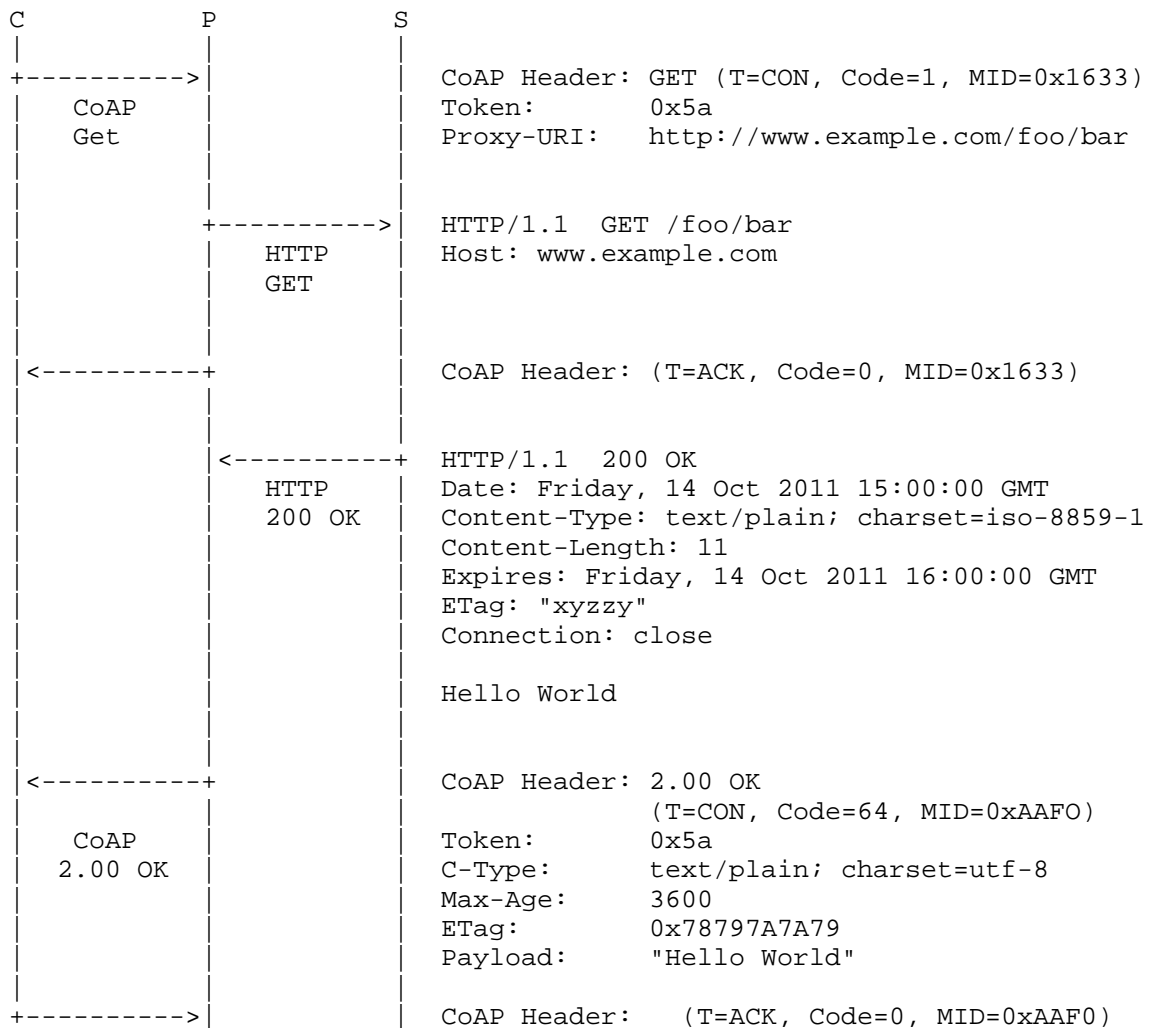


Figure 8: A Basic CoAP-HTTP GET Request

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

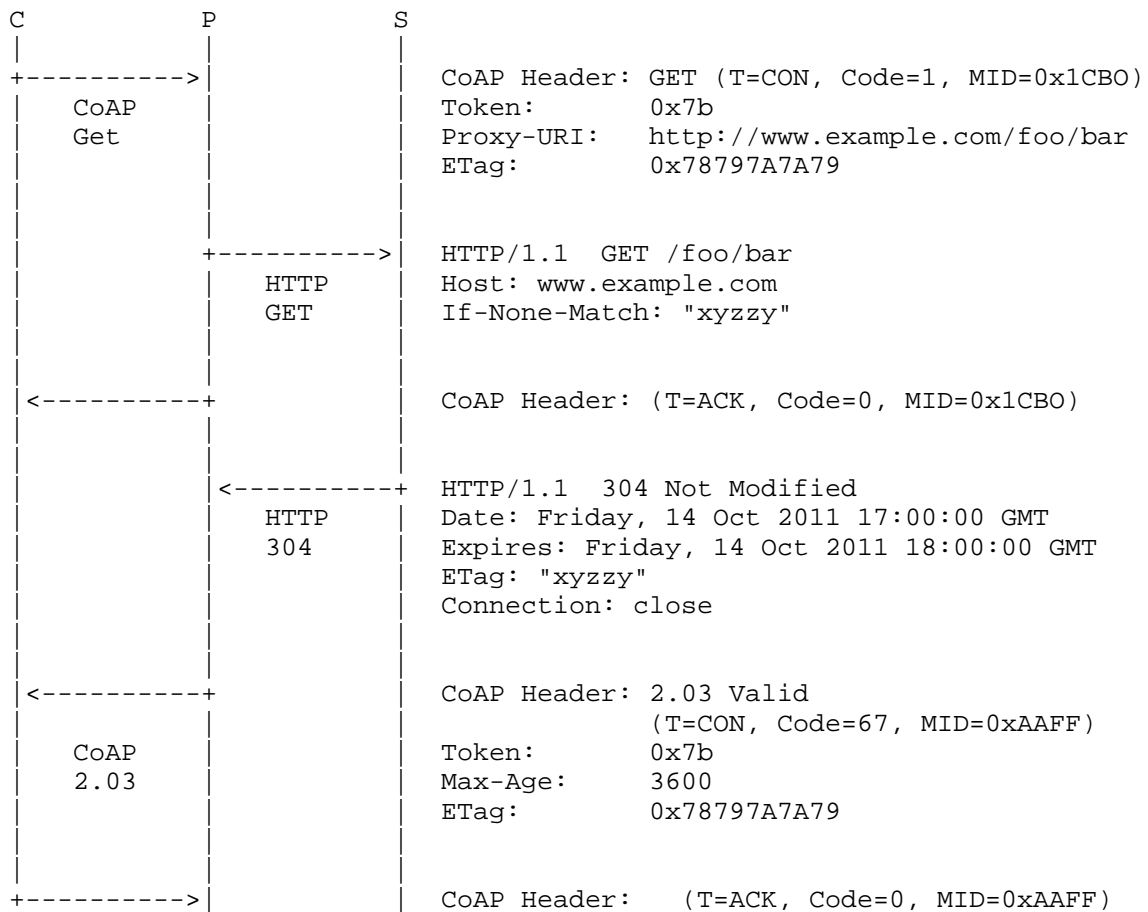


Figure 9: A CoAP-HTTP GET Request with an ETag Option

## 8. Acknowledgements

TBD.

## 9. IANA Considerations

This memo includes no request to IANA.

## 10. Security Considerations

### 10.1. Cross-protocol Security Policy Mapping

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

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

### 10.2. Subscription

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

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

## 11. References

### 11.1. Normative References

[I-D.castellani-core-http-mapping]

Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-http-mapping-06 (work in progress), October 2012.

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-10 (work in progress), October 2012.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.

[I-D.ietf-core-groupcomm]

Rahman, A. and E. Dijk, "Group Communication for CoAP",  
draft-ietf-core-groupcomm-04 (work in progress),  
December 2012.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-07 (work in progress),  
October 2012.

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

Fielding, R. and J. Reschke, "Hypertext Transfer Protocol  
(HTTP/1.1): Message Syntax and Routing",  
draft-ietf-httpbis-pl-messaging-21 (work in progress),  
October 2012.

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

Fielding, R. and J. Reschke, "Hypertext Transfer Protocol  
(HTTP/1.1): Semantics and Content",  
draft-ietf-httpbis-p2-semantic-21 (work in progress),  
October 2012.

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

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

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

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

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

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

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

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

## 11.2. Informative References

- [I-D.bormann-core-simple-server-discovery]  
Bormann, C., "CoRE Simple Server Discovery",  
draft-bormann-core-simple-server-discovery-01 (work in  
progress), March 2012.
- [I-D.shelby-core-resource-directory]  
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource  
Directory", draft-shelby-core-resource-directory-04 (work  
in progress), July 2012.
- [I-D.snell-http-prefer]  
Snell, J., "Prefer Header for HTTP",  
draft-snell-http-prefer-17 (work in progress),  
November 2012.
- [I-D.vanderstok-core-bc]  
Stok, P. and K. Lynn, "CoAP Utilization for Building  
Control", draft-vanderstok-core-bc-05 (work in progress),  
October 2011.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web  
Replication and Caching Taxonomy", RFC 3040, January 2001.
- [RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-  
Service Considerations", RFC 4732, December 2006.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins,  
"Known Issues and Best Practices for the Use of Long  
Polling and Streaming in Bidirectional HTTP", RFC 6202,  
April 2011.
- [W3C.HTML5]  
Hickson, I., "HTML5", World Wide Web Consortium WD (work  
in progress) WD-html5-20111018, October 2011,  
<<http://dev.w3.org/html5/spec/>>.

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

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

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



independent mapping source.

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

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

#### A.1. URL Map Algorithm

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

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

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

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

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

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

INPUT  
\* requested URL

OUTPUT  
\* target URL

SYNTAX  
url\_map [rule name] {  
 requested\_url <regex>  
 mapped\_url <regex match subst template>  
}

EXAMPLE 1  
url\_map homogeneous {  
 requested\_url '^http://.\*\$'  
 mapped\_url 'coap//\$1'  
}

EXAMPLE 2  
url\_map embedded {  
 requested\_url '^http://example.com/coap/.\*\$'  
 mapped\_url 'coap//\$1'  
}

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

#### A.2. Security Policy Map Algorithm

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

## INPUT

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

## OUTPUT

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

## SYNTAX

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

## EXAMPLE

```
<TBD>
```

## A.3. Content-Type Map Algorithm

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

## INPUT

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

## OUTPUT

- \* mapped content-type

## SYNTAX

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

## EXAMPLE

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

Authors' Addresses

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

Email: [angelo@castellani.net](mailto:angelo@castellani.net)

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

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

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

Phone: +1 514 585 0761  
Email: [Akbar.Rahman@InterDigital.com](mailto:Akbar.Rahman@InterDigital.com)

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

Phone: +39 051 644 82 68  
Email: [tho@koanlogic.com](mailto:tho@koanlogic.com)

Esko Dijk  
Philips Research

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



CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: August 29, 2013

A. Castellani  
University of Padova  
S. Loreto  
Ericsson  
A. Rahman  
InterDigital Communications, LLC  
T. Fossati  
KoanLogic  
E. Dijk  
Philips Research  
February 25, 2013

Best Practices for HTTP-CoAP Mapping Implementation  
draft-castellani-core-http-mapping-07

Abstract

This draft provides reference information for HTTP-CoAP protocol translation proxy implementors, focusing primarily on the reverse proxy case. It details deployment options, discusses possible approaches 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 August 29, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	3
3. Cross-Protocol Usage of URIs . . . . .	4
4. HTTP to CoAP URI Mapping . . . . .	5
4.1. Embedded Mapping . . . . .	5
4.2. Homogeneous Mapping . . . . .	5
4.3. Scheme Security Mapping . . . . .	6
5. HTTP-CoAP Reverse Proxy . . . . .	6
5.1. Proxy Placement . . . . .	7
5.2. Response Code Translations . . . . .	8
5.3. Media Type Translations . . . . .	10
5.4. Caching and Congestion Control . . . . .	11
5.5. Cache Refresh via Observe . . . . .	11
5.6. Use of CoAP Blockwise Transfer . . . . .	12
5.7. Security Translation . . . . .	13
5.8. Other guidelines . . . . .	13
6. IANA Considerations . . . . .	14
7. Security Considerations . . . . .	14
7.1. Traffic overflow . . . . .	14
7.2. Handling Secured Exchanges . . . . .	15
8. Acknowledgements . . . . .	15
9. References . . . . .	16
9.1. Normative References . . . . .	16
9.2. Informative References . . . . .	17
Authors' Addresses . . . . .	17

## 1. Introduction

CoAP [I-D.ietf-core-coap] has been designed with the twofold aim to be an application protocol specialized for constrained environments and to be easily used in REST architectures such as the Web. The latter goal has led to define CoAP to easily interoperate with HTTP [RFC2616] through an intermediary proxy which performs cross-protocol conversion.

Section 10 of [I-D.ietf-core-coap] describes the fundamentals of the CoAP-HTTP (and vice-versa) cross-protocol mapping process. However, implementing such a cross-protocol proxy can be complex, and many details regarding its internal procedures and design choices require further elaboration. Therefore a first goal of this document is to provide more detailed information to proxy designers and implementers, to help implement proxies that correctly inter-work with other CoAP and HTTP client/server implementations that adhere to the HTTP and CoAP specifications.

The second goal of this informational document is to define a consistent set of guidelines that a HTTP-to-CoAP proxy implementation MAY adhere to. The main reason of adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability. (As an example use case, a proxy conforming to these guidelines made by vendor A can be easily replaced by a proxy from vendor B that also conforms to the guidelines.)

This draft is organized as follows:

- o Section 2 describes terminology to identify proxy types, mapping approaches and proxy deployments;
- o Section 3 discusses how URIs refer to resources independent of access protocols;
- o Section 5 analyzes the mapping that allows HTTP clients to contact CoAP servers;
- o Section 7 discusses possible security impact related to HTTP/CoAP cross-protocol mapping.

## 2. Terminology

This document assumes readers are familiar with the terms Reverse Proxy as defined in [I-D.ietf-httpbis-pl-messaging] and Interception Proxy as defined in [RFC3040]. In addition, the following terms are defined:



Cross-Protocol Proxy (or Cross Proxy): is a proxy performing a cross-protocol mapping, in the context of this document a HTTP-CoAP (HC) mapping. A Cross-Protocol Proxy can behave as a Forward Proxy, Reverse Proxy or Interception Proxy. Note: In this document we focus on the Reverse Proxy mode of the Cross-Protocol Proxy.

Forward Proxy: a message forwarding agent that is selected by the client, usually via local configuration rules, to receive requests for some type(s) of absolute URI and to attempt to satisfy those requests via translation to the protocol indicated by the absolute URI. The user decides (is willing to) use the proxy as the forwarding/dereferencing agent for a predefined subset of the URI space.

Reverse Proxy: a receiving agent that acts as a layer above some other server(s) and translates the received requests to the underlying server's protocol. It behaves as an origin (HTTP) server on its connection towards the (HTTP) client and as a (CoAP) client on its connection towards the (CoAP) origin server. The (HTTP) client uses the "origin-form" [I-D.ietf-httpbis-pl-messaging] as a request-target URI.

Reverse and Forward proxies are technically very similar, with main differences being that the former appears to a client as an origin server while the latter does not, and that clients may be unaware they are communicating with a proxy.

Placement terms: a server-side (SS) proxy is placed in the same network domain as the server; conversely a client-side (CS) proxy is in the same network domain as the client. In any other case than SS or CS, the proxy is said to be External (E).

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

### 3. Cross-Protocol Usage of URIs

A Uniform Resource Identifier (URI) provides a simple and extensible method for identifying a resource. It enables uniform identification of resources via a separately defined extensible set of naming schemes [RFC3986].

URIs are formed of at least three components: scheme, authority and path. The scheme often corresponds to the protocol used to access the resource. However, as noted in Section 1.2.2 of [RFC3986] the

scheme does not imply that a particular protocol is used to access the resource. So, we can define the same resource to be accessible by different protocols i.e. the resource can have cross-protocol URIs referring to it.

HTTP clients typically only support 'http' and 'https' schemes. Therefore, they cannot directly access CoAP servers (which support 'coap' and/or 'coaps'). In this situation, communication is enabled by a Cross-Protocol Proxy, as shown in Figure 1, supporting URI mapping features. Such features are discussed in the following section.

#### 4. HTTP to CoAP URI Mapping

Assume that a HTTP client wants to access a CoAP resource and indicates a target resource of "http://node.something.net/foobar" to a Forward cross proxy. A possible URI mapping done by the proxy could result in "coap://node.coap.something.net/foo".

As shown in the above example, in a cross-protocol URI the scheme, authority and path parts of the URI may all change. The process of providing cross-protocol URIs may be complex, since a mechanism to statically or dynamically (e.g., discovery) map the URI is needed.

Two simple static URI mapping solutions are proposed in the following subsections. Note that other mapping approaches are possible as well.

##### 4.1. Embedded Mapping

In an embedded mapping approach, the HTTP URI has embedded inside it the authority and path part of the CoAP URI.

Example: The CoAP resource "//node.coap.something.net/foo" can be accessed by an HTTP client by inserting in the request "http://hc-proxy.something.net/coap/node.coap.something.net/foo". The Cross-Protocol Proxy then maps the URI to "coap://node.coap.something.net/foo"

##### 4.2. Homogeneous Mapping

In a homogeneous mapping approach, only the scheme portion of the URI needs to be mapped. The rest of the URI (i.e. authority, path, etc.) remains unchanged.

Example: The CoAP resource "coap://node.coap.something.net/foo" can be accessed by an HTTP client by requesting

"http://node.coap.something.net/foo". The Cross-Protocol Proxy receiving the request is responsible to map the URI to "coap://node.coap.something.net/foo"

Background info: The assumption in this case is that the HTTP client would be able to successfully resolve "node.coap.something.net" using DNS infrastructure to return the IP address of the HC proxy. Most likely this would be through a two step DNS lookup where the first DNS lookup would resolve "something.net" using public DNS infrastructure. Then the second DNS lookup on the subdomain "coap" and the host "node" would typically be resolved by a DNS server operated by the owner of domain "something.net". So this domain owner can manage its own internal node names and subdomain allocation which would correspond to the CoAP namespace

#### 4.3. Scheme Security Mapping

In general, regardless of the URI mapping scheme used in the Cross-Protocol Proxy, an "https" request SHOULD be translated to a "coaps" request. The exception case being cases where security on the CoAP side is not needed because the network is well enough protected already by other means (e.g. strong link-layer security, or the CoAP network runs inside a firewalled network, etc.).

#### 5. HTTP-CoAP Reverse Proxy

A HTTP-CoAP Reverse Cross-Protocol Proxy is accessed by web clients only supporting HTTP, and handles their requests by mapping these to CoAP requests, which are forwarded to CoAP servers; and mapping back the received CoAP responses to HTTP. This mechanism is transparent to the client, which may assume that it is communicating with the intended target HTTP server. In other words, the client accesses the proxy as an origin server using the "origin-form" [I-D.ietf-httpbis-pl-messaging] as a Request Target.

Normative requirements on the translation of HTTP requests to CoAP and of the CoAP responses back to HTTP responses are defined in Section 10.2 of [I-D.ietf-core-coap]. However, that section only considers the case of a HTTP-CoAP Forward Cross-Protocol Proxy in which a client explicitly indicates it targets a request to a CoAP server, and does not cover all aspects of proxy implementation in detail. The present section provides guidelines and more details for the implementation of a Reverse Cross-Protocol Proxy, which MAY be followed in addition to the normative requirements.

Translation of unicast HTTP requests into multicast CoAP requests is currently out of scope since in a reverse proxy scenario a HTTP

client typically expects to receive a single response, not multiple. However a Cross-Protocol Proxy MAY include custom application-specific functions to generate a multicast CoAP request based on a unicast HTTP request and aggregate multiple CoAP responses into a single HTTP response.

Note that the guidelines in this section also apply to an HTTP-CoAP Intercepting Cross-Protocol Proxy.

### 5.1. 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 functionalities 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.)

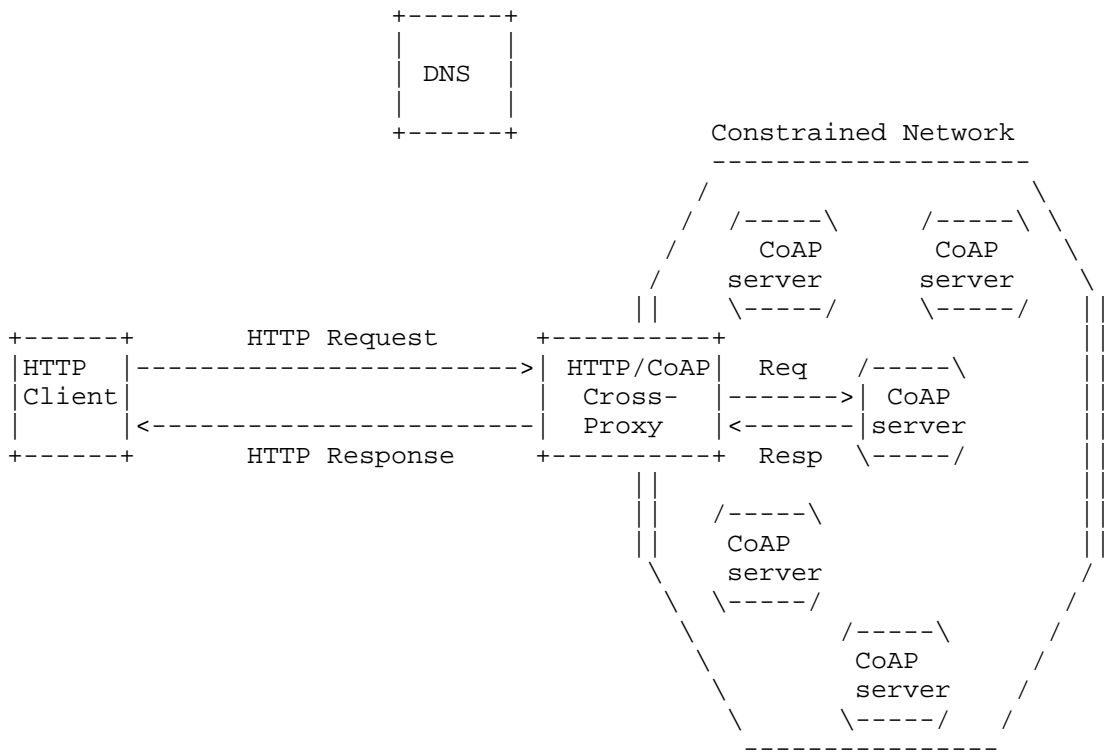


Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

5.2. Response Code Translations

Table 1 defines all possible CoAP responses along with the HTTP response to which each CoAP response SHOULD be translated. This table complies with the Section 10.2 requirements of [I-D.ietf-core-coap] and is intended to cover all possible cases. Multiple appearances of a HTTP status code in the second column indicates multiple equivalent HTTP responses are possible, depending on the conditions cited in the Notes (third column).

CoAP Response Code	HTTP Status Code	Notes
2.01 Created	201 Created	1
2.02 Deleted	200 OK	2
	204 No Content	2
2.03 Valid	304 Not Modified	3
	200 OK	4
2.04 Changed	200 OK	2
	204 No Content	2
2.05 Content	200 OK	
4.00 Bad Request	400 Bad Request	
4.01 Unauthorized	400 Bad Request	5
4.02 Bad Option	400 Bad Request	6
4.03 Forbidden	403 Forbidden	
4.04 Not Found	404 Not Found	
4.05 Method Not Allowed	400 Bad Request	7
4.06 Not Acceptable	406 Not Acceptable	
4.12 Precondition Failed	412 Precondition Failed	
4.13 Request Entity Too Large	413 Request Repr. Too Large	
4.15 Unsupported Media Type	415 Unsupported Media Type	
5.00 Internal Server Error	500 Internal Server Error	
5.01 Not Implemented	501 Not Implemented	
5.02 Bad Gateway	502 Bad Gateway	
5.03 Service Unavailable	503 Service Unavailable	8
5.04 Gateway Timeout	504 Gateway Timeout	
5.05 Proxying Not Supported	502 Bad Gateway	9

Table 1: HTTP-CoAP Response Mapping

## Notes:

1. A CoAP server may return an arbitrary format payload along with this response. This payload SHOULD be returned as entity in the HTTP 201 response. Section 7.3.2 of [I-D.ietf-httpbis-p2-semantics] does not put any requirement on the format of the payload. (In the past, [RFC2616] did.)
2. The HTTP code is 200 or 204 respectively for the case that a CoAP server returns a payload or not. [I-D.ietf-httpbis-p2-semantics] Section 5.3 requires code 200 in case a representation of the action result is returned for DELETE, POST and PUT and code 204 if not. Hence, a proxy SHOULD transfer any CoAP payload contained in a 2.02 response to the HTTP client in a 200 OK response.

3. A CoAP 2.03 (Valid) response only (1) confirms that the request ETag is valid and (2) provides a new Max-Age value. HTTP 304 (Not Modified) also updates some header fields of a stored response. A non-caching proxy may not have enough information to fill in the required values in the HTTP 304 (Not Modified) response, so it may not be advisable for a non-caching proxy to provoke the 2.03 (Valid) response by forwarding an ETag. A caching proxy will fill the information out of the cache.
4. A 200 response to a CoAP 2.03 occurs only when the proxy is caching and translated a HTTP request (without validation request) to a CoAP request that includes validation, for efficiency. The proxy receiving 2.03 updates the freshness of the cached representation and returns the entire representation to the HTTP client.
5. The HTTP code 401 Unauthorized MUST NOT be used, as long as in CoAP there is no equivalent defined of the required WWW-Authenticate header (Section 3.1 of [I-D.ietf-httpbis-p7-auth]).
6. In some cases a proxy receiving 4.02 may retry the request with less CoAP Options in the hope that the server will understand the newly formulated request. For example, if the proxy tried using a Block Option which was not recognized by the CoAP server it may retry without that Block Option.
7. The HTTP code "405 Method Not Allowed" MUST NOT be used since CoAP does not provide enough information to determine a value for the required "Allow" response-header field.
8. The value of the HTTP "Retry-After" response-header field is taken from the value of the CoAP Max-Age Option, if present.
9. This CoAP response can only happen if the proxy itself is configured to use a CoAP Forward Proxy to execute some, or all, of its CoAP requests.

### 5.3. Media Type Translations

A Cross-Protocol Proxy translates a media type string, carried in a HTTP Content-Type header in a request, to a CoAP Content-Format Option with the equivalent numeric value. The media types supported by CoAP are defined in the CoAP Content-Format Registry. Any HTTP request with a Content-Type for which the proxy does not know an equivalent CoAP Content-Format number, MUST lead to HTTP response 415 (Unsupported Media Type).

Also, a CoAP Content-Format value in a response is translated back to

the equivalent HTTP Content-Type. If a proxy receives a CoAP Content-Format value that it does not recognize (e.g. because the value is IANA-registered after the proxy software was deployed), and is unable to look up the equivalent HTTP Content-Type on the fly, the proxy SHOULD return an HTTP entity (payload) without Content-Type header (complying to Section 3.1.1.5 of [I-D.ietf-httpbis-p2-semantics]).

#### 5.4. Caching and Congestion Control

A Cross-Protocol Proxy SHOULD limit the number of requests to CoAP servers by responding, where applicable, with a cached representation of the resource.

Duplicate idempotent pending requests by a Cross-Protocol Proxy to the same CoAP resource SHOULD in general be avoided, by duplexing the response to the requesting HTTP clients without duplicating the CoAP request.

If the HTTP client times out and drops the HTTP session to the Cross-Protocol Proxy (closing the TCP connection) after the HTTP request was made, a Cross-Protocol Proxy SHOULD wait for the associated CoAP response and cache it if possible. Further requests to the Cross-Protocol Proxy for the same resource can use the result present in cache, or, if a response has still to come, the HTTP requests will wait on the open CoAP session.

According to [I-D.ietf-core-coap], a proxy MUST limit the number of outstanding interactions to a given CoAP server to NSTART. To limit the amount of aggregate traffic to a constrained network, the Cross-Protocol Proxy SHOULD also pose a limit to the number of concurrent CoAP requests pending on the same constrained network; further incoming requests MAY either be queued or dropped (returning 503 Service Unavailable). This limit and the proxy queueing/dropping behavior SHOULD be configurable. In order to efficiently apply this congestion control, the Cross-Protocol Proxy SHOULD be placed.

Resources experiencing a high access rate coupled with high volatility MAY be observed [I-D.ietf-core-observe] by the Cross-Protocol Proxy to keep their cached representation fresh while minimizing the number CoAP messages. See Section 5.5.

#### 5.5. Cache Refresh via Observe

There are cases where using the CoAP observe protocol [I-D.ietf-core-observe] to handle proxy cache refresh is preferable to the validation mechanism based on ETag as defined in [I-D.ietf-core-coap]. Such scenarios include, but are not limited



to, sleepy nodes -- with possibly high variance in requests' distribution -- which would greatly benefit from a server driven cache update mechanism. Ideal candidates would also be crowded or very low throughput networks, where reduction of the total number of exchanged messages is an important requirement.

This subsection aims at providing a practical evaluation method to decide whether the refresh of a cached resource R is more efficiently handled via ETag validation or by establishing an observation on R.

Let  $T_R$  be the mean time between two client requests to resource R, let  $F_R$  be the freshness lifetime of R representation, and let  $M_R$  be the total number of messages exchanged towards resource R. If we assume that the initial cost for establishing the observation is negligible, an observation on R reduces  $M_R$  iff  $T_R < 2 * F_R$  with respect to using ETag validation, that is iff the mean arrival time of requests for resource R is greater than half the refresh rate of R.

When using observations  $M_R$  is always upper bounded by  $2 * F_R$ : in the constrained network no more than  $2 * F_R$  messages will be generated towards resource R.

#### 5.6. Use of CoAP Blockwise Transfer

A Cross-Protocol Proxy SHOULD support CoAP blockwise transfers [I-D.ietf-core-block] to allow transport of large CoAP payloads while avoiding excessive link-layer fragmentation in LLNs, and to cope with small datagram buffers in CoAP end-points as described in [I-D.ietf-core-coap] Section 4.6.

A Cross-Protocol Proxy SHOULD attempt to retry a payload-carrying CoAP PUT or POST request with blockwise transfer if the destination CoAP server responded with 4.13 (Request Entity Too Large) to the original request. A Cross-Protocol Proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than a value `BLOCKWISE_THRESHOLD`. The value of `BLOCKWISE_THRESHOLD` MAY be implementation-specific, for example calculated based on a known or typical UDP datagram buffer size for CoAP end-points, or set to N times the size of a link-layer frame where e.g.  $N=5$ , or preset to a known IP MTU value, or set to a known Path MTU value. The value `BLOCKWISE_THRESHOLD` or parameters from which it is calculated SHOULD be configurable in a proxy implementation.

The Cross-Protocol Proxy SHOULD detect CoAP end-points not supporting blockwise transfers by checking for a 4.02 (Bad Option) response returned by an end-point in response to a CoAP request with a Block\*

Option. This allows the Cross-Protocol Proxy to be more efficient, not attempting repeated blockwise transfers to CoAP servers that do not support it. However if a request payload is too large to be sent as a single CoAP request and blockwise transfer would be unavoidable, the proxy still SHOULD attempt blockwise transfer on such an endpoint before returning 413 (Request Entity Too Large) to the HTTP client.

For improved latency a cross proxy MAY initiate a blockwise CoAP request triggered by an incoming HTTP request even when the HTTP request message has not yet been fully received, but enough data has been received to send one or more data blocks to a CoAP server already. This is particularly useful on slow client-to-proxy connections.

#### 5.7. 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.

#### 5.8. Other guidelines

For long delays of a CoAP server, the HTTP client or any other proxy in between MAY timeout. Further discussion of timeouts in HTTP is available in Section 6.2.4 of [I-D.ietf-httpbis-pl-messaging].

A cross proxy MUST define an internal timeout for each pending CoAP request, because the CoAP server may silently die before completing the request. The timeout value SHOULD be approximately less than or equal to MAX\_RTT defined in [I-D.ietf-core-coap].

When the DNS protocol is not used between CoAP nodes in a constrained network, defining valid FQDN (i.e., DNS entries) for constrained CoAP servers, where possible, MAY help HTTP clients to access the resources offered by these servers via a HC proxy.

HTTP connection pipelining (section 6.2.2.1 of [I-D.ietf-httpbis-pl-messaging]) MAY be supported by the proxy and is transparent to the CoAP network: the HC cross proxy will sequentially serve the pipelined requests by issuing different CoAP requests.

## 6. IANA Considerations

This memo includes no request to IANA.

## 7. Security Considerations

The security concerns raised in Section 15.7 of [RFC2616] also apply to the cross proxy scenario. In fact, the cross proxy is a trusted (not rarely a transparently trusted) component in the network path.

The trustworthiness assumption on the cross proxy cannot be dropped. Even if we had a blind, bi-directional, end-to-end, tunneling facility like the one provided by the CONNECT method in HTTP, and also assuming the existence of a DTLS-TLS transparent mapping, the two tunneled ends should be speaking the same application protocol, which is not the case. Basically, the protocol translation function is a core duty of the cross proxy that can't be removed, and makes it a necessarily trusted, impossible to bypass, component in the communication path.

A reverse proxy deployed at the boundary of a constrained network is an easy single point of failure for reducing availability. As such, a special care should be taken in designing, developing and operating it, keeping in mind that, in most cases, it could have fewer limitations than the constrained devices it is serving.

The following sub paragraphs categorize and argue about a set of specific security issues related to the translation, caching and forwarding functionality exposed by a cross proxy module.

### 7.1. 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 5.4), because inefficient implementations can be targeted by unconstrained Internet attackers. Bandwidth or complexity involved in such attacks is very low.

An amplification attack to the constrained network may be triggered by a multicast request generated by a single HTTP request mapped to a CoAP multicast resource, as considered in Section TBD of [I-D.ietf-core-coap].

The impact of this amplification technique is higher than an amplification attack carried out by a malicious constrained device (e.g. ICMPv6 flooding, like Packet Too Big, or Parameter Problem on a multicast destination [RFC4732]), since it does not require direct

access to the constrained network.

The feasibility of this attack, disruptive in terms of CoAP server availability, can be limited by access controlling the exposed HTTP multicast resource, so that only known/authorized users access such URIs.

## 7.2. Handling Secured Exchanges

It is possible that the request from the client to the cross proxy is sent over a secured connection. However, there may or may not exist a secure connection mapping to the other protocol. For example, a secure distribution method for multicast traffic is complex and MAY not be implemented (see [I-D.ietf-core-groupcomm]).

By default, a cross proxy SHOULD reject any secured client request if there is no configured security policy mapping. This recommendation MAY be relaxed in case the destination network is believed to be secured by other, complementary, means. E.g.: assumed that CoAP nodes are isolated behind a firewall (e.g. as the SS cross proxy deployment shown in Figure 1), the cross proxy may be configured to translate the incoming HTTPS request using plain CoAP (i.e. NoSec mode.)

The HC URI mapping MUST NOT map to HTTP (see Section 4) a CoAP resource intended to be accessed only using HTTPS.

A secured connection that is terminated at the cross proxy, i.e. the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The cross proxy SHOULD NOT cache any secured content to avoid any leak of secured information. However in some specific scenario, a security/efficiency trade-off could motivate caching secured information; in that case the caching behavior MAY be tuned to some extent on a per-resource basis.

## 8. Acknowledgements

An initial version of the table found in Section 5.2 has been provided in revision -05 of [I-D.ietf-core-coap]. Special thanks to Peter van der Stok for countless comments and discussions on this document, that contributed to its current structure and text.

Thanks to Carsten Bormann, Zach Shelby, Michele Rossi, Nicola Bui, Michele Zorzi, Klaus Hartke, Cullen Jennings, Kepeng Li, Brian Frank, Peter Saint-Andre, Kerry Lynn, Linyi Tian, Dorothy Gellert, Francesco Corazza for helpful comments and discussions that have shaped the document.

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n. [251557].

## 9. References

### 9.1. Normative References

- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-10 (work in progress), October 2012.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.ietf-core-groupcomm]  
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-05 (work in progress), February 2013.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-07 (work in progress), October 2012.
- [I-D.ietf-httpbis-pl-messaging]  
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", draft-ietf-httpbis-pl-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.ietf-httpbis-p7-auth]  
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", draft-ietf-httpbis-p7-auth-22 (work in progress), February 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

## 9.2. 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.shelby-core-resource-directory]  
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-shelby-core-resource-directory-04 (work in progress), July 2012.
- [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.

## Authors' Addresses

Angelo P. Castellani  
University of Padova  
Via Gradenigo 6/B  
Padova 35131  
Italy  
  
Email: [angelo@castellani.net](mailto:angelo@castellani.net)

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland  
  
Email: [salvatore.loreto@ericsson.com](mailto:salvatore.loreto@ericsson.com)

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

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

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

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

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

Email: esko.dijk@philips.com





CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: June 23, 2013

E. Dijk, Ed.  
Philips Research  
A. Rahman, Ed.  
InterDigital Communications, LLC  
December 20, 2012

Miscellaneous CoAP Group Communication Topics  
draft-dijk-core-groupcomm-misc-03

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on June 23, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

## Table of Contents

1. Introduction . . . . .	3
2. Potential Solutions for Group Communication . . . . .	3
3. Use Cases . . . . .	3
4. Requirements . . . . .	4
4.1. Background . . . . .	4
4.2. General Requirements . . . . .	5
4.3. Security Requirements . . . . .	7
5. Group Communication Solutions . . . . .	8
5.1. IP Multicast Transmission Methods . . . . .	8
5.1.1. Serial unicast . . . . .	8
5.1.2. Unreliable IP Multicast . . . . .	8
5.1.3. Reliable IP Multicast . . . . .	9
5.2. Overlay Multicast . . . . .	10
5.3. CoAP Application Layer Group Management . . . . .	10
6. DNS-SD Based Group Resource Manipulation . . . . .	13
7. Group Discovery and Member Discovery . . . . .	14
7.1. DNS-SD . . . . .	14
7.2. CoRE Resource Directory . . . . .	14
8. Deployment Guidelines . . . . .	15
8.1. Overview . . . . .	15
8.2. Implementation in Target Network Topologies . . . . .	15
8.2.1. Single LLN Topology . . . . .	15
8.2.2. Single LLN with Backbone Topology . . . . .	17
8.2.3. Multiple LLNs with Backbone Topology . . . . .	20
8.2.4. LLN(s) with Multiple 6LBRs . . . . .	20
8.2.5. Conclusions . . . . .	20
8.3. Implementation Considerations . . . . .	20
8.3.1. MLD Implementation on LLNs and MLD alternatives . . . . .	20
8.3.2. 6LBR Implementation . . . . .	21
8.3.3. Backbone IP Multicast Infrastructure . . . . .	22
9. Miscellaneous Topics . . . . .	22
9.1. CoAP Multicast and HTTP Unicast Interworking . . . . .	23
10. Acknowledgements . . . . .	24
11. IANA Considerations . . . . .	24
12. Security Considerations . . . . .	24
13. References . . . . .	25
13.1. Normative References . . . . .	25
13.2. Informative References . . . . .	26
Appendix A. Multicast Listener Discovery (MLD) . . . . .	28
Appendix B. CoAP-Observe Alternative to Group Communication . . . . .	28
Authors' Addresses . . . . .	29

## 1. Introduction

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

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

## 2. Potential Solutions for Group Communication

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

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

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

## 3. Use Cases

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

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

[RFC6690].

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

#### 4. Requirements

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

##### 4.1. Background

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

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

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

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

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

#### 4.2. General Requirements

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

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

- GEN-REQ 6:      Security: see Section 4.3 for security requirements for group communication.
- GEN-REQ 7:      Flexibility: support for one or many source(s), both dense and sparse networks, for high or low listener density, small or large number of groups, and multi-group membership.
- GEN-REQ 8:      Robust group management: functionality to join groups, leave groups, view group membership, and persistent group membership in failure or sleeping node situations.
- GEN-REQ 9:      Network layer independence: a solution is independent from specific unicast and/or IP multicast routing protocols.
- GEN-REQ 10:     Minimal specification overhead: a group communication solution should preferably re-use existing/established (IETF) protocols that are suitable for LLN deployments, instead of defining new protocols from scratch.
- GEN-REQ 11:     Minimal implementation overhead: e.g. a solution allows to re-use existing (software) components that are already present on constrained nodes such as (typical) 6LoWPAN/CoAP nodes.
- GEN-REQ 12:     Mixed backbone/LLN topology support: a solution should work within a single LLN, and in combined LLN/backbone network topologies, including multi-LLN topologies. Both the senders and receivers of CoAP group messages may be attached to different network links or be part of different LLNs, possibly with routers or switches in between group members. In addition, different routing protocols may operate on the LLN and backbone networks. Preferably a solution also works with existing, common backbone IP infrastructure (e.g. switches or routers).
- GEN-REQ 13:     CoAP Proxying support: a CoAP proxy can handle distribution of a message to a group on behalf of a (constrained) CoAP client.
- GEN-REQ 14:     Suitable for operation on LLNs with constrained nodes.

#### 4.3. Security Requirements

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

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

- SEC-REQ 1:    Group communications data encryption: Important CoAP group communications shall be encrypted (using a group key) to preserve confidentiality. It shall also be possible to send CoAP group communications in the clear (i.e. unencrypted) for low value data.
- SEC-REQ 2:    Group communications source data authentication: Important CoAP group communications shall be authenticated by verifying the source of the data (i.e. that it was generated by a given and trusted group member). It shall also be possible to send unauthenticated CoAP group communications for low value data.
- SEC-REQ 3:    Group communications limited data authentication: Less important CoAP group communications shall be authenticated by simply verifying that it originated from one of the group members (i.e. without explicitly identifying the source node). This is a weaker requirement (but simpler to implement) than REQ2. It shall also be possible to send unauthenticated CoAP group communications for low value data.
- SEC-REQ 4:    Group key management: There shall be a secure mechanism to manage the cryptographic keys (e.g. generation and distribution) belonging to the group; the state (e.g. current membership) associated with the keys; and other security parameters.
- SEC-REQ 5:    Use of Multicast IPSec: The CoAP protocol [I-D.ietf-core-coap] allows IPSec to be used as one option to secure CoAP. If IPSec is used as a way to security CoAP communications, then multicast IPSec [RFC5374] should be used for securing CoAP group

communications.

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

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

## 5. Group Communication Solutions

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

### 5.1. IP Multicast Transmission Methods

#### 5.1.1. Serial unicast

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

#### 5.1.2. Unreliable IP Multicast

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



### 5.1.3. Reliable IP Multicast

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

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

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

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

There are various approaches to achieve reliability, such as

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

## 5.2. Overlay Multicast

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

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

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

## 5.3. CoAP Application Layer Group Management

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

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

The following group management primitives are in general useful:

- o discover groups;
- o query group properties (e.g. related resource descriptions);
- o create a group;
- o remove a group;

- o add a group member;
- o remove a group member;
- o enumerate group members;
- o security and access control primitives.

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

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

[ TBD: how can a client send a request to a group? Does it only need to know the group name (string or URI) or also an IP multicast address? One way is to send a CoAP request to the CoAP Proxy with a group URI directly in the Proxy-URI field. This avoids having to know anything related to IP multicast addresses. ]

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

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

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

Figure 1: CoAP Header Options for Group Management

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

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
Ver  T   OC										Code										Message ID																			
delta   length										Join Group A (ID or URI)																													
0   length										Join Group B (ID or URI)																													
2   length										Leave Group C (ID or URI)																													

Figure 2: CoAP Message for Group Management

Header Fields for the above example:

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

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

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

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

group management can be piggy-backed in either Request or Response message.

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

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

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

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

## 6. DNS-SD Based Group Resource Manipulation

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

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

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

## 7. Group Discovery and Member Discovery

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

### 7.1. DNS-SD

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

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

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

### 7.2. CoRE Resource Directory

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

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

## 8. Deployment Guidelines

### 8.1. Overview

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

### 8.2. Implementation in Target Network Topologies

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

Starting from the simplest case of a single LLN topology, we move to more complex topologies involving a backbone network or multiple LLNs. With "backbone" we refer here typically to a corporate LAN or VLAN, which constitutes a single broadcast domain by design. It could also be an in-home network. A multi-link backbone is also possible, if there is proper IP multicast routing or forwarding configured between these links. (The term 6LoWPAN Border Router or "6LBR" is used here for a border router, though our evaluation is not necessarily restricted to 6LoWPAN networks.)

#### 8.2.1. Single LLN Topology

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

#### 8.2.1.1. Mesh-Under Multicast Routing

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

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

#### 8.2.1.2. RPL Multicast Routing

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

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

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

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

#### 8.2.1.3. RPL Routers with Non-RPL Hosts

Now we consider the case that hosts exist in a RPL network that are not RPL-aware themselves, but rely on RPL routers for their IP connectivity beyond link-local scope. Note that the current RPL specification [RFC6550] leaves this case for future specification (see Section 16.4). Non-RPL hosts cannot advertise their IP



multicast groups of interest via RPL DAO messages as defined above. Therefore in that case MLD could be used for such advertisements (State Change Report messages), with all or a subset of RPL routers acting in the role of MLD Routers as defined in [RFC3810]. However, as the MLD protocol is not designed specifically for LLNs it may be a burden for the constrained RPL router nodes to run the full MLD protocol. Alternatives are therefore proposed in Section 8.3.1.

#### 8.2.1.4. Trickle Multicast Forwarding

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

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

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

#### 8.2.1.5. Other Route-Over Methods

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

#### 8.2.2. Single LLN with Backbone Topology

A LLN may be connected via a Border Router (e.g. 6LBR) to a backbone network, on which IP multicast listeners and/or sources may be

present. This section analyzes cases in which IP multicast traffic needs to flow from/to the backbone, to/from the LLN.

#### 8.2.2.1. Mesh-Under Multicast Routing

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

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

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

#### 8.2.2.2. RPL Multicast Routing

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

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

previous section.

#### 8.2.2.3. RPL Routers with Non-RPL Hosts

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

#### 8.2.2.4. Trickle Multicast Forwarding

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

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

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

#### 8.2.2.5. Other Route-Over Methods

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

### 8.2.3. Multiple LLNs with Backbone Topology

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

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

### 8.2.4. LLN(s) with Multiple 6LBRs

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

### 8.2.5. Conclusions

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

## 8.3. Implementation Considerations

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

### 8.3.1. MLD Implementation on LLNs and MLD alternatives

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

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

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

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

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

[ TBD: consider what aspects are needed/not needed for CoAP/LLN applications. Will MLDv1 suffice? What to do with options like 'source specific' and include/exclude. Source-specific can also be dealt with at the destination host by filtering? Do we need limits on number of records per packet? Do we need a higher MLD reliability setting - see the parameters in the MLD RFC ]

#### 8.3.2. 6LBR Implementation

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

### 8.3.3. Backbone IP Multicast Infrastructure

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

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

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

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

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

## 9. Miscellaneous Topics

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

## 9.1. CoAP Multicast and HTTP Unicast Interworking

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

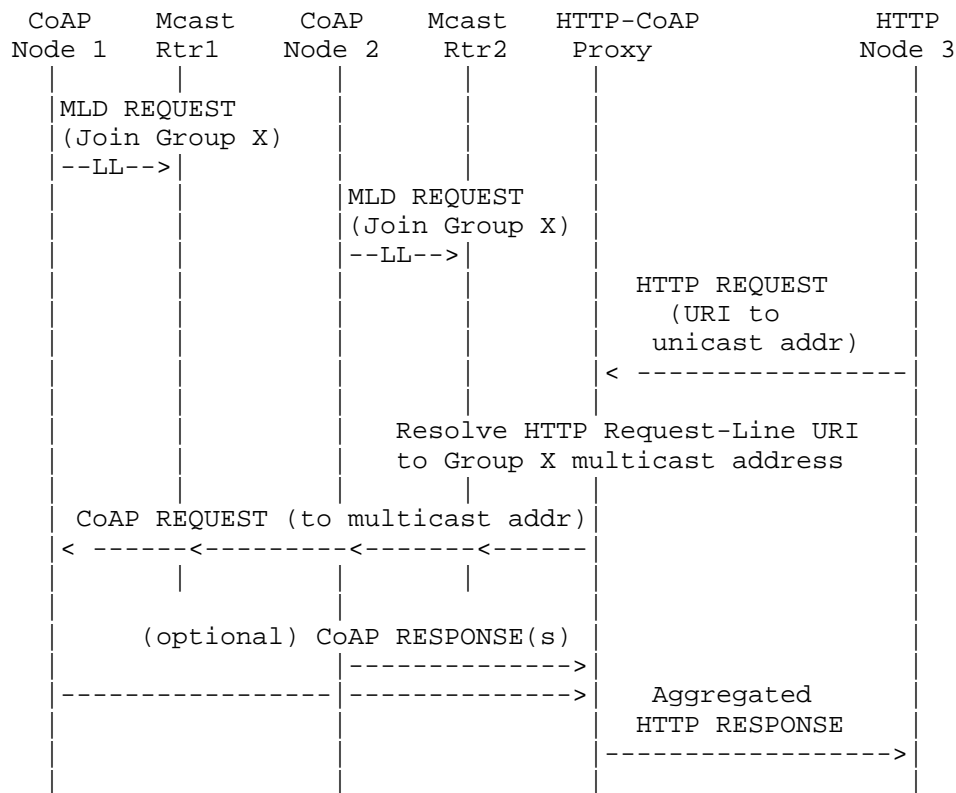


Figure 3: CoAP Multicast and HTTP Unicast Interworking

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

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

In terms of the HTTP Response, Figure 3 illustrates that it will be generated by the proxy node based on aggregated responses of the CoAP nodes and sent back to the client in the general Internet that sent the HTTP Request (i.e. node 1). In [I-D.castellani-core-advanced-http-mapping] the HTTP Response that the Proxy may use to aggregate multiple CoAP responses is described in more detail. So in terms of overall operation, the CoAP proxy can be considered to be a "non-transparent" proxy according to [RFC2616]. Specifically, [RFC2616] states that a "non-transparent proxy is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction or anonymity filtering."

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

## 10. Acknowledgements

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

## 11. IANA Considerations

This memo includes no request to IANA.

## 12. Security Considerations

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



### 13. References

#### 13.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-13 (work in progress), December 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, March 2004.
- [RFC3810] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.
- [RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, April 2005.
- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.
- [RFC4605] Fenner, B., He, H., Haberman, B., and H. Sandick, "Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying")", RFC 4605, August 2006.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5374] Weis, B., Gross, G., and D. Ignjatic, "Multicast Extensions to the Security Architecture for the Internet Protocol", RFC 5374, November 2008.
- [RFC5867] Martocci, J., De Mil, P., Riou, N., and W. Vermeylen, "Building Automation Routing Requirements in Low-Power and Lossy Networks", RFC 5867, June 2010.
- [RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R.,

Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, March 2012.

- [RFC6636] Asaeda, H., Liu, H., and Q. Wu, "Tuning the Behavior of the Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) for Routers in Mobile and Wireless Networks", RFC 6636, May 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.

### 13.2. Informative References

- [Banerjee01] Banerjee, B. and B. Bhattacharjee, "A Comparative Study of Application Layer Multicast Protocols", 2001, <<http://wmedia.grnet.gr/P2PBackground/a-comparative-study-ofALM.pdf>>.
- [I-D.castellani-core-advanced-http-mapping] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-advanced-http-mapping-00 (work in progress), July 2012.
- [I-D.cheshire-dnsext-dns-sd] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-11 (work in progress), December 2011.
- [I-D.eggert-core-congestion-control] Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.
- [I-D.ietf-6lowpan-routing-requirements] Kim, E., Kaspar, D., Gomez, C., and C. Bormann, "Problem Statement and Requirements for 6LoWPAN Routing", draft-ietf-6lowpan-routing-requirements-10 (work in progress), November 2011.

- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",  
draft-ietf-core-block-10 (work in progress), October 2012.
- [I-D.ietf-core-groupcomm]  
Rahman, A. and E. Dijk, "Group Communication for CoAP",  
draft-ietf-core-groupcomm-03 (work in progress),  
October 2012.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-07 (work in progress),  
October 2012.
- [I-D.ietf-roll-trickle-mcast]  
Hui, J. and R. Kelsey, "Multicast Protocol for Low power  
and Lossy Networks (MPL)",  
draft-ietf-roll-trickle-mcast-02 (work in progress),  
October 2012.
- [I-D.lynn-core-discovery-mapping]  
Lynn, K. and Z. Shelby, "CoRE Link-Format to DNS-Based  
Service Discovery Mapping",  
draft-lynn-core-discovery-mapping-02 (work in progress),  
October 2012.
- [I-D.rahman-core-groupcomm]  
Rahman, A. and E. Dijk, "Group Communication for CoAP",  
draft-rahman-core-groupcomm-07 (work in progress),  
October 2011.
- [I-D.shelby-core-coap-req]  
Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R.  
Kelsey, "CoAP Requirements and Features",  
draft-shelby-core-coap-req-02 (work in progress),  
October 2010.
- [I-D.shelby-core-resource-directory]  
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource  
Directory", draft-shelby-core-resource-directory-04 (work  
in progress), July 2012.
- [I-D.vanderstok-core-bc]  
Stok, P. and K. Lynn, "CoAP Utilization for Building  
Control", draft-vanderstok-core-bc-05 (work in progress),  
October 2011.
- [I-D.vanderstok-core-dna]

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

- [Lao05]    Lao, L., Cui, J., Gerla, M., and D. Maggiorini, "A Comparative Study of Multicast Protocols: Top, Bottom, or In the Middle?", 2005, <[http://www.cs.ucla.edu/NRL/hpi/AggMC/papers/comparison\\_gi\\_2005.pdf](http://www.cs.ucla.edu/NRL/hpi/AggMC/papers/comparison_gi_2005.pdf)>.

#### Appendix A. Multicast Listener Discovery (MLD)

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

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

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

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

#### Appendix B. CoAP-Observe Alternative to Group Communication

The CoAP Observation extension [I-D.ietf-core-observe] can be used as

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

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

#### Authors' Addresses

Esko Dijk (editor)  
Philips Research

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

Akbar Rahman (editor)  
InterDigital Communications, LLC

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



core  
Internet-Draft  
Intended status: Standards Track  
Expires: March 29, 2013

B. Greevenbosch  
Huawei Technologies  
September 25, 2012

CoAP Minimum Request Interval  
draft-greevenbosch-core-minimum-request-interval-00

Abstract

This document defines an "MinimumRequestInterval" option for CoAP, which can be used to negotiate the minimum time between two subsequent requests within a single client and server pair. It can be used for flow and congestion control, reducing the consumption of server and network resources when needed.

## Note

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

## Status of this Memo

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

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

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

This Internet-Draft will expire on March 29, 2013.

## Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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



## Table of Contents

1. Introduction . . . . .	4
2. Requirements notation . . . . .	5
3. Definitions . . . . .	6
4. Motivation . . . . .	7
5. The "MinimumRequestInterval" option . . . . .	8
6. Legacy behaviour . . . . .	9
7. Example . . . . .	10
8. Security Considerations . . . . .	12
9. IANA Considerations . . . . .	13
10. Acknowledgements . . . . .	14
11. Normative References . . . . .	15
Author's Address . . . . .	16

## 1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a RESTful protocol for constrained nodes and networks.

This document defines a "MinimumRequestInterval" option, which can be used to negotiate the minimum time between two subsequent requests within a single client and server pair.

Negotiating the minimum time between the requests can be used to limit the associated traffic, thereby reducing network congestion. In addition, it allows constrained servers to limit the number of requests they receive within a certain time period, preventing them from becoming overloaded.

The mechanism is especially useful for a block transaction, as defined in [I-D.ietf-core-block]. However it can also be used for other transactions involving multiple requests from the client, for example when the client browses the server's resources.

## 2. 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].

### 3. Definitions

#### transaction

A series of request/response pairs within a unique client and server pair.

#### block transaction

A transaction which consists of the transfer of a single source using the block mechanism.

#### two subsequent requests

Two requests within a single transaction, in which one request follows the other request, without a third request from the transaction in between.

#### request interval

The time between two subsequent requests.

#### request speed

The multiplicative inverse of the request interval.

#### ms

Milliseconds or mibiseconds, depending on the implementation.

#### mibisecond

1/1024 of a second.

#### 4. Motivation

It would be beneficial for the server to control the amount of requests it receives from the client within a certain time period. In this way, the server can achieve better usage of its internal resources, such as memory, processor load and message buffers. Limiting the number of incoming requests increases the reliability in responding to them, and decreases the chance on server overload.

One method to reduce the client's request speed is for the server to delay sending its ACKs. This indeed can slow down the client, especially in case the client only issues a new request after receipt of the ACK of the previous request. However, it has the disadvantage that the server has to keep the transaction open, and needs to use resources for delaying the ACK that could have been used to perform other tasks.

If, however, the server can explicitly signal the client's request speed, then the server does not need to keep track of its own minimum time to respond to each request, and can handle requests as soon as possible. This allows the server to use its resources for other tasks sooner. Since all clients will have a better probability that their requests are handled and that they will receive responses, the overall system's reliability is increased.

## 5. The "MinimumRequestInterval" option

Type	C/E	Name	Format	Length	Default
TBD	E	MinimumRequestInterval	uint	0-2B	0

Table 1: The "MinimumRequestInterval" option

The "MinimumRequestInterval" option is an elective option, which is used to negotiate the minimum time in ms that a client needs to wait between sending two subsequent requests.

In the remainder of this section, it is assumed that both the client and the server support the "MinimumRequestInterval" option.

If the client plans to perform a transaction consisting of multiple requests, it **SHOULD** include the "MinimumRequestInterval" option in the first request of the transaction.

The server **MUST** include the "MinimumRequestInterval" option in a response to a request that contained a "MinimumRequestInterval" option.

If a client receives a response with the "MinimumRequestInterval" option, it **MUST** include the "MinimumRequestInterval" in its subsequent request.

In the request, the option's value `T_C` is the request interval the client is currently using. An exception is the first request in the transaction, in which case the value `T_C` is a proposed request interval.

In a response, the option's value `T_S` indicates the minimum request interval in ms that the server can support at that particular moment. Depending on its workload, the server **MAY** increase or decrease the latest value of `T_C` to form `T_S`.

The client **SHALL** wait at least `T_S` ms between sending two subsequent requests. It **MAY** also send at a slower speed.

The "MinimumRequestInterval" option has a default value 0. A value `T_S=0` indicates the server does not put any restrictions on the transaction speed. Similarly value `T_C=0` in the first request indicates that the client prefers to send the following requests as quickly as possible.

## 6. Legacy behaviour

It is possible that either the client or server does not support the "MinimumRequestInterval" option. If the client does not support the option, then obviously it cannot take the server's preference into account. Similarly if the server does not support the option, it cannot use it to restrict the transaction speed.

In either case, or their combination, the client will choose the transaction speed as it prefers. This corresponds to the case  $T_S=0$ .

To allow the server to distinguish between a client that supports the "MinimumRequestInterval" option but wants to signal  $T_C=0$ , and a client that does not support the "MinimumRequestInterval" option, it is RECOMMENDED for the compliant client to include the option in the requests of a multiple request transaction, even when the client wants to signal  $T_C=0$ .

## 7. Example

Figure 1 contains an example of a block transaction with the "MinimumRequestInterval" option.

In the first request, the client proposes a minimum request interval of  $T_C=150\text{ms}$ . As the server is too busy, it wants to slow down the client and returns a minimum request interval of  $T_S=200\text{ms}$ .

The client uses this request interval for the timing of the next requests, and keeps informing the server of its current request speed. Likewise, in the first several messages the server echos the  $T_C$  in  $T_S$ , signalling that it is comfortable with the current request speed.

After sending three blocks, the server becomes less busy. It therefore increases the allowed request speed by signalling a new  $T_S=150\text{ms}$ . The client uses this speed until the end of the transaction.

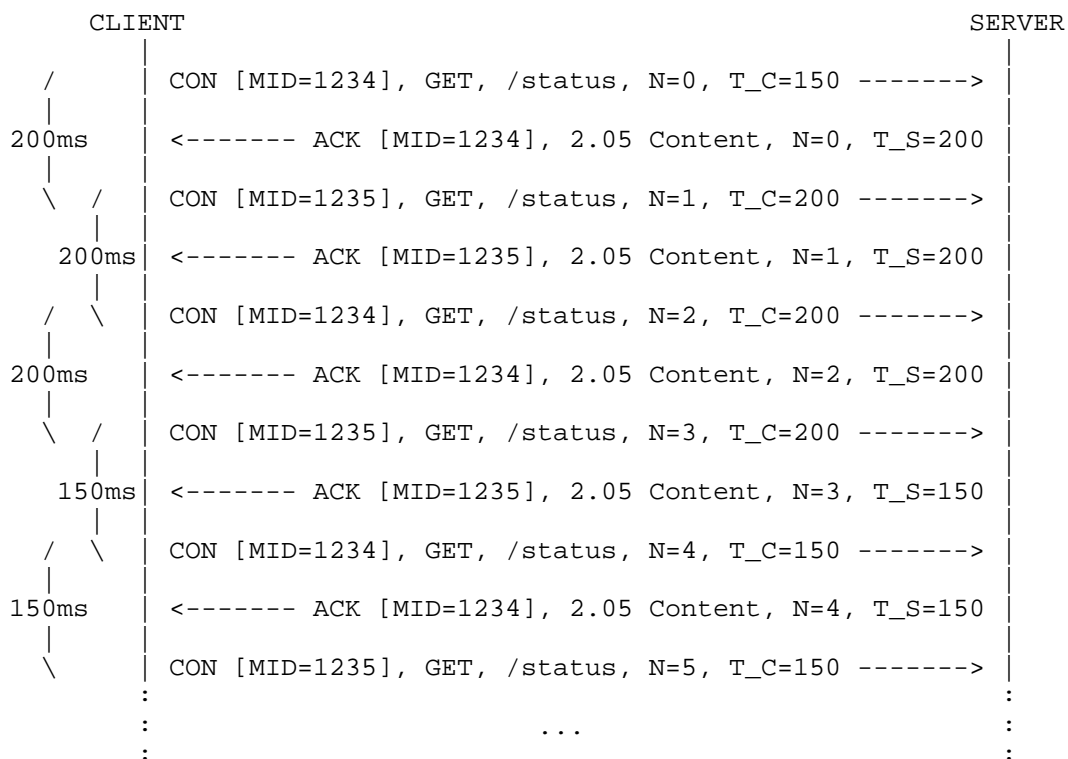




Figure 1: Example of transaction with "MinimumRequestInterval"

## 8. Security Considerations

By modifying the value of the "MinimumRequestInterval" option in a response to a higher value, a man-in-the-middle could increase the time used to perform a transaction. When the client encounters a response with a too high "MinimumRequestInterval" value, it MAY abort the transaction, and try to reinitiate it. However, to prevent overloading the server, the client MUST limit the number of these reinitiations.

By decreasing the value of the "MinimumRequestInterval" option in a response, the man-in-the-middle can induce the client to send requests at a speed too high for the server. The server should be prepared for this, for example by discarding requests that cannot be processed. This is similar to the case where the server or client does not support the "MinimumRequestInterval" option.

By altering the value of the "MinimumRequestInterval" option in a request, the man-in-the-middle can induce the server to believe that the client is using another transaction speed than it really is. This could lead to a false adjustment of the request interval.

All these attacks depend on the man-in-the-middle being able to modify multiple messages, as the speed would otherwise stabilise again after several adjustments by the server.

## 9. IANA Considerations

This draft adds the following option numbers to the CoAP Option Numbers registry of [I-D.ietf-core-coap].

Number	Name	Reference
TBD (elective)	MinimumRequestInterval	[RFCXXXX]

Table 2: CoAP option numbers

## 10. Acknowledgements

The author would like to thank Carsten Borrman, Esko Dijk and Kepeng Li for their feedback.

## 11. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [I-D.ietf-core-block]  
Shelby, Z. and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-block-08 (work in progress), February 2012.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-10 (work in progress), March 2012.

Author's Address

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

Phone: +86-755-28978088

Email: bert.greevenbosch@huawei.com



core  
Internet-Draft  
Intended status: Standards Track  
Expires: April 25, 2013

B. Greevenbosch  
Huawei Technologies  
J. Hoebeke  
I. Ishaq  
iMinds-IBCN/UGent  
October 22, 2012

CoAP Profile Description Format  
draft-greevenbosch-core-profile-description-01

Abstract

This document provides a profile description format, that can be used to express capabilities of a CoAP server.



## Note

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

## Status of this Memo

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

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

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

This Internet-Draft will expire on April 25, 2013.

## Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Requirements notation . . . . .	4
2. Introduction . . . . .	5
3. Obtaining the profile information . . . . .	6
4. The Profile Format . . . . .	7
4.1. The path "path" profile field . . . . .	7
4.2. The options "op" profile field . . . . .	7
4.3. The Content-Formats "cf" profile field . . . . .	7
4.4. The Block-Sizes "bls" and "b2s" profile fields . . . . .	8
5. Usage of URI Queries . . . . .	9
6. Forward compatibility . . . . .	10
7. Examples . . . . .	11
8. Open topics . . . . .	13
8.1. Open since v00 . . . . .	13
8.2. Open since v01 . . . . .	13
9. Change log . . . . .	14
9.1. Changes in v01 . . . . .	14
10. Security Considerations . . . . .	15
11. IANA Considerations . . . . .	16
12. Acknowledgements . . . . .	17
13. Normative References . . . . .	18
Authors' Addresses . . . . .	19

## 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 Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a RESTful protocol for constrained nodes and networks.

Often, a client first learns about a resource through the link format [I-D.ietf-core-link-format]. The link format only provides basic information, for example the resource URL. However, it would be good if the client could get more extensive information on the resources when required. This document defines a profile description format, which can be used to signal several parameters about resources and their servers.

One of the main features of the CoAP protocol is the ability to include CoAP options. These options can be either elective or critical. A message with an unsupported critical option will be rejected, whereas unsupported elective options will be ignored.

Even though it is well defined how the server must respond to unsupported options, it is useful for the client to know which options are supported in advance. In this way, it can determine which options are viable to use in a transaction, and which features cannot be exploited. This specification allows signalling of the supported options by the resource.

Another important feature of a CoAP server is which content formats it supports. CoAP provides a mechanism for the client to indicate to the server which content format the client prefers. The use of profiles allows signalling what content formats are supported by the server, so that the client can decide which content type it prefers.

It is anticipated that more profile descriptions will become available in the future.

### 3. Obtaining the profile information

Similar to the link format from [RFC6690], the profile interface uses a well-known resource URI as a pointer to the profile description.

The host component of the URI of the profile description should be equal to the URI of the associated resource, whereas the path component begins with ".well-known" as specified in [RFC5785]. The complete path component equals ".well-known/profile".

For example, if the client wants to get the profile description of a server with URI "www.example.org", it can send a GET request for "coap://www.example.org/.well-known/profile". In this case the server SHOULD respond with the profile details of all resources on the server. The server MAY use the reserved resource name "." to provide a default profile. This default profile applies to all resources that are not specifically listed in the profile (e.g. because they do not have their individual profile) and describes the general CoAP capabilities of the server. If a resource has its own profile, then this profile MUST be used and the default profile field MUST be ignored.

If only the profile of a particular resource is needed, the client SHOULD send a GET request including the "path" URI-query. If the resource has no specific profile the server MUST respond with the default profile.

For example, the profile of the sensor "coap://www.example.org/s" can be acquired with a GET to:  
"coap://www.example.org/.well-known/profile?path=s".

Section 5 covers this in more detail.

#### 4. The Profile Format

The profile description is formatted as a JSON document. It consists of several profile fields, each of which has associated parameters.

##### 4.1. The path "path" profile field

The "path" profile field contains the Uri-Path component associated with the resource. It can be used to filter on certain profile properties, as described in Section 5.

##### 4.2. The options "op" profile field

The options "op" profile field contains a list of options that are supported by a resource. It has the format depicted in Figure 1, where op1, op2, ... are integers representing the option numbers of the supported options, as defined in [I-D.ietf-core-coap] and/or through IANA. The option numbers MUST appear in numerical order, starting with the lowest number.

```
"op":[op1,op2,...]
```

Figure 1: "op" syntax

When including the "op" profile field in the profile description of a resource, all option numbers of the CoAP options supported by that resource MUST be included. Options that are not supported by the resource MUST NOT be included in the "op" profile field.

If the "op" profile field is available, the receiving party SHALL assume a non-listed option is not supported by the associated resource.

##### 4.3. The Content-Formats "cf" profile field

The content formats "cf" profile field indicates which content formats are supported. It has the format depicted in Figure 2, where cf1, cf2, ... are integers representing the numbers of the supported content formats, as defined in [I-D.ietf-core-coap] and/or through IANA. The content format numbers MUST appear in numerical order, starting with the lowest number.

```
"cf":[cf1,cf2,...]
```

Figure 2: "cf" syntax

When including the "cf" profile field in the profile description of a resource, all content formats of the CoAP options supported by that

resource MUST be included. Content formats that are not supported by the resource MUST NOT be included in the "cf" profile field.

If the "cf" profile field is available, the receiving party SHALL assume a non-listed content format is not supported by the associated resource.

#### 4.4. The Block-Sizes "bls" and "b2s" profile fields

The block sizes "bls" and "b2s" profile fields indicate which block sizes are supported for Block1 and Block2 options when block-wise transfer is used. It has the format depicted in Figure 3, where bls1, bls2, ... are three-bit unsigned integers indicating the size of a block to the power of two. Thus block size =  $2^{(bl + 4)}$ . The allowed values of bl are 0 to 6, i.e., the minimum block size is  $2^{(0+4)} = 16$  and the maximum is  $2^{(6+4)} = 1024$ . The block-size numbers MUST appear in numerical order, starting with the lowest number (see [I-D.ietf-core-block]).

```
"bls":[bls1,bls2,...]  
"b2s":[b2s1,b2s2,...]
```

Figure 3: "bls" and "b2s" syntax

When including the "bls" or the "b2s" profile fields in the profile description of a resource, all respective Block1 and Block2 sizes that are supported in block-wise transfer by that resource MUST be included. Block sizes that are not supported by the resource MUST NOT be included in the "bls" or the "b2s" profile fields.

If the "bls" or the "b2s" profile fields are available, the receiving party SHALL assume a non-listed block size is not supported by the associated resource. If only one of the "bls" and the "b2s" profile fields is available, the receiving party SHALL assume that the other block transfer is not supported by the associated resource.

## 5. Usage of URI Queries

To specify which information is needed, the client MAY include an "Uri-Query" option in its request for the profile description. The server SHOULD understand and process this information, although constraint servers MAY omit the functionality. In the latter case, they SHOULD return the same results as if the "Uri-Query" option was not included.

The URI Queries are of the form "N=V", where N is the name of the field to filter on, and V is the desired value.

For example, if the client wants to know all resources on the server that support content format "application/json", which has the number 50 (see [I-D.ietf-core-coap]), then it will include a "Uri-Query" option with the value "cf=50".

When the request contains multiple "Uri-Query" options, "AND" semantics hold.



## 6. Forward compatibility

To allow addition of new profile fields in the future, unknown profile fields SHOULD be ignored by the client.

## 7. Examples

The following is an example of a camera sensor at "coap://www.example.org/cam", that supports the "Uri-Host" (3), "ETag" (4), "Uri-Port" (7), "Uri-Path" (11), "Content-Format" (12), "Token" (19), "Block2" (23) and "Proxy-Uri" (35) options.

The supported content formats are "text/plain" (0), "application/link-format" (40) and "application/json" (50).

The camera can support Block2 with Block sizes of 256 or 512 bytes.

```
Req: GET coap://www.example.org/.well-known/profile
```

```
Res: 2.05 Content (application/json)
```

```
{
  "profile":
  {
    "path": "cam",
    "op": [3,4,7,11,12,19,23,35],
    "cf": [0,40,50]
    "b2s": [4,5]
  }
}
```

If the server also supports three other resources, such as a temperature sensor (which can do observe), a humidity and a fire detector, the request/response pair would look as follows:

```
Req: GET coap://www.example.org/.well-known/profile
```

```
Res: 2.05 Content (application/json)
```

```
{
  "profile":[
    {
      "path": ".",
      "op": [3,4,7,11,12,19,35],
      "cf": [0]
    },
    {
      "path": "cam",
      "op": [3,4,7,11,12,19,23,35],
      "cf": [0,40,50]
      "b2s": [4,5]
    },
    {
      "path": "temperature",
      "op": [3,4,6,7,11,12,19,35],
      "cf": [0]
    }
  ]
}
```

Please note that the response did not include profiles for the "fire" and "humidity" resources. Instead it included a default profile that applies for these two not explicitly mentioned resources.

If the client now wants to get the resources that support content-format "application/json" (50) it looks as follows:

```
Req: GET coap://www.example.org/.well-known/profile?cf=50
```

```
Res: 2.05 Content (application/json)
```

```
{
  "profile":
  {
    "path": "cam",
    "op": [3,4,7,11,12,19,23,35],
    "cf": [0,40,50]
    "b2s": [4,5]
  }
}
```

## 8. Open topics

### 8.1. Open since v00

- o How to signal the client profile?
- o Which other profile data needs signalling?
- o A natural content format for a camera would be JPEG. Therefore the "image/jpeg" content format may need CoAP registration.
- o Currently, support of observe can be signalled in the link format as well as through the mechanism described here.
- o Is it needed to signal the profile description on a resource basis, or would it be enough to have only one, associated with the server?
- o Fix the order in which the profile fields must appear?
- o Make a distinction between "critical" and "elective" profile fields?

### 8.2. Open since v01

- o Addition of the "path" option seems to create overlap with the link format.
- o For the time being, text about the hierarchy of profiles in servers, batches and resources has been removed. This leads to a requirement to provide the profile description for each separate resource. A mechanism to re-introduce hierarchy may make significantly reduce the profile description verbosity.

## 9. Change log

### 9.1. Changes in v01

- o Changed from /p suffix to usage of ".well-known/profile"
- o Added support of Uri-Query
- o Updated option numbering according to [I-D.ietf-core-coap]
- o Changed Media Type and "mt" to Content Format and "cf", in accordance with [I-D.ietf-core-coap]
- o Expanded examples
- o Removed text about the hierarchy
- o Added default profile "."
- o Added "b1s" and "b2s" fields for block size

## 10. Security Considerations

For general CoAP security considerations see [I-D.ietf-core-coap].

In an unprotected environment, an attacker can change the profile description. For example, the list of supported options may be changed. This could cause the client to make a wrong decision on which mechanisms to use. However, such a threat is normal in environments that lack secure authentication.

## 11. IANA Considerations

- o A registry for profile fields as well as possible values needs to be set up.
- o The ".well-known/profile" path component must be registered.

## 12. Acknowledgements

The authors would like to thank Kepeng Li for his ideas and feedback.



## 13. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",  
draft-ietf-core-block-10 (work in progress), October 2012.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-12 (work in progress), October 2012.
- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format",  
draft-ietf-core-link-format-12 (work in progress),  
May 2012.

## Authors' Addresses

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

Phone: +86-755-28978088  
Email: bert.greevenbosch@huawei.com

Jeroen Hoebeke  
iMinds-IBCN/UGent  
Department of Information Technology  
Internet Based Communication Networks and Services (IBCN)  
Ghent University - iMinds  
Gaston Crommenlaan 8 bus 201  
Ghent B-9050  
Belgium

Phone: +32-9-3314954  
Email: jeroen.hoebeke@intec.ugent.be

Isam Ishaq  
iMinds-IBCN/UGent  
Department of Information Technology  
Internet Based Communication Networks and Services (IBCN)  
Ghent University - iMinds  
Gaston Crommenlaan 8 bus 201  
Ghent B-9050  
Belgium

Phone: +32-9-3314946  
Email: isam.ishaq@intec.ugent.be



CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: April 24, 2013

C. Bormann  
Universitaet Bremen TZI  
Z. Shelby, Ed.  
Sensinode  
October 21, 2012

Blockwise transfers in CoAP  
draft-ietf-core-block-10

Abstract

CoAP is a RESTful transfer protocol for constrained nodes and networks. Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. Occasionally, however, applications will need to transfer larger payloads -- for instance, for firmware updates. With HTTP, TCP does the grunt work of slicing large payloads up into multiple packets and ensuring that they all arrive and are handled in the right order.

CoAP is based on datagram transports such as UDP or DTLS, which limits the maximum size of resource representations that can be transferred without too much fragmentation. Although UDP supports larger payloads through IP fragmentation, it is limited to 64 KiB and, more importantly, doesn't really work well for constrained applications and networks.

Instead of relying on IP fragmentation, this specification extends basic CoAP with a pair of "Block" options, for transferring multiple blocks of information from a resource representation in multiple request-response pairs. In many important cases, the Block options enable a server to be truly stateless: the server can handle each block transfer separately, with no need for a connection setup or other server-side memory of previous block transfers.

In summary, the Block options provide a minimal way to transfer larger representations in a block-wise fashion.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 24, 2013.

#### Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	4
2. Block-wise transfers . . . . .	6
2.1. The Block Options . . . . .	6
2.2. Structure of a Block Option . . . . .	7
2.3. Block Options in Requests and Responses . . . . .	9
2.4. Using the Block Options . . . . .	11
3. Examples . . . . .	14
4. The Size Option . . . . .	20
5. HTTP Mapping Considerations . . . . .	22
6. IANA Considerations . . . . .	24
7. Security Considerations . . . . .	25
7.1. Mitigating Resource Exhaustion Attacks . . . . .	25
7.2. Mitigating Amplification Attacks . . . . .	26
8. Acknowledgements . . . . .	27
9. References . . . . .	28
9.1. Normative References . . . . .	28
9.2. Informative References . . . . .	28
Authors' Addresses . . . . .	29

## 1. Introduction

The CoRE WG is tasked with standardizing an Application Protocol for Constrained Networks/Nodes, CoAP. This protocol is intended to provide RESTful [REST] services not unlike HTTP [RFC2616], while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

This objective requires restraint in a number of sometimes conflicting ways:

- o reducing implementation complexity in order to minimize code size,
- o reducing message sizes in order to minimize the number of fragments needed for each message (in turn to maximize the probability of delivery of the message), the amount of transmission power needed and the loading of the limited-bandwidth channel,
- o reducing requirements on the environment such as stable storage, good sources of randomness or user interaction capabilities.

CoAP is based on datagram transports such as UDP, which limit the maximum size of resource representations that can be transferred without creating unreasonable levels of IP fragmentation. In addition, not all resource representations will fit into a single link layer packet of a constrained network, which may cause adaptation layer fragmentation even if IP layer fragmentation is not required. Using fragmentation (either at the adaptation layer or at the IP layer) to enable the transport of larger representations is possible up to the maximum size of the underlying datagram protocol (such as UDP), but the fragmentation/reassembly process burdens the lower layers with conversation state that is better managed in the application layer.

The present specification defines a pair of CoAP options to enable \_block-wise\_ access to resource representations. The Block options provide a minimal way to transfer larger resource representations in a block-wise fashion. The overriding objective is to avoid the need for creating conversation state at the server for block-wise GET requests. (It is impossible to fully avoid creating conversation state for POST/PUT, if the creation/replacement of resources is to be atomic; where that property is not needed, there is no need to create server conversation state in this case, either.)

In summary, this specification adds a pair of Block options to CoAP that can be used for block-wise transfers. Benefits of using these

options include:

- o Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- o No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.
- o The transfer of each block is acknowledged, enabling retransmission if required.
- o Both sides have a say in the block size that actually will be used.
- o The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.
- o If needed, the Block options can also be used (without changes) to provide random access to power-of-two sized blocks within a resource representation.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119] and indicate requirement levels for compliant CoAP implementations.

In this document, the term "byte" is used in its now customary sense as a synonym for "octet".

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C, except that the operator "\*\*\*" stands for exponentiation.



## 2. Block-wise transfers

As discussed in the introduction, there are good reasons to limit the size of datagrams in constrained networks:

- o by the maximum datagram size (~ 64 KiB for UDP)
- o by the desire to avoid IP fragmentation (MTU of 1280 for IPv6)
- o by the desire to avoid adaptation layer fragmentation (60-80 bytes for 6LoWPAN [RFC4919])

When a resource representation is larger than can be comfortably transferred in the payload of a single CoAP datagram, a Block option can be used to indicate a block-wise transfer. As payloads can be sent both with requests and with responses, this specification provides two separate options for each direction of payload transfer.

In 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.

In most cases, all blocks being transferred for a body will be of the same size. The block size is not fixed by the protocol. To keep the implementation as simple as possible, the Block options support only a small range of power-of-two block sizes, from  $2^4$  (16) to  $2^{10}$  (1024) bytes. As bodies often will not evenly divide into the power-of-two block size chosen, the size need not be reached in the final block (but even for the final block, the chosen power-of-two size will still be indicated in the block size field of the Block option).

### 2.1. The Block Options

Type	C	U	N	R	Name	Format	Length	Default
23	C	U	-	-	Block2	uint	0-3 B	(none)
27	C	U	-	-	Block1	uint	0-3 B	(none)

Table 1: Block Option Numbers

Both Block1 and Block2 options can be present both in request and response messages. In either case, the Block1 Option pertains to the request payload, and the Block2 Option pertains to the response

payload.

Hence, for the methods defined in [I-D.ietf-core-coap], Block1 is useful with the payload-bearing POST and PUT requests and their responses. Block2 is useful with GET, POST, and PUT requests and their payload-bearing responses (2.01, 2.02, 2.04, 2.05 -- see section "Payload" of [I-D.ietf-core-coap]).

(As a memory aid: Block\_1\_ pertains to the payload of the \_1st\_ part of the request-response exchange, i.e. the request, and Block\_2\_ pertains to the payload of the \_2nd\_ part of the request-response exchange, i.e. the response.)

Where Block1 is present in a request or Block2 in a response (i.e., in that message to the payload of which it pertains) it indicates a block-wise transfer and describes how this block-wise payload forms part of the entire body being transferred ("descriptive usage"). Where it is present in the opposite direction, it provides additional control on how that payload will be formed or was processed ("control usage").

Implementation of either Block option is intended to be optional. However, when it is present in a CoAP message, it MUST be processed (or the message rejected); therefore it is identified as a critical option. It MUST NOT occur more than once.

## 2.2. Structure of a Block Option

Three items of information may need to be transferred in a Block option:

- o The size of the block (SZX);
- o whether more blocks are following (M);
- o the relative number of the block (NUM) within a sequence of blocks with the given size.

The value of the option is a 1-, 2- or 3-byte integer which encodes these three fields, see Figure 1.

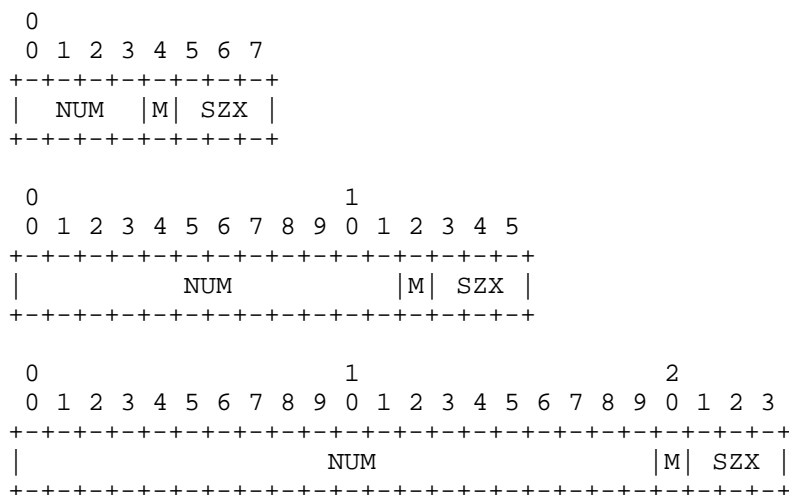


Figure 1: Block option value

The block size is encoded as 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 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. The block number is a variable-size (4, 12, or 20 bit) unsigned integer (uint, see Appendix A of [I-D.ietf-core-coap]) indicating the block number being requested or provided. Block number 0 indicates the first block of a 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 a 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. (Note that, as for any uint, the value 0 is efficiently indicated by a zero-length option; this, therefore, is different in semantics from the absence of the option.)

### 2.3. Block Options in Requests and Responses

The Block options are used in one of three roles:

- o In descriptive usage, i.e. a Block2 Option in a response (e.g., a 2.05 response for GET), or a Block1 Option in a request (e.g., 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 are required to complete the transfer of that body.
  - \* The block size given by SZX MUST match the size of the payload in bytes, if the M bit is set. (SZX does not govern the payload size if M is unset). For Block2, if the request suggested a larger value of SZX, the next request MUST move SZX

down to the size given here. (The effect is that, if the server uses the smaller of its preferred block size and the one 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 block numbers other than 0).
- 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 and plans to implement the request atomically (e.g., acts only upon reception of the last block of payload). 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.

## 2.4. Using the Block Options

Using one or both Block options, a single REST operation can be split into multiple CoAP message exchanges. As specified in [I-D.ietf-core-coap], each of these message exchanges uses their own CoAP Message ID.

When a request is answered with a response carrying a Block2 Option with the M bit set, the requester may retrieve additional blocks of the resource representation by sending further requests with the same options and a Block2 Option giving the block number and block size desired. In a request, the client **MUST** set the M bit of a Block2 Option to zero and the server **MUST** ignore it on reception.

To influence the block size used in a response, the requester also uses the Block2 Option, giving the desired size, a block number of zero and an M bit of zero. A server **MUST** use the block size indicated or a smaller size. Any further block-wise requests for blocks beyond the first one **MUST** indicate the same block size that was used by the server in the response for the first request that gave a desired size using a Block2 Option.

Once the Block2 Option is used by the requester and a first response has been received with a possibly adjusted block size, all further requests in a single block-wise transfer **SHOULD** ultimately use the same size, except that there may not be enough content to fill the last block (the one returned with the M bit not set). (Note that the client may start using the Block2 Option in a second request after a first request without a Block2 Option resulted in a Block 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.

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 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. If not all previous blocks are available at the server at this time, the transfer fails and error code 4.08 (Request Entity Incomplete) MUST be returned. 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 the Block1 than requested is a

hint to try a smaller SZX.)

Note that there is no way to perform multiple concurrently proceeding block-wise request payload transfer (e.g., PUT or POST) operations to the same URI. Starting a new block-wise sequence of requests to the same URI (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.)



### 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 separating the kind of Block option (1 or 2), block number (NUM), more bit (M), and block size exponent ( $2^{*(SZX+4)}$ ) 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.

The first example (Figure 2) shows a GET request that is split into three blocks. The server proposes a block size of 128, and the client agrees. The first two ACKs contain 128 bytes of payload each, and third ACK contains between 1 and 128 bytes.

CLIENT	SERVER
CON [MID=1234], GET, /status	----->
<----- ACK [MID=1234], 2.05 Content, 2/0/1/128	
CON [MID=1235], GET, /status, 2/1/0/128	----->
<----- ACK [MID=1235], 2.05 Content, 2/1/1/128	
CON [MID=1236], GET, /status, 2/2/0/128	----->
<----- ACK [MID=1236], 2.05 Content, 2/2/0/128	

Figure 2: Simple blockwise GET

In the second example (Figure 3), the client anticipates the blockwise transfer (e.g., because of a size indication in the link-format description [RFC6690]) and sends a size proposal. All ACK messages except for the last carry 64 bytes of payload; the last one carries between 1 and 64 bytes.

CLIENT		SERVER
CON [MID=1234], GET, /status, 2/0/0/64	----->	
<----- ACK [MID=1234], 2.05 Content, 2/0/1/64		
CON [MID=1235], GET, /status, 2/1/0/64	----->	
<----- ACK [MID=1235], 2.05 Content, 2/1/1/64		
:		:
:	...	:
:		:
CON [MID=1238], GET, /status, 2/4/0/64	----->	
<----- ACK [MID=1238], 2.05 Content, 2/4/1/64		
CON [MID=1239], GET, /status, 2/5/0/64	----->	
<----- ACK [MID=1239], 2.05 Content, 2/5/0/64		

Figure 3: Blockwise GET with early negotiation

In the third example (Figure 4), the client is surprised by the need for a blockwise transfer, and unhappy with the size chosen unilaterally by the server. As it did not send a size proposal initially, the negotiation only influences the size from the second message exchange onward. Since the client already obtained both the first and second 64-byte block in the first 128-byte exchange, it goes on requesting the third 64-byte block ("2/0/64"). None of this is (or needs to be) understood by the server, which simply responds to the requests as it best can.

CLIENT		SERVER
CON [MID=1234], GET, /status	----->	
<-----	ACK [MID=1234], 2.05 Content, 2/0/1/128	
CON [MID=1235], GET, /status, 2/2/0/64	----->	
<-----	ACK [MID=1235], 2.05 Content, 2/2/1/64	
CON [MID=1236], GET, /status, 2/3/0/64	----->	
<-----	ACK [MID=1236], 2.05 Content, 2/3/1/64	
CON [MID=1237], GET, /status, 2/4/0/64	----->	
<-----	ACK [MID=1237], 2.05 Content, 2/4/1/64	
CON [MID=1238], GET, /status, 2/5/0/64	----->	
<-----	ACK [MID=1238], 2.05 Content, 2/5/0/64	

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).

CLIENT		SERVER
CON [MID=1234], GET, /status	----->	
<-----	ACK [MID=1234], 2.05 Content, 2/0/1/128	
CON [MID=1235], GE//		
(timeout)		
CON [MID=1235], GET, /status, 2/2/0/64	----->	
<-----	ACK [MID=1235], 2.05 Content, 2/2/1/64	
:		:
:	...	:
:		:
CON [MID=1238], GET, /status, 2/5/0/64	----->	
<-----	ACK [MID=1238], 2.05 Content, 2/5/0/64	

Figure 5: Blockwise GET with late negotiation and lost CON

CLIENT		SERVER
CON [MID=1234], GET, /status	----->	
<-----	ACK [MID=1234], 2.05 Content, 2/0/1/128	
CON [MID=1235], GET, /status, 2/2/0/64	----->	
//tent, 2/2/1/64		
(timeout)		
CON [MID=1235], GET, /status, 2/2/0/64	----->	
<-----	ACK [MID=1235], 2.05 Content, 2/2/1/64	
:		:
:	...	:
:		:
CON [MID=1238], GET, /status, 2/5/0/64	----->	
<-----	ACK [MID=1238], 2.05 Content, 2/5/0/64	

Figure 6: Blockwise GET with late negotiation and lost ACK

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; only the final response tells the client that the PUT succeeded.

CLIENT		SERVER
CON [MID=1234], PUT, /options, 1/0/1/128	----->	
<-----	ACK [MID=1234], 2.04 Changed, 1/0/1/128	
CON [MID=1235], PUT, /options, 1/1/1/128	----->	
<-----	ACK [MID=1235], 2.04 Changed, 1/1/1/128	
CON [MID=1236], PUT, /options, 1/2/0/128	----->	
<-----	ACK [MID=1236], 2.04 Changed, 1/2/0/128	

Figure 7: Simple atomic blockwise PUT

A stateless server that simply builds/updates the resource in place (statelessly) may indicate this by not setting the more bit in the response (Figure 8); in this case, the response codes are valid separately for each block being updated. This is of course only an acceptable behavior of the server if the potential inconsistency present during the run of the message exchange sequence does not lead to problems, e.g. because the resource being created or changed is not yet or not currently in use.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1/0/1/128 ----->	
<----- ACK [MID=1234], 2.04 Changed, 1/0/0/128	
CON [MID=1235], PUT, /options, 1/1/1/128 ----->	
<----- ACK [MID=1235], 2.04 Changed, 1/1/0/128	
CON [MID=1236], PUT, /options, 1/2/0/128 ----->	
<----- ACK [MID=1236], 2.04 Changed, 1/2/0/128	

Figure 8: Simple stateless blockwise PUT

Finally, a server receiving a blockwise PUT or POST may want to indicate a smaller block size preference (Figure 9). In this case, the client SHOULD continue with a smaller block size; if it does, it MUST adjust the block number to properly count in that smaller size.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1/0/1/128 ----->	
<----- ACK [MID=1234], 2.04 Changed, 1/0/1/32	
CON [MID=1235], PUT, /options, 1/4/1/32 ----->	
<----- ACK [MID=1235], 2.04 Changed, 1/4/1/32	
CON [MID=1236], PUT, /options, 1/5/1/32 ----->	
<----- ACK [MID=1235], 2.04 Changed, 1/5/1/32	
CON [MID=1237], PUT, /options, 1/6/0/32 ----->	
<----- ACK [MID=1236], 2.04 Changed, 1/6/0/32	

Figure 9: Simple atomic blockwise PUT with negotiation

Block options may be used in both directions of a single exchange. The following example demonstrates a blockwise POST request, resulting in a separate blockwise response.

CLIENT	SERVER
CON [MID=1234], POST, /soap, 1/0/1/128	----->
<----- ACK [MID=1234], 2.01 Created, 1/0/1/128	
CON [MID=1235], POST, /soap, 1/1/1/128	----->
<----- ACK [MID=1235], 2.01 Created, 1/1/1/128	
CON [MID=1236], POST, /soap, 1/2/0/128	----->
<----- ACK [MID=1236], 0, 1/2/0/128	
<----- CON [MID=4712], 2.01 Created, 2/0/1/128	
ACK [MID=4712], 0, 2/0/1/128	----->
<----- CON [MID=4713], 2.01 Created, 2/1/1/128	
ACK [MID=4713], 0, 2/1/1/128	----->
<----- CON [MID=4714], 2.01 Created, 2/2/1/128	
ACK [MID=4714], 0, 2/2/1/128	----->
<----- CON [MID=4715], 2.01 Created, 2/3/0/128	
ACK [MID=4715], 0, 2/3/0/128	----->

Figure 10: Atomic blockwise POST with separate blockwise response

#### 4. The Size Option

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.

The Size Option may be used for three purposes:

- o in a request, to ask the server to provide a size estimate along with the usual response ("size request"). For this usage, the value MUST be set to 0.
- o in a response carrying a Block2 Option, to indicate the current estimate the server has of the total size of the resource representation.
- o in a request carrying a Block1 Option, to indicate the current estimate the client has of the total size of the resource representation.

In the latter two cases ("size indication"), the value of the option is the current estimate, measured in bytes.

A size request can be easily distinguished from a size indication, as the third case is not useful for a GET or DELETE, and an actual size indication of 0 would either be overridden by the actual size of the payload for a PUT or POST or would not be useful.

Apart from conveying/asking for size information, the Size option has 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 Option is "elective", i.e., a client MUST be prepared for the server to ignore the size estimate request. The Size Option MUST NOT occur more than once.

Type	C	U	N	R	Name	Format	Length	Default
28	-	-	N	-	Size	uint	0-4 B	(none)

Implementation Notes:

- o As a quality of implementation consideration, blockwise transfers for which the total size considerably exceeds the size of one block are expected to include size indications, whenever those can be provided without undue effort (preferably with the first block exchanged). If the size estimate does not change, the indication does not need to be repeated for every block.
- o The end of a blockwise transfer is governed by the M bits in the Block Options, not by exhausting the size estimates exchanged.
- o As usual for an option of type uint, the value 0 is best expressed as an empty option (0 bytes). There is no default value.
- o Size is neither critical nor unsafe, and is marked as No-Cache-Key.



## 5. HTTP Mapping Considerations

In this subsection, we give some brief examples for the influence the Block options might have on intermediaries that map between CoAP and HTTP.

For mapping CoAP requests to HTTP, the intermediary may want to map the sequence of block-wise transfers into a single HTTP transfer. E.g., for a GET request, the intermediary could perform the HTTP request once the first block has been requested and could then fulfill all further block requests out of its cache. A constrained implementation may not be able to cache the entire object and may use a combination of TCP flow control and (in particular if timeouts occur) HTTP range requests to obtain the information necessary for the next block transfer at the right time.

For PUT or POST requests, there is more variation in how HTTP servers might implement ranges. Some WebDAV servers do, but in general the CoAP-to-HTTP intermediary will have to try sending the payload of all the blocks of a block-wise transfer within one HTTP request. If enough buffering is available, this request can be started when the last CoAP block is received. A constrained implementation may want to relieve its buffering by already starting to send the HTTP request at the time the first CoAP block is received; any HTTP 408 status code that indicates that the HTTP server became impatient with the resulting transfer can then be mapped into a CoAP 4.08 response code (similarly, 413 maps to 4.13).

For mapping HTTP to CoAP, the intermediary may want to map a single HTTP transfer into a sequence of block-wise transfers. If the HTTP client is too slow delivering a request body on a PUT or POST, the CoAP server might time out and return a 4.08 response code, which in turn maps well to an HTTP 408 status code (again, 4.13 maps to 413). HTTP range requests received on the HTTP side may be served out of a cache and/or mapped to GET requests that request a sequence of blocks overlapping the range.

(Note that, while the semantics of CoAP 4.08 and HTTP 408 differ, this difference is largely due to the different way the two protocols are mapped to transport. HTTP has an underlying TCP connection, which supplies connection state, so a HTTP 408 status code can immediately be used to indicate that a timeout occurred during transmitting a request through that active TCP connection. The CoAP 4.08 response code indicates one or more missing blocks, which may be due to timeouts or resource constraints; as there is no connection state, there is no way to deliver such a response immediately; instead, it is delivered on the next block transfer. Still, HTTP 408 is probably the best mapping back to HTTP, as the timeout is the most

likely cause for a CoAP 4.08. Note that there is no way to distinguish a timeout from a missing block for a server without creating additional state, the need for which we want to avoid.)

## 6. IANA Considerations

This draft adds the following option numbers to the CoAP Option Numbers registry of [I-D.ietf-core-coap]:

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

Table 2: CoAP Option Numbers

This draft adds the following response code to the CoAP Response Codes registry of [I-D.ietf-core-coap]:

Code	Description	Reference
136	4.08 Request Entity Incomplete	[RFCXXXX]

Table 3: CoAP Response Codes

## 7. Security Considerations

Providing access to blocks within a resource may lead to surprising vulnerabilities. Where requests are not implemented atomically, an attacker may be able to exploit a race condition or confuse a server by inducing it to use a partially updated resource representation. Partial transfers may also make certain problematic data invisible to intrusion detection systems; it is RECOMMENDED that an intrusion detection system (IDS) that analyzes resource representations transferred by CoAP implement the Block options to gain access to entire resource representations. Still, approaches such as transferring even-numbered blocks on one path and odd-numbered blocks on another path, or even transferring blocks multiple times with different content and obtaining a different interpretation of temporal order at the IDS than at the server, may prevent an IDS from seeing the whole picture. These kinds of attacks are well understood from IP fragmentation and TCP segmentation; CoAP does not add fundamentally new considerations.

Where access to a resource is only granted to clients making use of a specific security association, all blocks of that resource MUST be subject to the same security checks; it MUST NOT be possible for unprotected exchanges to influence blocks of an otherwise protected resource. As a related consideration, where object security is employed, PUT/POST should be implemented in the atomic fashion, unless the object security operation is performed on each access and the creation of unusable resources can be tolerated.

A stateless server might be susceptible to an attack where the adversary sends a Block1 (e.g., PUT) block with a high block number: A naive implementation might exhaust its resources by creating a huge resource representation.

Misleading size indications may be used by an attacker to induce buffer overflows in poor implementations, for which the usual considerations apply.

### 7.1. Mitigating Resource Exhaustion Attacks

Certain blockwise requests may induce the server to create state, e.g. to create a snapshot for the blockwise GET of a fast-changing resource to enable consistent access to the same version of a resource for all blocks, or to create temporary resource representations that are collected until pressed into service by a final PUT or POST with the more bit unset. All mechanisms that induce a server to create state that cannot simply be cleaned up create opportunities for denial-of-service attacks. Servers SHOULD avoid being subject to resource exhaustion based on state created by

untrusted sources. But even if this is done, the mitigation may cause a denial-of-service to a legitimate request when it is drowned out by other state-creating requests. Wherever possible, servers should therefore minimize the opportunities to create state for untrusted sources, e.g. by using stateless approaches.

Performing segmentation at the application layer is almost always better in this respect than at the transport layer or lower (IP fragmentation, adaptation layer fragmentation), e.g. because there is application layer semantics that can be used for mitigation or because lower layers provide security associations that can prevent attacks. However, it is less common to apply timeouts and keepalive mechanisms at the application layer than at lower layers. Servers MAY want to clean up accumulated state by timing it out (cf. response code 4.08), and clients SHOULD be prepared to run blockwise transfers in an expedient way to minimize the likelihood of running into such a timeout.

## 7.2. Mitigating Amplification Attacks

[I-D.ietf-core-coap] discusses the susceptibility of CoAP end-points for use in amplification attacks.

A CoAP server can reduce the amount of amplification it provides to an attacker by offering large resource representations only in relatively small blocks. With this, e.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

## 8. Acknowledgements

Much of the content of this draft is the result of discussions with the [I-D.ietf-core-coap] authors, and via many CoRE WG discussions.

Charles Palmer provided extensive editorial comments to a previous version of this draft, some of which the authors hope to have covered in this version. Esko Dijk reviewed a more recent version, leading to a number of further editorial improvements as well as a solution to the 4.13 ambiguity problem. Markus Becker proposed getting rid of an ill-conceived default value for the Block2 and Block1 options.

Kepeng Li, Linyi Tian, and Barry Leiba wrote up an early version of the Size Option, which has informed this draft.

## 9. References

### 9.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-12 (work in progress), October 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,  
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext  
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

### 9.2. Informative References

- [REST] Fielding, R., "Architectural Styles and the Design of  
Network-based Software Architectures", Ph.D. Dissertation,  
University of California, Irvine, 2000, <[http://  
www.ics.uci.edu/~fielding/pubs/dissertation/  
fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6  
over Low-Power Wireless Personal Area Networks (6LoWPANs):  
Overview, Assumptions, Problem Statement, and Goals",  
RFC 4919, August 2007.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link  
Format", RFC 6690, August 2012.

Authors' Addresses

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

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

Zach Shelby (editor)  
Sensinode  
Kidekuja 2  
Vuokatti 88600  
Finland

Phone: +358407796297  
Email: zach@sensinode.com





CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: June 9, 2013

Z. Shelby  
Sensinode  
K. Hartke  
C. Bormann  
Universitaet Bremen TZI  
B. Frank  
SkyFoundry  
December 6, 2012

Constrained Application Protocol (CoAP)  
draft-ietf-core-coap-13

Abstract

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as 6LoWPAN often have high packet error rates and a typical throughput of 10s of kbit/s. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP easily interfaces with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

Status of this Memo

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

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

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

This Internet-Draft will expire on June 9, 2013.

## Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	6
1.1. Features . . . . .	6
1.2. Terminology . . . . .	7
2. Constrained Application Protocol . . . . .	10
2.1. Messaging Model . . . . .	11
2.2. Request/Response Model . . . . .	12
2.3. Intermediaries and Caching . . . . .	15
2.4. Resource Discovery . . . . .	15
3. Message Format . . . . .	16
3.1. Option Format . . . . .	17
3.2. Option Value Formats . . . . .	19
4. Message Transmission . . . . .	20
4.1. Messages and Endpoints . . . . .	20
4.2. Messages Transmitted Reliably . . . . .	21
4.3. Messages Transmitted Without Reliability . . . . .	22
4.4. Message Correlation . . . . .	22
4.5. Message Deduplication . . . . .	23
4.6. Message Size . . . . .	24
4.7. Congestion Control . . . . .	25
4.8. Transmission Parameters . . . . .	26
4.8.1. Changing The Parameters . . . . .	26
4.8.2. Time Values derived from Transmission Parameters . . . . .	27
5. Request/Response Semantics . . . . .	29
5.1. Requests . . . . .	29
5.2. Responses . . . . .	29
5.2.1. Piggy-backed . . . . .	30
5.2.2. Separate . . . . .	31
5.2.3. Non-Confirmable . . . . .	32
5.3. Request/Response Matching . . . . .	32
5.3.1. Token . . . . .	32

5.3.2.	Request/Response Matching Rules . . . . .	33
5.4.	Options . . . . .	33
5.4.1.	Critical/Elective . . . . .	34
5.4.2.	Proxy Unsafe/Safe and Cache-Key . . . . .	35
5.4.3.	Length . . . . .	35
5.4.4.	Default Values . . . . .	36
5.4.5.	Repeatable Options . . . . .	36
5.4.6.	Option Numbers . . . . .	36
5.5.	Payloads and Representations . . . . .	37
5.5.1.	Representation . . . . .	37
5.5.2.	Diagnostic Payload . . . . .	38
5.5.3.	Selected Representation . . . . .	38
5.5.4.	Content Negotiation . . . . .	38
5.6.	Caching . . . . .	39
5.6.1.	Freshness Model . . . . .	39
5.6.2.	Validation Model . . . . .	40
5.7.	Proxying . . . . .	40
5.7.1.	Proxy Operation . . . . .	41
5.7.2.	Forward-Proxies . . . . .	42
5.7.3.	Reverse-Proxies . . . . .	43
5.8.	Method Definitions . . . . .	43
5.8.1.	GET . . . . .	43
5.8.2.	POST . . . . .	44
5.8.3.	PUT . . . . .	44
5.8.4.	DELETE . . . . .	44
5.9.	Response Code Definitions . . . . .	45
5.9.1.	Success 2.xx . . . . .	45
5.9.2.	Client Error 4.xx . . . . .	46
5.9.3.	Server Error 5.xx . . . . .	47
5.10.	Option Definitions . . . . .	48
5.10.1.	Uri-Host, Uri-Port, Uri-Path and Uri-Query . . . . .	49
5.10.2.	Proxy-Uri and Proxy-Scheme . . . . .	50
5.10.3.	Content-Format . . . . .	51
5.10.4.	Accept . . . . .	51
5.10.5.	Max-Age . . . . .	51
5.10.6.	ETag . . . . .	52
5.10.7.	Location-Path and Location-Query . . . . .	53
5.10.8.	Conditional Request Options . . . . .	53
6.	CoAP URIs . . . . .	55
6.1.	coap URI Scheme . . . . .	55
6.2.	coaps URI Scheme . . . . .	56
6.3.	Normalization and Comparison Rules . . . . .	56
6.4.	Decomposing URIs into Options . . . . .	57
6.5.	Composing URIs from Options . . . . .	58
7.	Discovery . . . . .	59
7.1.	Service Discovery . . . . .	59
7.2.	Resource Discovery . . . . .	60
7.2.1.	'ct' Attribute . . . . .	60

8. Multicast CoAP . . . . .	61
8.1. Messaging Layer . . . . .	61
8.2. Request/Response Layer . . . . .	61
8.2.1. Caching . . . . .	62
8.2.2. Proxying . . . . .	63
9. Securing CoAP . . . . .	63
9.1. DTLS-secured CoAP . . . . .	64
9.1.1. Messaging Layer . . . . .	65
9.1.2. Request/Response Layer . . . . .	66
9.1.3. Endpoint Identity . . . . .	66
10. Cross-Protocol Proxying between CoAP and HTTP . . . . .	68
10.1. CoAP-HTTP Proxying . . . . .	69
10.1.1. GET . . . . .	69
10.1.2. PUT . . . . .	70
10.1.3. DELETE . . . . .	70
10.1.4. POST . . . . .	70
10.2. HTTP-CoAP Proxying . . . . .	71
10.2.1. OPTIONS and TRACE . . . . .	71
10.2.2. GET . . . . .	71
10.2.3. HEAD . . . . .	72
10.2.4. POST . . . . .	72
10.2.5. PUT . . . . .	73
10.2.6. DELETE . . . . .	73
10.2.7. CONNECT . . . . .	73
11. Security Considerations . . . . .	73
11.1. Protocol Parsing, Processing URIs . . . . .	73
11.2. Proxying and Caching . . . . .	74
11.3. Risk of amplification . . . . .	75
11.4. IP Address Spoofing Attacks . . . . .	76
11.5. Cross-Protocol Attacks . . . . .	76
12. IANA Considerations . . . . .	78
12.1. CoAP Code Registry . . . . .	78
12.1.1. Method Codes . . . . .	79
12.1.2. Response Codes . . . . .	80
12.2. Option Number Registry . . . . .	81
12.3. Content-Format Registry . . . . .	83
12.4. URI Scheme Registration . . . . .	84
12.5. Secure URI Scheme Registration . . . . .	85
12.6. Service Name and Port Number Registration . . . . .	86
12.7. Secure Service Name and Port Number Registration . . . . .	87
12.8. Multicast Address Registration . . . . .	88
13. Acknowledgements . . . . .	88
14. References . . . . .	89
14.1. Normative References . . . . .	89
14.2. Informative References . . . . .	91
Appendix A. Examples . . . . .	93
Appendix B. URI Examples . . . . .	98
Appendix C. Changelog . . . . .	99

Authors' Addresses . . . . .	108
------------------------------	-----

## 1. Introduction

The use of web services on the Internet has become ubiquitous in most applications, and depends on the fundamental Representational State Transfer [REST] architecture of the web.

The Constrained RESTful Environments (CoRE) work aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN, [RFC4944]). Constrained networks like 6LoWPAN support the expensive fragmentation of IPv6 packets into small link-layer frames. One design goal of CoAP has been to keep message overhead small, thus limiting the use of fragmentation.

One of the main goals of CoAP is to design a generic web protocol for the special requirements of this constrained environment, especially considering energy, building automation and other machine-to-machine (M2M) applications. The goal of CoAP is not to blindly compress HTTP [RFC2616], but rather to realize a subset of REST common with HTTP but optimized for M2M applications. Although CoAP could be used for compressing simple HTTP interfaces, it more importantly also offers features for M2M such as built-in discovery, multicast support and asynchronous message exchanges.

This document specifies the Constrained Application Protocol (CoAP), which easily translates to HTTP for integration with the existing web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments and M2M applications.

### 1.1. Features

CoAP has the following main features:

- o Constrained web protocol fulfilling M2M requirements.
- o UDP binding with optional reliability supporting unicast and multicast requests.
- o Asynchronous message exchanges.
- o Low header overhead and parsing complexity.
- o URI and Content-type support.
- o Simple proxy and caching capabilities.

- o A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.
- o Security binding to Datagram Transport Layer Security (DTLS).

## 1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC2616]. In addition, this specification defines the following terminology:

### Endpoint

An entity participating in the CoAP protocol. Colloquially, an endpoint lives on a "Node", although "Host" would be more consistent with Internet standards usage, and is further identified by transport layer multiplexing information that can include a UDP port number and a security association (Section 4.1).

### Sender

The originating endpoint of a message. When the aspect of identification of the specific sender is in focus, also "source endpoint".

### Recipient

The destination endpoint of a message. When the aspect of identification of the specific recipient is in focus, also "destination endpoint".

### Client

The originating endpoint of a request; the destination endpoint of a response.

### Server

The destination endpoint of a request; the originating endpoint of a response.



#### Origin Server

The server on which a given resource resides or is to be created.

#### Intermediary

A CoAP endpoint that acts both as a server and as a client towards (possibly via further intermediaries) an origin server. A common form of an intermediary is a proxy; several classes of such proxies are discussed in this specification.

#### Proxy

An intermediary that mainly is concerned with forwarding requests and relaying back responses, possibly performing caching, namespace translation, or protocol translation in the process. As opposed to intermediaries in the general sense, proxies generally do not implement specific application semantics. Based on the position in the overall structure of the request forwarding, there are two common forms of proxy: forward-proxy and reverse-proxy. In some cases, a single endpoint might act as an origin server, forward-proxy, or reverse-proxy, switching behavior based on the nature of each request.

#### Forward-Proxy

A "forward-proxy" is an endpoint selected by a client, usually via local configuration rules, to perform requests on behalf of the client, doing any necessary translations. Some translations are minimal, such as for proxy requests for "coap" URIs, whereas other requests might require translation to and from entirely different application-layer protocols.

#### Reverse-Proxy

A "reverse-proxy" is an endpoint that stands in for one or more other server(s) and satisfies requests on behalf of these, doing any necessary translations. Unlike a forward-proxy, the client may not be aware that it is communicating with a reverse-proxy; a reverse-proxy receives requests as if it was the origin server for the target resource.

#### Cross-Proxy

A cross-protocol proxy, or "cross-proxy" for short, is a proxy that translates between different protocols, such as a CoAP-to-HTTP proxy or an HTTP-to-CoAP proxy. While this specification makes very specific demands of CoAP-to-CoAP proxies, there is more variation possible in cross-proxies.

#### Confirmable Message

Some messages require an acknowledgement. These messages are called "Confirmable". When no packets are lost, each confirmable message elicits exactly one return message of type Acknowledgement

or type Reset.

#### Non-Confirmable Message

Some other messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual success is sufficient.

#### Acknowledgement Message

An Acknowledgement message acknowledges that a specific Confirmable Message arrived. It does not indicate success or failure of any encapsulated request.

#### Reset Message

A Reset message indicates that a specific message (confirmable or non-confirmable) was received, but some context is missing to properly process it. This condition is usually caused when the receiving node has rebooted and has forgotten some state that would be required to interpret the message. Provoking a Reset message (e.g., by sending an empty Confirmable message) is also useful as an inexpensive check of the liveness of an endpoint ("CoAP ping").

#### Piggy-backed Response

A Piggy-backed Response is included right in a CoAP Acknowledgement (ACK) message that is sent to acknowledge receipt of the Request for this Response (Section 5.2.1).

#### Separate Response

When a Confirmable message carrying a Request is acknowledged with an empty message (e.g., because the server doesn't have the answer right away), a Separate Response is sent in a separate message exchange (Section 5.2.2).

#### Critical Option

An option that would need to be understood by the endpoint receiving the message in order to properly process the message (Section 5.4.1). Note that the implementation of critical options is, as the name "Option" implies, generally optional: unsupported critical options lead to an error response or summary rejection of the message.

#### Elective Option

An option that is intended to be ignored by an endpoint that does not understand it. Processing the message even without understanding the option is acceptable (Section 5.4.1).

#### Unsafe Option

An option that would need to be understood by a proxy receiving the message in order to safely forward the message (Section 5.4.2).

#### Safe Option

An option that is intended to be safe for forwarding by a proxy that does not understand it. Forwarding the message even without understanding the option is acceptable (Section 5.4.2).

#### Resource Discovery

The process where a CoAP client queries a server for its list of hosted resources (i.e., links, Section 7).

#### Content-Format

The combination of an Internet media type, potentially with specific parameters given, and a content-coding (which is often the identity content-coding), identified by a numeric identifier defined by the CoAP Content-Format registry. When the focus is less on the numeric identifier than on the combination of these characteristics of a resource representation, this is also called "representation format".

In this specification, the term "byte" is used in its now customary sense as a synonym for "octet".

All multi-byte integers in this protocol are interpreted in network byte order.

Where arithmetic is used, this specification uses the notation familiar from the programming language C, except that the operator "\*\*\*" stands for exponentiation.

## 2. Constrained Application Protocol

The interaction model of CoAP is similar to the client/server model of HTTP. However, machine-to-machine interactions typically result in a CoAP implementation acting in both client and server roles. A CoAP request is equivalent to that of HTTP, and is sent by a client to request an action (using a method code) on a resource (identified by a URI) on a server. The server then sends a response with a response code; this response may include a resource representation.

Unlike HTTP, CoAP deals with these interchanges asynchronously over a datagram-oriented transport such as UDP. This is done logically using a layer of messages that supports optional reliability (with exponential back-off). CoAP defines four types of messages:

Confirmable, Non-Confirmable, Acknowledgement, Reset; method codes and response codes included in some of these messages make them carry requests or responses. The basic exchanges of the four types of messages are somewhat orthogonal to the request/response interactions; requests can be carried in Confirmable and Non-Confirmable messages, and responses can be carried in these as well as piggy-backed in Acknowledgement messages.

One could think of CoAP logically as using a two-layer approach, a CoAP messaging layer used to deal with UDP and the asynchronous nature of the interactions, and the request/response interactions using Method and Response codes (see Figure 1). CoAP is however a single protocol, with messaging and request/response just features of the CoAP header.

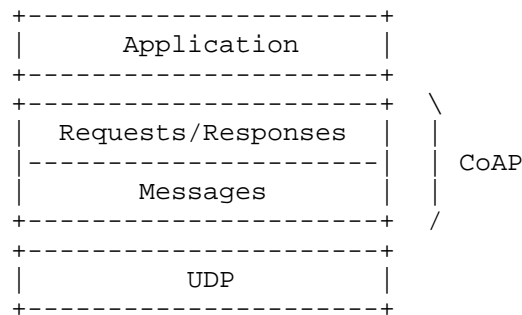


Figure 1: Abstract layering of CoAP

## 2.1. Messaging Model

The CoAP messaging model is based on the exchange of messages over UDP between endpoints.

CoAP uses a short fixed-length binary header (4 bytes) that may be followed by compact binary options and a payload. This message format is shared by requests and responses. The CoAP message format is specified in Section 3. Each message contains a Message ID used to detect duplicates and for optional reliability.

Reliability is provided by marking a message as Confirmable (CON). A Confirmable message is retransmitted using a default timeout and exponential back-off between retransmissions, until the recipient sends an Acknowledgement message (ACK) with the same Message ID (for example, 0x7d34) from the corresponding endpoint; see Figure 2. When a recipient is not at all able to process a Confirmable message (i.e., not even able to provide a suitable error response), it replies with a Reset message (RST) instead of an Acknowledgement

(ACK).

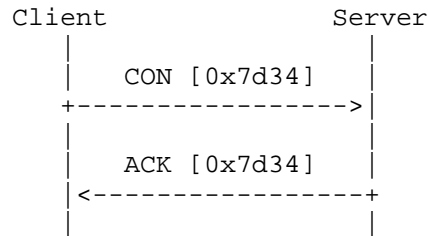


Figure 2: Reliable message transmission

A message that does not require reliable transmission, for example each single measurement out of a stream of sensor data, can be sent as a Non-confirmable message (NON). These are not acknowledged, but still have a Message ID for duplicate detection; see Figure 3. When a recipient is not able to process a Non-confirmable message, it may reply with a Reset message (RST).

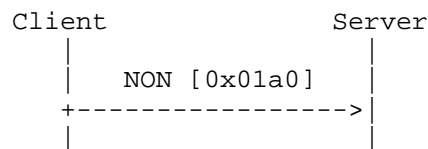


Figure 3: Unreliable message transmission

See Section 4 for details of CoAP messages.

As CoAP is based on UDP, it also supports the use of multicast IP destination addresses, enabling multicast CoAP requests. Section 8 discusses the proper use of CoAP messages with multicast addresses and precautions for avoiding response congestion.

Several security modes are defined for CoAP in Section 9 ranging from no security to certificate-based security. This document specifies a binding to DTLS for securing the protocol; the use of IPsec with CoAP is discussed in [I-D.bormann-core-ipsec-for-coap].

## 2.2. Request/Response Model

CoAP request and response semantics are carried in CoAP messages, which include either a Method code or Response code, respectively. Optional (or default) request and response information, such as the URI and payload media type are carried as CoAP options. A Token is used to match responses to requests independently from the underlying messages (Section 5.3).

A request is carried in a Confirmable (CON) or Non-confirmable (NON) message, and if immediately available, the response to a request carried in a Confirmable message is carried in the resulting Acknowledgement (ACK) message. This is called a piggy-backed response, detailed in Section 5.2.1. Two examples for a basic GET request with piggy-backed response are shown in Figure 4, one successful, one resulting in a 4.04 (Not Found) response.

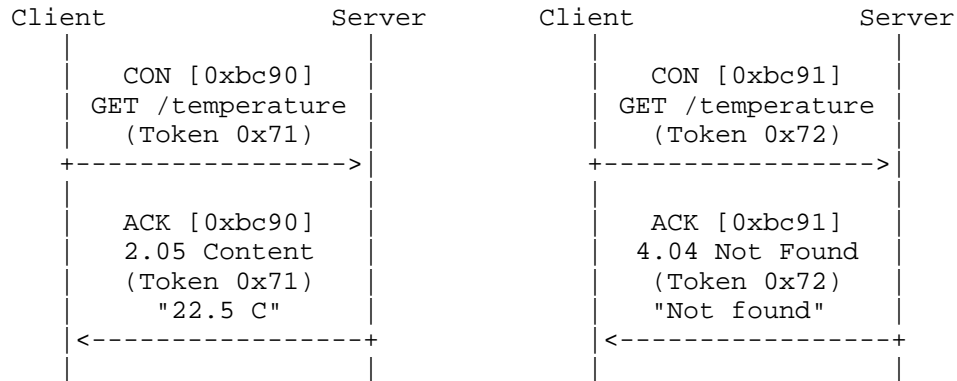


Figure 4: Two GET requests with piggy-backed responses

If the server is not able to respond immediately to a request carried in a Confirmable message, it simply responds with an empty Acknowledgement message so that the client can stop retransmitting the request. When the response is ready, the server sends it in a new Confirmable message (which then in turn needs to be acknowledged by the client). This is called a separate response, as illustrated in Figure 5 and described in more detail in Section 5.2.2.

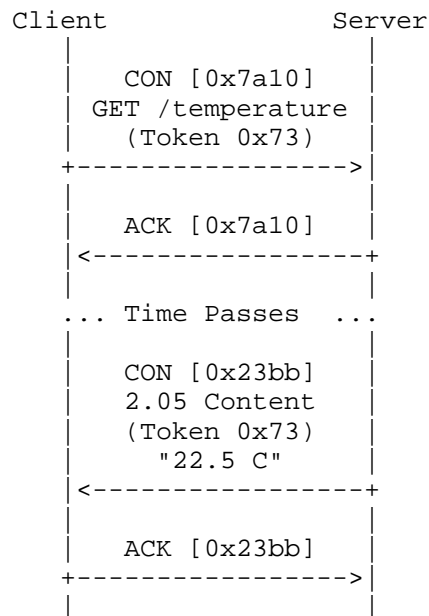


Figure 5: A GET request with a separate response

Likewise, if a request is sent in a Non-Confirmable message, then the response is usually sent using a new Non-Confirmable message, although the server may send a Confirmable message. This type of exchange is illustrated in Figure 6.

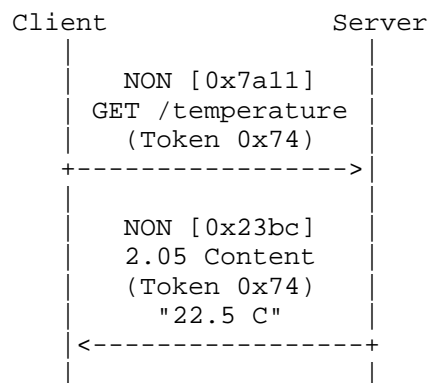


Figure 6: A NON request and response

CoAP makes use of GET, PUT, POST and DELETE methods in a similar manner to HTTP, with the semantics specified in Section 5.8. (Note that the detailed semantics of CoAP methods are "almost, but not

entirely unlike" those of HTTP methods: Intuition taken from HTTP experience generally does apply well, but there are enough differences that make it worthwhile to actually read the present specification.)

URI support in a server is simplified as the client already parses the URI and splits it into host, port, path and query components, making use of default values for efficiency. Response codes correspond to a small subset of HTTP response codes with a few CoAP specific codes added, as defined in Section 5.9.

### 2.3. Intermediaries and Caching

The protocol supports the caching of responses in order to efficiently fulfill requests. Simple caching is enabled using freshness and validity information carried with CoAP responses. A cache could be located in an endpoint or an intermediary. Caching functionality is specified in Section 5.6.

Proxying is useful in constrained networks for several reasons, including network traffic limiting, to improve performance, to access resources of sleeping devices or for security reasons. The proxying of requests on behalf of another CoAP endpoint is supported in the protocol. When using a proxy, the URI of the resource to request is included in the request, while the destination IP address is set to the address of the proxy. See Section 5.7 for more information on proxy functionality.

As CoAP was designed according to the REST architecture and thus exhibits functionality similar to that of the HTTP protocol, it is quite straightforward to map from CoAP to HTTP and from HTTP to CoAP. Such a mapping may be used to realize an HTTP REST interface using CoAP, or for converting between HTTP and CoAP. This conversion can be carried out by a cross-protocol proxy ("cross-proxy"), which converts the method or response code, media type, and options to the corresponding HTTP feature. Section 10 provides more detail about HTTP mapping.

### 2.4. Resource Discovery

Resource discovery is important for machine-to-machine interactions, and is supported using the CoRE Link Format [RFC6690] as discussed in Section 7.



### 3. Message Format

CoAP is based on the exchange of short messages which, by default, are transported over UDP (i.e. each CoAP message occupies the data section of one UDP datagram). CoAP may also be used over Datagram Transport Layer Security (DTLS) (see Section 9.1). It could also be used over other transports such as SMS, TCP or SCTP, the specification of which is out of this document's scope.

CoAP messages are encoded in a simple binary format. The message format starts with a fixed-size 4-byte header. This is followed by a variable-length Token value which can be between 0 and 8 bytes long. Following the Token value comes a sequence of zero or more CoAP Options in Type-Length-Value (TLV) format, optionally followed by a payload which takes up the rest of the datagram.

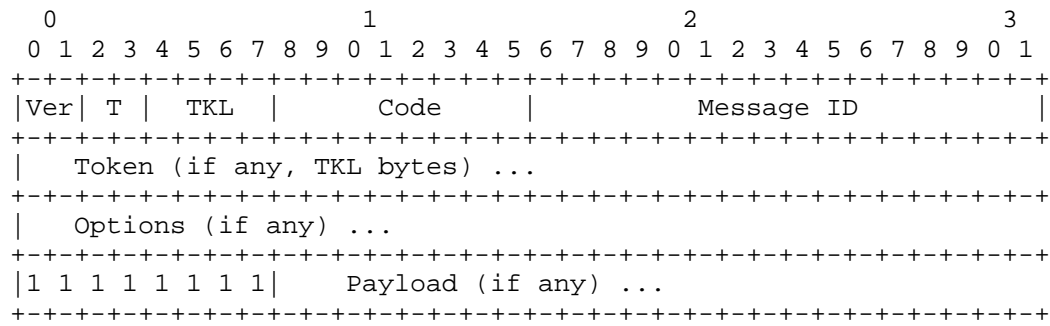


Figure 7: Message Format

The fields in the header are defined as follows:

Version (Ver): 2-bit unsigned integer. Indicates the CoAP version number. Implementations of this specification MUST set this field to 1. Other values are reserved for future versions.

Type (T): 2-bit unsigned integer. Indicates if this message is of type Confirmable (0), Non-Confirmable (1), Acknowledgement (2) or Reset (3). The semantics of these message types are defined in Section 4.

Token Length (TKL): 4-bit unsigned integer. Indicates the length of the variable-length Token field (0-8 bytes). Lengths 9-15 are reserved, MUST NOT be sent, and MUST be processed as a message format error.

Code: 8-bit unsigned integer. Indicates if the message carries a request (1-31) or a response (64-191), or is empty (0). (All other code values are reserved.) In case of a request, the Code field indicates the Request Method; in case of a response a Response Code. Possible values are maintained in the CoAP Code Registry (Section 12.1). The semantics of requests and responses are defined in Section 5.

Message ID: 16-bit unsigned integer in network byte order. Used for the detection of message duplication, and to match messages of type Acknowledgement/Reset to messages of type Confirmable/Non-confirmable. The rules for generating a Message ID and matching messages are defined in Section 4.

The header is followed by the Token value, which may be 0 to 8 bytes, as given by the Token Length field. The Token value is used to correlate requests and responses. The rules for generating a Token and correlating requests and responses are defined in Section 5.3.1.

Header and Token are followed by zero or more Options (Section 3.1). An Option can be followed by the end of the message, by another Option, or by the Payload Marker and the payload.

Following the header, token, and options, if any, comes the optional payload. If present and of non-zero length, it is prefixed by a fixed, one-byte Payload Marker (0xFF) which indicates the end of options and the start of the payload. The payload data extends from after the marker to the end of the UDP datagram, i.e., the Payload Length is calculated from the datagram size. The absence of the Payload Marker denotes a zero-length payload. The presence of a marker followed by a zero-length payload MUST be processed as a message format error.

### 3.1. Option Format

CoAP defines a number of options which can be included in a message. Each option instance in a message specifies the Option Number of the defined CoAP option, the length of the Option Value and the Option Value itself.

Instead of specifying the Option Number directly, the instances MUST appear in order of their Option Numbers and a delta encoding is used between them: The Option Number for each instance is calculated as the sum of its delta and the Option Number of the preceding instance in the message. For the first instance in a message, a preceding option instance with Option Number zero is assumed. Multiple instances of the same option can be included by using a delta of zero.

Option Numbers are maintained in the CoAP Option Number Registry (Section 12.2). See Section 5.4 for the semantics of the options defined in this document.

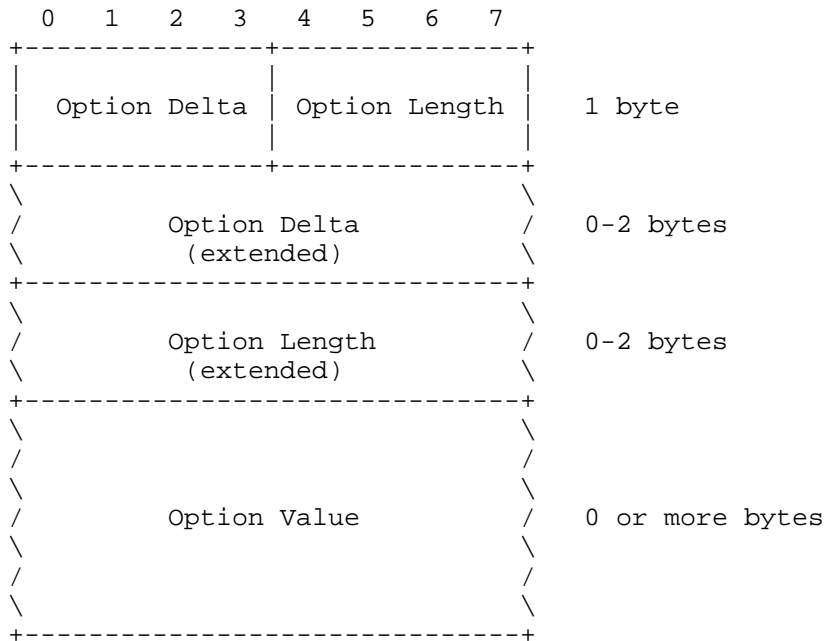


Figure 8: Option Format

The fields in an option are defined as follows:

Option Delta: 4-bit unsigned integer. A value between 0 and 12 indicates the Option Delta. Three values are reserved for special constructs:

- 13: An 8-bit unsigned integer follows the initial byte and indicates the Option Delta minus 13.
- 14: A 16-bit unsigned integer in network byte order follows the initial byte and indicates the Option Delta minus 269.
- 15: Reserved for the Payload Marker. If the field is set to this value but the entire byte is not the payload marker, this MUST be processed as a message format error.

The resulting Option Delta is used as the difference between the Option Number of this option and that of the previous option (or zero for the first option). In other words, the Option Number is

calculated by simply summing the Option Delta values of this and all previous options before it.

Option Length: 4-bit unsigned integer. A value between 0 and 12 indicates the length of the Option Value, in bytes. Three values are reserved for special constructs:

13: An 8-bit unsigned integer precedes the Option Value and indicates the Option Length minus 13.

14: A 16-bit unsigned integer in network byte order precedes the Option Value and indicates the Option Length minus 269.

15: Reserved for future use. If the field is set to this value, it MUST be processed as a message format error.

Value: A sequence of exactly Option Length bytes. The length and format of the Option Value depend on the respective option, which MAY define variable length values. See Section 3.2 for the formats used in this document; options defined in other documents MAY make use of other option value formats.

### 3.2. Option Value Formats

The options defined in this document make use of the following option value formats.

empty: A zero-length sequence of bytes.

opaque: An opaque sequence of bytes.

uint: A non-negative integer which is represented in network byte order using the number of bytes given by the Option Length field.

An option definition may specify a range of permissible numbers of bytes; if it has a choice, a sender SHOULD represent the integer with as few bytes as possible, i.e., without leading zeros. A recipient MUST be prepared to process values with leading zeros.

Implementation Note: The exceptional behavior permitted for the sender is intended for highly constrained, templated implementations (e.g., hardware implementations) that use fixed size options in the templates.

string:    A Unicode string which is encoded using UTF-8 [RFC3629] in Net-Unicode form [RFC5198].

Note that here and in all other places where UTF-8 encoding is used in the CoAP protocol, the intention is that the encoded strings can be directly used and compared as opaque byte strings by CoAP protocol implementations. There is no expectation and no need to perform normalization within a CoAP implementation (except where Unicode strings that are not known to be normalized are imported from sources outside the CoAP protocol). Note also that ASCII strings (that do not make use of special control characters) are always valid UTF-8 Net-Unicode strings.

#### 4. Message Transmission

CoAP messages are exchanged asynchronously between CoAP endpoints. They are used to transport CoAP requests and responses, the semantics of which are defined in Section 5.

As CoAP is bound to non-reliable transports such as UDP, CoAP messages may arrive out of order, appear duplicated, or go missing without notice. For this reason, CoAP implements a lightweight reliability mechanism, without trying to re-create the full feature set of a transport like TCP. It has the following features:

- o Simple stop-and-wait retransmission reliability with exponential back-off for Confirmable messages.
- o Duplicate detection for both Confirmable and Non-Confirmable messages.

##### 4.1. Messages and Endpoints

A CoAP endpoint is the source or destination of a CoAP message. It is identified depending on the security mode used (see Section 9): With no security, the endpoint is solely identified by an IP address and a UDP port number. With other security modes, the endpoint is identified as defined by the security mode.

There are different types of messages. The type of a message is specified by the Type field of the CoAP Header.

Separate from the message type, a message may carry a request, a response, or be empty. This is signaled by the Request/Response Code field in the CoAP Header and is relevant to the request/response model. Possible values for the field are maintained in the CoAP Code

Registry (Section 12.1).

An empty message has the Code field set to 0. The Token Length field MUST be set to 0 and no bytes MUST be present after the Message ID field. If there are any bytes, they MUST be processed as a message format error.

#### 4.2. Messages Transmitted Reliably

The reliable transmission of a message is initiated by marking the message as Confirmable in the CoAP header. A Confirmable message always carries either a request or response and MUST NOT be empty. A recipient MUST acknowledge such a message with an Acknowledgement message or, if it lacks context to process the message properly (including the case where the message is empty or has a message format error), MUST reject it; rejecting a Confirmable message is effected by sending a matching Reset message and otherwise ignoring it. The Acknowledgement message MUST echo the Message ID of the Confirmable message, and MUST carry a response or be empty (see Section 5.2.1 and Section 5.2.2). The Reset message MUST echo the Message ID of the confirmable message, and MUST be empty. Rejecting an Acknowledgement or Reset message is effected by silently ignoring it.

The sender retransmits the Confirmable message at exponentially increasing intervals, until it receives an acknowledgement (or Reset message), or runs out of attempts.

Retransmission is controlled by two things that a CoAP endpoint MUST keep track of for each Confirmable message it sends while waiting for an acknowledgement (or reset): a timeout and a retransmission counter. For a new Confirmable message, the initial timeout is set to a random number between ACK\_TIMEOUT and (ACK\_TIMEOUT \* ACK\_RANDOM\_FACTOR) (see Section 4.8), and the retransmission counter is set to 0. When the timeout is triggered and the retransmission counter is less than MAX\_RETRANSMIT, the message is retransmitted, the retransmission counter is incremented, and the timeout is doubled. If the retransmission counter reaches MAX\_RETRANSMIT on a timeout, or if the endpoint receives a Reset message, then the attempt to transmit the message is canceled and the application process informed of failure. On the other hand, if the endpoint receives an acknowledgement message in time, transmission is considered successful.

A CoAP endpoint that sent a Confirmable message MAY give up in attempting to obtain an ACK even before the MAX\_RETRANSMIT counter value is reached: E.g., the application has canceled the request as it no longer needs a response, or there is some other indication that

the CON message did arrive. In particular, a CoAP request message may have elicited a separate response, in which case it is clear to the requester that only the ACK was lost and a retransmission of the request would serve no purpose. However, a responder **MUST NOT** in turn rely on this cross-layer behavior from a requester, i.e. it **SHOULD** retain the state to create the ACK for the request, if needed, even if a Confirmable response was already acknowledged by the requester.

#### 4.3. Messages Transmitted Without Reliability

Some messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual success is sufficient.

As a more lightweight alternative, a message can be transmitted less reliably by marking the message as Non-confirmable. A Non-confirmable message always carries either a request or response and **MUST NOT** be empty. A Non-confirmable message **MUST NOT** be acknowledged by the recipient. If a recipient lacks context to process the message properly (including the case where the message is empty or has a message format error), it **MUST** reject the message; rejecting a Non-Confirmable message **MAY** involve sending a matching Reset message, and apart from the Reset message the rejected message **MUST** be silently ignored.

At the CoAP level, there is no way for the sender to detect if a Non-confirmable message was received or not. A sender **MAY** choose to transmit multiple copies of a Non-confirmable message within `MAX_TRANSMIT_SPAN`, or the network may duplicate the message in transit. To enable the receiver to act only once on the message, Non-confirmable messages specify a Message ID as well. (This Message ID is drawn from the same number space as the Message IDs for Confirmable messages.)

#### 4.4. Message Correlation

An Acknowledgement or Reset message is related to a Confirmable message or Non-confirmable message by means of a Message ID along with additional address information of the corresponding endpoint. The Message ID is a 16-bit unsigned integer that is generated by the sender of a Confirmable or Non-confirmable message and included in the CoAP header. The Message ID **MUST** be echoed in the Acknowledgement or Reset message by the recipient.

The same Message ID **MUST NOT** be re-used (in communicating with the same endpoint) within the `EXCHANGE_LIFETIME` (Section 4.8.2).

Implementation Note: Several implementation strategies can be employed for generating Message IDs. In the simplest case a CoAP endpoint generates Message IDs by keeping a single Message ID variable, which is changed each time a new confirmable or non-confirmable message is sent regardless of the destination address or port. Endpoints dealing with large numbers of transactions could keep multiple Message ID variables, for example per prefix or destination address. The initial variable value should be randomized.

For an Acknowledgement or Reset message to match a Confirmable or Non-confirmable message, the Message ID and source endpoint of the Acknowledgement or Reset message MUST match the Message ID and destination endpoint of the Confirmable or Non-confirmable message.

#### 4.5. Message Deduplication

A recipient MUST be prepared to receive the same Confirmable message (as indicated by the Message ID and source endpoint) multiple times within the `EXCHANGE_LIFETIME` (Section 4.8.2), for example, when its Acknowledgement went missing or didn't reach the original sender before the first timeout. The recipient SHOULD acknowledge each duplicate copy of a Confirmable message using the same Acknowledgement or Reset message, but SHOULD process any request or response in the message only once. This rule MAY be relaxed in case the Confirmable message transports a request that is idempotent (see Section 5.1) or can be handled in an idempotent fashion. Examples for relaxed message deduplication:

- o A server MAY relax the requirement to answer all retransmissions of an idempotent request with the same response (Section 4.2), so that it does not have to maintain state for Message IDs. For example, an implementation might want to process duplicate transmissions of a GET, PUT or DELETE request as separate requests if the effort incurred by duplicate processing is less expensive than keeping track of previous responses would be.
- o A constrained server MAY even want to relax this requirement for certain non-idempotent requests if the application semantics make this trade-off favorable. For example, if the result of a POST request is just the creation of some short-lived state at the server, it may be less expensive to incur this effort multiple times for a request than keeping track of whether a previous transmission of the same request already was processed.

A recipient MUST be prepared to receive the same Non-confirmable message (as indicated by the Message ID and source endpoint) multiple times within `NON_LIFETIME` (Section 4.8.2). As a general rule that



MAY be relaxed based on the specific semantics of a message, the recipient SHOULD silently ignore any duplicated Non-confirmable message, and SHOULD process any request or response in the message only once.

#### 4.6. Message Size

While specific link layers make it beneficial to keep CoAP messages small enough to fit into their link layer packets (see Section 1), this is a matter of implementation quality. The CoAP specification itself provides only an upper bound to the message size. Messages larger than an IP fragment result in undesired packet fragmentation. A CoAP message, appropriately encapsulated, SHOULD fit within a single IP packet (i.e., avoid IP fragmentation) and (by fitting into one UDP payload) obviously MUST fit within a single IP datagram. If the Path MTU is not known for a destination, an IP MTU of 1280 bytes SHOULD be assumed; if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size.

Implementation Note: CoAP's choice of message size parameters works well with IPv6 and with most of today's IPv4 paths. (However, with IPv4, it is harder to absolutely ensure that there is no IP fragmentation. If IPv4 support on unusual networks is a consideration, implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes; worse, the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload. Implementations extremely focused on this problem set might also set the IPv4 DF bit and perform some form of path MTU discovery; this should generally be unnecessary in most realistic use cases for CoAP, however.) A more important kind of fragmentation in many constrained networks is that on the adaptation layer (e.g., 6LoWPAN L2 packets are limited to 127 bytes including various overheads); this may motivate implementations to be frugal in their packet sizes and to move to block-wise transfers [I-D.ietf-core-block] when approaching three-digit message sizes.

Message sizes are also of considerable importance to implementations on constrained nodes. Many implementations will need to allocate a buffer for incoming messages. If an implementation is too constrained to allow for allocating the above-mentioned upper bound, it could apply the following implementation strategy: Implementations receiving a datagram into a buffer that is too small are usually able to determine if the trailing portion of a datagram was discarded and to retrieve the initial portion. So, if not all of the payload, at least the CoAP

header and options are likely to fit within the buffer. A server can thus fully interpret a request and return a 4.13 (Request Entity Too Large) response code if the payload was truncated. A client sending an idempotent request and receiving a response larger than would fit in the buffer can repeat the request with a suitable value for the Block Option [I-D.ietf-core-block].

#### 4.7. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.2.

In order not to cause congestion, Clients (including proxies) MUST strictly limit the number of simultaneous outstanding interactions that they maintain to a given server (including proxies) to NSTART. An outstanding interaction is either a CON for which an ACK has not yet been received but is still expected (message layer) or a request for which neither a response nor an Acknowledgment message has yet been received but is still expected (which may both occur at the same time, counting as one outstanding interaction). The default value of NSTART for this specification is 1.

Further congestion control optimizations and considerations are expected in the future, which may for example provide automatic initialization of the CoAP transmission parameters defined in Section 4.8, and thus may allow a value for NSTART greater than one.

A client stops expecting a response to a Confirmable request for which no acknowledgment message was received, after EXCHANGE\_LIFETIME. The specific algorithm by which a client stops to "expect" a response to a Confirmable request that was acknowledged, or to a Non-confirmable request, is not defined. Unless this is modified by additional congestion control optimizations, it MUST be chosen in such a way that an endpoint does not exceed an average data rate of PROBING\_RATE in sending to another endpoint that does not respond.

Note: CoAP places the onus of congestion control mostly on the clients. However, clients may malfunction or actually be attackers, e.g. to perform amplification attacks (Section 11.3). To limit the damage (to the network and to its own energy resources), a server SHOULD implement some rate limiting for its response transmission based on reasonable assumptions about application requirements. This is most helpful if the rate limit can be made effective for the misbehaving endpoints, only.

#### 4.8. Transmission Parameters

Message transmission is controlled by the following parameters:

name	default value
ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1
DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 Byte/second

##### 4.8.1. Changing The Parameters

The values for ACK\_TIMEOUT, ACK\_RANDOM\_FACTOR, MAX\_RETRANSMIT, NSTART, DEFAULT\_LEISURE, and PROBING\_RATE may be configured to values specific to the application environment (including dynamically adjusted values), however the configuration method is out of scope of this document. It is recommended that an application environment use consistent values for these parameters.

The transmission parameters have been chosen to achieve a behavior in the presence of congestion that is safe in the Internet. If a configuration desires to use different values, the onus is on the configuration to ensure these congestion control properties are not violated. In particular, a decrease of ACK\_TIMEOUT below 1 second would violate the guidelines of [RFC5405].

([I-D.allman-tcpm-rto-consider] provides some additional background.) CoAP was designed to enable implementations that do not maintain round-trip-time (RTT) measurements. However, where it is desired to decrease the ACK\_TIMEOUT significantly or increase NSTART, this can only be done safely when maintaining such measurements. Configurations MUST NOT decrease ACK\_TIMEOUT or increase NSTART without using mechanisms that ensure congestion control safety, either defined in the configuration or in future standards documents.

ACK\_RANDOM\_FACTOR MUST NOT be decreased below 1.0, and it SHOULD have a value that is sufficiently different from 1.0 to provide some protection from synchronization effects.

MAX\_RETRANSMIT can be freely adjusted, but a too small value will reduce the probability that a confirmable message is actually received, while a larger value than given here will require further adjustments in the time values (see discussion below).

If the choice of transmission parameters leads to an increase of derived time values (see below), the configuration mechanism MUST ensure the adjusted value is also available to all the endpoints in communicating with which these adjusted values are to be used.

#### 4.8.2. Time Values derived from Transmission Parameters

The combination of `ACK_TIMEOUT`, `ACK_RANDOM_FACTOR` and `MAX_RETRANSMIT` influences the timing of retransmissions, which in turn influences how long certain information items need to be kept by an implementation. To be able to unambiguously reference these derived time values, we give them names as follows:

- o `MAX_TRANSMIT_SPAN` is the maximum time from the first transmission of a confirmable message to its last retransmission. For the default transmission parameters, the value is  $(2+4+8+16)*1.5 = 45$  seconds, or more generally:

$$\text{ACK\_TIMEOUT} * (2 ** \text{MAX\_RETRANSMIT} - 1) * \text{ACK\_RANDOM\_FACTOR}$$

- o `MAX_TRANSMIT_WAIT` is the maximum time from the first transmission of a confirmable message to the time when the sender gives up on receiving an acknowledgement or reset. For the default transmission parameters, the value is  $(2+4+8+16+32)*1.5 = 93$  seconds, or more generally:

$$\text{ACK\_TIMEOUT} * (2 ** (\text{MAX\_RETRANSMIT} + 1) - 1) * \text{ACK\_RANDOM\_FACTOR}$$

In addition, some assumptions need to be made on the characteristics of the network and the nodes.

- o `MAX_LATENCY` is the maximum time a datagram is expected to take from the start of its transmission to the completion of its reception. This constant is related to the MSL (Maximum Segment Lifetime) of [RFC0793], which is "arbitrarily defined to be 2 minutes" ([RFC0793] glossary, page 81). Note that this is not necessarily smaller than `MAX_TRANSMIT_WAIT`, as `MAX_LATENCY` is not intended to describe a situation when the protocol works well, but the worst case situation against which the protocol has to guard. We, also arbitrarily, define `MAX_LATENCY` to be 100 seconds. Apart from being reasonably realistic for the bulk of configurations as well as close to the historic choice for TCP, this value also allows message ID lifetime timers to be represented in 8 bits (when measured in seconds). In these calculations, there is no assumption that the direction of the transmission is irrelevant (i.e. that the network is symmetric), just that the same value can reasonably be used as a maximum value for both directions. If

that is not the case, the following calculations become only slightly more complex.

- o PROCESSING\_DELAY is the time a node takes to turn around a confirmable message into an acknowledgement. We assume the node will attempt to send an ACK before having the sender time out, so as a conservative assumption we set it equal to ACK\_TIMEOUT.
- o MAX\_RTT is the maximum round-trip time, or:

$$2 * MAX\_LATENCY + PROCESSING\_DELAY$$

From these values, we can derive the following values relevant to the protocol operation:

- o EXCHANGE\_LIFETIME is the time from starting to send a confirmable message to the time when an acknowledgement is no longer expected, i.e. message layer information about the message exchange can be purged. EXCHANGE\_LIFETIME includes a MAX\_TRANSMIT\_SPAN, a MAX\_LATENCY forward, PROCESSING\_DELAY, and a MAX\_LATENCY for the way back. Note that there is no need to consider MAX\_TRANSMIT\_WAIT if the configuration is chosen such that the last waiting period ( $ACK\_TIMEOUT * (2 ** MAX\_RETRANSMIT)$ ) or the difference between MAX\_TRANSMIT\_SPAN and MAX\_TRANSMIT\_WAIT is less than MAX\_LATENCY -- which is a likely choice, as MAX\_LATENCY is a worst case value unlikely to be met in the real world. In this case, EXCHANGE\_LIFETIME simplifies to:

$$(ACK\_TIMEOUT * (2 ** MAX\_RETRANSMIT - 1) * ACK\_RANDOM\_FACTOR) + (2 * MAX\_LATENCY) + PROCESSING\_DELAY$$

or 248 seconds with the default transmission parameters.

- o NON\_LIFETIME is the time from sending a non-confirmable message to the time its message-ID can be safely reused. If multiple transmission of a NON message is not used, its value is MAX\_LATENCY, or 100 seconds. However, a CoAP sender might send a NON message multiple times, in particular for multicast applications. While the period of re-use is not bounded by the specification, an expectation of reliable detection of duplication at the receiver is in the timescales of MAX\_TRANSMIT\_SPAN. Therefore, for this purpose, it is safer to use the value:

$$MAX\_TRANSMIT\_SPAN + MAX\_LATENCY$$

or 145 seconds with the default transmission parameters; however, an implementation that just wants to use a single timeout value for retiring message-IDs can safely use the larger value for

EXCHANGE\_LIFETIME.

## 5. Request/Response Semantics

CoAP operates under a similar request/response model as HTTP: a CoAP endpoint in the role of a "client" sends one or more CoAP requests to a "server", which services the requests by sending CoAP responses. Unlike HTTP, requests and responses are not sent over a previously established connection, but exchanged asynchronously over CoAP messages.

### 5.1. Requests

A CoAP request consists of the method to be applied to the resource, the identifier of the resource, a payload and Internet media type (if any), and optional meta-data about the request.

CoAP supports the basic methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP. They have the same properties of safe (only retrieval) and idempotent (you can invoke it multiple times with the same effects) as HTTP (see Section 9.1 of [RFC2616]). The GET method is safe, therefore it MUST NOT take any other action on a resource other than retrieval. The GET, PUT and DELETE methods MUST be performed in such a way that they are idempotent. POST is not idempotent, because its effect is determined by the origin server and dependent on the target resource; it usually results in a new resource being created or the target resource being updated.

A request is initiated by setting the Code field in the CoAP header of a Confirmable or a Non-confirmable message to a Method Code and including request information.

The methods used in requests are described in detail in Section 5.8.

### 5.2. Responses

After receiving and interpreting a request, a server responds with a CoAP response, which is matched to the request by means of a client-generated token.

A response is identified by the Code field in the CoAP header being set to a Response Code. Similar to the HTTP Status Code, the CoAP Response Code indicates the result of the attempt to understand and satisfy the request. These codes are fully defined in Section 5.9. The Response Code numbers to be set in the Code field of the CoAP header are maintained in the CoAP Response Code Registry (Section 12.1.2).

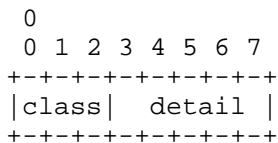


Figure 9: Structure of a Response Code

The upper three bits of the 8-bit Response Code number define the class of response. The lower five bits do not have any categorization role; they give additional detail to the overall class (Figure 9). There are 3 classes:

- 2 - Success: The request was successfully received, understood, and accepted.
- 4 - Client Error: The request contains bad syntax or cannot be fulfilled.
- 5 - Server Error: The server failed to fulfill an apparently valid request.

The response codes are designed to be extensible: Response Codes in the Client Error and Server Error class that are unrecognized by an endpoint MUST be treated as being equivalent to the generic Response Code of that class (4.00 and 5.00, respectively). However, there is no generic Response Code indicating success, so a Response Code in the Success class that is unrecognized by an endpoint can only be used to determine that the request was successful without any further details.

As a human readable notation for specifications and protocol diagnostics, the numeric value of a response code is indicated by giving the upper three bits in decimal, followed by a dot and then the lower five bits in a two-digit decimal. E.g., "Not Found" is written as 4.04 -- indicating a value of hexadecimal 0x84 or decimal 132. In other words, the dot "." functions as a short-cut for "\*32+".

The possible response codes are described in detail in Section 5.9.

Responses can be sent in multiple ways, which are defined below.

#### 5.2.1. Piggy-backed

In the most basic case, the response is carried directly in the Acknowledgement message that acknowledges the request (which requires that the request was carried in a Confirmable message). This is

called a "Piggy-backed" Response.

The response is returned in the Acknowledgement message independent of whether the response indicates success or failure. In effect, the response is piggy-backed on the Acknowledgement message, so no separate message is required to both acknowledge that the request was received and return the response.

Implementation Note: The protocol leaves the decision whether to piggy-back a response or not (i.e., send a separate response) to the server. The client **MUST** be prepared to receive either. On the quality of implementation level, there is a strong expectation that servers will implement code to piggy-back whenever possible -- saving resources in the network and both at the client and at the server.

#### 5.2.2. Separate

It may not be possible to return a piggy-backed response in all cases. For example, a server might need longer to obtain the representation of the resource requested than it can wait sending back the Acknowledgement message, without risking the client to repeatedly retransmit the request message. Responses to requests carried in a Non-Confirmable message are always sent separately (as there is no Acknowledgement message).

The server maybe initiates the attempt to obtain the resource representation and times out an acknowledgement timer, or it immediately sends an acknowledgement knowing in advance that there will be no piggy-backed response. The acknowledgement effectively is a promise that the request will be acted upon.

When the server finally has obtained the resource representation, it sends the response. When it is desired that this message is not lost, it is sent as a Confirmable message from the server to the client and answered by the client with an Acknowledgement, echoing the new Message ID chosen by the server. (It may also be sent as a Non-Confirmable message; see Section 5.2.3.)

Implementation Notes: Note that, as the underlying datagram transport may not be sequence-preserving, the Confirmable message carrying the response may actually arrive before or after the acknowledgement message for the request. Note also that, while the CoAP protocol itself does not make any specific demands here, there is an expectation that the response will come within a time frame that is reasonable from an application point of view; as there is no underlying transport protocol that could be instructed to run a keep-alive mechanism, the requester **MAY** want to set up a



timeout that is unrelated to CoAP's retransmission timers in case the server is destroyed or otherwise unable to send the response.)

An exchange is separate by definition when the Acknowledgement to the Confirmable request is an empty message. The Acknowledgement to the Confirmable response MUST also be an empty message, i.e. one that carries neither a request nor a response. However, a server MUST stop retransmitting its response on any matching Acknowledgement (silently ignoring any response code or payload) or Reset message.

#### 5.2.3. Non-Confirmable

If the request message is Non-confirmable, then the response SHOULD be returned in a Non-confirmable message as well. However, an endpoint MUST be prepared to receive a Non-confirmable response (preceded or followed by an empty acknowledgement message) in reply to a Confirmable request, or a Confirmable response in reply to a Non-confirmable request.

#### 5.3. Request/Response Matching

Regardless of how a response is sent, it is matched to the request by means of a token that is included by the client in the request, along with additional address information of the corresponding endpoint.

##### 5.3.1. Token

The Token is used to match a response with a request. The token value is a sequence of 0 to 8 bytes. (Note that every message carries a token, even if it is of zero length.) Every request carries a client-generated token, which the server MUST echo in any resulting response without modification.

A token is intended for use as a client-local identifier for differentiating between concurrent requests (see Section 5.3); it could have been called a "request ID".

The client SHOULD generate tokens in such a way that tokens currently in use for a given source/destination endpoint pair are unique. (Note that a client implementation can use the same token for any request if it uses a different endpoint each time, e.g. a different source port number.) An empty token value is appropriate e.g. when no other tokens are in use to a destination, or when requests are made serially per destination and receive piggy-backed responses. There are however multiple possible implementation strategies to fulfill this.

An endpoint receiving a token it did not generate MUST treat it as

opaque and make no assumptions about its format.

#### 5.3.2. Request/Response Matching Rules

The exact rules for matching a response to a request are as follows:

1. The source endpoint of the response MUST be the same as the destination endpoint of the original request.
2. In a piggy-backed response, both the Message ID of the Confirmable request and the Acknowledgement, and the token of the response and original request MUST match. In a separate response, just the token of the response and original request MUST match.

In case a message carrying a response is unexpected (i.e. the client is not waiting for a response at the endpoint addressed and/or with the given token), the response is rejected (Section 4.2, Section 4.3).

Implementation Note: A client that receives a response in a CON message may want to clean up the message state right after sending the ACK. If that ACK is lost and the server retransmits the CON, the client may no longer have any state to correlate this response to, making the retransmission an unexpected message; the client may send a Reset message so it does not receive any more retransmissions. This behavior is normal and not an indication of an error. (Clients that are not aggressively optimized in their state memory usage will still have message state that will identify the second CON as a retransmission. Clients that actually expect more messages from the server [I-D.ietf-core-observe] will have to keep state in any case.)

#### 5.4. Options

Both requests and responses may include a list of one or more options. For example, the URI in a request is transported in several options, and meta-data that would be carried in an HTTP header in HTTP is supplied as options as well.

CoAP defines a single set of options that are used in both requests and responses:

- o Content-Format
- o ETag

- o Location-Path
- o Location-Query
- o Max-Age
- o Proxy-Uri
- o Proxy-Scheme
- o Uri-Host
- o Uri-Path
- o Uri-Port
- o Uri-Query
- o Accept
- o If-Match
- o If-None-Match

The semantics of these options along with their properties are defined in detail in Section 5.10.

Not all options are defined for use with all methods and response codes. The possible options for methods and response codes are defined in Section 5.8 and Section 5.9 respectively. In case an option is not defined for a method or response code, it **MUST NOT** be included by a sender and **MUST** be treated like an unrecognized option by a recipient.

#### 5.4.1. Critical/Elective

Options fall into one of two classes: "critical" or "elective". The difference between these is how an option unrecognized by an endpoint is handled:

- o Upon reception, unrecognized options of class "elective" **MUST** be silently ignored.
- o Unrecognized options of class "critical" that occur in a confirmable request **MUST** cause the return of a 4.02 (Bad Option) response. This response **SHOULD** include a diagnostic payload describing the unrecognized option(s) (see Section 5.5.2).

- o Unrecognized options of class "critical" that occur in a confirmable response, or piggy-backed in an acknowledgement, MUST cause the response to be rejected (Section 4.2).
- o Unrecognized options of class "critical" that occur in a non-confirmable message MUST cause the message to be rejected (Section 4.3).

Note that, whether critical or elective, an option is never "mandatory" (it is always optional): These rules are defined in order to enable implementations to stop processing options they do not understand or implement.

Critical/Elective rules apply to non-proxying endpoints. A proxy processes options based on Unsafe/Safe classes as defined in Section 5.7.

#### 5.4.2. Proxy Unsafe/Safe and Cache-Key

In addition to an option being marked as Critical or Elective, options are also classified based on how a proxy is to deal with the option if it does not recognize it. For this purpose, an option can either be considered Unsafe to Forward (Unsafe is set) or Safe to Forward (Unsafe is clear).

In addition, for options that are marked Safe to Forward, the option indicates whether it is intended to be part of the Cache-Key in a request (NoCacheKey is not all set) or not (NoCacheKey is set).

Note: The Cache-Key indication is relevant only for proxies that do not implement the given option as a request option and instead rely on the Safe/Unsafe indication only. E.g., for ETag, actually using the request option as a cache key is grossly inefficient, but it is the best thing one can do if ETag is not implemented by a proxy, as the response is going to differ based on the presence of the request option. A more useful proxy that does implement the ETag request option is not using ETag as a cache key.

Proxy behavior with regard to these classes is defined in Section 5.7.

#### 5.4.3. Length

Option values are defined to have a specific length, often in the form of an upper and lower bound. If the length of an option value in a request is outside the defined range, that option MUST be treated like an unrecognized option (see Section 5.4.1).

#### 5.4.4. Default Values

Options may be defined to have a default value. If the value of option is intended to be this default value, the option SHOULD NOT be included in the message. If the option is not present, the default value MUST be assumed.

Where a critical option has a default value, this is chosen in such a way that the absence of the option in a message can be processed properly both by implementations unaware of the critical option and by implementations that interpret this absence as the presence of the default value for the option.

#### 5.4.5. Repeatable Options

The definition of an option MAY specify the option to be repeatable. An option that is repeatable MAY be included one or more times in a message. An option that is not repeatable MUST NOT be included more than once in a message.

If a message includes an option with more occurrences than the option is defined for, the additional option occurrences MUST be treated like an unrecognized option (see Section 5.4.1).

#### 5.4.6. Option Numbers

An Option is identified by an option number, which also provides some additional semantics information: e.g., odd numbers indicate a critical option, while even numbers indicate an elective option. Note that this is not just a convention, it is a feature of the protocol: Whether an option is elective or critical is entirely determined by whether its option number is even or odd.

More generally speaking, an Option number is constructed with a bit mask to indicate if an option is Critical/Elective, Unsafe/Safe and in the case of Safe, also a Cache-Key indication as shown by the following figure. When bit 7 (the least significant bit) is 1, an option is Critical (and likewise Elective when 0). When bit 6 is 1, an option is Unsafe (and likewise Safe when 0). When bit 6 is 0, i.e., the option is not Unsafe, it is not a Cache-Key (NoCacheKey) if and only if bits 3-5 are all set to 1; all other bit combinations mean that it indeed is a Cache-Key. These classes of options are explained in the next sections.

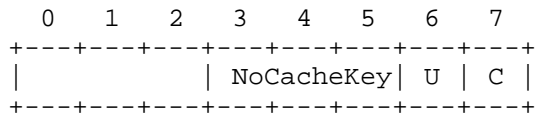


Figure 10: Option Number Mask

An endpoint may use an equivalent of the C code in Figure 11 to derive the characteristics of an option number "onum".

```
Critical = (onum & 1);
UnSafe = (onum & 2);
NoCacheKey = ((onum & 0x1e) == 0x1c);
```

Figure 11: Determining Characteristics from an Option Number

The option numbers for the options defined in this document are listed in the CoAP Option Number Registry (Section 12.2).

### 5.5. Payloads and Representations

Both requests and responses may include a payload, depending on the method or response code respectively. If a method or response code is not defined to have a payload, then a sender MUST NOT include one, and a recipient MUST ignore it.

#### 5.5.1. Representation

The payload of requests or of responses indicating success is typically a representation of a resource or the result of the requested action. Its format is specified by the Internet media type and content coding given by the Content-Format Option. In the absence of this option, no default value is assumed and the format will need to be inferred by the application (e.g., from the application context or by "sniffing" the payload).

Implementation Note: On a quality of implementation level, there is a strong expectation that a Content-Format indication will be provided with resource representations whenever possible. This is not a "SHOULD"-level requirement solely because it is not a protocol requirement, and it also would be difficult to outline exactly in what cases this expectation can be violated.

For responses indicating a client or server error, the payload is considered a representation of the result of the requested action only if a Content-Format Option is given. In the absence of this option, the payload is a Diagnostic Payload ({{diagnostic-message-payload}}).

#### 5.5.2. Diagnostic Payload

If no Content-Format option is given, the payload of responses indicating a client or server error is a brief human-readable diagnostic message, explaining the error situation. This diagnostic message **MUST** be encoded using UTF-8 [RFC3629], more specifically using Net-Unicode form [RFC5198].

The message is similar to the Reason-Phrase on an HTTP status line. It is not intended for end-users but for software engineers that during debugging need to interpret it in the context of the present, English-language specification; therefore no mechanism for language tagging is needed or provided. In contrast to what is usual in HTTP, the payload **SHOULD** be empty if there is no additional information beyond the response code.

#### 5.5.3. Selected Representation

Not all responses carry a payload that provides a representation of the resource addressed by the request. It is, however, sometimes useful to be able to refer to such a representation in relation to a response, independent of whether it actually was enclosed.

We use the term "selected representation" to refer to the current representation of a target resource that would have been selected in a successful response if the corresponding request had used the method GET and excluded any conditional request options (Section 5.10.8).

Certain response options provide metadata about the selected representation, which might differ from the representation included in the message for responses to some state-changing methods. Of the response options defined in this specification, only the ETag response option (Section 5.10.6) is defined as selected representation metadata.

#### 5.5.4. Content Negotiation

A server may be able to supply a representation for a resource in one of multiple representation formats. Without further information from the client, it will provide the representation in the format it prefers.

By using one or more instances of the Accept Option (Section 5.10.4) in a request, the client can indicate which content-formats it prefers to receive and provide a preference ranking between these content-formats.

## 5.6. Caching

CoAP endpoints MAY cache responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests.

The goal of caching in CoAP is to reuse a prior response message to satisfy a current request. In some cases, a stored response can be reused without the need for a network request, reducing latency and network round-trips; a "freshness" mechanism is used for this purpose (see Section 5.6.1). Even when a new request is required, it is often possible to reuse the payload of a prior response to satisfy the request, thereby reducing network bandwidth usage; a "validation" mechanism is used for this purpose (see Section 5.6.2).

Unlike HTTP, the cacheability of CoAP responses does not depend on the request method, but the Response Code. The cacheability of each Response Code is defined along the Response Code definitions in Section 5.9. Response Codes that indicate success and are unrecognized by an endpoint MUST NOT be cached.

For a presented request, a CoAP endpoint MUST NOT use a stored response, unless:

- o the presented request method and that used to obtain the stored response match,
- o all options match between those in the presented request and those of the request used to obtain the stored response (which includes the request URI), except that there is no need for a match of any request options marked as NoCacheKey (Section 5.4) or recognized by the Cache and fully interpreted with respect to its specified cache behavior (such as the ETag request option, Section 5.10.6, see also Section 5.4.2), and
- o the stored response is either fresh or successfully validated as defined below.

### 5.6.1. Freshness Model

When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.

The mechanism for determining freshness is for an origin server to provide an explicit expiration time in the future, using the Max-Age Option (see Section 5.10.5). The Max-Age Option indicates that the response is to be considered not fresh after its age is greater than



the specified number of seconds.

The Max-Age Option defaults to a value of 60. Thus, if it is not present in a cacheable response, then the response is considered not fresh after its age is greater than 60 seconds. If an origin server wishes to prevent caching, it **MUST** explicitly include a Max-Age Option with a value of zero seconds.

If a client has a fresh stored response and makes a new request matching the request for that stored response, the new response invalidates the old response.

#### 5.6.2. Validation Model

When an endpoint has one or more stored responses for a GET request, but cannot use any of them (e.g., because they are not fresh), it can use the ETag Option (Section 5.10.6) in the GET request to give the origin server an opportunity to both select a stored response to be used, and to update its freshness. This process is known as "validating" or "revalidating" the stored response.

When sending such a request, the endpoint **SHOULD** add an ETag Option specifying the entity-tag of each stored response that is applicable.

A 2.03 (Valid) response indicates the stored response identified by the entity-tag given in the response's ETag Option can be reused, after updating its freshness with the value of the Max-Age Option that is included (explicitly, or implicitly as a default value) with the response (see Section 5.9.1.3).

Any other response code indicates that none of the stored responses nominated in the request is suitable. Instead, the response **SHOULD** be used to satisfy the request and **MAY** replace the stored response.

#### 5.7. Proxying

A proxy is a CoAP endpoint that can be tasked by CoAP clients to perform requests on their behalf. This may be useful, for example, when the request could otherwise not be made, or to service the response from a cache in order to reduce response time and network bandwidth or energy consumption.

In an overall architecture for a Constrained RESTful Environment, proxies can serve quite different purposes. Proxies can be explicitly selected by clients, a role that we term "forward-proxy". Proxies can also be inserted to stand in for origin servers, a role that we term "reverse-proxy". Orthogonal to this distinction, a proxy can map from a CoAP request to a CoAP request (CoAP-to-CoAP

proxy) or translate from or to a different protocol ("cross-proxy"). Full definitions of these terms are provided in Section 1.2.

Notes: The terminology in this specification has been selected to be culturally compatible with the terminology used in the wider Web application environments, without necessarily matching it in every detail (which may not even be relevant to Constrained RESTful Environments). Not too much semantics should be ascribed to the components of the terms (such as "forward", "reverse", or "cross").

HTTP proxies, besides acting as HTTP proxies, often offer a transport protocol proxying function ("CONNECT") to enable end-to-end transport layer security through the proxy. No such function is defined for CoAP-to-CoAP proxies in this specification, as forwarding of UDP packets is unlikely to be of much value in Constrained RESTful environments. See also Section 10.2.7 for the cross-proxy case.

#### 5.7.1. Proxy Operation

A proxy generally needs a way to determine potential request parameters for a request to a destination based on the request it received. This way is fully specified for a forward-proxy, but may depend on the specific configuration for a reverse-proxy. In particular, the client of a reverse-proxy generally does not indicate a locator for the destination, necessitating some form of namespace translation in the reverse-proxy. However, some aspects of the operation of proxies are common to all its forms.

If a proxy does not employ a cache, then it simply forwards the translated request to the determined destination. Otherwise, if it does employ a cache but does not have a stored response that matches the translated request and is considered fresh, then it needs to refresh its cache according to Section 5.6. For options in the request that the proxy recognizes, it knows whether the option is intended to act as part of the key used in looking up the cached value or not. E.g., since requests for different Uri-Path values address different resources, Uri-Path values are always parts of the cache key, while, e.g., Token values are never part of the cache key. For options that the proxy does not recognize but that are marked Safe in the option number, the option also indicates whether it is to be included in the cache key (NoCacheKey is not all set) or not (NoCacheKey is all set). (Options that are unrecognized and marked Unsafe lead to 4.02 Bad Option.)

If the request to the destination times out, then a 5.04 (Gateway Timeout) response MUST be returned. If the request to the

destination returns a response that cannot be processed by the proxy (e.g, due to unrecognized critical options, message format errors), then a 5.02 (Bad Gateway) response MUST be returned. Otherwise, the proxy returns the response to the client.

If a response is generated out of a cache, it MUST be generated with a Max-Age Option that does not extend the max-age originally set by the server, considering the time the resource representation spent in the cache. E.g., the Max-Age Option could be adjusted by the proxy for each response using the formula:

$$\text{proxy-max-age} = \text{original-max-age} - \text{cache-age}$$

For example if a request is made to a proxied resource that was refreshed 20 seconds ago and had an original Max-Age of 60 seconds, then that resource's proxied max-age is now 40 seconds. Considering potential network delays on the way from the origin server, a proxy SHOULD be conservative in the max-age values offered.

All options present in a proxy request MUST be processed at the proxy. Unsafe options in a request that are not recognized by the proxy MUST lead to a 4.02 (Bad Option) response being returned by the proxy. A CoAP-to-CoAP proxy MUST forward to the origin server all Safe options that it does not recognize. Similarly, Unsafe options in a response that are not recognized by the CoAP-to-CoAP proxy server MUST lead to a 5.02 (Bad Gateway) response. Again, Safe options that are not recognized MUST be forwarded.

Additional considerations for cross-protocol proxying between CoAP and HTTP are discussed in Section 10.

#### 5.7.2. Forward-Proxies

CoAP distinguishes between requests made (as if) to an origin server and a request made through a forward-proxy. CoAP requests to a forward-proxy are made as normal confirmable or non-confirmable requests to the forward-proxy endpoint, but specify the request URI in a different way: The request URI in a proxy request is specified as a string in the Proxy-Uri Option (see Section 5.10.2), while the request URI in a request to an origin server is split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options (see Section 5.10.1); alternatively the URI in a proxy request can be assembled from a Proxy-Scheme option and the split options mentioned.

When a proxy request is made to an endpoint and the endpoint is unwilling or unable to act as proxy for the request URI, it MUST return a 5.05 (Proxying Not Supported) response. If the authority (host and port) is recognized as identifying the proxy endpoint

itself (see Section 5.10.2), then the request MUST be treated as a local (non-proxied) request.

Unless a proxy is configured to forward the proxy request to another proxy, it MUST translate the request as follows: The scheme of the request URI defines the outgoing protocol and its details (e.g., CoAP is used over UDP for the "coap" scheme and over DTLS for the "coaps" scheme.) For a CoAP-to-CoAP proxy, the origin server's IP address and port are determined by the authority component of the request URI, and the request URI is decoded and split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options. This consumes the Proxy-Uri or Proxy-Scheme option, which is therefore not forwarded to the origin server.

#### 5.7.3. Reverse-Proxies

Reverse-proxies do not make use of the Proxy-Uri or Proxy-Scheme options, but need to determine the destination (next hop) of a request from information in the request and information in their configuration. E.g., a reverse-proxy might offer various resources the existence of which it has learned through resource discovery as if they were its own resources. The reverse-proxy is free to build a namespace for the URIs that identify these resources. A reverse-proxy may also build a namespace that gives the client more control over where the request goes, e.g. by embedding host identifiers and port numbers into the URI path of the resources offered.

In processing the response, a reverse-proxy has to be careful about namespacing the ETag option. In many cases, it can be forwarded unchanged. If the mapping from a resource offered by the reverse-proxy to resources offered by its various origin servers is not unique, the reverse-proxy may need to generate a new ETag, making sure the semantics of this option are properly preserved.

#### 5.8. Method Definitions

In this section each method is defined along with its behavior. A request with an unrecognized or unsupported Method Code MUST generate a 4.05 (Method Not Allowed) piggy-backed response.

##### 5.8.1. GET

The GET method retrieves a representation for the information that currently corresponds to the resource identified by the request URI. If the request includes one or more Accept Options, they indicate the preferred content-format of a response. If the request includes an ETag Option, the GET method requests that ETag be validated and that the representation be transferred only if validation failed. Upon

success a 2.05 (Content) or 2.03 (Valid) response code SHOULD be present in the response.

The GET method is safe and idempotent.

#### 5.8.2. POST

The POST method requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and dependent on the target resource. It usually results in a new resource being created or the target resource being updated.

If a resource has been created on the server, the response returned by the server SHOULD have a 2.01 (Created) response code and SHOULD include the URI of the new resource in a sequence of one or more Location-Path and/or Location-Query Options (Section 5.10.7). If the POST succeeds but does not result in a new resource being created on the server, the response SHOULD have a 2.04 (Changed) response code. If the POST succeeds and results in the target resource being deleted, the response SHOULD have a 2.02 (Deleted) response code.

POST is neither safe nor idempotent.

#### 5.8.3. PUT

The PUT method requests that the resource identified by the request URI be updated or created with the enclosed representation. The representation format is specified by the media type and content coding given in the Content-Format Option, if provided.

If a resource exists at the request URI the enclosed representation SHOULD be considered a modified version of that resource, and a 2.04 (Changed) response code SHOULD be returned. If no resource exists then the server MAY create a new resource with that URI, resulting in a 2.01 (Created) response code. If the resource could not be created or modified, then an appropriate error response code SHOULD be sent.

Further restrictions to a PUT can be made by including the If-Match (see Section 5.10.8.1) or If-None-Match (see Section 5.10.8.2) options in the request.

PUT is not safe, but is idempotent.

#### 5.8.4. DELETE

The DELETE method requests that the resource identified by the request URI be deleted. A 2.02 (Deleted) response code SHOULD be

used on success or in case the resource did not exist before the request.

DELETE is not safe, but is idempotent.

## 5.9. Response Code Definitions

Each response code is described below, including any options required in the response. Where appropriate, some of the codes will be specified in regards to related response codes in HTTP [RFC2616]; this does not mean that any such relationship modifies the HTTP mapping specified in Section 10.

### 5.9.1. Success 2.xx

This class of status code indicates that the clients request was successfully received, understood, and accepted.

#### 5.9.1.1. 2.01 Created

Like HTTP 201 "Created", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

If the response includes one or more Location-Path and/or Location-Query Options, the values of these options specify the location at which the resource was created. Otherwise, the resource was created at the request URI. A cache receiving this response MUST mark any stored response for the created resource as not fresh.

This response is not cacheable.

#### 5.9.1.2. 2.02 Deleted

Like HTTP 204 "No Content", but only used in response to DELETE requests. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache MUST mark any stored response for the deleted resource as not fresh.

#### 5.9.1.3. 2.03 Valid

Related to HTTP 304 "Not Modified", but only used to indicate that the response identified by the entity-tag identified by the included ETag Option is valid. Accordingly, the response MUST include an ETag Option, and MUST NOT include a payload.

When a cache that recognizes and processes the ETag response option receives a 2.03 (Valid) response, it MUST update the stored response with the value of the Max-Age Option included in the response (see Section 5.6.2).

#### 5.9.1.4. 2.04 Changed

Like HTTP 204 "No Content", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache MUST mark any stored response for the changed resource as not fresh.

#### 5.9.1.5. 2.05 Content

Like HTTP 200 "OK", but only used in response to GET requests.

The payload returned with the response is a representation of the target resource.

This response is cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1) and (if present) the ETag Option for validation (see Section 5.6.2).

#### 5.9.2. Client Error 4.xx

This class of response code is intended for cases in which the client seems to have erred. These response codes are applicable to any request method.

The server SHOULD include a diagnostic payload under the conditions detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

##### 5.9.2.1. 4.00 Bad Request

Like HTTP 400 "Bad Request".

##### 5.9.2.2. 4.01 Unauthorized

The client is not authorized to perform the requested action. The client SHOULD NOT repeat the request without previously improving its authentication status to the server. Which specific mechanism can be used for this is outside this document's scope; see also Section 9.

5.9.2.3.    4.02 Bad Option

The request could not be understood by the server due to one or more unrecognized or malformed options. The client SHOULD NOT repeat the request without modification.

5.9.2.4.    4.03 Forbidden

Like HTTP 403 "Forbidden".

5.9.2.5.    4.04 Not Found

Like HTTP 404 "Not Found".

5.9.2.6.    4.05 Method Not Allowed

Like HTTP 405 "Method Not Allowed", but with no parallel to the "Allow" header field.

5.9.2.7.    4.06 Not Acceptable

Like HTTP 406 "Not Acceptable", but with no response entity.

5.9.2.8.    4.12 Precondition Failed

Like HTTP 412 "Precondition Failed".

5.9.2.9.    4.13 Request Entity Too Large

Like HTTP 413 "Request Entity Too Large".

5.9.2.10.   4.15 Unsupported Content-Format

Like HTTP 415 "Unsupported Media Type".

5.9.3.    Server Error 5.xx

This class of response code indicates cases in which the server is aware that it has erred or is incapable of performing the request. These response codes are applicable to any request method.

The server SHOULD include a diagnostic payload under the conditions detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.



5.9.3.1.    5.00 Internal Server Error

Like HTTP 500 "Internal Server Error".

5.9.3.2.    5.01 Not Implemented

Like HTTP 501 "Not Implemented".

5.9.3.3.    5.02 Bad Gateway

Like HTTP 502 "Bad Gateway".

5.9.3.4.    5.03 Service Unavailable

Like HTTP 503 "Service Unavailable", but using the Max-Age Option in place of the "Retry-After" header field to indicate the number of seconds after which to retry.

5.9.3.5.    5.04 Gateway Timeout

Like HTTP 504 "Gateway Timeout".

5.9.3.6.    5.05 Proxying Not Supported

The server is unable or unwilling to act as a forward-proxy for the URI specified in the Proxy-Uri Option or using Proxy-Scheme (see Section 5.10.2).

5.10.    Option Definitions

The individual CoAP options are summarized in Table 1 and explained below.

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
16				x	Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Table 1: Options

#### 5.10.1. Uri-Host, Uri-Port, Uri-Path and Uri-Query

The Uri-Host, Uri-Port, Uri-Path and Uri-Query Options are used to specify the target resource of a request to a CoAP origin server. The options encode the different components of the request URI in a way that no percent-encoding is visible in the option values and that the full URI can be reconstructed at any involved endpoint. The syntax of CoAP URIs is defined in Section 6.

The steps for parsing URIs into options is defined in Section 6.4. These steps result in zero or more Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in a request, where each option holds the following values:

- o the Uri-Host Option specifies the Internet host of the resource being requested,
- o the Uri-Port Option specifies the transport layer port number of the resource,
- o each Uri-Path Option specifies one segment of the absolute path to the resource, and

- o each Uri-Query Option specifies one argument parameterizing the resource.

Note: Fragments ([RFC3986], Section 3.5) are not part of the request URI and thus will not be transmitted in a CoAP request.

The default value of the Uri-Host Option is the IP literal representing the destination IP address of the request message. Likewise, the default value of the Uri-Port Option is the destination UDP port. The default values for the Uri-Host and Uri-Port Options are sufficient for requests to most servers. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers.

The Uri-Path and Uri-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Uri-Path Option MUST NOT be "." or ".." (as the request URI must be resolved before parsing it into options).

The steps for constructing the request URI from the options are defined in Section 6.5. Note that an implementation does not necessarily have to construct the URI; it can simply look up the target resource by looking at the individual options.

Examples can be found in Appendix B.

#### 5.10.2. Proxy-Uri and Proxy-Scheme

The Proxy-Uri Option is used to make a request to a forward-proxy (see Section 5.7). The forward-proxy is requested to forward the request or service it from a valid cache, and return the response.

The option value is an absolute-URI ([RFC3986], Section 4.3).

Note that the forward-proxy MAY forward the request on to another proxy or directly to the server specified by the absolute-URI. In order to avoid request loops, a proxy MUST be able to recognize all of its server names, including any aliases, local variations, and the numeric IP addresses.

An endpoint receiving a request with a Proxy-Uri Option that is unable or unwilling to act as a forward-proxy for the request MUST cause the return of a 5.05 (Proxying Not Supported) response.

The Proxy-Uri Option MUST take precedence over any of the Uri-Host, Uri-Port, Uri-Path or Uri-Query options (which MUST NOT be included at the same time in a request containing the Proxy-Uri Option).

As a special case to simplify many proxy clients, the absolute-URI can be constructed from the Uri-\* options. When a Proxy-Scheme Option is present, the absolute-URI is constructed as follows: A CoAP URI is constructed from the Uri-\* options as defined in Section 6.5. In the resulting URI, the initial scheme up to, but not including the following colon is then replaced by the content of the Proxy-Scheme Option.

#### 5.10.3. Content-Format

The Content-Format Option indicates the representation format of the message payload. The representation format is given as a numeric content format identifier that is defined in the CoAP Content Format registry (Section 12.3). In the absence of the option, no default value is assumed, i.e. the representation format of any representation message payload is indeterminate (Section 5.5).

#### 5.10.4. Accept

The CoAP Accept option indicates when included one or more times in a request, one or more Content-Formats, each of which is an acceptable Content-Format for the client, in the order of preference (most preferred first). The representation format is given as a numeric Content-Format identifier that is defined in the CoAP Content-Format registry (Section 12.3). If no Accept options are given, the client does not express a preference (thus no default value is assumed). The client prefers the representation returned by the server to be in one of the Content-Formats indicated. The server SHOULD return one of the preferred Content-Formats if available. If none of the preferred Content-Formats can be returned, then a 4.06 "Not Acceptable" SHOULD be sent as a response.

Note that as a server might not support the Accept option (and thus would ignore it as it is elective), the client needs to be prepared to receive a representation in a different Content-Format. The client can simply discard a representation it can not make use of.

#### 5.10.5. Max-Age

The Max-Age Option indicates the maximum time a response may be cached before it MUST be considered not fresh (see Section 5.6.1).

The option value is an integer number of seconds between 0 and  $2^{32}-1$  inclusive (about 136.1 years). A default value of 60 seconds is assumed in the absence of the option in a response.

The value is intended to be current at the time of transmission. Servers that provide resources with strict tolerances on the value of

Max-Age SHOULD update the value before each retransmission. (See also Section 5.7.1.)

#### 5.10.6. ETag

An entity-tag is intended for use as a resource-local identifier for differentiating between representations of the same resource that vary over time. It is generated by the server providing the resource, which may generate it in any number of ways including a version, checksum, hash or time. An endpoint receiving an entity-tag MUST treat it as opaque and make no assumptions about its format. (Endpoints that generate an entity-tag are encouraged to use the most compact representation possible, in particular in regards to clients and intermediaries that may want to store multiple ETag values.)

##### 5.10.6.1. ETag as a Response Option

The ETag Option in a response provides the current value (i.e., after the request was processed) of the entity-tag for the "tagged representation". If no Location-\* options are present, the tagged representation is the selected representation (Section 5.5.3) of the target resource. If one or more Location-\* options are present and thus a location URI is indicated (Section 5.10.7), the tagged representation is the representation that would be retried by a GET request to the location URI.

An ETag response option can be included with any response for which there is a tagged representation (e.g., it would not be meaningful in a 4.04 or 4.00 response). The ETag Option MUST NOT occur more than once in a response.

There is no default value for the ETag Option; if it is not present in a response, the server makes no statement about the entity-tag for the tagged representation.

##### 5.10.6.2. ETag as a Request Option

In a GET request, an endpoint that has one or more representations previously obtained from the resource, and has obtained ETag response options with these, can specify an instance of the ETag Option for one or more of these stored responses.

A server can issue a 2.03 Valid response (Section 5.9.1.3) in place of a 2.05 Content response if one of the ETags given is the entity-tag for the current representation, i.e. is valid; the 2.03 Valid response then echoes this specific ETag in a response option.

In effect, a client can determine if any of the stored

representations is current (see Section 5.6.2) without needing to transfer them again.

The ETag Option MAY occur zero, one or more times in a request.

#### 5.10.7. Location-Path and Location-Query

The Location-Path and Location-Query Options together indicate a relative URI that consists either of an absolute path, a query string or both. A combination of these options is included in a 2.01 (Created) response to indicate the location of the resource created as the result of a POST request (see Section 5.8.2). The location is resolved relative to the request URI.

If a response with one or more Location-Path and/or Location-Query Options passes through a cache that interprets these options and the implied URI identifies one or more currently stored responses, those entries MUST be marked as not fresh.

Each Location-Path Option specifies one segment of the absolute path to the resource, and each Location-Query Option specifies one argument parameterizing the resource. The Location-Path and Location-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Location-Path Option MUST NOT be "." or "..".

The steps for constructing the location URI from the options are analogous to Section 6.5, except that the first five steps are skipped and the result is a relative URI-reference, which is then interpreted relative to the request URI. Note that the relative URI-reference constructed this way always includes an absolute-path (e.g., leaving out Location-Path but supplying Location-Query means the path component in the URI is "/").

The options that are used to compute the relative URI-reference are collectively called Location-\* options. Beyond Location-Path and Location-Query, more Location-\* options may be defined in the future, and have been reserved option numbers 128, 132, 136, and 140. If any of these reserved option numbers occurs in addition to Location-Path and/or Location-Query and are not supported, then a 4.02 (Bad Option) error MUST be returned.

#### 5.10.8. Conditional Request Options

Conditional request options enable a client to ask the server to perform the request only if certain conditions specified by the option are fulfilled.

For each of these options, if the condition given is not fulfilled, then the the server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

If the condition is fulfilled, the server performs the request method as if the conditional request options were not present.

If the request would, without the conditional request options, result in anything other than a 2.xx or 4.12 response code, then any conditional request options MAY be ignored.

#### 5.10.8.1. If-Match

The If-Match Option MAY be used to make a request conditional on the current existence or value of an ETag for one or more representations of the target resource. If-Match is generally useful for resource update requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource (i.e., the "lost update" problem).

The value of an If-Match option is either an ETag or the empty string. An If-Match option with an ETag matches a representation with that exact ETag. An If-Match option with an empty value matches any existing representation (i.e., it places the precondition on the existence of any current representation for the target resource).

The If-Match Option can occur multiple times. If any of the options match, then the condition is fulfilled.

If there is one or more If-Match Option, but none of the options match, then the condition is not fulfilled.

#### 5.10.8.2. If-None-Match

The If-None-Match Option MAY be used to make a request conditional on the non-existence of the target resource. If-None-Match is useful for resource creation requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource. The If-None-Match Option carries no value.

If the target resource does exist, then the condition is not fulfilled.

## 6. CoAP URIs

CoAP uses the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. Resources are organized hierarchically and governed by a potential CoAP origin server listening for CoAP requests ("coap") or DTLS-secured CoAP requests ("coaps") on a given UDP port. The CoAP server is identified via the generic syntax's authority component, which includes a host component and optional UDP port number. The remainder of the URI is considered to be identifying a resource which can be operated on by the methods defined by the CoAP protocol. The "coap" and "coaps" URI schemes can thus be compared to the "http" and "https" URI schemes respectively.

The syntax of the "coap" and "coaps" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", "query", "segment", "IP-literal", "IPv4address" and "reg-name" are adopted from [RFC3986].

Implementation Note: Unfortunately, over time the URI format has acquired significant complexity. Implementers are encouraged to examine [RFC3986] closely. E.g., the ABNF for IPv6 addresses is more complicated than maybe expected. Also, implementers should take care to perform the processing of percent decoding/encoding exactly once on the way from a URI to its decoded components or back. Percent encoding is crucial for data transparency, but may lead to unusual results such as a slash in a path component.

### 6.1. coap URI Scheme

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

If the host component is provided as an IP-literal or IPv4address, then the CoAP server can be reached at that IP address. If host is a registered name, then that name is considered an indirect identifier and the endpoint might use a name resolution service, such as DNS, to find the address of that host. The host MUST NOT be empty; if a URI is received with a missing authority or an empty host, then it MUST be considered invalid. The port subcomponent indicates the UDP port at which the CoAP server is located. If it is empty or not given, then the default port 5683 is assumed.

The path identifies a resource within the scope of the host and port. It consists of a sequence of path segments separated by a slash character (U+002F SOLIDUS "/").

The query serves to further parameterize the resource. It consists of a sequence of arguments separated by an ampersand character



(U+0026 AMPERSAND "&"). An argument is often in the form of a "key=value" pair.

The "coap" URI scheme supports the path prefix `"/.well-known/"` defined by [RFC5785] for "well-known locations" in the name-space of a host. This enables discovery of policy or other information about a host ("site-wide metadata"), such as hosted resources (see Section 7).

Application designers are encouraged to make use of short, but descriptive URIs. As the environments that CoAP is used in are usually constrained for bandwidth and energy, the trade-off between these two qualities should lean towards the shortness, without ignoring descriptiveness.

## 6.2. coaps URI Scheme

```
coaps-URI = "coaps:" "//" host [ ":" port ] path-abempty
           [ "?" query ]
```

All of the requirements listed above for the "coap" scheme are also requirements for the "coaps" scheme, except that a default UDP port of [IANA\_TBD\_PORT] is assumed if the port subcomponent is empty or not given, and the UDP datagrams MUST be secured for privacy through the use of DTLS as described in Section 9.1.

Considerations for caching of responses to "coaps" identified requests are discussed in Section 11.2.

Resources made available via the "coaps" scheme have no shared identity with the "coap" scheme even if their resource identifiers indicate the same authority (the same host listening to the same UDP port). They are distinct name spaces and are considered to be distinct origin servers.

## 6.3. Normalization and Comparison Rules

Since the "coap" and "coaps" schemes conform to the URI generic syntax, such URIs are normalized and compared according to the algorithm defined in [RFC3986], Section 6, using the defaults described above for each scheme.

If the port is equal to the default port for a scheme, the normal form is to elide the port subcomponent. Likewise, an empty path component is equivalent to an absolute path of `"/"`, so the normal form is to provide a path of `"/"` instead. The scheme and host are case-insensitive and normally provided in lowercase; IP-literals are in recommended form [RFC5952]; all other components are compared in a

case-sensitive manner. Characters other than those in the "reserved" set are equivalent to their percent-encoded octets (see [RFC3986], Section 2.1): the normal form is to not encode them.

For example, the following three URIs are equivalent, and cause the same options and option values to appear in the CoAP messages:

```
coap://example.com:5683/~sensors/temp.xml
coap://EXAMPLE.com/%7Esensors/temp.xml
coap://EXAMPLE.com:/%7esensors/temp.xml
```

#### 6.4. Decomposing URIs into Options

The steps to parse a request's options from a string `/url/` are as follows. These steps either result in zero or more of the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in the request, or they fail.

1. If the `/url/` string is not an absolute URI ([RFC3986]), then fail this algorithm.
2. Resolve the `/url/` string using the process of reference resolution defined by [RFC3986], with the URL character encoding set to UTF-8 [RFC3629].

NOTE: It doesn't matter what it is resolved relative to, since we already know it is an absolute URL at this point.

3. If `/url/` does not have a `<scheme>` component whose value, when converted to ASCII lowercase, is "coap" or "coaps", then fail this algorithm.
4. If `/url/` has a `<fragment>` component, then fail this algorithm.
5. If the `<host>` component of `/url/` does not represent the request's destination IP address as an IP-literal or IPv4address, include a Uri-Host Option and let that option's value be the value of the `<host>` component of `/url/`, converted to ASCII lowercase, and then converting all percent-encodings ("% followed by two hexadecimal digits) to the corresponding characters.

NOTE: In the usual case where the request's destination IP address is derived from the host part, this ensures that a Uri-Host Option is only used for a `<host>` component of the form `reg-name`.

6. If `/url/` has a `<port>` component, then let `/port/` be that component's value interpreted as a decimal integer; otherwise,

let /port/ be the default port for the scheme.

7. If /port/ does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be /port/.
8. If the value of the <path> component of /url/ is empty or consists of a single slash character (U+002F SOLIDUS "/"), then move to the next step.

Otherwise, for each segment in the <path> component, include a Uri-Path Option and let that option's value be the segment (not including the delimiting slash characters) after converting all percent-encodings ("% followed by two hexadecimal digits) to the corresponding characters.

9. If /url/ has a <query> component, then, for each argument in the <query> component, include a Uri-Query Option and let that option's value be the argument (not including the question mark and the delimiting ampersand characters) after converting all percent-encodings to the corresponding characters.

Note that these rules completely resolve any percent-encoding.

#### 6.5. Composing URIs from Options

The steps to construct a URI from a request's options are as follows. These steps either result in a URI, or they fail. In these steps, percent-encoding a character means replacing each of its (UTF-8 encoded) bytes by a "%" character followed by two hexadecimal digits representing the byte, where the digits A-F are in upper case (as defined in [RFC3986] Section 2.1; to reduce variability, the hexadecimal notation for percent-encoding in CoAP URIs MUST use uppercase letters). The definitions of "unreserved" and "sub-delims" are adopted from [RFC3986].

1. If the request is secured using DTLS, let /url/ be the string "coaps://". Otherwise, let /url/ be the string "coap://".
2. If the request includes a Uri-Host Option, let /host/ be that option's value, where any non-ASCII characters are replaced by their corresponding percent-encoding. If /host/ is not a valid reg-name or IP-literal or IPv4address, fail the algorithm. If the request does not include a Uri-Host Option, let /host/ be the IP-literal (making use of the conventions of [RFC5952]) or IPv4address representing the request's destination IP address.
3. Append /host/ to /url/.

4. If the request includes a Uri-Port Option, let /port/ be that option's value. Otherwise, let /port/ be the request's destination UDP port.
5. If /port/ is not the default port for the scheme, then append a single U+003A COLON character (:) followed by the decimal representation of /port/ to /url/.
6. Let /resource name/ be the empty string. For each Uri-Path Option in the request, append a single character U+002F SOLIDUS (/) followed by the option's value to /resource name/, after converting any character that is not either in the "unreserved" set, "sub-delims" set, a U+003A COLON (:) or U+0040 COMMERCIAL AT (@) character, to its percent-encoded form.
7. If /resource name/ is the empty string, set it to a single character U+002F SOLIDUS (/).
8. For each Uri-Query Option in the request, append a single character U+003F QUESTION MARK (?) (first option) or U+0026 AMPERSAND (&) (subsequent options) followed by the option's value to /resource name/, after converting any character that is not either in the "unreserved" set, "sub-delims" set (except U+0026 AMPERSAND (&)), a U+003A COLON (:), U+0040 COMMERCIAL AT (@), U+002F SOLIDUS (/) or U+003F QUESTION MARK (?) character, to its percent-encoded form.
9. Append /resource name/ to /url/.
10. Return /url/.

Note that these steps have been designed to lead to a URI in normal form (see Section 6.3).

## 7. Discovery

### 7.1. Service Discovery

A server is discovered by a client by the client knowing or learning a URI that references a resource in the namespace of the server. Alternatively, clients can use Multicast CoAP (see Section 8) and the "All CoAP Nodes" multicast address to find CoAP servers.

Unless the port subcomponent in a "coap" or "coaps" URI indicates the UDP port at which the CoAP server is located, the server is assumed to be reachable at the default port.

The CoAP default port number 5683 MUST be supported by a server that offers resources for resource discovery (see Section 7.2 below) and SHOULD be supported for providing access to other resources. The default port number [IANA\_TBD\_PORT] for DTLS-secured CoAP MAY be supported by a server for resource discovery and for providing access to other resources. In addition other endpoints may be hosted at other ports, e.g. in the dynamic port space.

Implementation Note: When a CoAP server is hosted by a 6LoWPAN node, header compression efficiency is improved when it also supports a port number in the 61616-61631 compressed UDP port space defined in [RFC4944] (note that, as its UDP port differs from the default port, it is a different endpoint from the server at the default port).

## 7.2. Resource Discovery

The discovery of resources offered by a CoAP endpoint is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. A CoAP endpoint SHOULD support the CoRE Link Format of discoverable resources as described in [RFC6690]. It is up to the server which resources are made discoverable (if any).

### 7.2.1. 'ct' Attribute

This section defines a new Web Linking [RFC5988] attribute for use with [RFC6690]. The Content-Format code "ct" attribute provides a hint about the Content-Formats this resource returns. Note that this is only a hint, and does not override the Content-Format Option of a CoAP response obtained by actually requesting the representation of the resource. The value is in the CoAP identifier code format as a decimal ASCII integer and MUST be in the range of 0-65535 (16-bit unsigned integer). For example application/xml would be indicated as "ct=41". If no Content-Format code attribute is present then nothing about the type can be assumed. The Content-Format code attribute MAY include a space-separated sequence of Content-Format codes, indicating that multiple content-formats are available. The syntax of the attribute value is summarized in the production ct-value in Figure 12, where cardinal, SP and DQUOTE are defined as in [RFC6690].

```
ct-value = cardinal
          / DQUOTE cardinal *( 1*SP cardinal ) DQUOTE
```

Figure 12

## 8. Multicast CoAP

CoAP supports making requests to a IP multicast group. This is defined by a series of deltas to Unicast CoAP.

CoAP endpoints that offer services that they want other endpoints to be able to find using multicast service discovery, join one or more of the appropriate all-CoAP-nodes multicast addresses (Section 12.8) and listen on the default CoAP port. Note that an endpoint might receive multicast requests on other multicast addresses, including the all-nodes IPv6 address (or via broadcast on IPv4); an endpoint **MUST** therefore be prepared to receive such messages but **MAY** ignore them if multicast service discovery is not desired.

### 8.1. Messaging Layer

A multicast request is characterized by being transported in a CoAP message that is addressed to an IP multicast address instead of a CoAP endpoint. Such multicast requests **MUST** be Non-Confirmable.

A server **SHOULD** be aware that a request arrived via multicast, e.g. by making use of modern APIs such as IPV6\_RECVPKTINFO [RFC3542], if available.

When a server is aware that a request arrived via multicast, it **MUST NOT** return a RST in reply to NON. If it is not aware, it **MAY** return a RST in reply to NON as usual. Because such a Reset message will look identical to an RST for a unicast message from the sender, the sender **MUST** avoid using a Message ID that is also still active from this endpoint with any unicast endpoint that might receive the multicast message.

### 8.2. Request/Response Layer

When a server is aware that a request arrived via multicast, the server **MAY** always pretend it did not receive the request, in particular if it doesn't have anything useful to respond (e.g., if it only has an empty payload or an error response). The decision for this may depend on the application. (For example, in [RFC6690] query filtering, a server should not respond to a multicast request if the filter does not match.)

If a server does decide to respond to a multicast request, it should not respond immediately. Instead, it should pick a duration for the period of time during which it intends to respond. For purposes of this exposition, we call the length of this period the *Leisure*. The specific value of this *Leisure* may depend on the application, or **MAY** be derived as described below. The server **SHOULD** then pick a random

point of time within the chosen Leisure period to send back the unicast response to the multicast request. If further responses need to be sent based on the same multicast address membership, a new leisure period starts at the earliest after the previous one finishes.

To compute a value for Leisure, the server should have a group size estimate  $G$ , a target data transfer rate  $R$  (which both should be chosen conservatively) and an estimated response size  $S$ ; a rough lower bound for Leisure can then be computed as

$$\text{lb\_Leisure} = S * G / R$$

E.g., for a multicast request with link-local scope on an 2.4 GHz IEEE 802.15.4 (6LoWPAN) network,  $G$  could be (relatively conservatively) set to 100,  $S$  to 100 bytes, and the target rate to a conservative 8 kbit/s = 1 kB/s. The resulting lower bound for the Leisure is 10 seconds.

If a CoAP endpoint does not have suitable data to compute a value for Leisure, it MAY resort to DEFAULT\_LEISURE.

When matching a response to a multicast request, only the token MUST match; the source endpoint of the response does not need to (and will not) be the same as the destination endpoint of the original request.

For the purposes of interpreting the Location-\* options and any links embedded in the representation and, the request URI (base URI) relative to which the response is interpreted, is formed by replacing the multicast address in the Host component of the original request URI by the literal IP address of the endpoint actually responding.

#### 8.2.1. Caching

When a client makes a multicast request, it always makes a new request to the multicast group (since there may be new group members that joined meanwhile or ones that did not get the previous request). It MAY update the cache with the received responses. Then it uses both cached-still-fresh and 'new' responses as the result of the request.

A response received in reply to a GET request to a multicast group MAY be used to satisfy a subsequent request on the related unicast request URI. The unicast request URI is obtained by replacing the authority part of the request URI with the transport layer source address of the response message.

A cache MAY revalidate a response by making a GET request on the related unicast request URI.

A GET request to a multicast group MUST NOT contain an ETag option. A mechanism to suppress responses the client already has is left for further study.

#### 8.2.2. Proxying

When a forward-proxy receives a request with a Proxy-Uri or URI constructed from Proxy-Scheme that indicates a multicast address, the proxy obtains a set of responses as described above and sends all responses (both cached-still-fresh and new) back to the original client.

This specification does not provide a way to indicate the unicast-modified request URI (base URI) in responses thus forwarded. A proposal to address this can be found in section 3 of [I-D.bormann-coap-misc].

### 9. Securing CoAP

This section defines the DTLS binding for CoAP.

During the provisioning phase, a CoAP device is provided with the security information that it needs, including keying materials and access control lists. This specification defines provisioning for the RawPublicKey mode in Section 9.1.3.2.1. At the end of the provisioning phase, the device will be in one of four security modes with the following information for the given mode. The NoSec and RawPublicKey modes are mandatory to implement for this specification.

**NoSec:** There is no protocol level security (DTLS is disabled). Alternative techniques to provide lower layer security SHOULD be used when appropriate. The use of IPsec is discussed in [I-D.bormann-core-ipsec-for-coap].

**PreSharedKey:** DTLS is enabled and there is a list of pre-shared keys [RFC4279] and each key includes a list of which nodes it can be used to communicate with as described in Section 9.1.3.1. At the extreme there may be one key for each node this CoAP node needs to communicate with (1:1 node/key ratio).

**RawPublicKey:** DTLS is enabled and the device has an asymmetric key pair without a certificate (a raw public key) that is validated using an out-of-band mechanism [I-D.ietf-tls-oob-pubkey] as described in Section 9.1.3.2. The device also has an identity calculated from the public key and a list of identities of the nodes it can communicate with.



Certificate: DTLS is enabled and the device has an asymmetric key pair with an X.509 certificate [RFC5280] that binds it to its Authority Name and is signed by some common trust root as described in Section 9.1.3.3. The device also has a list of root trust anchors that can be used for validating a certificate.

In the "NoSec" mode, the system simply sends the packets over normal UDP over IP and is indicated by the "coap" scheme and the CoAP default port. The system is secured only by keeping attackers from being able to send or receive packets from the network with the CoAP nodes; see Section 11.5 for an additional complication with this approach.

The other three security modes are achieved using DTLS and are indicated by the "coaps" scheme and DTLS-secured CoAP default port. The result is a security association that can be used to authenticate (within the limits of the security model) and, based on this authentication, authorize the communication partner. CoAP itself does not provide protocol primitives for authentication or authorization; where this is required, it can either be provided by communication security (i.e., IPsec or DTLS) or by object security (within the payload). Devices that require authorization for certain operations are expected to require one of these two forms of security. Necessarily, where an intermediary is involved, communication security only works when that intermediary is part of the trust relationships; CoAP does not provide a way to forward different levels of authorization that clients may have with an intermediary to further intermediaries or origin servers -- it therefore may be required to perform all authorization at the first intermediary.

#### 9.1. DTLS-secured CoAP

Just as HTTP is secured using Transport Layer Security (TLS) over TCP, CoAP is secured using Datagram TLS (DTLS) [RFC6347] over UDP (see Figure 13). This section defines the CoAP binding to DTLS, along with the minimal mandatory-to-implement configurations appropriate for constrained environments. The binding is defined by a series of deltas to Unicast CoAP. DTLS is in practice TLS with added features to deal with the unreliable nature of the UDP transport.

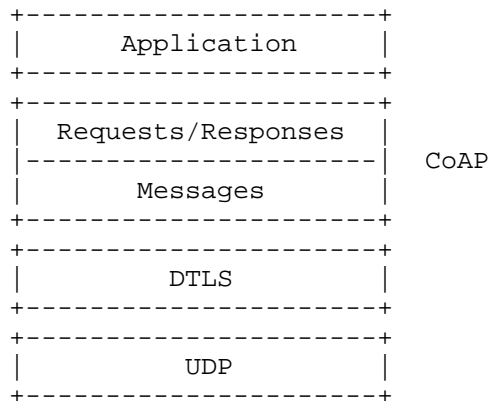


Figure 13: Abstract layering of DTLS-secured CoAP

In some constrained nodes (limited flash and/or RAM) and networks (limited bandwidth or high scalability requirements), and depending on the specific cipher suites in use, all modes of DTLS may not be applicable. Some DTLS cipher suites can add significant implementation complexity as well as some initial handshake overhead needed when setting up the security association. Once the initial handshake is completed, DTLS adds a limited per-datagram overhead of approximately 13 bytes, not including any initialization vectors/nonces (e.g., 8 bytes with `TLS_PSK_WITH_AES_128_CCM_8` [RFC6655]), integrity check values (e.g., 8 bytes with `TLS_PSK_WITH_AES_128_CCM_8` [RFC6655]) and padding required by the cipher suite. Whether and which mode of using DTLS is applicable for a CoAP-based application should be carefully weighed considering the specific cipher suites that may be applicable, and whether the session maintenance makes it compatible with application flows and sufficient resources are available on the constrained nodes and for the added network overhead. DTLS is not applicable to group keying (multicast communication); however, it may be a component in a future group key management protocol.

#### 9.1.1.1. Messaging Layer

The endpoint acting as the CoAP client should also act as the DTLS client. It should initiate a session to the server on the appropriate port. When the DTLS handshake has finished, the client may initiate the first CoAP request. All CoAP messages **MUST** be sent as DTLS "application data".

The following rules are added for matching an ACK or RST to a CON message or a RST to a NON message: The DTLS session **MUST** be the same and the epoch **MUST** be the same.

A message is the same when it is sent within the same DTLS session and same epoch and has the same Message ID.

Note: When a confirmable message is retransmitted, a new DTLS sequence\_number is used for each attempt, even though the CoAP Message ID stays the same. So a recipient still has to perform deduplication as described in Section 4.5. Retransmissions MUST NOT be performed across epochs.

DTLS connections in RawPublicKey and Certificate mode are set up using mutual authentication so they can remain up and be reused for future message exchanges in either direction. Devices can close a DTLS connection when they need to recover resources but in general they should keep the connection up for as long as possible. Closing the DTLS connection after every CoAP message exchange is very inefficient.

#### 9.1.2. Request/Response Layer

The following rules are added for matching a response to a request: The DTLS session MUST be the same and the epoch MUST be the same.

#### 9.1.3. Endpoint Identity

Devices SHOULD support the Server Name Indication (SNI) to indicate their Authority Name in the SNI HostName field as defined in Section 3 of [RFC6066]. This is needed so that when a host that acts as a virtual server for multiple Authorities receives a new DTLS connection, it knows which keys to use for the DTLS session.

##### 9.1.3.1. Pre-Shared Keys

When forming a connection to a new node, the system selects an appropriate key based on which nodes it is trying to reach and then forms a DTLS session using a PSK (Pre-Shared Key) mode of DTLS. Implementations in these modes MUST support the mandatory to implement cipher suite TLS\_PSK\_WITH\_AES\_128\_CCM\_8 as specified in [RFC6655].

The security considerations of [RFC4279] (Section 7) apply. In particular, applications should carefully weigh whether they need Perfect Forward Secrecy (PFS) or not and select an appropriate cipher suite (7.1). The entropy of the PSK must be sufficient to mitigate against brute-force and (where the PSK is not chosen randomly but by a human) dictionary attacks (7.2). The cleartext communication of client identities may leak data or compromise privacy (7.3).

#### 9.1.3.2. Raw Public Key Certificates

In this mode the device has an asymmetric key pair but without an X.509 certificate (called a raw public key). A device MAY be configured with multiple raw public keys. The type and length of the raw public key depends on the cipher suite used. Implementations in RawPublicKey mode MUST support the mandatory to implement cipher suite TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CCM\_8 as specified in [I-D.mcgregor-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. The mechanism for using raw public keys with TLS is specified in [I-D.ietf-tls-oob-pubkey].

##### 9.1.3.2.1. Provisioning

The RawPublicKey mode was designed to be easily provisioned in M2M deployments. It is assumed that each device has an appropriate asymmetric public key pair installed. An identifier is calculated from the public key as described in Section 2 of [I-D.farrell-decade-ni]. All implementations that support checking RawPublicKey identities MUST support at least the sha-256-120 mode (SHA-256 truncated to 120 bits). Implementations SHOULD support also longer length identifiers and MAY support shorter lengths. Note that the shorter lengths provide less security against attacks and their use is NOT RECOMMENDED.

Depending on how identifiers are given to the system that verifies them, support for URI, binary, and/or human-speakable format [I-D.farrell-decade-ni] needs to be implemented. All implementations SHOULD support the binary mode and implementations that have a user interface SHOULD also support the human-speakable format.

During provisioning, the identifier of each node is collected, for example by reading a barcode on the outside of the device or by obtaining a pre-compiled list of the identifiers. These identifiers are then installed in the corresponding endpoint, for example an M2M data collection server. The identifier is used for two purposes, to associate the endpoint with further device information and to perform access control. During provisioning, an access control list of identifiers the device may start DTLS sessions with SHOULD also be installed.

#### 9.1.3.3. X.509 Certificates

Implementations in Certificate Mode MUST support the mandatory to implement cipher suite TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CCM\_8 as specified in [RFC5246].

The Authority Name in the certificate is the name that would be used

in the Host part of a CoAP URI. It is worth noting that this would typically not be either an IP address or DNS name built in the usual way but would instead be built out of a long term unique identifier for the device such as the EUI-64 [EUI64]. The discovery process used in the system would build up the mapping between IP addresses of the given devices and the Authority Name for each device. Some devices could have more than one Authority and would need more than a single certificate.

When a new connection is formed, the certificate from the remote device needs to be verified. If the CoAP node has a source of absolute time, then the node SHOULD check that the validity dates of the certificate are within range. The certificate MUST also be signed by an appropriate chain of trust. If the certificate contains a SubjectAltName, then the Authority Name MUST match at least one of the authority names of any CoAP URI found in a field of URI type in the SubjectAltName set. If there is no SubjectAltName in the certificate, then the Authoritative Name must match the CN found in the certificate using the matching rules defined in [RFC2818] with the exception that certificates with wildcards are not allowed.

If the system has a shared key in addition to the certificate, then a cipher suite that includes the shared key such as TLS\_RSA\_PSK\_WITH\_AES\_128\_CBC\_SHA [RFC4279] SHOULD be used.

#### 10. Cross-Protocol Proxying between CoAP and HTTP

CoAP supports a limited subset of HTTP functionality, and thus cross-protocol proxying to HTTP is straightforward. There might be several reasons for proxying between CoAP and HTTP, for example when designing a web interface for use over either protocol or when realizing a CoAP-HTTP proxy. Likewise, CoAP could equally be proxied to other protocols such as XMPP [RFC6120] or SIP [RFC3264]; the definition of these mechanisms is out of scope of this specification.

There are two possible directions to access a resource via a forward-proxy:

CoAP-HTTP Proxying: Enables CoAP clients to access resources on HTTP servers through an intermediary. This is initiated by including the Proxy-Uri or Proxy-Scheme Option with an "http" or "https" URI in a CoAP request to a CoAP-HTTP proxy.

HTTP-CoAP Proxying: Enables HTTP clients to access resources on CoAP servers through an intermediary. This is initiated by specifying a "coap" or "coaps" URI in the Request-Line of an HTTP request to an HTTP-CoAP proxy.

Either way, only the Request/Response model of CoAP is mapped to HTTP. The underlying model of confirmable or non-confirmable messages, etc., is invisible and MUST have no effect on a proxy function. The following sections describe the handling of requests to a forward-proxy. Reverse proxies are not specified as the proxy function is transparent to the client with the proxy acting as if it was the origin server. However, similar considerations apply to reverse-proxies as to forward-proxies, and there generally will be an expectation that reverse-proxies operate in a similar way forward-proxies would. As an implementation note, HTTP client libraries may make it hard to operate an HTTP-CoAP forward proxy by not providing a way to put a CoAP URI on the HTTP Request-Line; reverse-proxying may therefore lead to wider applicability of a proxy. A separate specification may define a convention for URIs operating such a HTTP-CoAP reverse proxy [I-D.bormann-core-cross-reverse-convention].

#### 10.1. CoAP-HTTP Proxying

If a request contains a Proxy-Uri or Proxy-Scheme Option with an 'http' or 'https' URI [RFC2616], then the receiving CoAP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated HTTP resource and return the result to the client.

This section specifies for any CoAP request the CoAP response that the proxy should return to the client. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to an HTTP origin server.

Since HTTP and CoAP share the basic set of request methods, performing a CoAP request on an HTTP resource is not so different from performing it on a CoAP resource. The meanings of the individual CoAP methods when performed on HTTP resources are explained below.

If the proxy is unable or unwilling to service a request with an HTTP URI, a 5.05 (Proxying Not Supported) response is returned to the client. If the proxy services the request by interacting with a third party (such as the HTTP origin server) and is unable to obtain a result within a reasonable time frame, a 5.04 (Gateway Timeout) response is returned; if a result can be obtained but is not understood, a 5.02 (Bad Gateway) response is returned.

##### 10.1.1. GET

The GET method requests the proxy to return a representation of the HTTP resource identified by the request URI.

Upon success, a 2.05 (Content) response code SHOULD be returned. The payload of the response MUST be a representation of the target HTTP resource, and the Content-Format Option be set accordingly. The response MUST indicate a Max-Age value that is no greater than the remaining time the representation can be considered fresh. If the HTTP entity has an entity tag, the proxy SHOULD include an ETag Option in the response and process ETag Options in requests as described below.

A client can influence the processing of a GET request by including the following option:

Accept: The request MAY include one or more Accept Options, identifying the preferred response content-format.

ETag: The request MAY include one or more ETag Options, identifying responses that the client has stored. This requests the proxy to send a 2.03 (Valid) response whenever it would send a 2.05 (Content) response with an entity tag in the requested set otherwise. Note that CoAP ETags are always strong ETags in the HTTP sense; CoAP does not have the equivalent of HTTP weak ETags, and there is no good way to make use of these in a cross-proxy.

#### 10.1.1.2. PUT

The PUT method requests the proxy to update or create the HTTP resource identified by the request URI with the enclosed representation.

If a new resource is created at the request URI, a 2.01 (Created) response MUST be returned to the client. If an existing resource is modified, a 2.04 (Changed) response MUST be returned to indicate successful completion of the request.

#### 10.1.1.3. DELETE

The DELETE method requests the proxy to delete the HTTP resource identified by the request URI at the HTTP origin server.

A 2.02 (Deleted) response MUST be returned to client upon success or if the resource does not exist at the time of the request.

#### 10.1.1.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the HTTP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request

URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 2.04 (Changed) response MUST be returned to the client. If a resource has been created on the origin server, a 2.01 (Created) response MUST be returned.

## 10.2. HTTP-CoAP Proxying

If an HTTP request contains a Request-URI with a 'coap' or 'coaps' URI, then the receiving HTTP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated CoAP resource and return the result to the client.

This section specifies for any HTTP request the HTTP response that the proxy should return to the client. Unless otherwise specified all the statements made are RECOMMENDED behavior; some highly constrained implementations may need to resort to shortcuts. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to a CoAP origin server. The meanings of the individual HTTP methods when performed on CoAP resources are explained below.

If the proxy is unable or unwilling to service a request with a CoAP URI, a 501 (Not Implemented) response is returned to the client. If the proxy services the request by interacting with a third party (such as the CoAP origin server) and is unable to obtain a result within a reasonable time frame, a 504 (Gateway Timeout) response is returned; if a result can be obtained but is not understood, a 502 (Bad Gateway) response is returned.

### 10.2.1. OPTIONS and TRACE

As the OPTIONS and TRACE methods are not supported in CoAP a 501 (Not Implemented) error MUST be returned to the client.

### 10.2.2. GET

The GET method requests the proxy to return a representation of the CoAP resource identified by the Request-URI.

Upon success, a 200 (OK) response is returned. The payload of the response MUST be a representation of the target CoAP resource, and the Content-Type and Content-Encoding header fields be set accordingly. The response MUST indicate a max-age directive that indicates a value no greater than the remaining time the representation can be considered fresh. If the CoAP response has an



ETag option, the proxy should include an ETag header field in the response.

A client can influence the processing of a GET request by including the following options:

Accept: Each individual Media-type of the HTTP Accept header in a request is mapped to a CoAP Accept option. HTTP Accept Media-type ranges, parameters and extensions are not supported by the CoAP Accept option. If the proxy cannot send a response which is acceptable according to the combined Accept field value, then the proxy sends a 406 (not acceptable) response.

Conditional GETs: Conditional HTTP GET requests that include an "If-Match" or "If-None-Match" request-header field can be mapped to a corresponding CoAP request. The "If-Modified-Since" and "If-Unmodified-Since" request-header fields are not directly supported by CoAP, but are implemented locally by a caching proxy.

#### 10.2.3. HEAD

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response.

Although there is no direct equivalent of HTTP's HEAD method in CoAP, an HTTP-CoAP proxy responds to HEAD requests for CoAP resources, and the HTTP headers are returned without a message-body.

Implementation Note: An HTTP-CoAP proxy may want to try using a block-wise transfer [I-D.ietf-core-block] option to minimize the amount of data actually transferred, but needs to be prepared for the case that the origin server does not support block-wise transfers.

#### 10.2.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the CoAP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 200 (OK) or 204 (No Content) response MUST be returned to the client. If a resource has been created on the origin server, a 201 (Created) response MUST be returned.

If any of the Location-\* Options are present in the CoAP response, a Location header field constructed from the values of these options is returned.

#### 10.2.5. PUT

The PUT method requests the proxy to update or create the CoAP resource identified by the Request-URI with the enclosed representation.

If a new resource is created at the Request-URI, a 201 (Created) response is returned to the client. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes is sent to indicate successful completion of the request.

#### 10.2.6. DELETE

The DELETE method requests the proxy to delete the CoAP resource identified by the Request-URI at the CoAP origin server.

A successful response is 200 (OK) if the response includes an entity describing the status or 204 (No Content) if the action has been enacted but the response does not include an entity.

#### 10.2.7. CONNECT

This method can not currently be satisfied by an HTTP-CoAP proxy function as TLS to DTLS tunneling has not yet been specified. For now, a 501 (Not Implemented) error is returned to the client.

### 11. Security Considerations

This section analyzes the possible threats to the protocol. It is meant to inform protocol and application developers about the security limitations of CoAP as described in this document. As CoAP realizes a subset of the features in HTTP/1.1, the security considerations in Section 15 of [RFC2616] are also pertinent to CoAP. This section concentrates on describing limitations specific to CoAP.

#### 11.1. Protocol Parsing, Processing URIs

A network-facing application can exhibit vulnerabilities in its processing logic for incoming packets. Complex parsers are well-known as a likely source of such vulnerabilities, such as the ability to remotely crash a node, or even remotely execute arbitrary code on it. CoAP attempts to narrow the opportunities for introducing such vulnerabilities by reducing parser complexity, by giving the entire

range of encodable values a meaning where possible, and by aggressively reducing complexity that is often caused by unnecessary choice between multiple representations that mean the same thing. Much of the URI processing has been moved to the clients, further reducing the opportunities for introducing vulnerabilities into the servers. Even so, the URI processing code in CoAP implementations is likely to be a large source of remaining vulnerabilities and should be implemented with special care. The most complex parser remaining could be the one for the CoRE Link Format, although this also has been designed with a goal of reduced implementation complexity [RFC6690]. (See also section 15.2 of [RFC2616].)

#### 11.2. Proxying and Caching

As mentioned in 15.7 of [RFC2616], proxies are by their very nature men-in-the-middle, breaking any IPsec or DTLS protection that a direct CoAP message exchange might have. They are therefore interesting targets for breaking confidentiality or integrity of CoAP message exchanges. As noted in [RFC2616], they are also interesting targets for breaking availability.

The threat to confidentiality and integrity of request/response data is amplified where proxies also cache. Note that CoAP does not define any of the cache-suppressing Cache-Control options that HTTP/1.1 provides to better protect sensitive data.

For a caching implementation, any access control considerations that would apply to making the request that generated the cache entry also need to be applied to the value in the cache. This is relevant for clients that implement multiple security domains, as well as for proxies that may serve multiple clients. Also, a caching proxy **MUST NOT** make cached values available to requests that have lesser transport security properties than to which it would make available the process of forwarding the request in the first place.

Unlike the "coap" scheme, responses to "coaps" identified requests are never "public" and thus **MUST NOT** be reused for shared caching unless the cache is able to make equivalent access control decisions to the ones that led to the cached entry. They can, however, be reused in a private cache if the message is cacheable by default in CoAP.

Finally, a proxy that fans out Separate Responses (as opposed to Piggy-backed Responses) to multiple original requesters may provide additional amplification (see below).

### 11.3. Risk of amplification

CoAP servers generally reply to a request packet with a response packet. This response packet may be significantly larger than the request packet. An attacker might use CoAP nodes to turn a small attack packet into a larger attack packet, an approach known as amplification. There is therefore a danger that CoAP nodes could become implicated in denial of service (DoS) attacks by using the amplifying properties of the protocol: An attacker that is attempting to overload a victim but is limited in the amount of traffic it can generate, can use amplification to generate a larger amount of traffic.

This is particularly a problem in nodes that enable NoSec access, that are accessible from an attacker and can access potential victims (e.g. on the general Internet), as the UDP protocol provides no way to verify the source address given in the request packet. An attacker need only place the IP address of the victim in the source address of a suitable request packet to generate a larger packet directed at the victim.

As a mitigating factor, many constrained networks will only be able to generate a small amount of traffic, which may make CoAP nodes less attractive for this attack. However, the limited capacity of the constrained network makes the network itself a likely victim of an amplification attack.

A CoAP server can reduce the amount of amplification it provides to an attacker by using slicing/blocking modes of CoAP [I-D.ietf-core-block] and offering large resource representations only in relatively small slices. E.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

CoAP also supports the use of multicast IP addresses in requests, an important requirement for M2M. Multicast CoAP requests may be the source of accidental or deliberate denial of service attacks, especially over constrained networks. This specification attempts to reduce the amplification effects of multicast requests by limiting when a response is returned. To limit the possibility of malicious use, CoAP servers SHOULD NOT accept multicast requests that can not be authenticated in some way, cryptographically or by some multicast boundary limiting the potential sources. If possible a CoAP server SHOULD limit the support for multicast requests to the specific resources where the feature is required.

On some general purpose operating systems providing a Posix-style

API, it is not straightforward to find out whether a packet received was addressed to a multicast address. While many implementations will know whether they have joined a multicast group, this creates a problem for packets addressed to multicast addresses of the form FF0x::1, which are received by every IPv6 node. Implementations SHOULD make use of modern APIs such as IPV6\_RECVPKTINFO [RFC3542], if available, to make this determination.

#### 11.4. IP Address Spoofing Attacks

Due to the lack of a handshake in UDP, a rogue endpoint which is free to read and write messages carried by the constrained network (i.e. NoSec or PreSharedKey deployments with nodes/key ratio > 1:1), may easily attack a single endpoint, a group of endpoints, as well as the whole network e.g. by:

1. spoofing RST in response to a CON or NON message, thus making an endpoint "deaf"; or
2. spoofing the entire response with forged payload/options (this has different levels of impact: from single response disruption, to much bolder attacks on the supporting infrastructure, e.g. poisoning proxy caches, or tricking validation / lookup interfaces in resource directories and, more generally, any component that stores global network state and uses CoAP as the messaging facility to handle state set/update's is a potential target.); or
3. spoofing a multicast request for a target node which may result in both network congestion/collapse and victim DoS'ing / forced wakeup from sleeping; or
4. spoofing observe messages, etc.

In principle, spoofing can be detected by CoAP only in case CON semantics is used, because of unexpected ACK/RSTs coming from the deceived endpoint. But this imposes keeping track of the used Message IDs which is not always possible, and moreover detection becomes available usually after the damage is already done. This kind of attack can be prevented using security modes other than NoSec.

#### 11.5. Cross-Protocol Attacks

The ability to incite a CoAP endpoint to send packets to a fake source address can be used not only for amplification, but also for cross-protocol attacks against a victim listening to UDP packets at a given address (IP address and port):

- o the attacker sends a message to a CoAP endpoint with the given address as the fake source address,
- o the CoAP endpoint replies with a message to the given source address,
- o the victim at the given address receives a UDP packet that it interprets according to the rules of a different protocol.

This may be used to circumvent firewall rules that prevent direct communication from the attacker to the victim, but happen to allow communication from the CoAP endpoint (which may also host a valid role in the other protocol) to the victim.

Also, CoAP endpoints may be the victim of a cross-protocol attack generated through an endpoint of another UDP-based protocol such as DNS. In both cases, attacks are possible if the security properties of the endpoints rely on checking IP addresses (and firewalling off direct attacks sent from outside using fake IP addresses). In general, because of their lack of context, UDP-based protocols are relatively easy targets for cross-protocol attacks.

Finally, CoAP URIs transported by other means could be used to incite clients to send messages to endpoints of other protocols.

One mitigation against cross-protocol attacks is strict checking of the syntax of packets received, combined with sufficient difference in syntax. As an example, it might help if it were difficult to incite a DNS server to send a DNS response that would pass the checks of a CoAP endpoint. Unfortunately, the first two bytes of a DNS reply are an ID that can be chosen by the attacker, which map into the interesting part of the CoAP header, and the next two bytes are then interpreted as CoAP's Message ID (i.e., any value is acceptable). The DNS count words may be interpreted as multiple instances of a (non-existent, but elective) CoAP option 0, or possibly as a Token. The echoed query finally may be manufactured by the attacker to achieve a desired effect on the CoAP endpoint; the response added by the server (if any) might then just be interpreted as added payload.

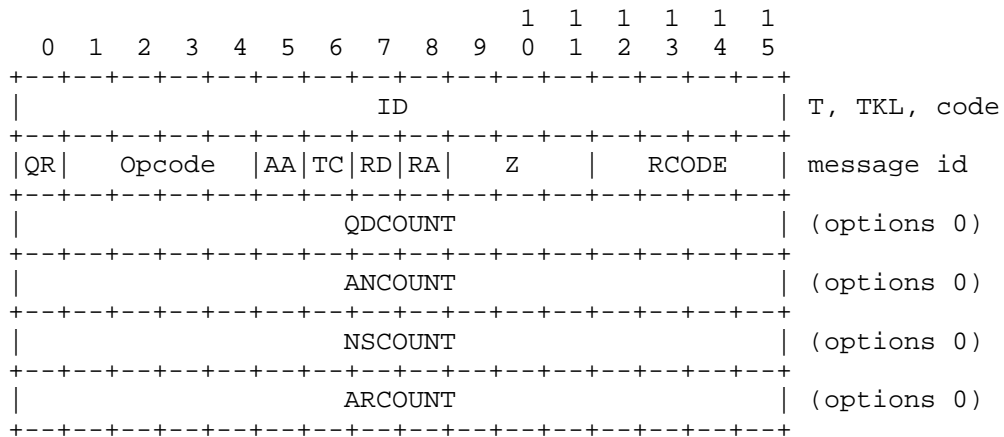


Figure 14: DNS Header vs. CoAP Message

In general, for any pair of protocols, one of the protocols can very well have been designed in a way that enables an attacker to cause the generation of replies that look like messages of the other protocol. It is often much harder to ensure or prove the absence of viable attacks than to generate examples that may not yet completely enable an attack but might be further developed by more creative minds. Cross-protocol attacks can therefore only be completely mitigated if endpoints don't authorize actions desired by an attacker just based on trusting the source IP address of a packet. Conversely, a NoSec environment that completely relies on a firewall for CoAP security not only needs to firewall off the CoAP endpoints but also all other endpoints that might be incited to send UDP messages to CoAP endpoints using some other UDP-based protocol.

In addition to the considerations above, the security considerations for DTLS with respect to cross-protocol attacks apply. E.g., if the same DTLS security association ("connection") is used to carry data of multiple protocols, DTLS no longer provides protection against cross-protocol attacks between these protocols.

## 12. IANA Considerations

### 12.1. CoAP Code Registry

This document defines a registry for the values of the Code field in the CoAP header. The name of the registry is "CoAP Codes".

All values are assigned by sub-registries according to the following ranges:

- 0            Indicates an empty message (see Section 4.1).
- 1-31        Indicates a request. Values in this range are assigned by the "CoAP Method Codes" sub-registry (see Section 12.1.1).
- 32-63       Reserved
- 64-191     Indicates a response. Values in this range are assigned by the "CoAP Response Codes" sub-registry (see Section 12.1.2).
- 192-255    Reserved

#### 12.1.1.1. Method Codes

The name of the sub-registry is "CoAP Method Codes".

Each entry in the sub-registry must include the Method Code in the range 1-31, the name of the method, and a reference to the method's documentation.

Initial entries in this sub-registry are as follows:

Code	Name	Reference
1	GET	[RFCXXXX]
2	POST	[RFCXXXX]
3	PUT	[RFCXXXX]
4	DELETE	[RFCXXXX]

Table 2: CoAP Method Codes

All other Method Codes are Unassigned.

The IANA policy for future additions to this registry is "IETF Review or IESG approval" as described in [RFC5226].

The documentation of a method code should specify the semantics of a request with that code, including the following properties:

- o The response codes the method returns in the success case.
- o Whether the method is idempotent, safe, or both.



## 12.1.1.2. Response Codes

The name of the sub-registry is "CoAP Response Codes".

Each entry in the sub-registry must include the Response Code in the range 64-191, a description of the Response Code, and a reference to the Response Code's documentation.

Initial entries in this sub-registry are as follows:

Code	Description	Reference
65	2.01 Created	[RFCXXXX]
66	2.02 Deleted	[RFCXXXX]
67	2.03 Valid	[RFCXXXX]
68	2.04 Changed	[RFCXXXX]
69	2.05 Content	[RFCXXXX]
128	4.00 Bad Request	[RFCXXXX]
129	4.01 Unauthorized	[RFCXXXX]
130	4.02 Bad Option	[RFCXXXX]
131	4.03 Forbidden	[RFCXXXX]
132	4.04 Not Found	[RFCXXXX]
133	4.05 Method Not Allowed	[RFCXXXX]
134	4.06 Not Acceptable	[RFCXXXX]
140	4.12 Precondition Failed	[RFCXXXX]
141	4.13 Request Entity Too Large	[RFCXXXX]
143	4.15 Unsupported Content-Format	[RFCXXXX]
160	5.00 Internal Server Error	[RFCXXXX]
161	5.01 Not Implemented	[RFCXXXX]
162	5.02 Bad Gateway	[RFCXXXX]
163	5.03 Service Unavailable	[RFCXXXX]
164	5.04 Gateway Timeout	[RFCXXXX]
165	5.05 Proxying Not Supported	[RFCXXXX]

Table 3: CoAP Response Codes

The Response Codes 96-127 are Reserved for future use. All other Response Codes are Unassigned.

The IANA policy for future additions to this registry is "IETF Review or IESG approval" as described in [RFC5226].

The documentation of a response code should specify the semantics of a response with that code, including the following properties:

- o The methods the response code applies to.
- o Whether payload is required, optional or not allowed.
- o The semantics of the payload. For example, the payload of a 2.05 (Content) response is a representation of the target resource; the payload in an error response is a human-readable diagnostic payload.
- o The format of the payload. For example, the format in a 2.05 (Content) response is indicated by the Content-Format Option; the format of the payload in an error response is always Net-Unicode text.
- o Whether the response is cacheable according to the freshness model.
- o Whether the response is validatable according to the validation model.
- o Whether the response causes a cache to mark responses stored for the request URI as not fresh.

#### 12.2. Option Number Registry

This document defines a registry for the Option Numbers used in CoAP options. The name of the registry is "CoAP Option Numbers".

Each entry in the registry must include the Option Number, the name of the option and a reference to the option's documentation.

Initial entries in this registry are as follows:

Number	Name	Reference
0	(Reserved)	
1	If-Match	[RFCXXXX]
3	Uri-Host	[RFCXXXX]
4	ETag	[RFCXXXX]
5	If-None-Match	[RFCXXXX]
7	Uri-Port	[RFCXXXX]
8	Location-Path	[RFCXXXX]
11	Uri-Path	[RFCXXXX]
12	Content-Format	[RFCXXXX]
14	Max-Age	[RFCXXXX]
15	Uri-Query	[RFCXXXX]
16	Accept	[RFCXXXX]
20	Location-Query	[RFCXXXX]
35	Proxy-Uri	[RFCXXXX]
39	Proxy-Scheme	[RFCXXXX]
128	(Reserved)	[RFCXXXX]
132	(Reserved)	[RFCXXXX]
136	(Reserved)	[RFCXXXX]
140	(Reserved)	[RFCXXXX]

Table 4: CoAP Option Numbers

The IANA policy for future additions to this registry is split into three tiers as follows. The range of 0..255 is reserved for options defined by the IETF (IETF Review or IESG approval). The range of 256..2047 is reserved for commonly used options with public specifications (Specification Required). The range of 2048..64999 is for all other options including private or vendor specific ones, which undergo a Designated Expert review to help ensure that the option semantics are defined correctly. The option numbers between 65000 and 65535 inclusive are reserved for experiments. They are not meant for vendor specific use of any kind and MUST NOT be used in operational deployments.

Option Number	Policy [RFC5226]
0..255	IETF Review or IESG approval
256..2047	Specification Required
2048..64999	Designated Expert
65000..65535	Reserved for experiments

The documentation of an Option Number should specify the semantics of

an option with that number, including the following properties:

- o The meaning of the option in a request.
- o The meaning of the option in a response.
- o Whether the option is critical or elective, as determined by the Option Number.
- o Whether the option is Safe, and, if yes, whether it is part of the Cache-Key, as determined by the Option Number (see Section 5.4.2).
- o The format and length of the option's value.
- o Whether the option must occur at most once or whether it can occur multiple times.
- o The default value, if any. For a critical option with a default value, a discussion on how the default value enables processing by implementations not implementing the critical option (Section 5.4.4).

### 12.3. Content-Format Registry

Internet media types are identified by a string, such as "application/xml" [RFC2046]. In order to minimize the overhead of using these media types to indicate the format of payloads, this document defines a registry for a subset of Internet media types to be used in CoAP and assigns each, in combination with a content-coding, a numeric identifier. The name of the registry is "CoAP Content-Formats".

Each entry in the registry must include the media type registered with IANA, the numeric identifier in the range 0-65535 to be used for that media type in CoAP, the content-coding associated with this identifier, and a reference to a document describing what a payload with that media type means semantically.

CoAP does not include a separate way to convey content-encoding information with a request or response, and for that reason the content-encoding is also specified for each identifier (if any). If multiple content-encodings will be used with a media type, then a separate Content-Format identifier for each is to be registered. Similarly, other parameters related to an Internet media type, such as level, can be defined for a CoAP Content-Format entry.

Initial entries in this registry are as follows:

Media type	Encoding	Id.	Reference
text/plain; charset=utf-8	-	0	[RFC2046][RFC3676][RFC5147]
application/ link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/ octet-stream	-	42	[RFC2045][RFC2046]
application/exi	-	47	[EXIMIME]
application/json	-	50	[RFC4627]

Table 5: CoAP Content-Formats

The identifiers between 65000 and 65535 inclusive are reserved for experiments. They are not meant for vendor specific use of any kind and MUST NOT be used in operational deployments. The identifiers between 256 and 9999 are reserved for future use in IETF specifications (IETF review or IESG approval). All other identifiers are Unassigned.

Because the name space of single-byte identifiers is so small, the IANA policy for future additions in the range 0-255 inclusive to the registry is "Expert Review" as described in [RFC5226]. The IANA policy for additions in the range 10000-64999 inclusive is "First Come First Served" as described in [RFC5226].

In machine to machine applications, it is not expected that generic Internet media types such as text/plain, application/xml or application/octet-stream are useful for real applications in the long term. It is recommended that M2M applications making use of CoAP will request new Internet media types from IANA indicating semantic information about how to create or parse a payload. For example, a Smart Energy application payload carried as XML might request a more specific type like application/se+xml or application/se-exi.

#### 12.4. URI Scheme Registration

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap". The registration request complies with [RFC4395].

URI scheme name.  
coap

Status.

Permanent.

URI scheme syntax.

Defined in Section 6.1 of [RFCXXXX].

URI scheme semantics.

The "coap" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP). The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "http" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.

The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.

The scheme is used by CoAP endpoints to access CoAP resources.

Interoperability considerations.

None.

Security considerations.

See Section 11.1 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

References.

[RFCXXXX]

## 12.5. Secure URI Scheme Registration

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coaps". The registration request complies with [RFC4395].

URI scheme name.

coaps

Status.

Permanent.

URI scheme syntax.

Defined in Section 6.2 of [RFCXXXX].

URI scheme semantics.

The "coaps" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using Datagram Transport Layer Security (DTLS) for transport security. The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "https" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.

The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.

The scheme is used by CoAP endpoints to access CoAP resources using DTLS.

Interoperability considerations.

None.

Security considerations.

See Section 11.1 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

References.

[RFCXXXX]

## 12.6. Service Name and Port Number Registration

One of the functions of CoAP is resource discovery: a CoAP client can ask a CoAP server about the resources offered by it (see Section 7). To enable resource discovery just based on the knowledge of an IP address, the CoAP port for resource discovery needs to be standardized.

IANA has assigned the port number 5683 and the service name "coap", in accordance with [RFC6335].

Besides unicast, CoAP can be used with both multicast and anycast.

Service Name.  
coap

Transport Protocol.  
UDP

Assignee.  
IESG <iesg@ietf.org>

Contact.  
IETF Chair <chair@ietf.org>

Description.  
Constrained Application Protocol (CoAP)

Reference.  
[RFCXXXX]

Port Number.  
5683

#### 12.7. Secure Service Name and Port Number Registration

CoAP resource discovery may also be provided using the DTLS-secured CoAP "coaps" scheme. Thus the CoAP port for secure resource discovery needs to be standardized.

This document requests the assignment of the port number [IANA\_TBD\_PORT] and the service name "coaps", in accordance with [RFC6335].

Besides unicast, DTLS-secured CoAP can be used with anycast.

Service Name.  
coaps

Transport Protocol.  
UDP

Assignee.  
IESG <iesg@ietf.org>



Contact.

IETF Chair <chair@ietf.org>

Description.

DTLS-secured CoAP

Reference.

[RFCXXXX]

Port Number.

[IANA\_TBD\_PORT]

## 12.8. Multicast Address Registration

Section 8, "Multicast CoAP", defines the use of multicast. This document requests the assignment of the following multicast addresses for use by CoAP nodes:

IPv4 -- "All CoAP Nodes" address [TBD1], from the IPv4 Multicast Address Space Registry. As the address is used for discovery that may span beyond a single network, it should come from the Internetwork Control Block (224.0.1.x, RFC 5771).

IPv6 -- "All CoAP Nodes" address [TBD2], from the IPv6 Multicast Address Space Registry, in the Variable Scope Multicast Addresses space (RFC3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only. The address should be of the form FF0x::nn, where nn is a single byte, to ensure good compression of the local-scope address with [RFC6282].

[The explanatory text to be removed upon allocation of the addresses, except for the note about the distinct multicast addresses.]

## 13. Acknowledgements

Special thanks to Peter Bigot, Esko Dijk and Cullen Jennings for substantial contributions to the ideas and text in the document, along with countless detailed reviews and discussions.

Thanks to Ed Beroaset, Angelo P. Castellani, Gilbert Clark, Robert Cragie, Esko Dijk, Lisa Dussealt, Thomas Fossati, Tom Herbst, Richard Kelsey, Ari Keranen, Matthias Kovatsch, Salvatore Loreto, Kerry Lynn, Alexey Melnikov, Guido Moritz, Petri Mutka, Colin O'Flynn, Charles Palmer, Adriano Pezzuto, Robert Quattlebaum, Akbar Rahman, Eric Rescorla, David Ryan, Szymon Sasin, Michael Scharf, Dale Seed, Robby Simpson, Peter van der Stok, Michael Stuber, Linyi Tian, Gilman

Tolle, Matthieu Vial and Alper Yegin for helpful comments and discussions that have shaped the document.

Some of the text has been borrowed from the working documents of the IETF httpbis working group.

## 14. References

### 14.1. Normative References

- [I-D.farrell-decade-ni]  
Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", draft-farrell-decade-ni-10 (work in progress), August 2012.
- [I-D.ietf-tls-oob-pubkey]  
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-06 (work in progress), October 2012.
- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [RFC3676] Gellens, R., "The Text/Plain Format and DelSp Parameters", RFC 3676, February 2004.

- [RFC3986]    Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4279]    Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4395]    Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.
- [RFC5147]    Wilde, E. and M. Duerst, "URI Fragment Identifiers for the text/plain Media Type", RFC 5147, April 2008.
- [RFC5198]    Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5226]    Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5234]    Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5246]    Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5280]    Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5785]    Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC5952]    Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.
- [RFC5988]    Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6066]    Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.
- [RFC6347]    Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

- [RFC6690]    Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

#### 14.2. Informative References

- [EUI64]        "GUIDELINES FOR 64-BIT GLOBAL IDENTIFIER (EUI-64) REGISTRATION AUTHORITY", April 2010, <<http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>>.
- [EXIMIME]      "Efficient XML Interchange (EXI) Format 1.0", December 2009, <<http://www.w3.org/TR/2009/CR-exi-20091208/#mediaTypeRegistration>>.
- [I-D.allman-tcpm-rto-consider]  
Allman, M., "Retransmission Timeout Considerations", draft-allman-tcpm-rto-consider-01 (work in progress), May 2012.
- [I-D.bormann-coap-misc]  
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-21 (work in progress), October 2012.
- [I-D.bormann-core-cross-reverse-convention]  
Bormann, C., "A convention for URIs operating a HTTP-CoAP reverse proxy", draft-bormann-core-cross-reverse-convention-00 (work in progress), December 2012.
- [I-D.bormann-core-ipsec-for-coap]  
Bormann, C., "Using CoAP with IPsec", draft-bormann-core-ipsec-for-coap-00 (work in progress), December 2012.
- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-10 (work in progress), October 2012.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-07 (work in progress), October 2012.
- [I-D.mcgrew-tls-aes-ccm-ecc]  
McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM ECC Cipher Suites for TLS", draft-mcgrew-tls-aes-ccm-ecc-05 (work in progress), July 2012.

- [REST]        Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <[http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.
- [RFC0793]    Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC2818]    Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC3264]    Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [RFC3542]    Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.
- [RFC4492]    Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.
- [RFC4627]    Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC4944]    Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5405]    Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.
- [RFC6120]    Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [RFC6282]    Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, September 2011.
- [RFC6335]    Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.
- [RFC6655]    McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for

Transport Layer Security (TLS)", RFC 6655, July 2012.

## Appendix A. Examples

This section gives a number of short examples with message flows for GET requests. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and multicast.

Figure 15 shows a basic GET request causing a piggy-backed response: The client sends a Confirmable GET request for the resource `coap://server/temperature` to the server with a Message ID of 0x7d34. The request includes one Uri-Path Option (Delta 0 + 11 = 11, Length 11, Value "temperature"); the Token is left empty. This request is a total of 16 bytes long. A 2.05 (Content) response is returned in the Acknowledgement message that acknowledges the Confirmable request, echoing both the Message ID 0x7d34 and the empty Token value. The response includes a Payload of "22.3 C" and is 10 bytes long.

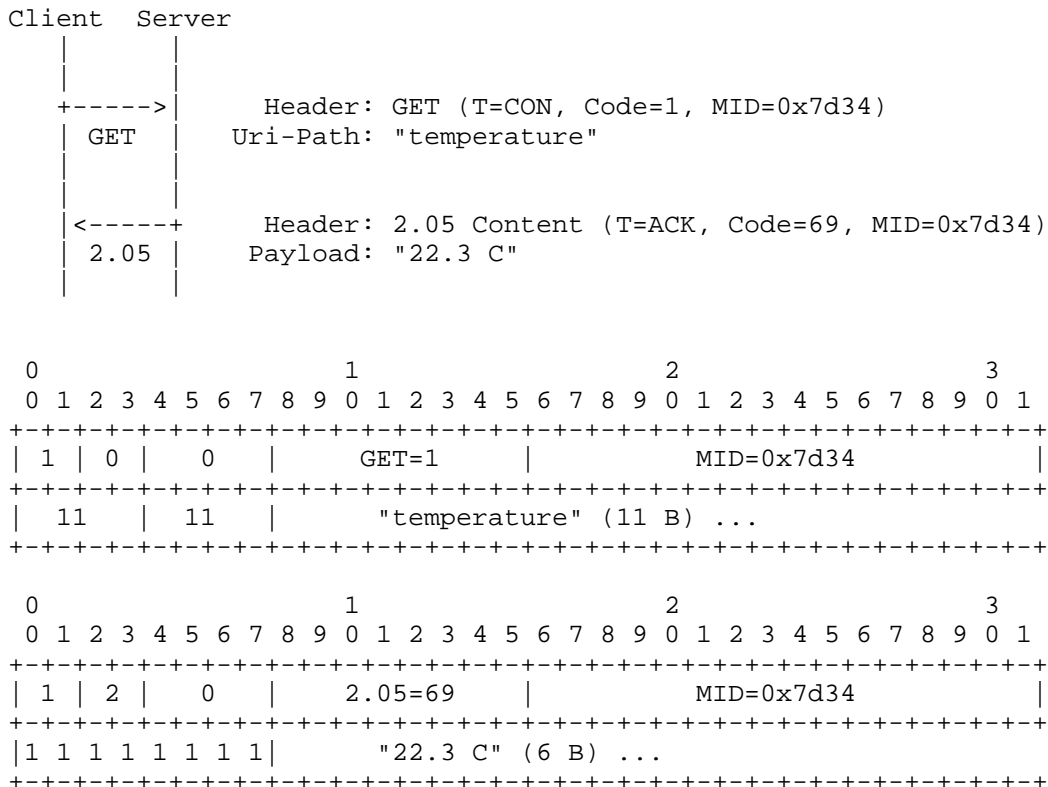


Figure 15: Confirmable request; piggy-backed response

Figure 16 shows a similar example, but with the inclusion of a non-empty Token (Value 0x20) in the request and (Jump 15 + 4 = 19) in the response, increasing the sizes to 18 and 12 bytes, respectively.

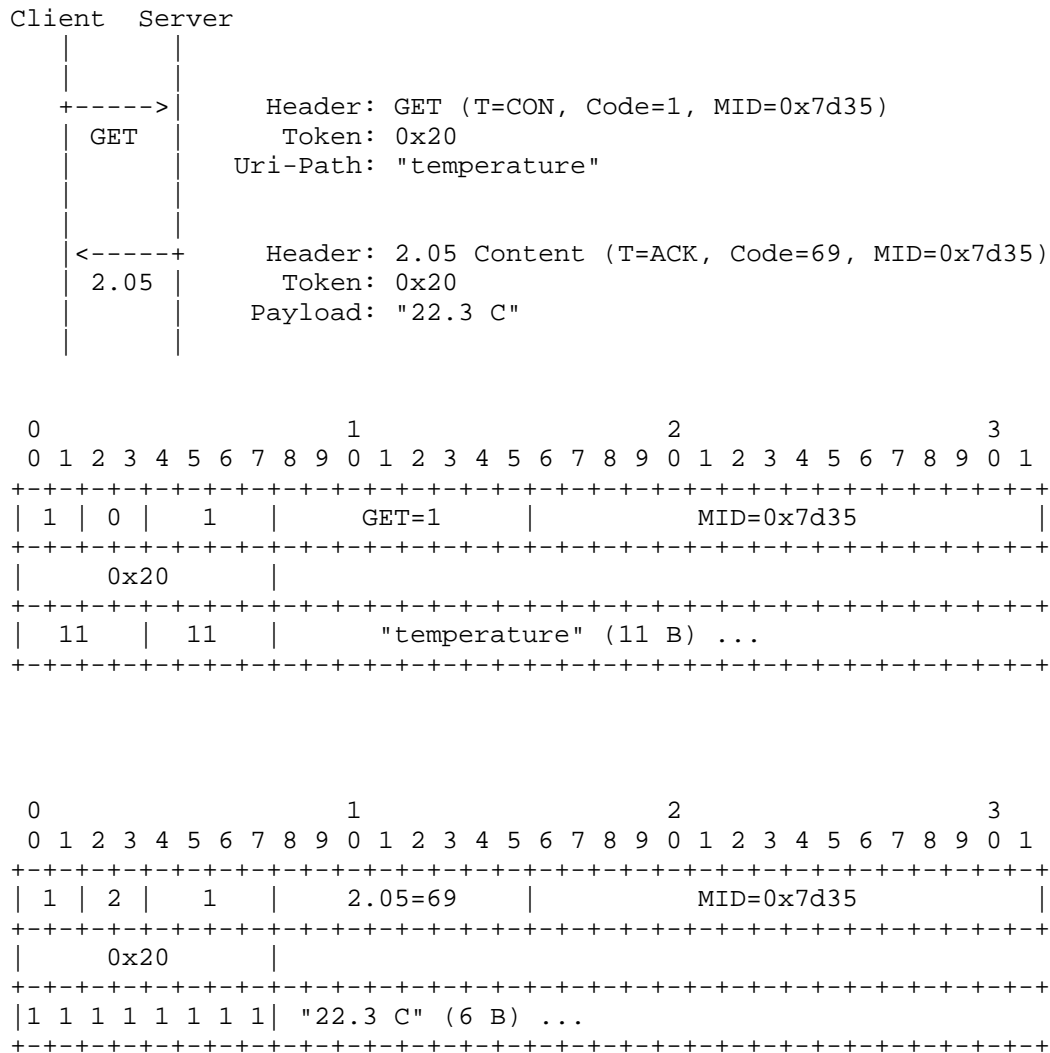


Figure 16: Confirmable request; piggy-backed response

In Figure 17, the Confirmable GET request is lost. After ACK\_TIMEOUT seconds, the client retransmits the request, resulting in a piggy-backed response as in the previous example.

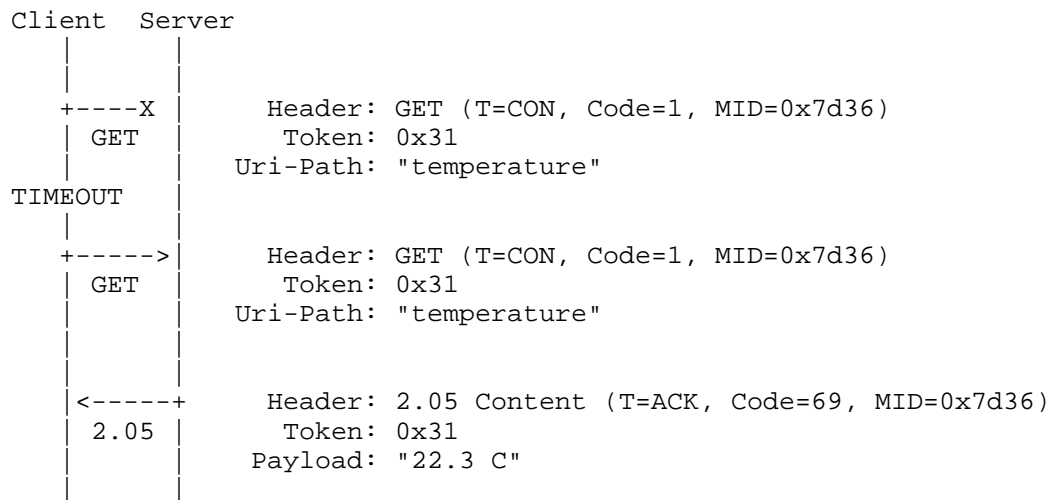


Figure 17: Confirmable request (retransmitted); piggy-backed response

In Figure 18, the first Acknowledgement message from the server to the client is lost. After ACK\_TIMEOUT seconds, the client retransmits the request.

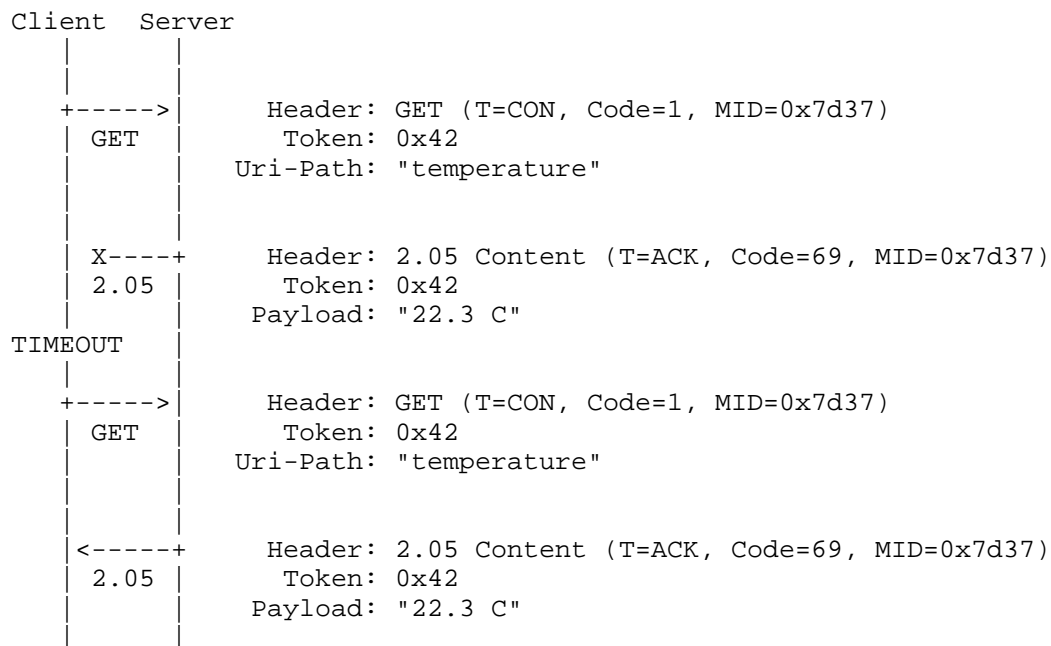
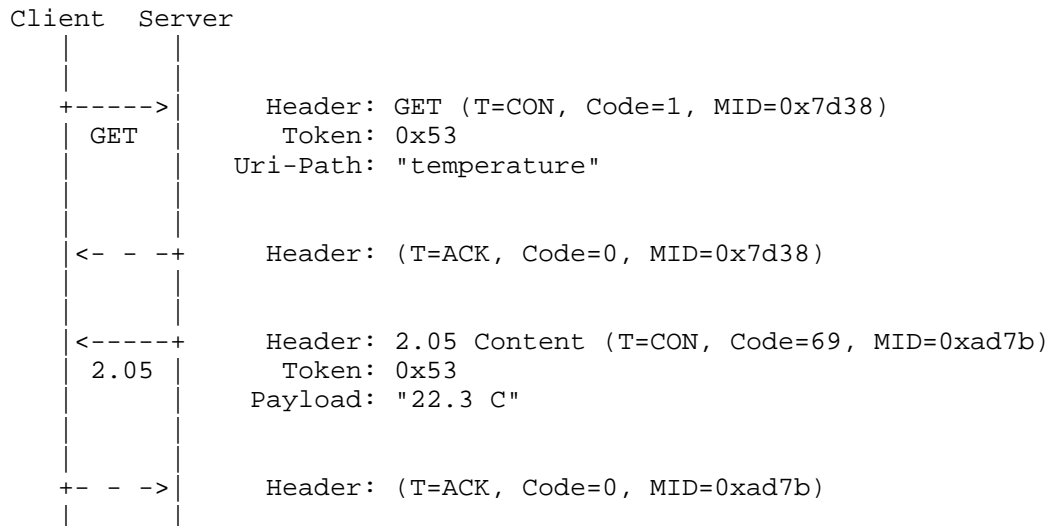


Figure 18: Confirmable request; piggy-backed response (retransmitted)



In Figure 19, the server acknowledges the Confirmable request and sends a 2.05 (Content) response separately in a Confirmable message. Note that the Acknowledgement message and the Confirmable response do not necessarily arrive in the same order as they were sent. The client acknowledges the Confirmable response.



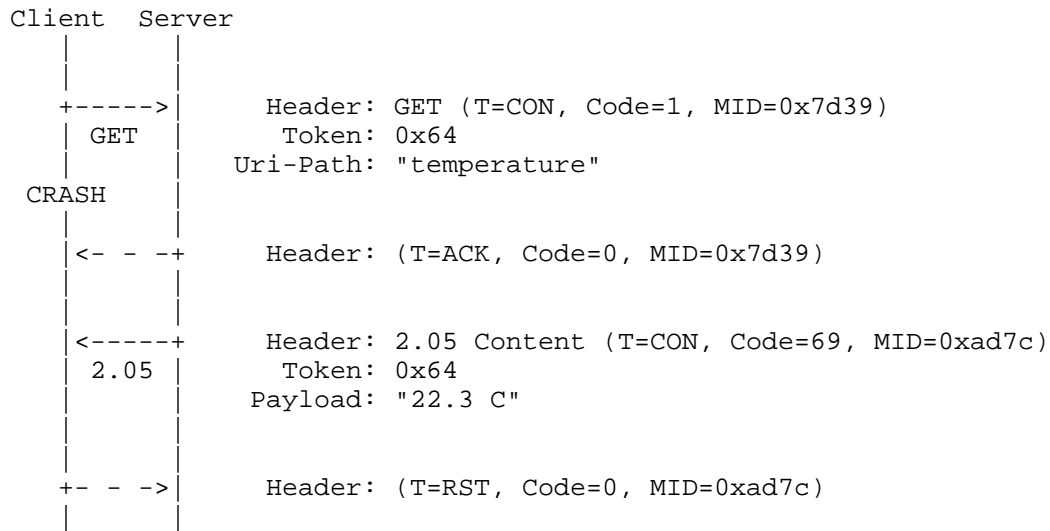


Figure 20: Confirmable request; separate response (unexpected)

Figure 21 shows a basic GET request where the request and the response are non-confirmable, so both may be lost without notice.

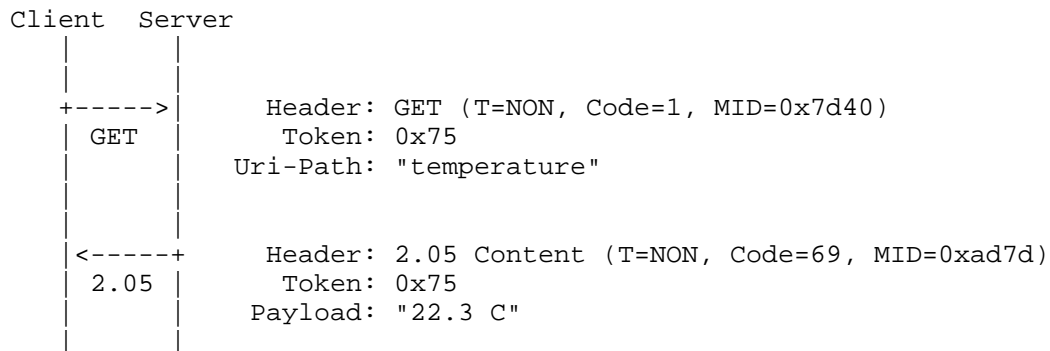


Figure 21: Non-confirmable request; Non-confirmable response

In Figure 22, the client sends a Non-confirmable GET request to a multicast address: all nodes in link-local scope. There are 3 servers on the link: A, B and C. Servers A and B have a matching resource, therefore they send back a Non-confirmable 2.05 (Content) response. The response sent by B is lost. C does not have matching response, therefore it sends a Non-confirmable 4.04 (Not Found) response.

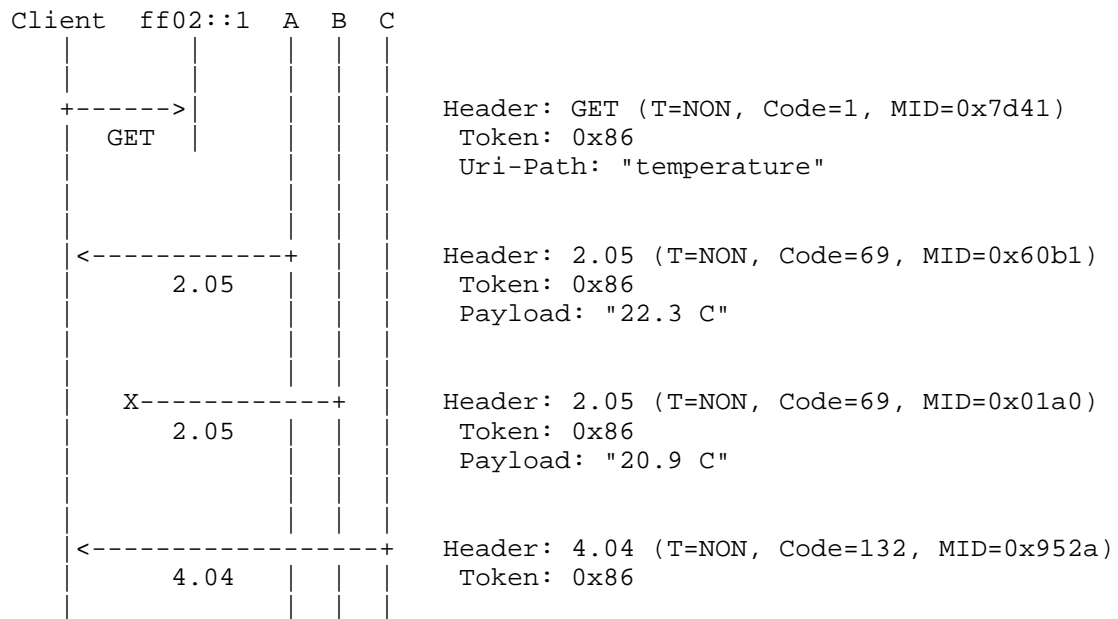


Figure 22: Non-confirmable request (multicast); Non-confirmable response

## Appendix B. URI Examples

The following examples demonstrate different sets of Uri options, and the result after constructing an URI from them.

- o `coap://[2001:db8::2:1]/`  
     Destination IP Address = [2001:db8::2:1]  
     Destination UDP Port = 5683
- o `coap://example.net/`  
     Destination IP Address = [2001:db8::2:1]  
     Destination UDP Port = 5683  
     Uri-Host = "example.net"
- o `coap://example.net/.well-known/core`

Destination IP Address = [2001:db8::2:1]

Destination UDP Port = 5683

Uri-Host = "example.net"

Uri-Path = ".well-known"

Uri-Path = "core"

- o coap://  
xn--18j4d.example/%E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF

Destination IP Address = [2001:db8::2:1]

Destination UDP Port = 5683

Uri-Host = "xn--18j4d.example"

Uri-Path = the string composed of the Unicode characters U+3053 U+3093 U+306b U+3061 U+306f, usually represented in UTF-8 as E38193E38293E381ABE381A1E381AF hexadecimal

- o coap://198.51.100.1:61616//%2F//?%2F%2F&?%26

Destination IP Address = 198.51.100.1

Destination UDP Port = 61616

Uri-Path = ""

Uri-Path = "/"

Uri-Path = ""

Uri-Path = ""

Uri-Query = "//"

Uri-Query = "?&"

## Appendix C. Changelog

Changed from ietf-12 to ietf-13:

- o Simplified message format.

- \* Removed the OC (Option Count) field in the CoAP Header.
- \* Changed the End-of-Options Marker into the Payload Marker.
- \* Changed the format of Options: use 4 bits for option length and delta; insert one or two additional bytes after the option header if necessary.
- \* Promoted the Token Option to a field following the CoAP Header.
- o Clarified when a payload is a diagnostic payload (#264).
- o Moved IPsec discussion to separate draft (#262).
- o Added a reference to a separate draft on reverse-proxy URI embedding (#259).
- o Clarified the use of ETags and of 2.03 responses (#265, #254, #256).
- o Added reserved Location-\* numbers and clarified Location-\*.
- o Added Proxy-Scheme proposal.
- o Clarified terms such as content negotiation, selected representation, representation-format, message format error.
- o Numerous clarifications and a few bugfixes.

Changed from ietf-11 to ietf-12:

- o Extended options to support lengths of up to 1034 bytes (#202).
- o Added new Jump mechanism for options and removed Fenceposting (#214).
- o Added new IANA option number registration policy (#214).
- o Added Proxy Unsafe/Safe and Cache-Key masking to option numbers (#241).
- o Re-numbered option numbers to use Unsafe/Safe and Cache-Key compliant numbers (#241).
- o Defined NSTART and restricted the value to 1 with a MUST (#215).
- o Defined PROBING\_RATE and set it to 1 Byte/second (#215).

- o Defined DEFAULT\_LEISURE (#246).
- o Renamed Content-Type into Content-Format, and Media Type registry into Content-Format registry.
- o A large number of small editorial changes, clarifications and improvements have been made.

Changed from ietf-10 to ietf-11:

- o Expanded section 4.8 on Transmission Parameters, and used the derived values defined there (#201). Changed parameter names to be shorter and more to the point.
- o Several more small editorial changes, clarifications and improvements have been made.

Changed from ietf-09 to ietf-10:

- o Option deltas are restricted to 0 to 14; the option delta 15 is used exclusively for the end-of-options marker (#239).
- o Option numbers that are a multiple of 14 are not reserved, but are required to have an empty default value (#212).
- o Fixed misleading language that was introduced in 5.10.2 in coap-07 re Uri-Host and Uri-Port (#208).
- o Segments and arguments can have a length of zero characters (#213).
- o The Location-\* options describe together describe one location. The location is a relative URI, not an "absolute path URI" (#218).
- o The value of the Location-Path Option must not be '.' or '..' (#218).
- o Added a sentence on constructing URIs from Location-\* options (#231).
- o Reserved option numbers for future Location-\* options (#230).
- o Fixed response codes with payload inconsistency (#233).
- o Added advice on default values for critical options (#207).
- o Clarified use of identifiers in RawPublicKey Mode Provisioning (#222).

- o Moved "Securing CoAP" out of the "Security Considerations" (#229).
- o Added "All CoAP Nodes" multicast addresses to "IANA Considerations" (#216).
- o Over 100 small editorial changes, clarifications and improvements have been made.

Changed from ietf-08 to ietf-09:

- o Improved consistency of statements about RST on NON: RST is a valid response to a NON message (#183).
- o Clarified that the protocol constants can be configured for specific application environments.
- o Added implementation note recommending piggy-backing whenever possible (#182).
- o Added a content-encoding column to the media type registry (#181).
- o Minor improvements to Appendix D.
- o Added text about multicast response suppression (#177).
- o Included the new End-of-options Marker (#176).
- o Added a reference to draft-ietf-tls-oob-pubkey and updated the RPK text accordingly.

Changed from ietf-07 to ietf-08:

- o Clarified matching rules for messages (#175)
- o Fixed a bug in Section 8.2.2 on Etags (#168)
- o Added an IP address spoofing threat analysis contribution (#167)
- o Re-focused the security section on raw public keys (#166)
- o Added an 4.06 error to Accept (#165)

Changed from ietf-06 to ietf-07:

- o application/link-format added to Media types registration (#160)
- o Moved content-type attribute to the document from link-format.

- o Added coaps scheme and DTLS-secured CoAP default port (#154)
- o Allowed 0-length Content-type options (#150)
- o Added congestion control recommendations (#153)
- o Improved text on PUT/POST response payloads (#149)
- o Added an Accept option for content-negotiation (#163)
- o Added If-Match and If-None-Match options (#155)
- o Improved Token Option explanation (#147)
- o Clarified mandatory to implement security (#156)
- o Added first come first server policy for 2-byte Media type codes (#161)
- o Clarify matching rules for messages and tokens (#151)
- o Changed OPTIONS and TRACE to always return 501 in HTTP-CoAP mapping (#164)

Changed from ietf-05 to ietf-06:

- o HTTP mapping section improved with the minimal protocol standard text for CoAP-HTTP and HTTP-CoAP forward proxying (#137).
- o Eradicated percent-encoding by including one Uri-Query Option per &-delimited argument in a query.
- o Allowed RST message in reply to a NON message with unexpected token (#135).
- o Cache Invalidation only happens upon successful responses (#134).
- o 50% jitter added to the initial retransmit timer (#142).
- o DTLS cipher suites aligned with ZigBee IP, DTLS clarified as default CoAP security mechanism (#138, #139)
- o Added a minimal reference to draft-kivinen-ipsecme-ikev2-minimal (#140).
- o Clarified the comparison of UTF-8s (#136).



- o Minimized the initial media type registry (#101).

Changed from ietf-04 to ietf-05:

- o Renamed Immediate into Piggy-backed and Deferred into Separate -- should finally end the confusion on what this is about.
- o GET requests now return a 2.05 (Content) response instead of 2.00 (OK) response (#104).
- o Added text to allow 2.02 (Deleted) responses in reply to POST requests (#105).
- o Improved message deduplication rules (#106).
- o Section added on message size implementation considerations (#103).
- o Clarification made on human readable error payloads (#109).
- o Definition of CoAP methods improved (#108).
- o Max-Age removed from requests (#107).
- o Clarified uniqueness of tokens (#112).
- o Location-Query Option added (#113).
- o ETag length set to 1-8 bytes (#123).
- o Clarified relation between elective/critical and option numbers (#110).
- o Defined when to update Version header field (#111).
- o URI scheme registration improved (#102).
- o Added review guidelines for new CoAP codes and numbers.

Changes from ietf-03 to ietf-04:

- o Major document reorganization (#51, #63, #71, #81).
- o Max-age length set to 0-4 bytes (#30).
- o Added variable unsigned integer definition (#31).

- o Clarification made on human readable error payloads (#50).
- o Definition of POST improved (#52).
- o Token length changed to 0-8 bytes (#53).
- o Section added on multiplexing CoAP, DTLS and STUN (#56).
- o Added cross-protocol attack considerations (#61).
- o Used new Immediate/Deferred response definitions (#73).
- o Improved request/response matching rules (#74).
- o Removed unnecessary media types and added recommendations for their use in M2M (#76).
- o Response codes changed to base 32 coding, new Y.XX naming (#77).
- o References updated as per AD review (#79).
- o IANA section completed (#80).
- o Proxy-Uri Option added to disambiguate between proxy and non-proxy requests (#82).
- o Added text on critical options in cached states (#83).
- o HTTP mapping sections improved (#88).
- o Added text on reverse proxies (#72).
- o Some security text on multicast added (#54).
- o Trust model text added to introduction (#58, #60).
- o AES-CCM vs. AES-CCB text added (#55).
- o Text added about device capabilities (#59).
- o DTLS section improvements (#87).
- o Caching semantics aligned with RFC2616 (#78).
- o Uri-Path Option split into multiple path segments.
- o MAX\_RETRANSMIT changed to 4 to adjust for RESPONSE\_TIME = 2.

Changes from ietf-02 to ietf-03:

- o Token Option and related use in asynchronous requests added (#25).
- o CoAP specific error codes added (#26).
- o Erroring out on unknown critical options changed to a MUST (#27).
- o Uri-Query Option added.
- o Terminology and definitions of URIs improved.
- o Security section completed (#22).

Changes from ietf-01 to ietf-02:

- o Sending an error on a critical option clarified (#18).
- o Clarification on behavior of PUT and idempotent operations (#19).
- o Use of Uri-Authority clarified along with server processing rules; Uri-Scheme Option removed (#20, #23).
- o Resource discovery section removed to a separate CoRE Link Format draft (#21).
- o Initial security section outline added.

Changes from ietf-00 to ietf-01:

- o New cleaner transaction message model and header (#5).
- o Removed subscription while being designed (#1).
- o Section 2 re-written (#3).
- o Text added about use of short URIs (#4).
- o Improved header option scheme (#5, #14).
- o Date option removed while being designed (#6).
- o New text for CoAP default port (#7).
- o Completed proxying section (#8).
- o Completed resource discovery section (#9).

- o Completed HTTP mapping section (#10).
- o Several new examples added (#11).
- o URI split into 3 options (#12).
- o MIME type defined for link-format (#13, #16).
- o New text on maximum message size (#15).
- o Location Option added.

Changes from shelby-01 to ietf-00:

- o Removed the TCP binding section, left open for the future.
- o Fixed a bug in the example.
- o Marked current Sub/Notify as (Experimental) while under WG discussion.
- o Fixed maximum datagram size to 1280 for both IPv4 and IPv6 (for CoAP-CoAP proxying to work).
- o Temporarily removed the Magic Byte header as TCP is no longer included as a binding.
- o Removed the Uri-code Option as different URI encoding schemes are being discussed.
- o Changed the rel= field to desc= for resource discovery.
- o Changed the maximum message size to 1024 bytes to allow for IP/UDP headers.
- o Made the URI slash optimization and method idempotence MUSTs
- o Minor editing and bug fixing.

Changes from shelby-00 to shelby-01:

- o Unified the message header and added a notify message type.
- o Renamed methods with HTTP names and removed the NOTIFY method.
- o Added a number of options field to the header.

- o Combines the Option Type and Length into an 8-bit field.
- o Added the magic byte header.
- o Added new ETag Option.
- o Added new Date Option.
- o Added new Subscription Option.
- o Completed the HTTP Code - CoAP Code mapping table appendix.
- o Completed the Content-type Identifier appendix and tables.
- o Added more simplifications for URI support.
- o Initial subscription and discovery sections.
- o A Flag requirements simplified.

#### Authors' Addresses

Zach Shelby  
Sensinode  
Kidekuja 2  
Vuokatti 88600  
Finland

Phone: +358407796297  
Email: zach@sensinode.com

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

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

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

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

Brian Frank  
SkyFoundry  
Richmond, VA  
USA

Phone:  
Email: brian@skyfoundry.com



CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 29, 2013

K. Hartke  
Universitaet Bremen TZI  
February 25, 2013

Observing Resources in CoAP  
draft-ietf-core-observe-08

Abstract

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

Editor's Note

This is an interim revision which will receive further modifications during the resolution of open tickets, in particular #204 and #281.

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 August 29, 2013.

Copyright Notice

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



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

## Table of Contents

1. Introduction . . . . .	3
1.1. Background . . . . .	3
1.2. Protocol Overview . . . . .	3
1.3. Requirements Notation . . . . .	6
2. The Observe Option . . . . .	6
3. Client-side Requirements . . . . .	7
3.1. Request . . . . .	7
3.2. Notifications . . . . .	7
3.3. Caching . . . . .	8
3.4. Reordering . . . . .	9
3.5. Transmission . . . . .	10
3.6. Cancellation . . . . .	10
4. Server-side Requirements . . . . .	10
4.1. Request . . . . .	10
4.2. Notifications . . . . .	11
4.3. Caching . . . . .	12
4.4. Reordering . . . . .	13
4.5. Transmission . . . . .	13
5. Intermediaries . . . . .	15
6. Blockwise Transfers . . . . .	16
7. Web Linking . . . . .	16
8. Security Considerations . . . . .	17
9. IANA Considerations . . . . .	17
10. Acknowledgements . . . . .	17
11. References . . . . .	18
11.1. Normative References . . . . .	18
11.2. Informative References . . . . .	18
Appendix A. Examples . . . . .	19
A.1. Proxying . . . . .	23
A.2. Blockwise Transfer . . . . .	25
Appendix B. Modeling Resources to Tailor Notifications . . . . .	26
Appendix C. Changelog . . . . .	26
Author's Address . . . . .	31

## 1. Introduction

### 1.1. Background

CoAP [I-D.ietf-core-coap] is an application protocol for constrained nodes and networks. It is intended to provide RESTful services [REST] not unlike HTTP [RFC2616] while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

The model of REST is that of a client exchanging representations of resources with a server. A representation captures the current or intended state of a resource. The server is the definitive source for representations of the resources in its namespace. A client interested in the state of a resource initiates a request to the server; the server then returns a response with a representation of the resource that is current at the time of the request.

This model does not work well when a client is interested in having a current representation of a resource over a period of time. Existing approaches from HTTP, such as repeated polling or HTTP long polling [RFC6202], generate significant complexity and/or overhead and thus are less applicable in a constrained environment.

The protocol specified in this document extends the CoAP core protocol with a mechanism for a CoAP client to "observe" a resource on a CoAP server: the client can retrieve a representation of the resource and keep this representation updated by the server over a period of time.

The protocol keeps the architectural properties of REST. It enables high scalability and efficiency through the support of caches and proxies. There is no intention, 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, it must register separately for all of them.

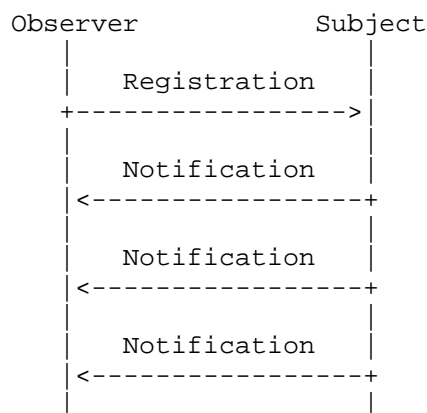


Figure 1: The Observer Design Pattern

The observer design pattern is realized in CoAP as follows:

**Subject:** In the context of CoAP, the subject is a resource in the namespace of a CoAP server. The state of the resource can change over time, ranging from infrequent updates to continuous state transformations.

**Observer:** An observer is a CoAP client that is interested in having a current representation of the resource at any given time.

**Registration:** A client registers its interest in a resource by initiating an extended GET request to the server. In addition to returning a representation of the target resource, this request causes the server to add the client to the list of observers of the resource.

**Notification:** Whenever the state of a resource changes, the server notifies each client in the list of observers of the resource. Each notification is an additional CoAP response sent by the server in reply to the GET request and includes a complete, updated representation of the new resource state.

Figure 2 below shows an example of a CoAP client registering its interest in a resource and receiving three notifications: the first upon registration with the current state, and then two upon two 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 provides a sequence number for reordering detection. All notifications carry the token specified by the client in the request, so the client can easily correlate them to the request.

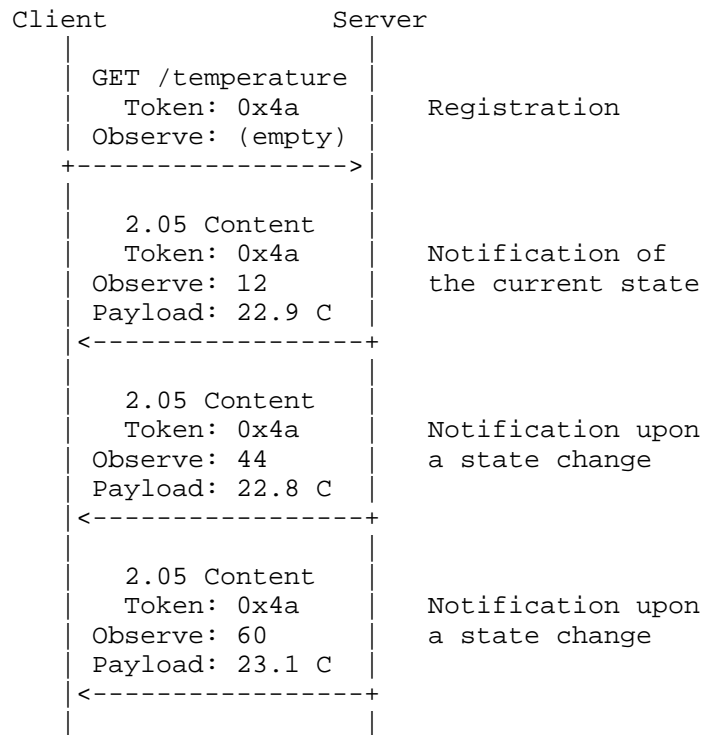


Figure 2: Observing a Resource in CoAP

The server is the authority for determining under what conditions resources change their state and how often observers are notified. The protocol does not offer explicit means for setting up triggers, thresholds or other conditions; it is up to the server to expose observable resources that change their state in a way that is useful in the application context. Resources can be parameterized to achieve similar effects, though; see Appendix B for examples.

A client remains on the list of observers as long as the server can determine the client's continued interest in the resource. The interest is determined from the client's acknowledgement of notifications sent in confirmable messages by the server. If the client actively rejects a notification or if the transmission of a notification ultimately fails, then the client is assumed to be no longer interested and is removed from the list of observers.

While a client is in the list of observers of a resource, it is the goal of the protocol to keep the resource state observed by the client as closely in sync with the actual state at the server as possible.

Becoming out of sync at times cannot be avoided: First, there is always some latency between the change of the resource state and the receipt of the notification. Second, messages with notifications can get lost, which will cause the client assume an old state until it receives the next notification. And third, the server may erroneously come to the conclusion that the client is no longer interested in the resource, which will cause it to stop sending notifications and the client to assume an old state until it registers its interest again.

The protocol addresses this issue as follows:

- o It follows a best-effort approach for sending the current representation to the client after a state change: Clients should see the new state after a state change as soon as possible, and they should see as many states as possible. However, a client cannot rely on observing every single state that a resource goes through.
- o It labels notifications with a maximum duration up to which it is acceptable for the observed state and the actual state to be out of sync. When the age of the notification received reaches this maximum, the client cannot use the enclosed representation until it has received a new notification.
- o It is designed on the principle of eventual consistency: The protocol guarantees that, if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the latest resource state.

### 1.3. Requirements Notation

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

## 2. The Observe Option

No.	C	U	N	R	Name	Format	Length	Default
6		x	-		Observe	empty/uint	0 B/0-3 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Table 1: The Observe Option

The Observe Option, when present in a request, extends the GET method so it does not only retrieve a current representation of the target resource, but also requests the server to add the client to the list of observers of the resource. The exact semantics are defined in the following sections. The value of the option in a request **MUST** be empty on transmission and **MUST** be ignored on reception.

The Observe Option is not critical for processing the request. If the server is unwilling or unable to add the client to the list of observers of the resource identified by the request URI, then the request falls back to a normal GET request.

In a response, the Observe Option identifies the message as a notification. This implies that the server has added the client to the list of observers and that it will notify the client of changes to the resource state. The value of the option is a 24-bit sequence number for reordering detection; see Section 3.4 and Section 4.4 for the client- and server-side respectively. The sequence number is encoded in network byte order using a variable number of bytes, as specified in Section 3.2 of RFC XXXX [I-D.ietf-core-coap].

The Observe Option is not part of the cache-key: a cacheable response obtained with an Observe Option in the request can be used to satisfy a request without an Observe Option, and vice versa. When a stored response with an Observe Option is used to satisfy a normal GET request, the option **MUST** be removed before the response is returned to the client.

### 3. Client-side Requirements

#### 3.1. Request

A client can register its interest in a resource by issuing a GET request that includes an empty Observe Option. If the server returns a 2.xx response that includes an Observe Option as well, the server has added the client successfully to the list of observers of the target resource and the client will be notified of changes to the resource state.

#### 3.2. Notifications

Notifications are additional responses sent by the server in reply to the GET request. Each notification includes the token specified by the client in the GET request, an Observe Option with a sequence number for reordering detection (see Section 3.4) and a payload in the same Content-Format as the initial response.

Notifications have a 2.05 (Content) response code, or a 2.03 (Valid) response code if the client has included one or more ETag Options in the request (see Section 3.3). In the event that the resource changes in a way that would cause a normal GET request at that time to return a non-2.xx response (for example, when the resource is deleted), the server sends a notification with a matching response code and removes the client from the list of observers.

### 3.3. Caching

As notifications are just additional responses to a GET request, notifications partake in caching as defined in Section 5.6 of RFC XXXX [I-D.ietf-core-coap]. Both the freshness model and the validation model are supported.

#### 3.3.1. Freshness

A client MAY store a notification like a response in its cache and use a stored notification that is fresh without contacting the origin 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 if 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 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 use the enclosed representation until it is validated or a new notification is received.

To make sure it has a current representation and/or to re-register its interest in a resource, a client MAY issue a new GET request with an Observe Option at any time. The client SHOULD specify a new token in the GET request, as the token serves as an epoch identifier for the sequence numbers in the Observe Option (see Section 3.4).

It is RECOMMENDED that the client does not issue the request while it still has a fresh notification/response for a resource in its cache. Additionally, the client SHOULD wait for a random amount of time between 5 and 15 seconds before issuing the new request to avoid synchronicity with other clients.

### 3.3.2. Validation

When a client has one or more notifications stored in its cache for a resource, it can use the ETag Option in the GET request to give the server an opportunity to select a stored notification to be used.

The client MAY include an ETag Option for each stored response that is applicable in the GET request. Whenever the observed resource changes to a representation identified by one of the ETag Options, the server can select a stored response by sending a 2.03 (Valid) notification with an appropriate ETag Option instead of a 2.05 (Content) notification. The client needs to keep all candidate responses in its cache until it is no longer interested in the target resource or it issues a new GET request with a new set of entity-tags.

### 3.4. Reordering

Messages with notifications can arrive in a different order than they were sent. Since the goal is to keep the observed state as closely in sync with the actual state as possible, a client SHOULD NOT update the observed state with a notification that arrives later than a newer notification.

For reordering detection, the server sets the value of the Observe Option in each notification to a 24-bit sequence number. An incoming notification is newer than the newest notification received so far when one of the following conditions is met:

$$\begin{aligned} & (V1 < V2 \text{ and } V2 - V1 < 2^{23}) \text{ or} \\ & (V1 > V2 \text{ and } V1 - V2 > 2^{23}) \text{ or} \\ & (T2 > T1 + 128 \text{ seconds}) \end{aligned}$$

where V1 is the value of the Observe Option of the newest notification received so far, V2 the value of the Observe Option of the incoming notification, T1 a client-local timestamp of the newest notification received so far, and T2 a client-local timestamp of the incoming notification.

Design Note: The first two conditions verify that V1 is less than V2 in 24-bit sequence 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 sequence number; in other words, after 128 seconds have elapsed without any notification, a client does not need to check the sequence numbers in order to assume an incoming notification is new.



### 3.5. Transmission

A notification can be confirmable or non-confirmable, i.e., be sent in a confirmable or a non-confirmable message. The message type used is independent from the type used for the request or 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 client from the list of observers.

### 3.6. Cancellation

When a client rejects a confirmable notification with a Reset message or when it issues a GET request without an Observe Option for a currently observed resource, the server will remove the client from the list of observers of this resource. The client **MAY** use either method to indicate that it is no longer interested in receiving further notifications for the resource until it eventually registers again.

When a client rejects non-confirmable notification, the server may also (but is not required to) remove the client from the list of observers of this resource. The client **MAY** try this method as well, and **MAY** rate-limit the Reset messages it sends if the server appears to persistently ignore them.

Implementation Note: A client that does not mediate all its requests through its cache might inadvertently cancel an observation by making an unrelated GET to the same resource. To avoid this, without incurring a need for synchronization, such clients can use a different source endpoint for these unrelated GET requests.

## 4. Server-side Requirements

### 4.1. Request

A GET request that includes an Observe Option requests the server not only to return a current representation of the resource identified by

the request URI, but also to add the client to the list of observers of the target resource. If no error occurs, the server MUST return a 2.05 (Content) response with the representation of the current resource state and MUST notify the client of subsequent changes to the state as long as the client is on the list of observers.

A server that is unable or unwilling to add the client to the list of observers of the target resource MAY silently ignore the Observe Option and process the GET request as usual. The resulting response MUST NOT include an Observe Option, the absence of which signals to the client that it will not be notified of changes to the resource and, e.g., needs to poll the resource for its state instead.

If the client is already on the list of observers, the server MUST NOT add it a second time but MUST replace or update the existing entry. If the server receives a GET request for the a resource that does not include an Observe Option, the server MUST remove any existing entry from the list of observers.

Two requests relate to the same list entry if and only if both the request URI and the source endpoint of the requests are the same. Message IDs, tokens and other options are not taken into account.

Any request with an unrecognized critical option or a method other than GET MUST NOT have a direct effect on a list of observers of a resource. However, a non-GET request can have the indirect consequence of causing the server to send a non-2.xx notification which does affect the list of observers (for example, when a DELETE request is successful and the observed resource no longer exists).

#### 4.2. Notifications

A client is notified of changes to the resource state by additional responses sent by the server in reply to the GET request. Each such notification response (including the initial response) MUST include an Observe Option and MUST echo the token specified by the client in the GET request. If there are multiple clients on the list of observers, the order in which they 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 a matching response code and MUST remove the client from the list of observers of the resource.

The Content-Format used in a notification MUST be the same as the one used in the initial response to the GET request. If the server is unable to continue sending notifications using this Content-Format, it SHOULD send a notification with a 5.00 (Internal Server Error) response code and MUST remove the client from the list of observers of the resource.

#### 4.3. Caching

As notifications are just additional responses sent by the server, they are subject to caching as defined in Section 5.6 of RFC XXXX [I-D.ietf-core-coap].

##### 4.3.1. Freshness

After returning the initial response, the server MUST try to keep the returned representation current, i.e., keep the resource state observed by the client as closely in sync with the actual resource state as possible.

Since becoming out of sync at times cannot be avoided, the server MUST indicate for each representation an age up to which it is acceptable that the observed state and the actual state are inconsistent. This age is application-dependent and MUST be specified in notifications using the Max-Age Option.

When the resource does not change and the client has a current representation, the server does not need to send a notification. However, if the client does not receive a notification, it cannot tell if the observed state and the actual state are still in sync. So, when the the age of the latest notification becomes greater than its indicated Max-Age, then the client will assume that the states are inconsistent until the representation is validated or a new notification is received. The server MAY wish to prevent that by sending a notification with the unchanged representation before Max-Age expires.

##### 4.3.2. Validation

A client can include a set of entity-tags in its request using the ETag Option. When a observed resource changes its state and the origin server is about to send a 2.05 (Content) notification, then, whenever that notification has an entity-tag in the set of entity-tags specified by the client, the server MAY send a 2.03 (Valid) response with an appropriate ETag Option instead. The server MUST NOT assume that the client has any response stored other than those identified by the entity-tags in the most recent GET request received for the resource.

#### 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  $2^{24}$  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 for the same resource with the same token. The value of the Observe Option **MUST** be current at the time of transmission; if a notification is retransmitted, the server **MUST** update value of the Observe Option before sending the message.

Implementation Note: A simple implementation that satisfies the requirements is to obtain a timestamp from a local clock. The sequence number then is the timestamp in ticks, where 1 tick =  $(256 \text{ seconds}) / (2^{24}) = 15.26 \text{ microseconds}$ . It is not necessary that the clock reflects the current time/date or that it ticks in a precisely periodical way.

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  $2^{24}$  times in the next 256 seconds after any state change). This alleviates the need to update the value of the Observe Option in a message when the resource state did not change.

Design Note: The choice of a 24-bit option value and a time span of 256 seconds allows for a notification rate of up to 65536 notifications per second. 64K ought to be enough for anybody.

#### 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.

The acknowledgement of a confirmable notification signals to the server that the client is interested in receiving further notifications. If a client rejects a confirmable notification with a Reset message, the client is no longer interested and the server **MUST** remove the client from the list of observers. If the client rejects a non-confirmable notification, the server **MAY** remove the client from the list of observers as well. (It is expected that the server does remove the client if it has the information available that is needed to match the Reset message to the non-confirmable notification, but the server is not required to keep this information.)

At a minimum, the server **MUST** send a notification in a confirmable message instead of a non-confirmable message at least every 24 hours.

A server **MAY** choose to skip a notification if it knows that it will send another notification soon (e.g., when the state is changing frequently). Similarly, it **MAY** choose to send a notification more than once. 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 after the burst by repeating the last notification in a confirmable message.

The server **MUST** limit the number of confirmable notifications for which an acknowledgement has not been received yet to **NSTART** (see Section 4.7 of RFC XXXX [I-D.ietf-core-coap]), and it **SHOULD NOT** send more than one non-confirmable notification every 3 seconds on average.

When the state of an observed resource changes while the server is still waiting for a confirmable notification to be acknowledged or the 3 seconds for a non-confirmable notification to elapse, then the server **MUST** proceed as follows:

1. Wait for the current transmission attempt to complete.
2. If the result is a Reset message or the transmission was the last attempt to deliver a notification, remove the client from the list of observers of the observed resource.
3. If the client is still in the list of observers, transmit a notification with a representation of the current resource state. Should the resource have changed its state more than once in the meantime, skip the notifications for the intermediate states.
4. If the previously completed transmission timed out, increment the retransmission counter and double the timeout; otherwise, reinitialize the retransmission counter and the timeout.

If CoAP is used over a connection-oriented or session-based transport such as DTLS, the server **MUST** remove the client from the list of observers when the connection or session is closed.

## 5. Intermediaries

A client may be interested in a resource in the namespace of an origin server that is reached through a chain of one or more CoAP-to-CoAP intermediaries. In this case, the client registers its interest with the first intermediary towards the origin server, acting as if it was communicating with the origin server itself as specified in Section 3. It is the task of this intermediary to provide the client with a current representation of the target resource and send notifications upon changes to the target resource state, much like an origin server as specified in Section 4.

To perform this task, the intermediary **SHOULD** make use of the protocol specified in this document, taking the role of the client and registering its own interest in the target resource with the next hop towards the origin server. If the next hop does not return a response with an Observe Option, the intermediary **MAY** resort to polling the next hop or **MAY** itself return a response without an Observe Option.

The communication between each pair of hops is independent; each hop in the server role **MUST** determine individually how many notifications to send, of which message type, and so on. Each hop **MUST** generate its own values for the Observe Option, and **MUST** set the value of the Max-Age Option according to the age of the local current representation.

Because a client (or an intermediary in the client role) can only be once on the list of observers of a resource on a server (or an intermediary in the server role) -- it is useless to observe the same resource multiple times -- an intermediary **MUST** observe a resource only once, even if there are multiple clients for which it observes the resource.

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.1 for examples.

## 6. Blockwise Transfers

Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. CoAP provides a blockwise transfer mechanism to address this problem [I-D.ietf-core-block]. The following rules apply to the combination of blockwise transfers with notifications.

As with basic GET transfers, the client can indicate its desired block size in a Block2 Option in the GET request. If the server supports blockwise transfers, it SHOULD take note of the block size for all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the server receives a new GET request for the resource from the client).

When sending a 2.05 (Content) notification, the server always sends all blocks of the representation, suitably sequenced by its congestion control mechanism, even if only some of the blocks have changed with respect to a previous notification. The server performs the blockwise transfer by making use of the Block2 Option in each block. When reassembling representations that are transmitted in multiple blocks, the client MUST NOT combine blocks carrying different Observe Option values.

Blockwise transfers of notifications MUST use confirmable messages and MUST NOT use non-confirmable messages.

See Appendix A.2 for an example.

## 7. 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.

## 8. Security Considerations

The security considerations of RFC XXXX [I-D.ietf-core-coap] apply.

The considerations about amplification attacks are somewhat amplified when observing resources. Without client authentication, a server MUST therefore strictly limit the number of notifications that it sends between receiving acknowledgements that confirm the actual interest of the client in the data; i.e., any notifications sent in non-confirmable messages MUST be interspersed with confirmable messages. (An attacker may still spoof the acknowledgements if the confirmable messages are sufficiently predictable.)

As with any protocol that creates state, attackers may attempt to exhaust the resources that the server has available for maintaining the list of observers for each resource. Servers may want to access-control this creation of state. As degraded behavior, the server can always fall back to processing the request as a normal GET request (without an Observe Option) if it is unwilling or unable to add a client to the list of observers of a resource, including if system resources are exhausted or nearing exhaustion.

Intermediaries must be careful to ensure that notifications cannot be employed to create a loop. A simple way to break any loops is to employ caches for forwarding notifications in intermediaries.

## 9. IANA Considerations

The following entry is added to the CoAP Option Numbers registry:

Number	Name	Reference
6	Observe	[RFCXXXX]

## 10. Acknowledgements

Carsten Bormann was an original author of this draft and is acknowledged for significant contribution to this document.

Thanks to Daniele Alessandrelli, Jari Arkko, Peter Bigot, Angelo P. Castellani, Gilbert Clark, Esko Dijk, Thomas Fossati, Brian Frank, Jeroen Hoebeke, Cullen Jennings, Matthias Kovatsch, Salvatore Loreto, Charles Palmer, Zach Shelby and Floris Van den Abeele for helpful comments and discussions that have shaped the document.



## 11. References

### 11.1. Normative References

- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",  
draft-ietf-core-block-10 (work in progress), October 2012.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-13 (work in progress), December 2012.
- [RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982,  
August 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

### 11.2. Informative References

- [GOF] Gamma, E., Helm, R., Johnson, R., and J. Vlissides,  
"Design Patterns: Elements of Reusable Object-Oriented  
Software", Addison-Wesley, Reading, MA, USA,  
November 1994.
- [REST] Fielding, R., "Architectural Styles and the Design of  
Network-based Software Architectures", Ph.D. Dissertation,  
University of California, Irvine, 2000, <[http://  
www.ics.uci.edu/~fielding/pubs/dissertation/  
fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,  
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext  
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC5989] Roach, A., "A SIP Event Package for Subscribing to Changes  
to an HTTP Resource", RFC 5989, October 2010.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins,  
"Known Issues and Best Practices for the Use of Long  
Polling and Streaming in Bidirectional HTTP", RFC 6202,  
April 2011.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link  
Format", RFC 6690, August 2012.

## Appendix A. Examples

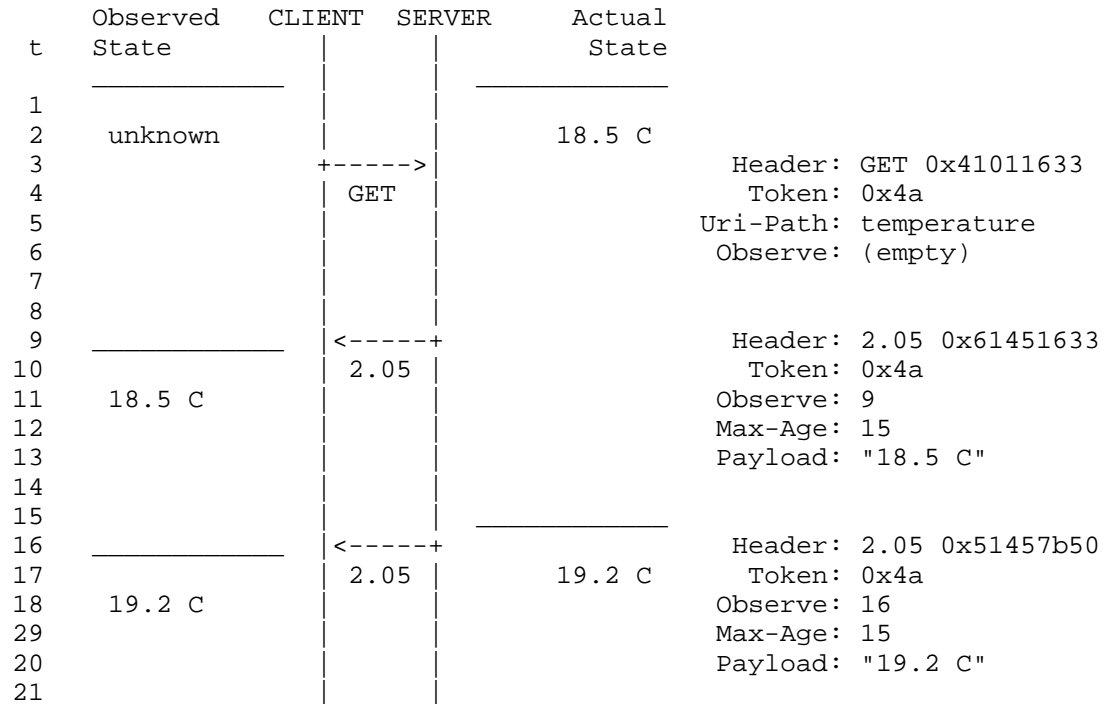


Figure 3: A client registers and receives one notification of the current state and one of a new state upon a state change

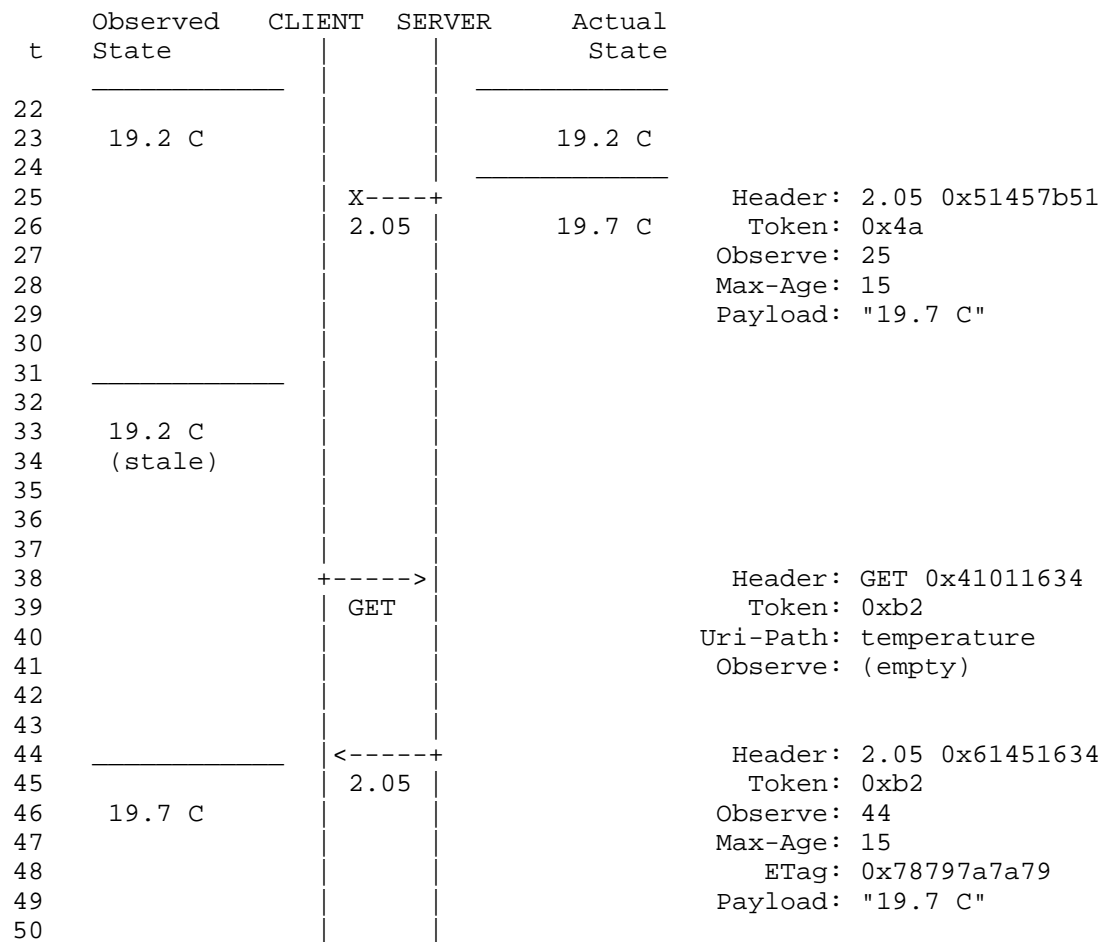


Figure 4: The client re-registers after Max-Age ends

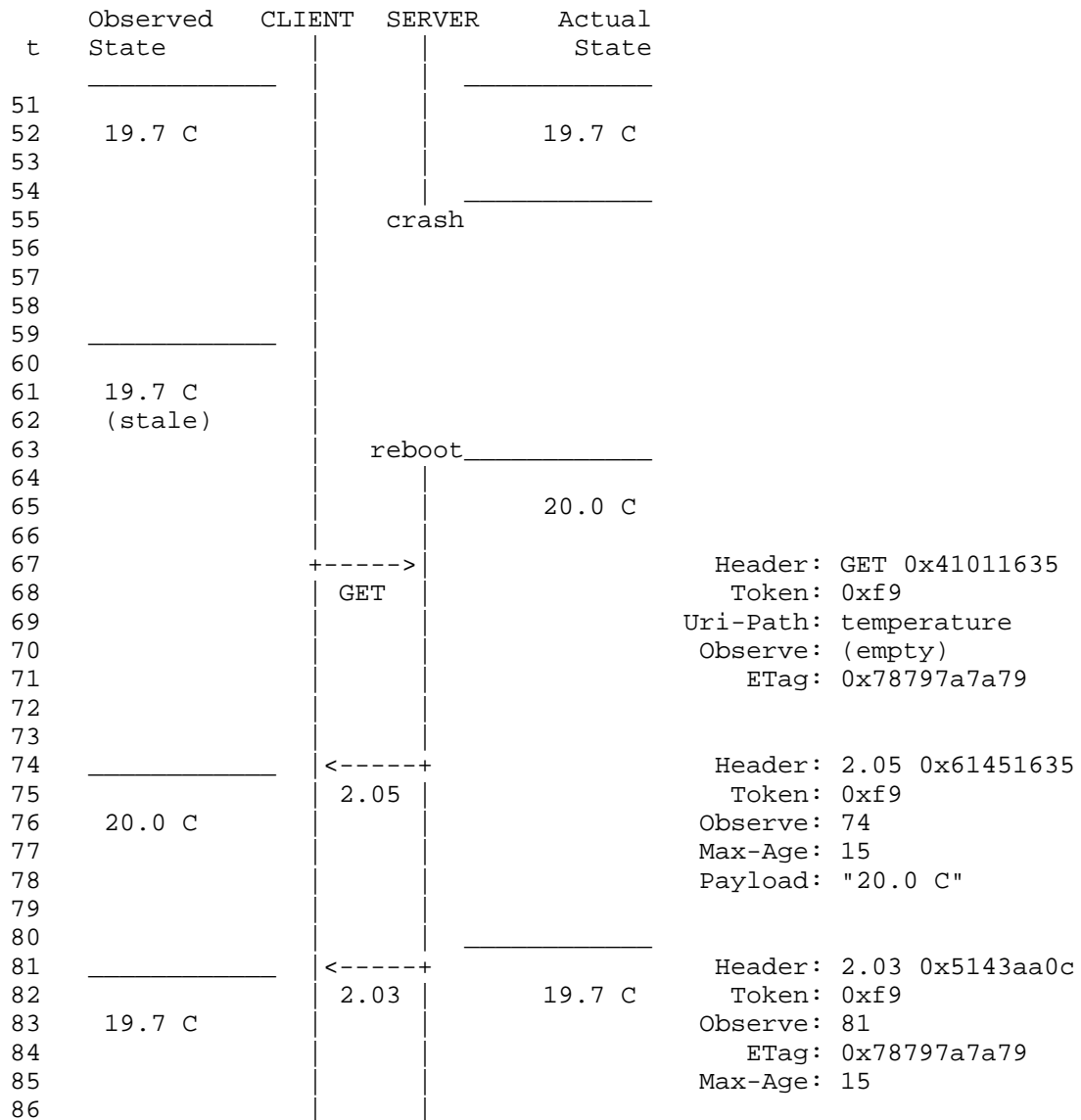


Figure 5: The client re-registers and gives the server the opportunity to select a stored response

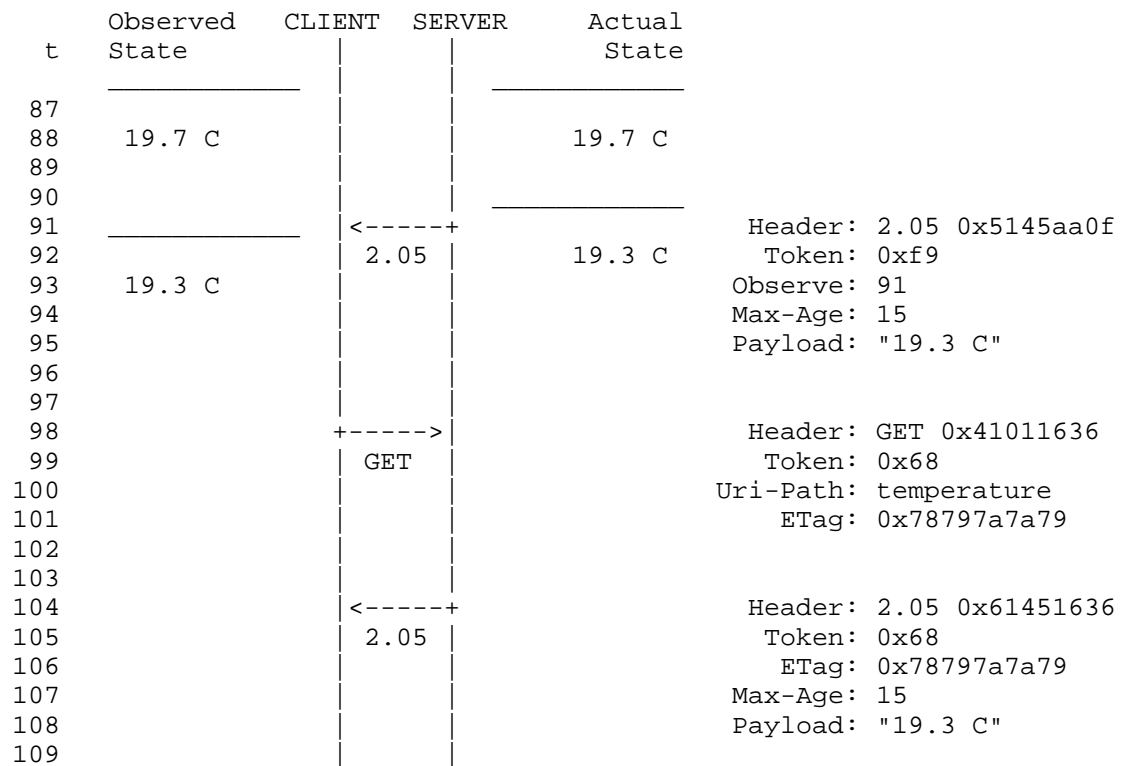


Figure 6: The client makes a normal GET request and thereby cancels the observation

## A.1. Proxying

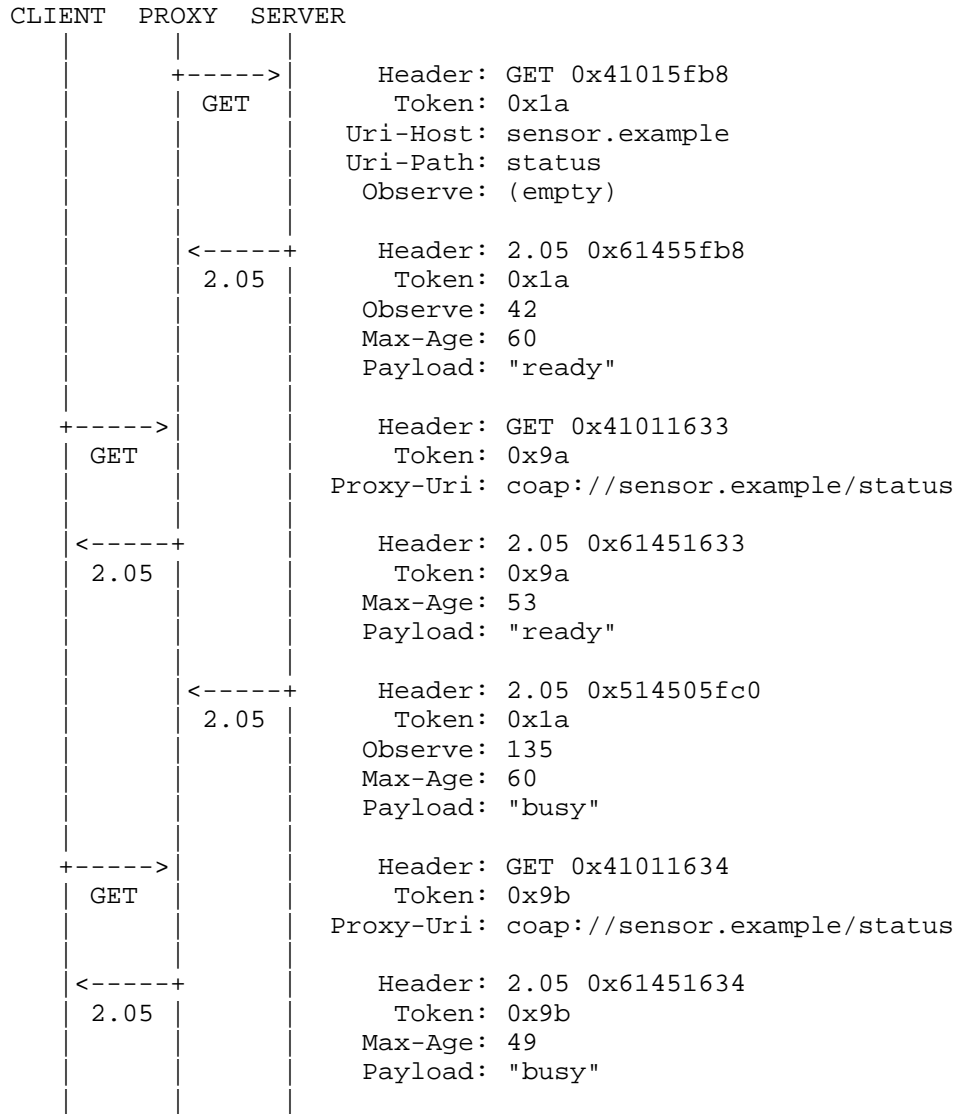


Figure 7: A proxy observes a resource to keep its cache up to date

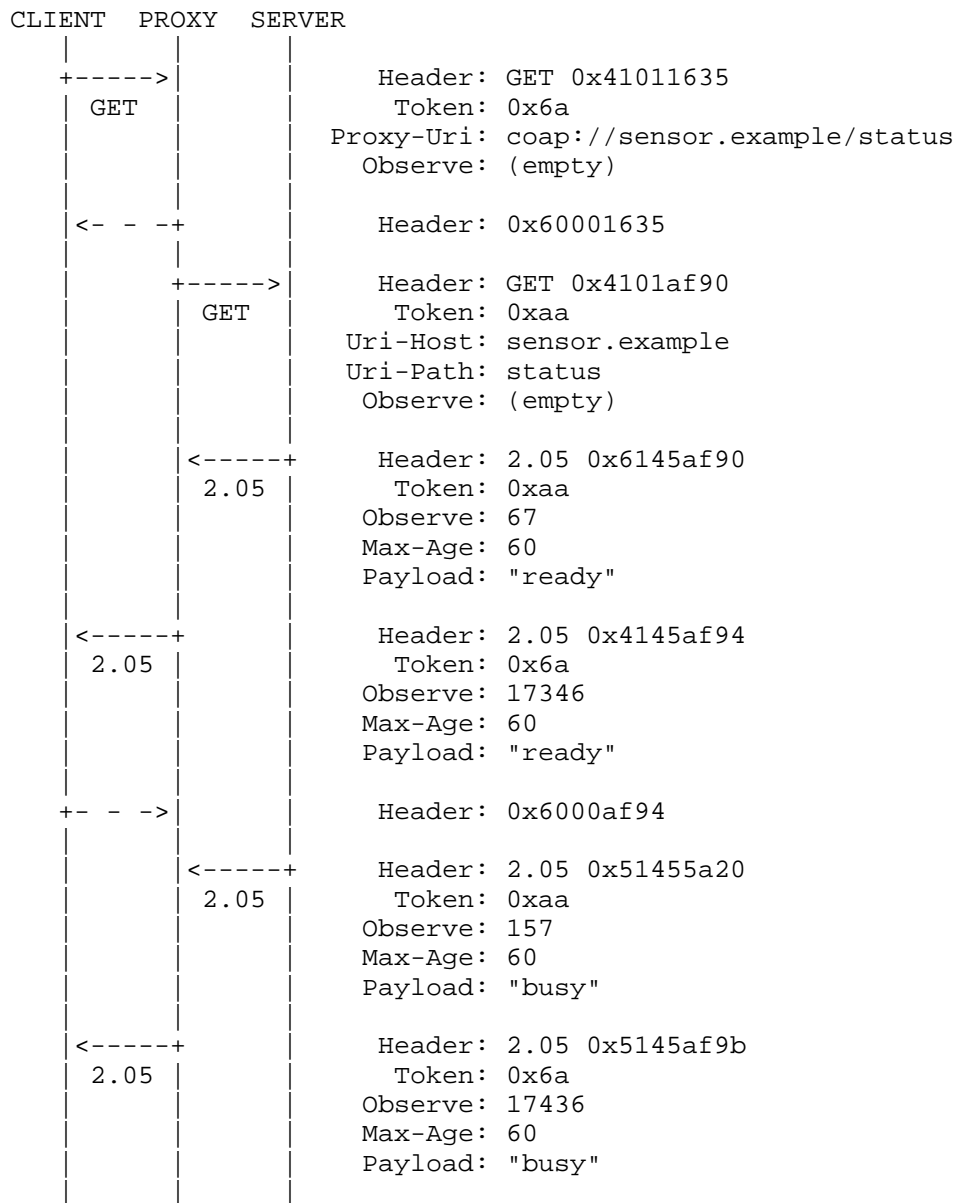


Figure 8: A client observes a resource through a proxy

## A.2. Blockwise Transfer

CLIENT	SERVER
<pre> +-----&gt; GET </pre>	<pre> Header: GET 0x41011636 Token: 0xfb Uri-Path: status-icon Observe: (empty) </pre>
<pre> &lt;-----+ 2.05 </pre>	<pre> Header: 2.05 0x61451636 Token: 0xfb Block2: 0/1/128 Observe: 62354 Max-Age: 60 Payload: [128 bytes] </pre>
<pre> &lt;-----+ 2.05 </pre>	<pre> Header: 2.05 0x4145af9c Token: 0xfb Block2: 1/0/128 Observe: 62354 Max-Age: 60 Payload: [27 bytes] </pre>
<pre> + - - -&gt; </pre>	<pre> Header: 0x6000af9c </pre>
<pre> &lt;-----+ 2.05 </pre>	<pre> Header: 2.05 0x4145af9d Token: 0xfb Block2: 0/1/128 Observe: 62444 Max-Age: 60 Payload: [128 bytes] </pre>
<pre> + - - -&gt; </pre>	<pre> Header: 60005af9d </pre>
<pre> &lt;-----+ 2.05 </pre>	<pre> Header: 2.05 0x4145af9e Token: 0xfb Block2: 1/0/128 Observe: 62444 Max-Age: 60 Payload: [27 bytes] </pre>
<pre> + - - -&gt; </pre>	<pre> Header: 0x6000af9e </pre>

Figure 9: A server sends two notifications of two blocks each



## Appendix B. Modeling Resources to Tailor Notifications

A server may want to provide notifications that respond to very specific conditions on some state. This is best done by modeling the resources that the server exposes according to these needs.

For example, for a CoAP server with an attached temperature sensor,

- o the server could, in the simplest form, expose a resource `<coap://server/temperature>` that changes its state every second to the current temperature measured by the sensor;
- o the server could, however, also expose a resource `<coap://server/temperature/felt>` that changes its state to "cold" when it's warm and the temperature drops below a preconfigured threshold, and to "warm" when it's cold and the temperature exceeds a second, higher threshold;
- o the server could expose a parameterized resource `<coap://server/temperature/critical?above=45>` that changes its state every second to the current temperature if the sensor reading exceeds the specified parameter value, and that changes its state to "OK" when the temperature drops below; or
- o the server could expose a parameterized resource `<coap://server/temperature?query=select+avg(temperature)+from+Sensor.window:time(30sec)>` that accepts expressions of arbitrary complexity and changes its state accordingly.

In any case, the client is notified about the current state of the resource whenever the state of the appropriately modeled resource changes. By designing resources that change their state on certain conditions, it is possible to notify the client only when these conditions occur instead of continuously supplying it with information it doesn't need.

By parametrizing resources, this is not limited to conditions defined by the server, but can be extended to arbitrarily complex conditions defined by the client. Thus, the server designer can choose exactly the right level of complexity for the application envisioned and devices used, and is not constrained to a "one size fits all" mechanism built into the protocol.

## Appendix C. Changelog

Changes from ietf-07 to ietf-08:

- o Expanded text on transmitting notification while a previous transmission is pending (#242).
- o Changed reordering detection to use a fixed time span of 128 seconds instead of EXCHANGE\_LIFETIME (#276).
- o Removed the use of the freshness model to determine if the client is still on the list of observers. This includes removing that
  - \* the client assumes that it has been removed from the list of observers when Max-Age ends;
  - \* the server sets the Max-Age Option of a notification to a value that indicates when the server will send the next notification;
  - \* the server uses a number of retransmit attempts such that removing a client from the list of observers before Max-Age ends is avoided (#235);
  - \* the server may remove the client from all lists of observers when the transmission of a confirmable notification ultimately times out.
- o Changed that an unrecognized critical option in a request must actually have no effect on the state of any observation relationship to any resource, as the option could lead to a different target resource.
- o Clarified that client implementations must be prepared to receive each notification equally as a confirmable or a non-confirmable message, regardless of the message type of the request and of any previous notification.
- o Added a requirement for sending a confirmable notification at least every 24 hours before continuing with non-confirmable notifications (#221).
- o Added congestion control considerations from [I-D.bormann-core-congestion-control-02].
- o Recommended that the client waits for a randomized time after the freshness of the latest notification expired before re-registering. This prevents that multiple clients observing a resource perform a GET request at the same time when the need to re-register arises.
- o Changed reordering detection from 'MAY' to 'SHOULD', as the goal of the protocol (to keep the observed state as closely in sync

with the actual state as possible) is not optional.

- o Fixed the length of the Observe (3 bytes) in the table in Section 2.
- o Replaced the 'x' in the No-Cache-Key column in the table in Section 2 with a '-', as the Observe Option doesn't have the No-Cache-Key flag set, even though it is not part of the cache key.
- o Updated examples.

Changes from ietf-06 to ietf-07:

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

Changes from ietf-05 to ietf-06:

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

Changes from ietf-04 to ietf-05:

- o Recommended that a client does not re-register while a new notification from the server is still likely to arrive. This is

to avoid that the request of the client and the last notification after max-age cross over each other (#174).

- o Relaxed requirements when sending a Reset message in reply to non-confirmable notifications.
- o Added an implementation note about careless GET requests (#184).
- o Updated examples.

Changes from ietf-03 to ietf-04:

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

Changes from ietf-02 to ietf-03:

- o Separated client-side and server-side requirements.
- o Fixed uncertainty if client is still on the list of observers by introducing a liveliness model based on Max-Age and a new option called "Max-OFE" (#174).
- o Simplified the text on message reordering (#129).
- o Clarified requirements for intermediaries.
- o Clarified the combination of blockwise transfers with notifications (#172).
- o Updated examples to show how the state observed by the client becomes eventually consistent with the actual state on the server.
- o Added examples for parameterization of observable resource.

Changes from ietf-01 to ietf-02:

- o Removed the requirement of periodic refreshing (#126).
- o The new "Observe" Option replaces the "Lifetime" Option.
- o Introduced a new mechanism to detect message reordering.

- o Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:

- o Changed terminology from "subscriptions" to "observation relationships" (#33).
- o Changed the name of the option to "Lifetime".
- o Clarified establishment of observation relationships.
- o Clarified that an observation is only identified by the URI of the observed resource and the identity of the client (#66).
- o Clarified rules for establishing observation relationships (#68).
- o Clarified conditions under which an observation relationship is terminated.
- o Added explanation on how clients can terminate an observation relationship before the lifetime ends (#34).
- o Clarified that the overriding objective for notifications is eventual consistency of the actual and the observed state (#67).
- o Specified how a server needs to deal with clients not acknowledging confirmable messages carrying notifications (#69).
- o Added a mechanism to detect message reordering (#35).
- o Added an explanation of how notifications can be cached, supporting both the freshness and the validation model (#39, #64).
- o Clarified that non-GET requests do not affect observation relationships, and that GET requests without "Lifetime" Option affecting relationships is by design (#65).
- o Described interaction with blockwise transfers (#36).
- o Added Resource Discovery section (#99).
- o Added IANA Considerations.
- o Added Security Considerations (#40).
- o Added examples (#38).

Author's Address

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

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



Network Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: April 16, 2013

C. Jennings  
Cisco  
October 13, 2012

Transitive Trust Enrollment for Constrained Devices  
draft-jennings-core-transitive-trust-enrollment-01

Abstract

This document provides a sketch of a rendezvous protocol that allows constrained internet devices such as sensors to securely connect into a system that uses them. The solution is based on the idea that each device will be manufactured with a one time password that can be used by the customer to tell the device which controller to enroll with, and the device will be manufactured to contact a given Transfer Server that is used to tell the device which system to connect to. The administrator of the device will be able to get this one time password from the device using a QR code, and then will be able to use that one time password to inform a Transfer Server which system the device should connect to. The device will contact the Transfer Agent, get this information, and then connect to the appropriate system.

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, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal



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

## Table of Contents

1. Introduction . . . . .	3
2. Enrollment Information Flow . . . . .	5
3. Terminology . . . . .	6
4. QR Code . . . . .	6
5. Transfer Agent API . . . . .	7
5.1. Create . . . . .	7
5.2. Setup . . . . .	8
5.3. Bind . . . . .	8
5.4. Fetch . . . . .	9
6. Controller API . . . . .	9
6.1. Test Alive . . . . .	10
6.2. Add . . . . .	10
6.3. Sensor Update . . . . .	10
7. Security Considerations . . . . .	10
8. Variations . . . . .	11
8.1. LED Based Enrollment . . . . .	11
8.2. Bulk Enrollment . . . . .	12
8.3. Public Key Crypto . . . . .	12
9. Implementation Notes . . . . .	12
9.1. Random Numbers . . . . .	12
10. Open Issues . . . . .	13
11. IANA Considerations . . . . .	13
12. Acknowledgments . . . . .	13
13. Appendix A: JOSE SHA224-CFB . . . . .	13
13.1. Example . . . . .	14
14. References . . . . .	14
14.1. Normative References . . . . .	14
14.2. Informative References . . . . .	15
Author's Address . . . . .	15

## 1. Introduction

Secure enrollment of devices into internet-based systems has never been easy. The constrained devices that need to be enrolled into systems today face many challenges. Typically, simple devices such as keyboards and screens have no user interface; they may have only a single button or LED. At the time they are installed, there may not be a working network or even power. However, these devices are being used for applications that are increasingly important and safety-critical, so they need to have reasonable security and privacy characteristics. This document specifies an enrollment system for such devices.

In many systems, there is a need to configure a Device, such as a sensor or actuator, so that it is controlled by some specific Controller. With Devices like a switch and light, it may be that all the Controller does is later configure the switch to control the light. To make this happen, both Devices need to be under the control of a common Controller that is authorized to make changes to the Devices.

The simplified high-level information flow is illustrated in the following figure. The goal is to get to the point where the Device knows that it should be talking to the Controller.

TODO - See PDF Version of draft

When the Manufacturer builds the Device, it includes a One Time Password (OTP) that the Introducer can use to enroll the Device with the Controller. The Manufacturer also runs a website known as the Transfer Agent that knows the OTP for every device that uses that Transfer Agent. The Device can include the OTP as a QR code on the outside of the Device. When the Device is installed, the network administrator or installer uses a software agent known as the Introducer. The Introducer would typically be simply a normal browser running on a smart phone with a camera that can read QR codes. When the Device is installed, the Introducer can scan the QR code on the Device. This QR code includes a URL to the Transfer Agent along with the OTP and a separate Device secret DevSecret. (Message 1). The Introducer then contacts the Transfer Agent and uses the OTP to tell the Transfer Agent which Controller this Device should use (Message 2). The Introducer can also tell the Controller the DevSecret (Message 3) so that the Controller and Device can authenticate each other. Later, the first time the Device boots up and gets network connectivity, it contacts the Transfer Agent, and the Transfer Agent tells the Device which Controller to talk to (Message 4). From that point on, any time the Device boots, the Device can communicate directly with the Controller (Message 5). The

actual message flow is slightly more complicated and shown in Section 2, but it uses the same basic idea as this simplified flow.

The system is designed to achieve several desirable properties:

- o Can work for Devices with very limited memory and processing power.
- o Does not require network or power to be available when the Device is installed.
- o Is fairly secure (see more in the security section).
- o Minimal addition to manufacturing costs.
- o The installer can detect if the OTP has already been used.
- o Provides a work flow in which a Device does not need to be taken out of the box to be enrolled. This can be very important to enable consumers themselves to enroll devices they buy from a service provider.
- o Works with common firewall and NAT network topologies.

One of the key steps in making this system work is getting the OTP from the Device to the Introducer. The approach used here is to use a QR code representing the URL. The QR code is printed on the Device and/or box it comes in.

The Device uses HTTP or COAP [I-D.ietf-core-coap] to communicate with the Controller. The Transfer Agent and Introducer use HTTPS to communicate with each other. There are three pieces of keying material used for cryptographic operations. The first is the One Time Password (OTP) that is passed via a QR code from the device to the Introducer and that the Introducer then uses to authorize itself to the Transfer Agent. There is also a DevSecret that is used to secure communications between the Device and the Controller. Finally there is a TaSecret that is used to secure communications between the Device and the TransferAgent. The Transfer Agent needs a normal certificate to use HTTPS.

It is assumed that the Device may have a NAT between it and the Controller and that the Device is on the inside of the NAT. The Transfer Agent is assumed to be a generally accessible server on the internet, but the Controller and Device can be on the inside of a firewall or NAT between them and the Transfer Agent.

The semantic level information in each message is discussed in Section 2 and the syntax of the messages is discussed in Section 5 and Section 6. The security properties of the system are described in Section 7.

## 2. Enrollment Information Flow

In the following message flow we use the following definitions:

**TaURL** An http URL that can be used to reach the root resource on the Transfer Agent.

**DevURN** A globally unique URN assigned by the Manufacturer to uniquely identify this Device.

**OTP** The One Time Password created by the Manufacturer for enrolling the Device.

**TaSecret** The secret created by the Manufacturer for the Device to communicate with the Transfer Agent.

**DevSecret** The secret created by the Manufacturer for the device to communicate with the Controller.

**ContURL** This is a URL that provides the address to reach the controller. It can have a scheme of http, https, coap, or coaps.

**DevLabel** A locally significant string that the Introducer can assign to a Device. For example, the convention for a thermostat in building 30, floor 2, office 361 might be assign the string "BLD30/2/361 - Thermostat". This string is provided purely as a way to let the Introducer and Controller exchange information that may be useful for the user installing the system.

The information flow is illustrated in the following figure. The goal is get to the point where the Device knows that it should be talking to the Controller, the Controller knows it should be talking the Device, and the Device and Controller can communicate and authenticate each other using the DevSecret.

TODO - See PDF Version of draft

When the Manufacturer builds the Device, it includes a TaSecret on the Device, a DevSecret, and the URN for the Device (DevURN). It also creates a QR code on the Device that contains the URL to the transfer agent (TaURL), the URN for the Device (DevURL), the OTP, and the DevSecret. This QR codes is described in detail in section TODO. The Manufacturer also tells the Transfer Agent the OTP, TaSecret and DevURN for this Device.

When the Device is installed, the Introducer reads OTP, DevSecret, DeviceURN, and the URL for the Transfer Agent (TaURL) from the Device by scanning the QR code on the device (Message 1). If the Introducer is a web browser, it uses the Transfer Agent URL to fetch an HTML user interface to perform the next steps (Message 2). The user interface on the Introducer allows the user to user to enter the URL for the Controller (ContURL) and an optional label for the device (DevLabel).

Next the controller tells the Transfer Agent the Controller URL to

use for this DeviceURN. This request is authenticated by the Transfer Agent using the OTP (Message 3). Open Issue: right now the OTP is transferred in the request (which is over HTTPS). A better design might be to have the Introducer prove possession of the OTP to the Transfer Agent and not send the OTP over the wire.

The Introducer also tells the controller the DevSecret for this Device and the optional DevLabel (message 4).

Later the Device contacts the Transfer Agent and the Transfer Agent tells the Device the URL of the Controller to talk to (ContURL) in message 5 and 6). The information from the Transfer Agent to Device is encrypted and signed with the TaSecret.

From that point on, any time the Device boots, it can directly communicate with the Controller (Messages 7 and 8). The Controller and Device both know the DevSecret for the Device and can use that to authenticate and encrypt communications between them. It is suggested that the first thing the Controller and Device should do is to use this DevSecret to securely replace it with some different secret that is not known to anyone that saw the QR code.

Open Issue: should we add in an additional ContSecret that is picked by the Controller, passed to Introducer, then passed to the Trust Agent, and finally passed to Device?

### 3. Terminology

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

When this draft says Base64, it means the URL safe Base64 encoding from TODO.

### 4. QR Code

The QR code for the Device MUST be an HTTPS URL that points at the appropriate Transfer Agent. The authority MUST be formed by using the authority from the TaURL. The path is formed by concatenating ".well-known/tt1/" followed the DevURN followed by "/s". The DevURN SHOULD be one of the URNs defined in [I-D.arkko-core-dev-urn]. It MUST include the OTP as a Base64 encoded value for a parameter called otp. The secret MUST be encoded in Base64 and used as the fragment identifier of the URL. The secret is put as a fragment so that if the Introducer scans the QR code and dereferences the URL with a web

browser, the fragment identifier will not be transferred in the request to the Transfer Agent.

As an example, if the Transfer Agent's domain is example.net, a valid URL for the QR code would be:

```
TODO - change hex to base64
https://example.net/.well-known/ttel/urn:dev:mac:90a2da001a0c/s
?otp=88F5EC91493E473B758159C7792C#00004DCFDCDBD9F54C1B2E71FC22
```

The QR code SHOULD use an error coding level of "H". This would generate the following QR code:

QR code in ASCII art left as an exercise to the reader but there is one in the PDF version.

## 5. Transfer Agent API

Note that future version of the API that needed to increment a version number would do it by changing the ttel to tte2.

TODO - still need to define all the error responses but basic approach will be a simple JSON object with the error responses.

### 5.1. Create

The HTTP REST API allows the manufacturer to tell the Transfer Device about the DevURN and OTP for a given device the Manufacturer has created.

```
Path: .well-known/ttel/d/{DevURN}
Methods: POST
Parameters:
  otp: Base64 encoded version of the OTP
Return Values: TODO
```

This creates an entry for the device in the database and stores the OTP associated with this Device. The Transfer Agent SHOULD authenticate this request to authorize it. Note the "d" in the path is short for "device"; having this path segment allows for future extensibility.

Example Request:

```
POST https://example.net/.well-known/ttel/d/urn:dev:mac:90a2da001a0c
?otp=88F5EC91493E473B758159C7792C
```

Example Response:

TODO

## 5.2. Setup

The Transfer Agent MUST return a web page that allows the user to provide the information needed for the bind, and then the Introducer must call the bind and add methods.

Path: .well-known/ttel/d/{DevURN}/s

Methods: GET

Parameters:

otp: Base64 encoded version of the OTP

Return Values: HTML5 web page

This MUST return a HTML web page that has a suitable user interface to allow the user to enter the address of the the Controller. It is suggested that the page validate that this controller entry is correct using the "alive" API in Section TODO. Once the Controller is verified, the web page MUST tell the Transfer Agent the Controller address using the "bind" API in Section TODO. The page MUST tell the controller the DevURN and DevSecret for the Device using the "add" API in Section TODO. MUST be done over HTTPS.

Example Request:

GET https://example.net/.well-known/ttel/d/urn:dev:mac:90a2da001a0c/s  
?otp=88F5EC91493E473B758159C7792C

## 5.3. Bind

TODO MUST be sent over TLS, and the Introducer MUST verify that the HTTPS certificate of the Transfer Agent matches the URL.

Once the Transfer Agent has successfully stored the Controller's address for a given OTP, it MUST NOT allow that OTP to be used again to store an address for that Device.

Path: .well-known/ttel/d/{DevURN}/c

Methods: PUT

Parameters:

otp: Base64 encoded version of the OTP

conturl: URL to controller escaped as necessary for placement in  
a URL

Return Values: TODO

This request using the

Example Request:

TODO

PUT https://example.net/.well-known/ttel/d/urn:dev:mac:90a2da001a0c/c  
?otp=88F5EC91493E473B758159C7792C

#### 5.4. Fetch

This API allows a Device to get the information about the controller it should connect to. It is provided in a JSON object which is encrypted using the OTP.

Path: .well-known/ttel/{DevURN}/c

Methods: GET

Parameters: None

Success Return Values: JSON object as defined in TODO that contains the encrypted URL to the Controller.

Error Return Values: Returns HTTP 404 if the DevID can not be found or if the controller URL has not yet been set for this DevURN.

The Transfer Agent looks up the OTP and ContURL for the requested DevURN. If the DevURN cannot be found, or the ContURL has not yet been set for this DevURN, then the Transfer Agent returns an HTTP 404 error. If they are found, it then the Authenticated Encryption with Associated Data (AEAD) algorithm described in Appendix A (TODO ref) to form the JSON object that is returned. The TaSecret is used as the key for the AEAD, the ContURL is the input data to be encrypted, and the DevURN is used as Associated Data for the authentication.

Example Request:

GET https://example.net/.well-known/ttel/d/urn:dev:mac:90a2da001a0c/c

Example Response:

TODO

#### 6. Controller API

The Alive and Add API need to include a CORS (TODO REF) header to allow AJAX from the Transfer Agent to call these APIs. They MUST include an HTTP header in the response that sets the header field Access-Control-Allow-Origin to a value of "\*". TODO security section



needs to discuss implications.

### 6.1. Test Alive

This method allows the Introducer to validate that a valid Controller address has been entered. It simply returns an HTTP 200 if the controller is operational.

Path: .well-known/ttel/alive  
Methods: GET  
Parameters: None  
Return Values: TODO

### 6.2. Add

This method is used by the Introducer to add a new Device to the Controller. (Open issues - should it result in redirect to a controller web page to configure device? )

Path: .well-known/ttel/c/{DevURN}/k  
Methods: PUT  
Parameters:  
    devSecret: Base64 encoded version of the secret  
    devLabel:  
Return Values: TODO

### 6.3. Sensor Update

TODO

Path: .well-known/ttel/s/{DevURN}/v  
Methods: PUT  
Parameters: None  
Body: Encrypted SENML  
Return Values: TODO

The body is a Encrypted JOSE object, as specified in Appendix A (TODO REF). The secret for this Device is used as the key to encrypt the data. The DevURN is used as the associated data. The decrypted data is a SENML sensor reading for this Device as described in [I-D.jennings-senml].

## 7. Security Considerations

This section has not really been started and needs lots of work.

TODO - Discuss how one can replace a dead Controller with a new one

in an operational network. The short answer is likely that one needs to back up the keys of the old Controller and move these to the new Controller.

What happens if the OTP is stolen during Device transit? The short answer is that the Device is compromised at this point and needs to be discarded or returned to the manufacturer to get a new OTP. The Introducer needs to detect that this has happened and warn the user.

There are additional concerns about Devices that may be operational without ever being introduced to a Controller. For example, if a light switch supported this protocol but could also be used just as a stand alone light switch, there would be a risk that the OTP could be stolen by an attacker, with the attacker enrolling the Device to the attacker's Controller. If the correct user installed the light switch but did not Introduce it to anything, the fact it had been compromised would go undetected. One way to mitigate this risk might be to include some manual configuration on the Device to indicate that it is to be used in stand-alone mode, such as a jumper that can be cut.

Network topology consideration - Introducer can install firewall rules that allow Devices to contact Transfer Agent.

Explain why works with NATs / FWs.

## 8. Variations

### 8.1. LED Based Enrollment

An alternative to QR codes is to have an LED on the Device flash out the relevant information to the Introducer. The output string is formed by concatenating a 16-bit start of message constant value of 0x0001, followed by the 8 bit length of TaURN, TaURN, 8 bit length of DevURN, the DevURN, 8 bit length of OTP, OTP, 8 bit length of DevSect, DevSecret and then an 8-bit two's compliment checksum value computed over the previous bytes, including the start of message constant. All values are in network byte order. The resulting string is output using Non-Return-to-Zero Inverted (NRZI) encoding on the LED at a baud rate of 15 bps. This allows a Device such as a smartphone with video capture to detect the signal and recover the information.

TODO - see if this works at 30 bps. See about encoding multiple intensity levels or colors in the LED. Initial experiments indicate this does not work very well, as auto contrast in the video camera tends to saturate LED range. Would an Adler-32 checksum be better?

Could multiple colors of intensity improve the speed of this as this is very slow.

## 8.2. Bulk Enrollment

Imagine one wants to enroll a whole box of sensors. We should define some scheme where one could simply bar code something on the outside of a box so that all the sensors in the box could be bulk enrolled. Perhaps there could be a master secret and start and end DevURN on the outside of the box bar code. Then the OTP for a given Device would be generated using the master secret and DevURN of that Device. Work is needed to sort out details of a scheme like this.

## 8.3. Public Key Crypto

This specification assumes that COAP is being used with DTLS in Pre Shared Key (PSK) mode. It would also be possible to use DTLS with self signed certificates with a very similar flow, where the Introducer provided the Transfer Agent with the fingerprint of the certificate or public key of the Controller.

## 9. Implementation Notes

The system described here can be implemented on a very small device. An implementations for Arduino with ethernet was done that includes all the parts described here, including SENML, JSON, the encryption and signing, HTTP, DNS, and DHCP. It also included libraries for reading a 1-wire temperature sensor. This fits in under 32k of flash, and uses less than 4k of ram on an 8 bit AVR processor. That means that the cost for an embedded processor in volume with this much flash, ram, etc. is very roughly \$1.50 USD in 2012. A key part of getting this to be small is the extremely small crypto footprint from using SHA224-CFB.

### 9.1. Random Numbers

Note: This section would be better in a separate draft.

TODO - Explain how to use SHA224\_DRBG as defined in NIST SP800-90A. TODO REF. Store reseed counter in eeprom every 24 hours. Explain what to store in eeprom on reseed. TODO REF RFC 4086 and XKCD 221.

Todo - Discuss sources of randomness in use: 16 bytes of random data created during manufacturing. A 32 bit boot counter that increments every time the device boots. 8 byte pool of random data from sensor readings. 8 byte pool of random data from timing of receiving or sending network packets. A 32 bit counter that increments each time

a random number is generated but resets to 0 on reboot.

#### 10. Open Issues

The references section is in serious need of work - let me know stuff that should be added to it.

Does QR encoding of L work out better than H?

Is there any advantage in having the HTTP URL in well-known space?

Is there some clever way (perhaps zeroconf) for the Introducer to discover the ContURL?

#### 11. IANA Considerations

TODO register .well-known/ttel

#### 12. Acknowledgments

Some of the fundamental ideas in this draft were inspired by Max Pritikin's work on Transitive Trust Introduction. Randy Bush provided crisp and excellent advice on what the security properties of the solutions should be. I'd like to thank the following people for review comments: Eric Rescorla, Jari Arkko, Lyndsay Campbell, and Zach Shelby.

#### 13. Appendix A: JOSE SHA224-CFB

Note: This section will eventually be moved to an experimental draft submitted to JOSE WG.

This section describes how to create a JOSE object as described by [I-D.barnes-jose-jsms] that is encrypted and signed with SHA224-CFB as specified in [HashCFB].

This adds a new ENCRYPTION algorithm called sha224-cfb to [I-D.barnes-jose-jsms]. This takes one parameter named "n" which represents the nonce as defined in [HashCFB]. It is RECOMMENDED that the key be 14 bytes long and that the nonce be 24 bytes long. The authentication information from the algorithm is stored in the "mac" field.

### 13.1. Example

TODO example. Todo fix to base64 instead of hex encoding. TOOD talk to Barnes about keyID and case with no key wrap. TODO - state of sha224-cfb and this is all experimental.

Input Key (Hex) = 88F5EC91493E473B758159C7792C  
Input Associated Data = "urn:dev:mac:90a2da001a0c"  
Input plain text = "http://example.com" - TODO

Result =  
{  
 TODO  
}

## 14. References

### 14.1. Normative References

- [HashCFB] Forler, C., McGrew, D., Lucks, S., and J. Wenzel, "Hash-CFB: Authenticated Encryptions without a Block Cipher", Directions in Authenticated Ciphers Workshop , July 2012.
- [I-D.barnes-jose-jsms]  
Barnes, R., "JavaScript Message Security Format",  
draft-barnes-jose-jsms-00 (work in progress), June 2012.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-08 (work in progress), October 2011.
- [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.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

## 14.2. Informative References

[I-D.arkko-core-dev-urn]

Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-01 (work in progress), October 2011.

[I-D.jennings-senml]

Jennings, C., Shelby, Z., and J. Arkko, "Media Types for Sensor Markup Language (SENML)", draft-jennings-senml-07 (work in progress), October 2011.

## Author's Address

Cullen Jennings  
Cisco  
170 West Tasman Drive  
San Jose, CA 95134  
USA

Phone: +1 408 421-9990  
Email: fluffy@iii.ca



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: April 25, 2013

C. Jennings  
Cisco  
Z. Shelby  
Sensinode  
J. Arkko  
Ericsson  
October 22, 2012

Media Types for Sensor Markup Language (SENML)  
draft-jennings-senml-10

Abstract

This specification defines media types for representing simple sensor measurements and device parameters in the Sensor Markup Language (SenML). Representations are defined in JavaScript Object Notation (JSON), eXtensible Markup Language (XML) and Efficient XML Interchange (EXI), which share the common SenML data model. A simple sensor, such as a temperature sensor, could use this media type in protocols such as HTTP or CoAP to transport the measurements of the sensor or to be configured.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 25, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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



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

## Table of Contents

1. Overview . . . . .	3
2. Requirements and Design Goals . . . . .	3
3. Terminology . . . . .	4
4. Semantics . . . . .	4
5. Associating Meta-data . . . . .	7
6. JSON Representation (application/senml+json) . . . . .	7
6.1. Examples . . . . .	9
6.1.1. Single Datapoint . . . . .	9
6.1.2. Multiple Datapoints . . . . .	9
6.1.3. Multiple Measurements . . . . .	10
6.1.4. Collection of Resources . . . . .	10
7. XML Representation (application/senml+xml) . . . . .	11
8. EXI Representation (application/senml-exi) . . . . .	12
9. Usage Considerations . . . . .	14
10. IANA Considerations . . . . .	15
10.1. Units Registry . . . . .	15
10.2. Media Type Registration . . . . .	18
10.2.1. senml+json Media Type Registration . . . . .	18
10.2.2. senml+xml Media Type Registration . . . . .	19
10.2.3. senml-exi Media Type Registration . . . . .	20
10.3. XML Namespace Registration . . . . .	21
11. Security Considerations . . . . .	21
12. Privacy Considerations . . . . .	21
13. Acknowledgement . . . . .	21
14. References . . . . .	22
14.1. Normative References . . . . .	22
14.2. Informative References . . . . .	22
Authors' Addresses . . . . .	23

## 1. Overview

Connecting sensors to the internet is not new, and there have been many protocols designed to facilitate it. This specification defines new media types for carrying simple sensor information in a protocol such as HTTP or CoAP[I-D.ietf-core-coap] called the Sensor Markup Language (SenML). This format was designed so that processors with very limited capabilities could easily encode a sensor measurement into the media type, while at the same time a server parsing the data could relatively efficiently collect a large number of sensor measurements. There are many types of more complex measurements and measurements that this media type would not be suitable for. A decision was made not to carry most of the meta data about the sensor in this media type to help reduce the size of the data and improve efficiency in decoding. Instead meta-data about a sensor resource can be described out-of-band using the CoRE Link Format [I-D.ietf-core-link-format]. The markup language can be used for a variety of data flow models, most notably data feeds pushed from a sensor to a collector, and the web resource model where the sensor is requested as a resource representation (GET /sensor/temperature).

SenML is defined by a data model for measurements and simple meta-data about measurements and devices. The data is structured as a single object (with attributes) that contains an array of entries. Each entry is an object that has attributes such as a unique identifier for the sensor, the time the measurement was made, and the current value. Serializations for this data model are defined for JSON [RFC4627], XML and Efficient XML Interchange (EXI) [W3C.REC-exi-20110310].

For example, the following shows a measurement from a temperature gauge encoded in the JSON syntax.

```
{"e":[{"n": "urn:dev:ow:10e2073a01080063", "v":23.5, "u":"Cel" }]}
```

In the example above, the array in the object has a single measurement for a sensor named "urn:dev:ow:10e2073a01080063" with a temperature of 23.5 degrees Celsius.

## 2. Requirements and Design Goals

The design goal is to be able to send simple sensor measurements in small packets on mesh networks from large numbers of constrained devices. Keeping the total size under 80 bytes makes this easy to use on a wireless mesh network. It is always difficult to define what small code is, but there is a desire to be able to implement this in roughly 1 KB of flash on a 8 bit microprocessor. Experience with Google power meter and large scale deployments has indicated

that the solution needs to support allowing multiple measurements to be batched into a single HTTP or CoAP request. This "batch" upload capability allows the server side to efficiently support a large number of devices. It also conveniently supports batch transfers from proxies and storage devices, even in situations where the sensor itself sends just a single data item at a time. The multiple measurements could be from multiple related sensors or from the same sensor but at different times.

### 3. Terminology

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

### 4. Semantics

Each representation carries a single SenML object that represents a set of measurements and/or parameters. This object contains several optional attributes described below and a mandatory array of one or more entries.

#### Base Name

This is a string that is prepended to the names found in the entries. This attribute is optional.

#### Base Time

A base time that is added to the time found in an entry. This attribute is optional.

#### Base Units

A base unit that is assumed for all entries, unless otherwise indicated. This attribute is optional.

#### Version

Version number of media type format. This attribute is optional positive integer and defaults to 1 if not present.

#### Measurement or Parameter Entries

Array of values for sensor measurements or other generic parameters (such as configuration parameters). If present there must be at least one entry in the array.

Each array entry contains several attributes, some of which are optional and some of which are mandatory.

#### Name

Name of the sensor or parameter. When appended to the Base Name attribute, this must result in a globally unique identifier for the resource. The name is optional, if the Base Name is present. If the name is missing Base Name must uniquely identify the resource. This can be used to represent a large array of measurements from the same sensor without having to repeat its identifier on every measurement.

#### Units

Units for a measurement value.

#### Value

Value of the entry. Optional if a Sum value is present, otherwise required. Values are represented using three basic data types, Floating point numbers ("v" field for "Value"), Booleans ("bv" for "Boolean Value") and Strings ("sv" for "String Value"). Exactly one of these three fields MUST appear.

#### Sum

Integrated sum of the values over time. Optional. This attribute is in the units specified in the Unit value multiplied by seconds.

#### Time

Time when value was recorded. Optional.

#### Update Time

Update time. A time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement. This can be used to detect the failure of sensors or communications path from the sensor. Optional.

The SenML format can be extended with further custom attributes placed in the base object, or in an entry. Extensions in the base object pertain to all entries, whereas extensions in an entry object only pertain to that.

Systems reading one of the objects MUST check for the Version attribute. If this value is a version number larger than the version which the system understands, the system SHOULD NOT use this object. This allows the version number to indicate that the object contains mandatory to understand attributes. New version numbers can only be defined in RFC which updates this specification or its successors.

The Name value is concatenated to the Base Name value to get the name of the sensor. The resulting name needs to uniquely identify and differentiate the sensor from all others. If the object is a representation resulting from the request of a URI [RFC3986], then in the absence of the Base Name attribute, this URI is used as the default value of Base Name. Thus in this case the Name field needs to be unique for that URI, for example an index or subresource name of sensors handled by the URI.

Alternatively, for objects not related to a URI, a unique name is required. In any case, it is RECOMMENDED that the full names are represented as URIs or URNs [RFC2141]. One way to create a unique name is to include a EUI-48 or EUI-64 identifier (A MAC address) or some other bit string that is guaranteed uniqueness (such as a 1-wire address) that is assigned to the device. Some of the examples in this draft use the device URN type as specified in [I-D.arkko-core-dev-urn]. UUIDs [RFC4122] are another way to generate a unique name.

The resulting concatenated name MUST consist only of characters out of the set "A" to "Z", "a" to "z", "0" to "9", "-", ":", ".", or "\_" and it MUST start with a character out of the set "A" to "Z", "a" to "z", or "0" to "9". This restricted character set was chosen so that these names can be directly used as in other types of URI including segments of an HTTP path with no special encoding. [RFC5952] contains advice on encoding an IPv6 address in a name.

If either the Base Time or Time value is missing, the missing attribute is considered to have a value of zero. The Base Time and Time values are added together to get the time of measurement. A time of zero indicates that the sensor does not know the absolute time and the measurement was made roughly "now". A negative value is used to indicate seconds in the past from roughly "now". A positive value is used to indicate the number of seconds, excluding leap seconds, since the start of the year 1970 in UTC.

Representing the statistical characteristics of measurements can be very complex. Future specification may add new attributes to provide better information about the statistical properties of the measurement.

## 5. Associating Meta-data

SenML is designed to carry the minimum dynamic information about measurements, and for efficiency reasons does not carry more static meta-data about the device, object or sensors. Instead, it is assumed that this meta-data is carried out of band. For web resources using SenML representations, this meta-data can be made available using the CoRE Link Format [I-D.ietf-core-link-format].

The CoRE Link Format provides a simple way to describe Web Links, and in particular allows a web server to describe resources it is hosting. The list of links that a web server has available, can be discovered by retrieving the /.well-known/core resource, which returns the list of links in the CoRE Link Format. Each link may contain attributes, for example title, resource type, interface description and content-type.

The most obvious use of this link format is to describe that a resource is available in a SenML format in the first place. The relevant media type indicator is included in the Content-Type (ct=) attribute.

Further semantics about a resource can be included in the Resource Type and Interface Description attributes. The Resource Type (rt=) attribute is meant to give a semantic meaning to that resource. For example rt="outdoor-temperature" would indicate static semantic meaning in addition to the unit information included in SenML. The Interface Description (if=) attribute is used to describe the REST interface of a resource, and may include e.g. a reference to a WADL description [WADL].

## 6. JSON Representation (application/senml+json)

Root variables:

	SenML	JSON	Type
Base Name	bn		String
Base Time	bt		Number
Base Units	bu		Number
Version	ver		Number
Measurement or Parameters	e		Array

Measurement or Parameter Entries:

	SenML	JSON	Notes
Name	n	String	
Units	u	String	
Value	v	Floating point	
String Value	sv	String	
Boolean Value	bv	Boolean	
Value Sum	s	Floating point	
Time	t	Number	
Update Time	ut	Number	

All of the data is UTF-8, but since this is for machine to machine communications on constrained systems, only characters with code points between U+0001 and U+007F are allowed which corresponds to the ASCII[RFC0020] subset of UTF-8.

The root contents MUST consist of exactly one JSON object as specified by [RFC4627]. This object MAY contain a "bn" attribute with a value of type string. This object MAY contain a "bt" attribute with a value of type number. The object MAY contain a "bu" attribute with a value of type string. The object MAY contain a "ver" attribute with a value of type number. The object MAY contain other attribute value pairs, and the object MUST contain exactly one "e" attribute with a value of type array. The array MUST have one or more measurement or parameter objects.

Inside each measurement or parameter object the "n", "u", and "sv" attributes are of type string, the "t" and "ut" attributes are of type number, the "bv" attribute is of type boolean, and the "v" and "s" attributes are of type floating point. All the attributes are optional, but as specified in Section 4, one of the "v", "sv", or "bv" attributes MUST appear unless the "s" attribute is also present. The "v", and "sv", and "bv" attributes MUST NOT appear together.

Systems receiving measurements MUST be able to process the range of floating point numbers that are representable as an IEEE double-precision floating-point numbers [IEEE.754.1985]. The number of significant digits in any measurement is not relevant, so a reading of 1.1 has exactly the same semantic meaning as 1.10. If the value has an exponent, the "e" MUST be in lower case. The mantissa SHOULD be less than 19 characters long and the exponent SHOULD be less than 5 characters long. This allows time values to have better than micro second precision over the next 100 years.

## 6.1. Examples

### 6.1.1. Single Datapoint

The following shows a temperature reading taken approximately "now" by a 1-wire sensor device that was assigned the unique 1-wire address of 10e2073a01080063:

```
{"e":[{"n": "urn:dev:ow:10e2073a01080063", "v":23.5 }]}
```

### 6.1.2. Multiple Datapoints

The following example shows voltage and current now, i.e., at an unspecified time. The device has an EUI-64 MAC address of 0024beffffe804ff1.

```
{"e":[{"n": "voltage", "t": 0, "u": "V", "v": 120.1 },
      {"n": "current", "t": 0, "u": "A", "v": 1.2 }],
  "bn": "urn:dev:mac:0024beffffe804ff1/"
}
```

The next example is similar to the above one, but shows current at Tue Jun 8 18:01:16 UTC 2010 and at each second for the previous 5 seconds.

```
{"e":[{"n": "voltage", "u": "V", "v": 120.1 },
      {"n": "current", "t": -5, "v": 1.2 },
      {"n": "current", "t": -4, "v": 1.30 },
      {"n": "current", "t": -3, "v": 0.14e1 },
      {"n": "current", "t": -2, "v": 1.5 },
      {"n": "current", "t": -1, "v": 1.6 },
      {"n": "current", "t": 0, "v": 1.7 }],
  "bn": "urn:dev:mac:0024beffffe804ff1/",
  "bt": 1276020076,
  "ver": 1,
  "bu": "A"
}
```



```
}
```

#### 6.1.1.3. Multiple Measurements

The following example shows humidity measurements from a mobile device with an IPv6 address 2001:db8::1, starting at Mon Oct 31 13:24:24 UTC 2011. The device also provide position data, which is provided in the same measurement or parameter array as separate entries. Note time is used to for correlating data that belongs together, e.g., a measurement and a parameter associated with it. Finally, the device also reports extra data about its battery status at a separate time.

```
{ "e": [
  { "v": 20.0, "t": 0 },
  { "sv": "E 24' 30.621", "u": "lon", "t": 0 },
  { "sv": "N 60' 7.965", "u": "lat", "t": 0 },
  { "v": 20.3, "t": 60 },
  { "sv": "E 24' 30.622", "u": "lon", "t": 60 },
  { "sv": "N 60' 7.965", "u": "lat", "t": 60 },
  { "v": 20.7, "t": 120 },
  { "sv": "E 24' 30.623", "u": "lon", "t": 120 },
  { "sv": "N 60' 7.966", "u": "lat", "t": 120 },
  { "v": 98.0, "u": "%EL", "t": 150 },
  { "v": 21.2, "t": 180 },
  { "sv": "E 24' 30.628", "u": "lon", "t": 180 },
  { "sv": "N 60' 7.967", "u": "lat", "t": 180 } ],
  "bn": "http://[2001:db8::1]",
  "bt": 1320067464,
  "bu": "%RH"
}
```

#### 6.1.1.4. Collection of Resources

The following example shows how to query one device that can provide multiple measurements. The example assumes that a client has fetched information from a device at 2001:db8::2 by performing a GET operation on `http://[2001:db8::2]` at Mon Oct 31 16:27:09 UTC 2011, and has gotten two separate values as a result, a temperature and humidity measurement.

```
{ "e": [
  { "n": "temperature", "v": 27.2, "u": "Cel" },
  { "n": "humidity", "v": 80, "u": "%RH" } ],
  "bn": "http://[2001:db8::2]/",
  "bt": 1320078429,
  "ver": 1
}
```

## 7. XML Representation (application/senml+xml)

A SenML object can also be represented in XML format as defined in this section. The following example shows an XML example for the same sensor measurement as in Section 6.1.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<senml xmlns="urn:ietf:params:xml:ns:senml"
  bn="urn:dev:mac:0024beffffe804ff1/"
  bt="1276020076"
  ver="1" bu="A">

  <e n="voltage" u="V" v="120.1" />

  <e n="current" t="-5" v="1.2" />

  <e n="current" t="-4" v="1.30" />

  <e n="current" t="-3" v="0.14e1" />

  <e n="current" t="-2" v="1.5" />

  <e n="current" t="-1" v="1.6" />

  <e n="current" t="0" v="1.7" />

</senml>
```

The RelaxNG schema for the XML is:

```

default namespace = "urn:ietf:params:xml:ns:senml"
namespace rng = "http://relaxng.org/ns/structure/1.0"

e = element e {
  attribute n { xsd:string }?,
  attribute u { xsd:string }?,
  attribute v { xsd:float }?,
  attribute sv { xsd:string }?,
  attribute bv { xsd:boolean }?,
  attribute s { xsd:decimal }?,
  attribute t { xsd:int }?,
  attribute ut { xsd:int }?,
  p*
}

senml =
  element senml {
    attribute bn { xsd:string }?,
    attribute bt { xsd:int }?,
    attribute bu { xsd:string }?,
    attribute ver { xsd:int }?,
    e*
  }

start = senml

```

## 8. EXI Representation (application/senml-exi)

For efficient transmission of SenML over e.g. a constrained network, Efficient XML Interchange (EXI) can be used. This encodes the XML Schema structure of SenML into binary tags and values rather than ASCII text. An EXI representation of SenML SHOULD be made using the strict schema-mode of EXI. This mode however does not allow tag extensions to the schema, and therefore any extensions will be lost in the encoding. For uses where extensions need to be preserved in EXI, the non-strict schema mode of EXI MAY be used.

The EXI header option MUST be included. An EXI schemaID options MUST be set to the value of "a" indicating the scheme provided in this specification. Future revisions to the schema can change this schemaID to allow for backwards compatibility. When the data will be transported over COAP or HTTP, an EXI Cookie SHOULD NOT be used as it simply makes things larger as is redundant to information provided in the Content-Type header.

The following XSD Schema is generated from the RelaxNG and used for strict schema guided EXI processing.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="urn:ietf:params:xml:ns:senml"
  xmlns:ns1="urn:ietf:params:xml:ns:senml">

  <xs:element name="e">
    <xs:complexType>
      <xs:attribute name="n" type="xs:string" minOccurs="0"/>
      <xs:attribute name="u" type="xs:string" minOccurs="0"/>
      <xs:attribute name="v" type="xs:float" minOccurs="0"/>
      <xs:attribute name="sv" type="xs:string" minOccurs="0"/>
      <xs:attribute name="bv" type="xs:boolean" minOccurs="0"/>
      <xs:attribute name="s" type="xs:decimal" minOccurs="0"/>
      <xs:attribute name="t" type="xs:int" minOccurs="0"/>
      <xs:attribute name="ut" type="xs:int" minOccurs="0"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="senml">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="ns1:e"/>
      </xs:sequence>
      <xs:attribute name="bn" type="xs:string"/>
      <xs:attribute name="bt" type="xs:int"/>
      <xs:attribute name="bu" type="xs:string"/>
      <xs:attribute name="ver" type="xs:int"/>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

The following shows a hexdump of the EXI produced from encoding the following XML example. Note that while this example is similar to the first example in Section 6.1.2 in JSON format.

```

<?xml version="1.0" encoding="UTF-8"?>
<senml xmlns="urn:ietf:params:xml:ns:senml"
  bn="urn:dev:ow:10e2073a01080063" >
  <e n="voltage" t="0" v="120.1" u="V" />
  <e n="current" t="0" v="1.2" u="A" />
</senml>

```

Which compresses to the following displayed in hexdump:

```

00000000  a0 30 0d 85 01 d7 57 26  e3 a6 46 57 63 a6 f7 73
00000010  a3 13 06 53 23 03 73 36  13 03 13 03 83 03 03 63
00000020  36 21 2e cd ed 8e 8c 2c  ec a8 00 00 d5 95 88 4c
00000030  02 08 4b 1b ab 93 93 2b  73 a2 00 00 34 14 19 00

```

00000040 c0

The above example used the bit packed form of EXI but it is also possible to use a byte packed form of EXI which can makes it easier for a simple sensor to produce valid EXI without really implementing EXI. Consider the example of a temperature sensor that produces a value in tenths of degrees Celsius over a range of 0.0 to 55.0. = It would produce XML SenML file such as:

```
<?xml version="1.0" encoding="UTF-8"?>
<senml xmlns="urn:ietf:params:xml:ns:senml"
  bn="urn:dev:ow:10e2073a01080063" >
  <e n="temp" v="23.1" u="degC" />
</senml>
```

The compressed form, using the byte alignment option of EXI, for the above XML is the following:

```
00000000 a00048806c200200 1d75726e3a646576 |..H.1 ...urn:dev|
00000010 3a6f773a31306532 3037336130313038 |:ow:10e2073a0108|
00000020 3030363303010674 656d700306646567 |0063...temp..deg|
00000030 430100e701010001 02 |C.....|
```

A small temperature sensor devices that only generates this one EXI file does not really need an full EXI implementation. It can simple hard code the output replacing the one wire device ID starting at byte 0x14 and going to byte 0x23 with it's device ID , and replacing the value "0xe7 0x01" at location 0x33 to 0x34 with the current temperature. The EXI Specification[W3C.REC-exi-20110310] contains the full information on how floating point numbers are represented, but for the purpose of this sensor, the temperature can be converted to an integer in tenths of degrees ( 231 in this example ). EXI stores 7 bits of the integer in each byte with the top bit set to one if there are further bytes. So the first bytes at location 0x33 is set to low 7 bits of the integer temperature in tenths of degrees plus 0x80. In this example  $231 \& 0x7F + 0x80 = 0xE7$ . The second byte at location 0x34 is set to the integer temperature in tenths of degrees right shifted 7 bits. In this example  $231 \gg 7 = 0x01$ .

## 9. Usage Considerations

The measurements support sending both the current value of a sensor as well as the an integrated sum. For many types of measurements, the sum is more useful than the current value. For example, an electrical meter that measures the energy a given computer uses will typically want to measure the cumulative amount of energy used. This is less prone to error than reporting the power each second and

trying to have something on the server side sum together all the power measurements. If the network between the sensor and the meter goes down over some period of time, when it comes back up, the cumulative sum helps reflect what happened while the network was down. A meter like this would typically report a measurement with the units set to watts, but it would put the sum of energy used in the "s" attribute of the measurement. It might optionally include the current power in the "v" attribute.

While the benefit of using the integrated sum is fairly clear for measurements like power and energy, it is less obvious for something like temperature. Reporting the sum of the temperature makes it easy to compute averages even when the individual temperature values are not reported frequently enough to compute accurate averages. Implementors are encouraged to report the cumulative sum as well as the raw value of a given sensor.

Applications that use the cumulative sum values need to understand they are very loosely defined by this specification, and depending on the particular sensor implementation may behave in unexpected ways. Applications should be able to deal with the following issues:

1. Many sensors will allow the cumulative sums to "wrap" back to zero after the value gets sufficiently large.
2. Some sensors will reset the cumulative sum back to zero when the device is reset, loses power, or is replaced with a different sensor.
3. Applications cannot make assumptions about when the device started accumulating values into the sum.

Typically applications can make some assumptions about specific sensors that will allow them to deal with these problems. A common assumption is that for sensors whose measurement values are always positive, the sum should never get smaller; so if the sum does get smaller, the application will know that one of the situations listed above has happened.

## 10. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification.

### 10.1. Units Registry

IANA will create a registry of unit symbols. The primary purpose of this registry is to make sure that symbols uniquely map to give type of measurement. Definitions for many of these units can be found in

[NIST822] and [BIPM].

In addition to the units in this table, any of the Unified Code for Units of Measure [UCUM] in case sensitive form (c/s column) can be prepended by the string "UCUM:" and used in SEML.

Symbol	Description	Reference
m	meter	RFC-AAAA
kg	kilogram	RFC-AAAA
s	second	RFC-AAAA
A	ampere	RFC-AAAA
K	kelvin	RFC-AAAA
cd	candela	RFC-AAAA
mol	mole	RFC-AAAA
Hz	hertz	RFC-AAAA
rad	radian	RFC-AAAA
sr	steradian	RFC-AAAA
N	newton	RFC-AAAA
Pa	pascal	RFC-AAAA
J	joule	RFC-AAAA
W	watt	RFC-AAAA
C	coulomb	RFC-AAAA
V	volt	RFC-AAAA
F	farad	RFC-AAAA
Ohm	ohm	RFC-AAAA
S	siemens	RFC-AAAA
Wb	weber	RFC-AAAA
T	tesla	RFC-AAAA
H	henry	RFC-AAAA
Cel	degrees Celsius	RFC-AAAA
lm	lumen	RFC-AAAA
lx	lux	RFC-AAAA
Bq	becquerel	RFC-AAAA
Gy	gray	RFC-AAAA
Sv	sievert	RFC-AAAA
kat	katal	RFC-AAAA
pH	pH acidity	RFC-AAAA
%	Value of a switch. A value of 0.0 indicates the switch is off while 100.0 indicates on.	RFC-AAAA
count	counter value	RFC-AAAA
%RH	Relative Humidity	RFC-AAAA
m2	area	RFC-AAAA
l	volume in liters	RFC-AAAA
m/s	velocity	RFC-AAAA
m/s2	acceleration	RFC-AAAA
l/s	flow rate in liters per second	RFC-AAAA

W/m2	irradiance	RFC-AAAA
cd/m2	luminance	RFC-AAAA
Bspl	bel sound pressure level	RFC-AAAA
bit/s	bits per second	RFC-AAAA
lat	degrees latitude. Assumed to be in WGS84 unless another reference frame is known for the sensor.	RFC-AAAA
lon	degrees longitude. Assumed to be in WGS84 unless another reference frame is known for the sensor.	RFC-AAAA
%EL	remaining battery energy level in percents	RFC-AAAA
EL	remaining battery energy level in seconds	RFC-AAAA
beet/m	Heart rate in beats per minute	RFC-AAAA
beets	Cumulative number of heart beats	RFC-AAAA

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. Experts should exercise their own good judgment but need to consider the following guidelines:

1. There needs to be a real and compelling use for any new unit to be added.
2. Units should define the semantic information and be chosen carefully. Implementors need to remember that the same word may be used in different real-life contexts. For example, degrees when measuring latitude have no semantic relation to degrees when measuring temperature; thus two different units are needed.
3. These measurements are produced by computers for consumption by computers. The principle is that conversion has to be easily be done when both reading and writing the media type. The value of a single canonical representation outweighs the convenience of easy human representations or loss of precision in a conversion.
4. Use of SI prefixes such as "k" before the unit is not allowed. Instead one can represent the value using scientific notation such a 1.2e3.
5. For a given type of measurement, there will only be one unit type defined. So for length, meters are defined and other lengths such as mile, foot, light year are not allowed. For most cases, the SI unit is preferred.
6. Symbol names that could be easily confused with existing common units or units combined with prefixes should be avoided. For example, selecting a unit name of "mph" to indicate something that had nothing to do with velocity would be a bad choice, as "mph" is commonly used to mean miles per hour.
7. The following should not be used because they are common SI prefixes: Y, Z, E, P, T, G, M, k, h, da, d, c, n, u, p, f, a, z, y, Ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi.



8. The following units should not be used as they are commonly used to represent other measurements Ky, Gal, dyn, etg, P, St, Mx, G, Oe, Gb, sb, Lmb, ph, Ci, R, RAD, REM, gal, bbl, qt, degF, Cal, BTU, HP, pH, B/s, psi, Torr, atm, at, bar, kWh.
9. The unit names are case sensitive and the correct case needs to be used, but symbols that differ only in case should not be allocated.
10. A number after a unit typically indicates the previous unit raised to that power, and the / indicates that the units that follow are the reciprocal. A unit should have only one / in the name.

#### 10.2. Media Type Registration

The following registrations are done following the procedure specified in [RFC4288] and [RFC3023].

Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification.

##### 10.2.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC4627]. Specifically, only the ASCII[RFC0020] subset of the UTF-8 characters are allowed. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any JSON

key value pairs that they do not understand. This allows backwards compatibility extensions to this specification. The "ver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

#### 10.2.2. senml+xml Media Type Registration

Type name: application

Subtype name: senml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

#### 10.2.3. senml-exi Media Type Registration

Type name: application

Subtype name: senml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

### 10.3. XML Namespace Registration

This document registers the following XML name spaces in the IETF XML registry defined in [RFC3688].

URI: urn:ietf:params:xml:ns:senml

Registrant Contact: The IESG.

XML: N/A, the requested URIs are XML namespaces

## 11. Security Considerations

See Section 12. Further discussion of security proprieties can be found in Section 10.2.

## 12. Privacy Considerations

Sensor data can range from information with almost no security considerations, such as the current temperature in a given city, to highly sensitive medical or location data. This specification provides no security protection for the data but is meant to be used inside another container or transport protocol such as S/MIME or HTTP with TLS that can provide integrity, confidentiality, and authentication information about the source of the data.

## 13. Acknowledgement

We would like to thank Lisa Dusseault, Joe Hildebrand, Lyndsay Campbell, Martin Thomson, John Klensin, Bjoern Hoehrmann, and Carsten Bormann for their review comments.

## 14. References

## 14.1. Normative References

- [BIPM] Bureau International des Poids et Mesures, "The International System of Units (SI)", 8th edition, 2006 .
- [IEEE.754.1985] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE Standard 754, August 1985.
- [NIST822] Thompson, A. and B. Taylor, "Guide for the Use of the International System of Units (SI)", NIST Special Publication 811, 2008 Edition .
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, May 2008.
- [UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure (UCUM)", Regenstrief Institute and Indiana University School of Informatics .
- [W3C.REC-exi-20110310] Kamiya, T. and J. Schneider, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, March 2011, <<http://www.w3.org/TR/2011/REC-exi-20110310>>.

## 14.2. Informative References

- [I-D.arkko-core-dev-urn] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-01 (work in progress), October 2011.

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-10 (work in progress), October 2011.
- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format",  
draft-ietf-core-link-format-14 (work in progress),  
November 2011.
- [RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20,  
October 1969.
- [RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform  
Resource Identifier (URI): Generic Syntax", STD 66,  
RFC 3986, January 2005.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally  
Unique IDentifier (UUID) URN Namespace", RFC 4122,  
July 2005.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6  
Address Text Representation", RFC 5952, August 2010.
- [WADL] Hadley, M., "Web Application Description Language (WADL)",  
2009, <[http://java.net/projects/wadl/sources/svn/content/  
trunk/www/wadl20090202.pdf](http://java.net/projects/wadl/sources/svn/content/trunk/www/wadl20090202.pdf)>.

#### Authors' Addresses

Cullen Jennings  
Cisco  
170 West Tasman Drive  
San Jose, CA 95134  
USA

Phone: +1 408 421-9990  
Email: [fluffy@cisco.com](mailto:fluffy@cisco.com)

Zach Shelby  
Sensinode  
Kidekuja 2  
Vuokatti 88600  
FINLAND

Phone: +358407796297  
Email: zach@sensinode.com

Jari Arkko  
Ericsson  
Jorvas 02420  
Finland

Email: jari.arkko@piuha.net





CORE WG  
Internet-Draft  
Intended status: Standards Track  
Expires: August 28, 2013

A. Rahman  
InterDigital Communications, LLC  
February 24, 2013

Enhanced Sleepy Node Support for CoAP  
draft-rahman-core-sleepy-02

Abstract

CoAP is a RESTful application protocol for constrained devices. These devices typically have some combination of limited battery power, small memory footprint and low throughput links. It is expected that in CoAP networks there will be a certain portion of devices that are "sleepy" and which may occasionally go into a sleep mode (i.e. go into a low power state to conserve power) and temporarily suspend CoAP protocol communication. This document proposes a minimal and efficient mechanism building on the Resource Directory concept to enhance sleepy node support in CoAP networks.

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 August 28, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Terminology and Conventions . . . . .	3
2. Introduction . . . . .	3
3. Proposal . . . . .	4
3.1. RD Based Sleep Tracking . . . . .	4
3.2. Example of Synchronous RD Based Sleep Tracking . . . . .	5
3.3. Example of Asynchronous RD Based Sleep Tracking . . . . .	7
3.4. RD Caching Proxy . . . . .	11
3.5. Experimental Results . . . . .	12
4. Acknowledgements . . . . .	13
5. IANA Considerations . . . . .	13
6. Security Considerations . . . . .	13
7. References . . . . .	14
7.1. Normative References . . . . .	14
7.2. Informative References . . . . .	14
Author's Address . . . . .	14

## 1. Terminology and Conventions

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

This document assumes readers are familiar with the terms and concepts that are used in [I-D.ietf-core-coap] and [RFC6690]. In addition, this document defines the following terminology:

### Sleepy Node

A sleepy node is a CoAP client or server that may sometimes go into a sleep mode (i.e. go into a low power state to conserve power) and temporarily suspend CoAP protocol communication. A sleepy node may also sometimes remain in a fully powered on state where it has the capability to perform full CoAP protocol communication.

### Non-Sleepy Node

A non-sleepy node is a CoAP client or server that always remains in a fully powered on state (i.e. always awake) where it has the capability to perform full CoAP protocol communication. The general operation of non-sleepy nodes are assumed to be well known and so are not explicitly spelled out in this document except where needed for clarity.

## 2. Introduction

The current CoAP approach assumes a minimal support of sleepy nodes as follows:

- o [I-D.ietf-core-coap] defines CoAP proxies which can cache and service requests for sleepy CoAP servers. A client explicitly sends a CoAP request (GET) to a proxy (identified by its IP address) while indicating the URI (of the resource of interest) associated to a sleepy CoAP origin server. If the proxy has a valid representation of the resource in its cache it can then respond directly to the client regardless of the current sleep state of the origin server. Otherwise the proxy has to attempt to retrieve (GET) the resource from the sleepy origin server. The attempt may or may not be successful depending on the sleep state of the origin server.
- o [RFC6690] and [I-D.shelby-core-resource-directory] defines a Resource Directory (RD) mechanism where sleepy CoAP servers can register/update (POST/PUT to "/.well-known/core") their list of resources on a central (non-sleepy) RD server. This allows

clients to discover the list of resources from the RD (GET /rd-lookup/...) for a sleepy server, regardless of its current sleep state. Unlike a proxy, the RD stores only the URIs (i.e. CORE Link Format) for other nodes, and not the actual resource representation. The client then may attempt to retrieve (GET) the actual representation of the desired resource from the sleepy origin server. The attempt may or may not be successful depending on the sleep state of the origin server.

- o Lower layer (i.e. below the IP layer) support for sleepy nodes exist in most wireless technologies (e.g. IEEE 802.11 (WiFi), and IEEE 802.15.4 (ZigBee)). For example, most wireless technologies support limited functionality such as packet scheduling to account for sleepy nodes in their physical and MAC layer protocols. These lower layer functionalities are not aware of any specific timing or operational aspects of application layer protocols like CoAP.

### 3. Proposal

#### 3.1. RD Based Sleep Tracking

The current CoAP approach to support sleepy nodes can be significantly improved by introducing RD based mechanisms for a CoAP client to determine whether:

- o A targeted resource is located on a sleepy server.
- o A sleepy server is currently in sleep mode or not.

We define the following new parameters to characterize a sleepy node:

- o SleepState - Indicates whether the node is currently in sleep mode or not (i.e. Sleeping or Awake).
- o SleepDuration - Indicates the maximum duration of time that the node stays in sleep mode.
- o TimeSleeping - Indicates the length of time the node has been sleeping (i.e. if Sleep State = Sleeping).
- o NextSleep - Indicates the next time the node will go to sleep (i.e. if Sleep State = Awake).

These parameters are all server (node) level and are new parameters added to the RD URI Template Variables defined in [I-D.shelby-core-resource-directory].

We also define a new lookup-type ("ss") for the RD lookup interface specified in [I-D.shelby-core-resource-directory]. This new lookup-type supports looking up the SleepState of a specified end-point.

The three time based parameters (SleepDuration, TimeSleeping, NextSleep) can be based on either an absolute network time (for a time synchronized network) or a relative local time (measured at the local node).

Following the approach of [RFC6690] and [I-D.shelby-core-resource-directory], sleep parameters for sleepy servers can be stored by the server in the RD and accessed by all interested clients. Examples of using these parameters in a synchronous or asynchronous manner are shown in the following sections.

### 3.2. Example of Synchronous RD Based Sleep Tracking

Figure 1 shows an example of using RD based sleep tracking in a synchronous fashion:

(1) SleepyNode-1 is awake and having previously discovered the local RD, stores its CORE link format in the RD (POST/rd) identified by its entry point (?ep=SleepyNode-1). The sleep parameters are also updated as part of this step.

(2)-(3) RD services the POST and stores the CORE link format and starts the sleep timers for this node.

(4) SleepyNode-1 falls asleep.

(5) A client is interested in temperature sensors in this domain and does a lookup on the RD.

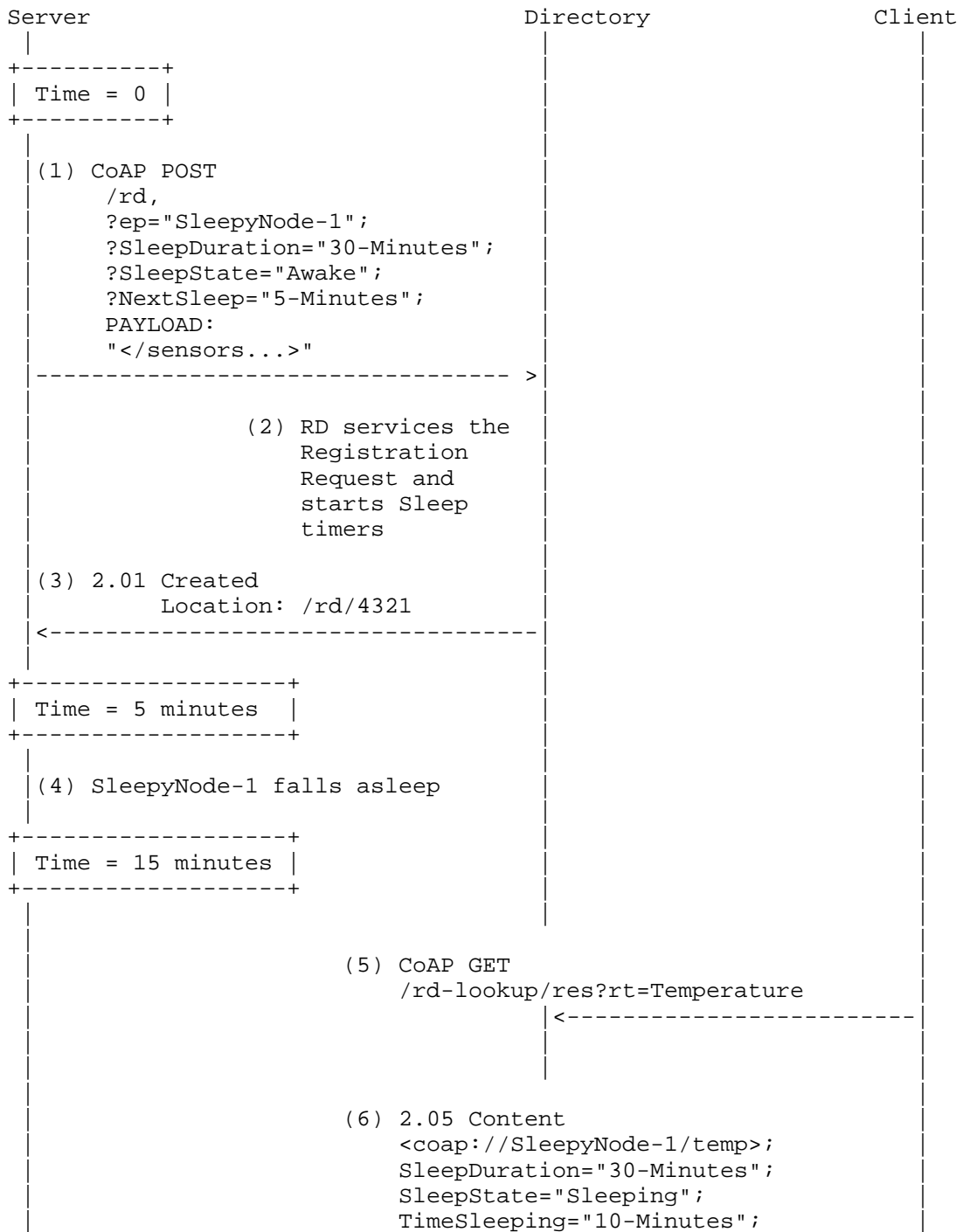
(6) The RD does a lookup and finds that SleepyNode-1 is the only node meeting the match and sends back the required info including the Sleep parameters.

(7) From the sleep parameters, the client knows that the node is currently asleep and so internally schedules to send the GET request when the node wakes up (plus a small safety hysteresis).

(8)-(9) Client sends GET request for temperature sensors and successfully receives the content as SleepyNode-1 is awake.

SleepyNode-1

Resource



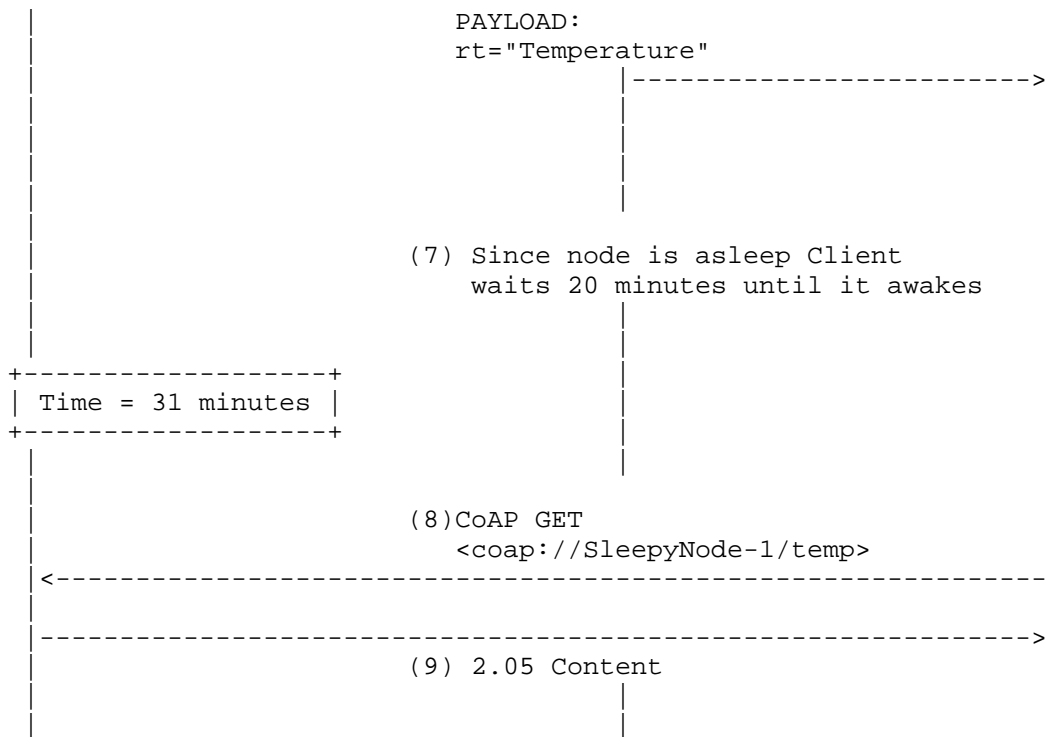


Figure 1: Synchronous Resource Directory Based Sleep Tracking

### 3.3. Example of Asynchronous RD Based Sleep Tracking

Figure 2 shows an example of using RD based sleep tracking in an asynchronous fashion:

(1) SleepyNode-1 is awake and having previously discovered the local RD, stores its CORE link format in the RD (POST/rd) identified by its entry point (?ep=SleepyNode-1).

(2)-(3) RD services the POST and stores the CORE link format.

(4) A client is interested in temperature sensors in this domain and does a lookup on the RD for all sensors that are currently awake.

(5) The RD does a lookup and finds that SleepyNode-1 is the only node meeting the match and sends back the required info.

(6)-(7) Using the sleep state lookup functionality (lookup-type := "ss"), the client adds itself to the list of observers to get

SleepState updates from RD for SleepyNode-1 [I-D.ietf-core-observe].

(8)-(9) Client performs RD 'resource' lookup to find URI of temperature sensor of resource hosted on SleepyNode-1.

(10)-(13) SleepyNode-1 prepares to go to sleep and updates the SleepState in the RD.

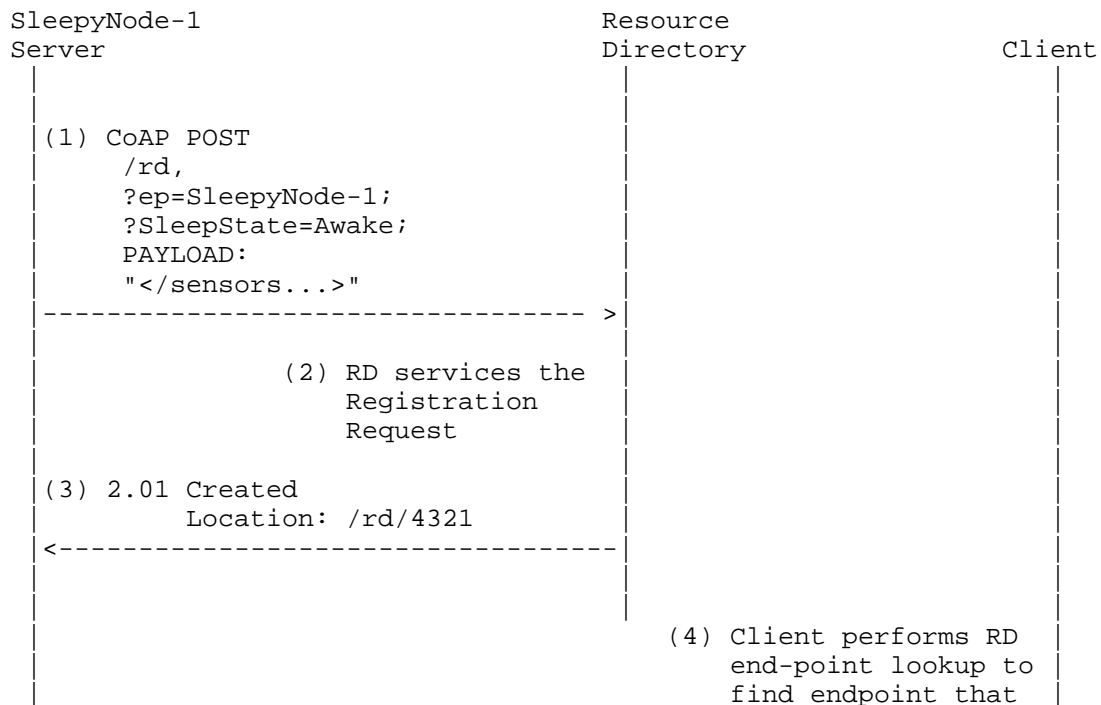
(14) RD notifies the client through previously established observe relationship.

(15) Client application wants to get the temperature now but does not send the request as it knows SleepyNode-1 is currently sleeping.

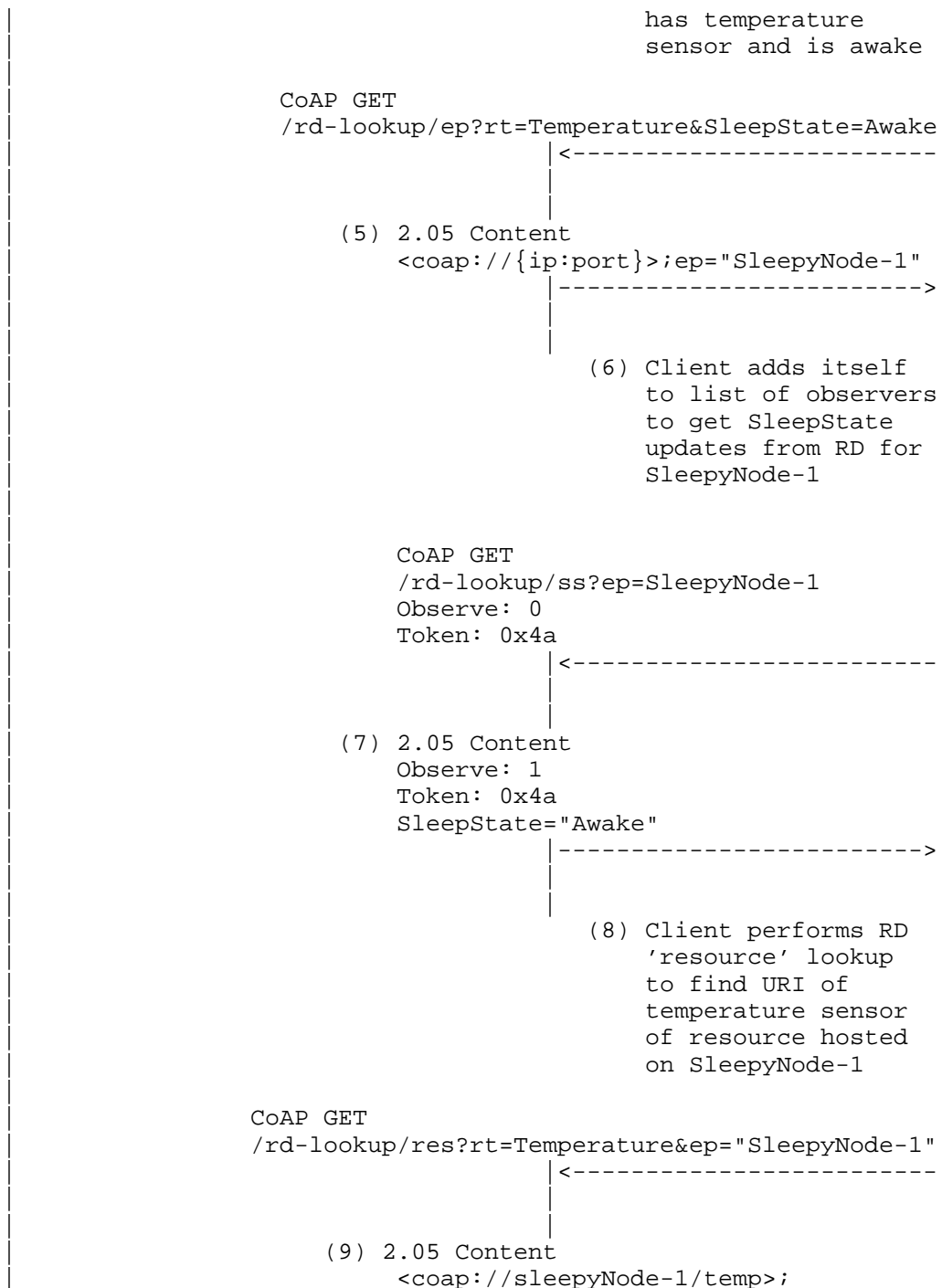
(16)-(19) SleepyNode-1 wakes up and updates the SleepState in the RD.

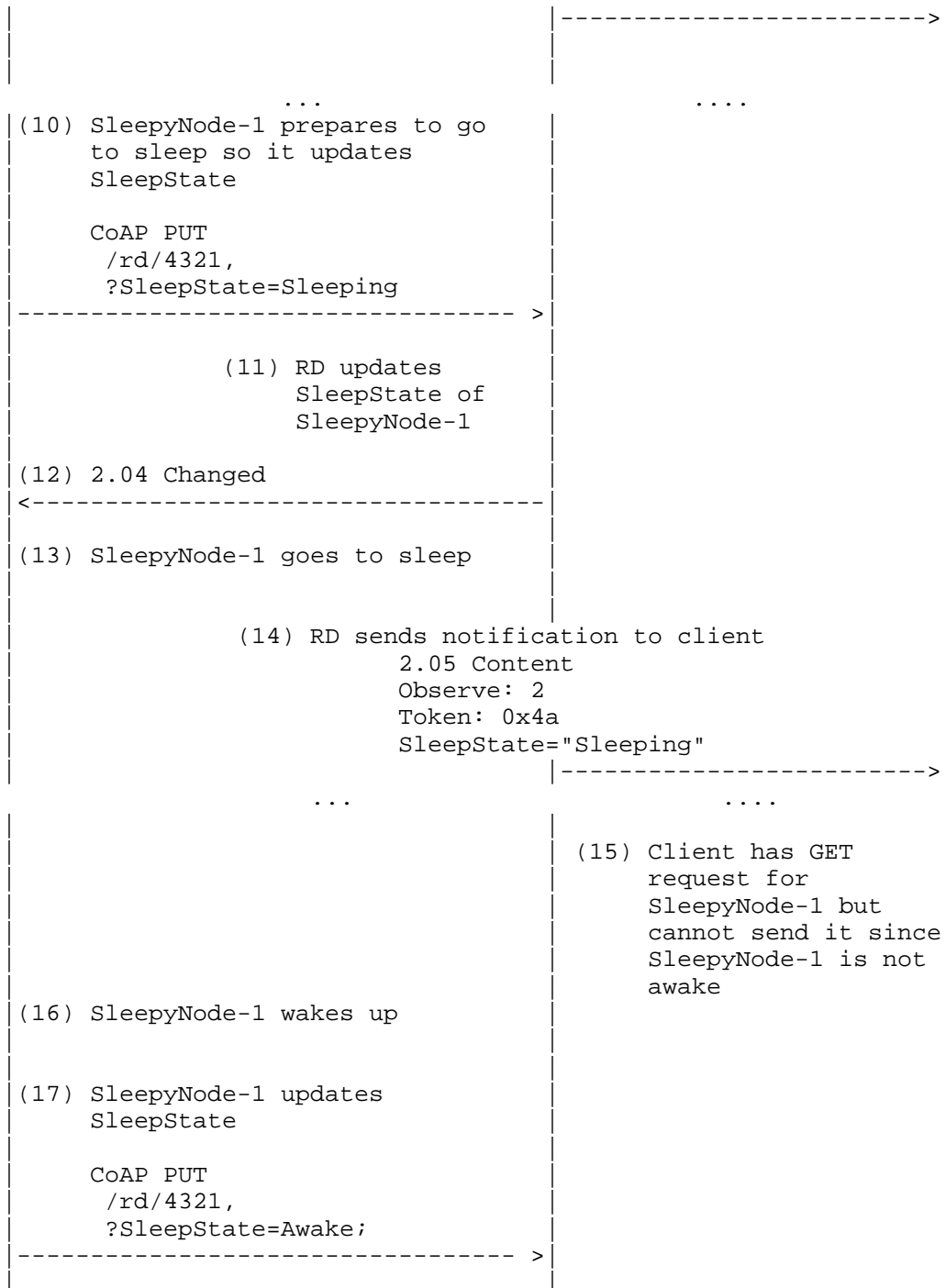
(20)-(21) RD notifies the client through previously established observe relationship.

(22)-(23) Client sends GET request for temperature sensors and successfully receives the content as SleepyNode-1 is awake.









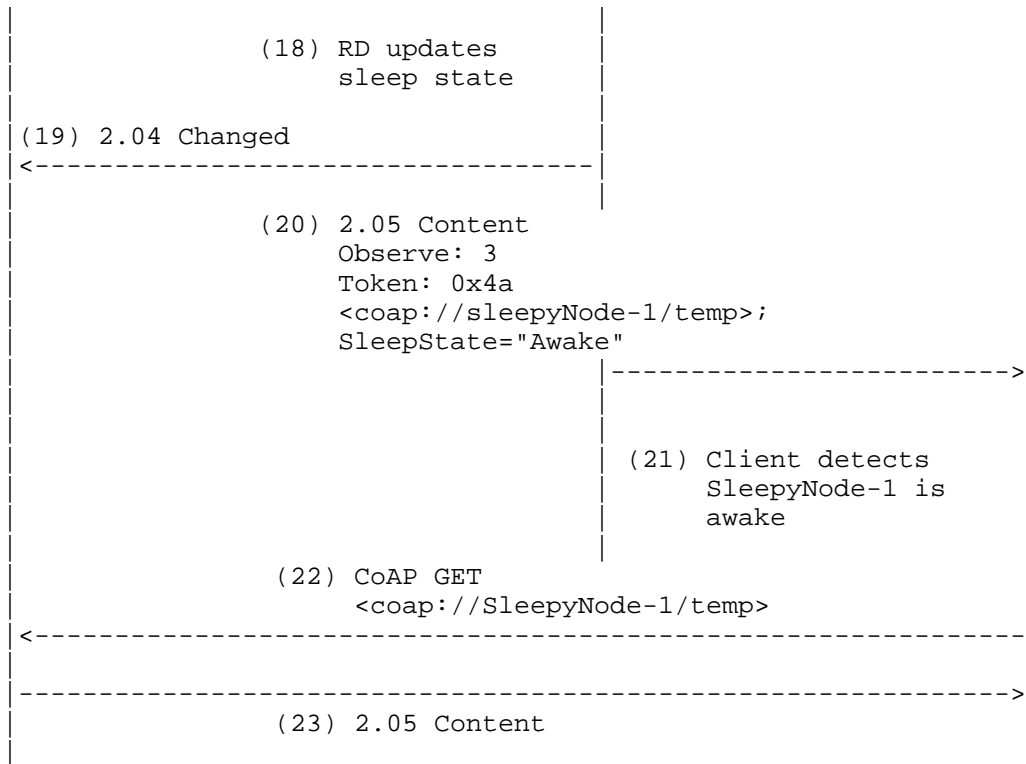


Figure 2: Asynchronous Resource Directory Based Sleep Tracking

### 3.4. RD Caching Proxy

It would be useful for an RD to be able to indicate which proxy performs caching for Sleepy CoAP nodes (see Section 2). This would be done through a new RD "CachingProxy" attribute for each device (similar to the attributes defined in Section 3.1):

- o An RD may be co-located with a proxy that performs caching for CoAP nodes. In this case, the RD automatically adds itself to each CachingProxy entry.
- o The sleepy node itself could suggest the CachingProxy if it is peered to a specific proxy.

This parameter would be added to the RD URI Template Variables defined in [I-D.shelby-core-resource-directory].

### 3.5. Experimental Results

A simple prototype was implemented to validate certain aspects of the performance of the proposed CoAP sleepy node support protocol enhancement. The network for the prototype is shown in Figure 3.

Key aspects of the prototype are as follows:

- o There are two modes of operation: "Caching Only" and "Caching and Sleep Aware"
- o In "Caching Only" mode the Reverse Proxy will cache and service requests according to the "Max-Age" parameter as defined in [I-D.ietf-core-coap].
- o In "Caching and Sleep Aware" mode the Reverse Proxy will also cache and service requests according to "Max-Age". In addition, the proxy will also be aware of the "SleepDuration", "NextSleep" and "SleepState" sleepy node parameters as defined in Section 3.1. Based on these sleep parameters, the proxy will send a "5.03 Service Unavailable (and retry after)" for servers that are currently asleep and for which no cache is available.
- o The key variables in the experiment are: (1) Number of clients; (2) Number of sleepy servers; (3) Delay between client requests; (4) MaxAge; (5) SleepDuration; (6) NextSleep; and (7) SleepState.
- o The target of the experiment will be to measure the amount of traffic over the network in the two modes of operation (for the same input client requests profile). It is hypothesized that the "Caching and Sleep Aware" mode will have the minimal amount of network traffic indicating that the Sleep Aware network will be the most efficient.

Experimental results are being generated now and will be available for discussion during the IETF-86 meeting.

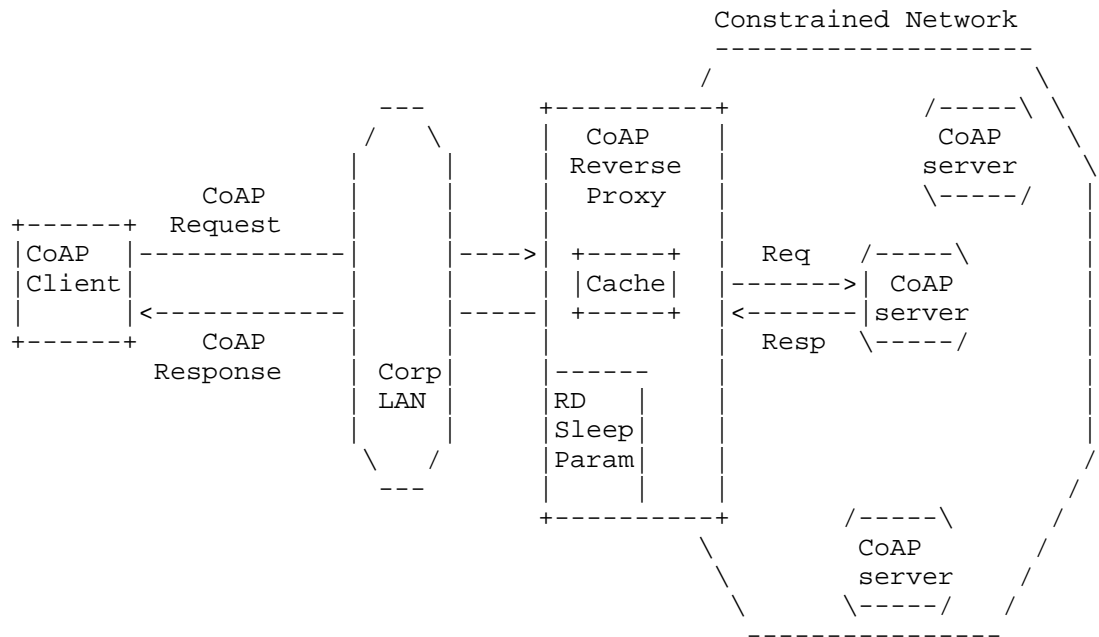


Figure 3: Experimental Setup

#### 4. Acknowledgements

Thanks to Thomas Fossati, Salvatore Loreto, Dale Seed, and Zach Shelby for valuable discussions and feedback on this document.

#### 5. IANA Considerations

This memo includes no request to IANA.

#### 6. Security Considerations

TBD. (All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.)

#### 7. References

## 7.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-07 (work in progress),  
October 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link  
Format", RFC 6690, August 2012.

## 7.2. Informative References

- [I-D.shelby-core-resource-directory]  
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource  
Directory", draft-shelby-core-resource-directory-04 (work  
in progress), July 2012.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC  
Text on Security Considerations", BCP 72, RFC 3552,  
July 2003.

## Author's Address

Akbar Rahman  
InterDigital Communications, LLC  
Montreal, Quebec H3A 3G4  
Canada

Phone: +1-514-585-0761  
Email: akbar.rahman@interdigital.com



Core  
Internet-Draft  
Intended status: Standards Track  
Expires: August 22, 2013

B. Sarikaya  
Huawei USA  
February 18, 2013

Security Bootstrapping Solution for Resource-Constrained Devices  
draft-sarikaya-core-secure-bootsolution-00

Abstract

We present a solution to initially configure the network of resource constrained nodes securely, a.k.a., security bootstrapping. The solution is based on EAP-TLS authentication with the use of raw public keys as certificates.

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 August 22, 2013.

Copyright Notice

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

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



Table of Contents

1. Introduction . . . . .	3
2. Secure Bootstrapping Architecture . . . . .	3
3. Secure Bootstrapping Solution using Raw Public Keys . . . . .	4
4. Transporting EAP Messages . . . . .	6
5. Future Work . . . . .	7
6. Security Considerations . . . . .	8
7. IANA Considerations . . . . .	9
8. Contributors . . . . .	9
9. Acknowledgements . . . . .	9
10. References . . . . .	9
10.1. Normative References . . . . .	9
10.2. Informative References . . . . .	10
Author's Address . . . . .	12

## 1. Introduction

Bootstrapping is any processing required before the network can operate. The bootstrapping problem is not specific to any MAC or PHY. This problem exists across any two nodes which have no previous knowledge of each other. In particular, this problem is complicated when the nodes are resource-constrained and may not have an advanced user interface.

Bootstrapping needs to be secure to make sure that the network operation is secure and hence secure bootstrapping ensures that only the authorized nodes can get access to the network. Because of this secure bootstrapping needs to precede IP address configuration.

[I-D.jennings-core-transitive-trust-enrollment] defines a protocol that enables sensors to securely connect into a system that uses them. The protocol which is being defined is based on the Device using HTTP or COAP [I-D.ietf-core-coap] to communicate with the Controller. This seems to assume that the device is already configured with an IP address. Such an assumption violates the assumption we have in this document on secure bootstrapping.

Transport Layer Security (TLS) is commonly used protocol to secure web browsing, emailing, or other client-server applications. In TLS, the client and the server present their certificates and authenticate each other. Recently, raw public key extension is defined to be used as certificates [I-D.ietf-tls-oob-pubkey]. In this document we use the raw public keys in EAP-TLS.

The document continues in Section 2 on bootstrapping architecture, in Section 3 on secure bootstrapping solution, in Section 4 on transporting EAP messages, in Section 5 on future work.

## 2. Secure Bootstrapping Architecture

Security bootstrapping architecture is structured in a hierarchy of nodes going from the least resource constraint to the most resource constraint. At the top there is a root node. The root node is called Coordinator or Trust Center in Zigbee and 6LoWPAN Border Router (6LBR) in 6LoWPAN ND.

At the next level there are interior Routers. Routers are able to run a routing protocol between other routers and the root. Routers are called 6LoWPAN Routers (6BR) in 6LoWPAN ND.

At the lowest level there are the nodes. The nodes do not run a routing protocol. They can connect to the nearest router over a

single radio link. The nodes are called End Devices in Zigbee and hosts in 6LoWPAN ND.

Routers first join the network as a node and go through security bootstrapping operations in order to create a Master Session Key (MSK). Next, routers execute routing protocol, e.g. [RFC6550] specific steps to create session keys with their neighbors and to establish upstream and downstream next hop parents.

At each node hierarchy level described above, there are lower-layer and higher-layer protocols to bootstrap their ciphering keys, where the lower-layer refers to layers below IP layer including IEEE 802.15.4 MAC layer and LoWPAN adaptation layer and the higher-layer refers to IP layer and the above. In general, required bootstrapping procedures depend on the bootstrapping protocols to use. Section 3 describes the bootstrapping procedures where EAP (Extensible Authentication Protocol) [RFC3748] and other protocols are used as the bootstrapping protocols.

### 3. Secure Bootstrapping Solution using Raw Public Keys

When a new resource-constrained device is deployed, it configures its global unique IPv6 address first. This is done by 6LoWPAN Neighbor Discovery (6LoWPAN-ND)'s Router Solicitation/Router Advertisement message exchange [RFC6775]. The newly generated IPv6 address can not be used until the joining device is authenticated and securely joins the network. After the authentication, the joining device receives the current group key of the network, so that the IPv6 registration and further communication can be protected by the link layer ciphering e.g. 802.15.4, then it can start using its global unique IPv6 address for communication.

For authentication, Extensible Authentication Protocol (EAP) MUST be used. EAP authentication framework is explained in [RFC5247].

The EAP method EAP-TLS [RFC5216] can be used for the resource-constrained device authentication. Instead of X.509 certificates, raw public key of the device MUST be used. EAP-TLS is executed between the joining device and the AAA server which acts as the Authentication Server (AS). After a successful authentication, the device and the AAA server establish a Master Session Key (MSK), and then the AAA server exports the MSK to the authenticator. Upon receipt of the MSK, the authenticator distributes the group key to the joining device within the authentication success message. The group key is encrypted by a Key Encryption Key derived from the MSK.

The resource-constrained device initiates the EAP authentication

process by sending a message of initiation to the authenticator, i.e. the root node or 6LBR. The root node requests the identity from the device by sending an EAP-Request/Identity packet. The device replies with an EAP-Response/Identity containing the device's ID. The identity information includes the device's network access ID (NAI). When the root node receives NAI of the device, it sends the identity information to the AS.

The AS starts the EAP-TLS authentication process by sending a EAP-TLS/Start packet which is an EAP-Request packet with EAP-Type=EAP-TLS to the device. The device generates a client random number and responds with an EAP-Response/TLS-Client-Hello message which contains the TLS version, a client random number, a set of cipher suites. Only one cipher suite MUST be offered in Client-Hello message with RC4-SHA1. EAP-Response packet MUST have the EAP-Type value set at EAP-TLS Figure 1.

The device MUST add an extension of type client certificate type and server certificate type defined in [I-D.ietf-tls-oob-pubkey] to Client-Hello message. Both of these types MUST be set to RawPublicKey.

Upon receipt of Client Hello, if the AS supports raw public key extension, it generates a server random number, a new session ID, server certificate type set to RawPublicKey and includes only the SubjectPublicKeyInfo part of the certificate with its raw public key, rather than the whole certificate in the Certificate message and then sends them to the device with an EAP-Request/TLS-Server-Hello message. Server-Key-Exchange message contains a temporary key for the client to encrypt Client Key Exchange message. For the device, the server adds certificate request message to ask for the device's RawPublicKey using client certificate type message.

Device receives AS's RawPublicKey. Device SHOULD verify the key using out of band mechanisms. Device sends Client Certificate message containing the device's RawPublicKey. With the client and server random number, the device generates a pre\_master\_secret, then sends it in Client-Key-Exchange field of EAP-Response/TLS-Client-Finished message to the AS encrypting pre\_master\_secret with the temporary key in Server-Key-Exchange message. Device includes Change Cypher Spec message to indicate that all messages that follow Client Finished message will be encrypted.

The AS derives the Master Session Key (MSK) and replies with EAP-Request/TLS-Server-Finished message. In this message, the server includes Change Cypher Spec message to indicate that the server will begin encrypting messages with the keys negotiated. The device also derives the MSK after receiving the Server Finished and acknowledges

with EAP-Response/EAP-TLS message.

The AS then exports the MSK to the authenticator in RADIUS Access-Accept message, the authenticator subsequently sends the EAP-Success message to the device. The AS MUST send the group key in this message and the EAP-TLS ends.

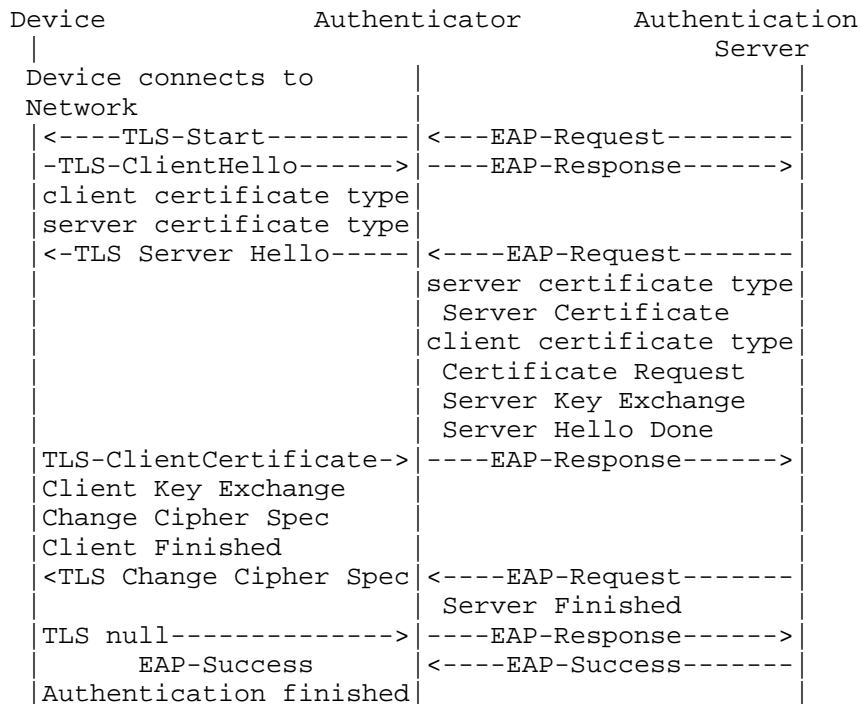


Figure 1: Authentication Call Flow

#### 4. Transporting EAP Messages

EAP can be transported between the device and the authenticator either in Layer 3 using PANA [RFC5191] or in Layer 2 using IEEE 802.1X [802.1x].

EAP is transported using RADIUS [RFC2865] between the authenticator and authentication server.

When a device is not a direct neighbor of the authenticator, its parent node MUST act as relay. Different EAP encapsulation protocols

have different mechanisms for the relay function, such as the PANA Relay Element (PRE).

After the keys are established from a successful EAP method (such as EAP-TLS), the device runs neighbor discovery protocol to get an IPv6 address assigned [RFC6775]. Data transfer can be secured using DTLS or IPSec. Keys derived from EAP TLS are used in either generating DTLS ciphering keys after a successful DTLS handshake or IPSec ESP ciphering keys after a successful IKEv2 handshake.

## 5. Future Work

The nodes in a constrained network called devices have wide range of capabilities and are used in diverse number of applications. Different secure bootstrapping solutions may apply to different applications and different types of nodes. In all cases, it is assumed in this document that the devices are IPv6 enabled.

The solution described in Section 3 has the most stringent requirements on the devices and therefore is not suitable on less constrained nodes. It seems that the devices used in smart metering may have enough resources to run the bootstrapping protocol and they do not suffer from power constraints compared with most other devices such as light switches.

One possible optimization in Figure 1 applies to the case where the device does not have a RawPublicKey. In this case the device sends only `server_certificate_type` set to RawPublicKey in Client-Hello message. In response, AS sends its RawPublicKey in Server Hello message. As a result the messages are much simpler than in Figure 1.

Further optimizations to the EAP-TLS call flow in Figure 1 are TBD.

Simpler devices such as light switches, environmental sensors, etc. may have much less resources, much less constrained IPv6 stack and they may not stay on for long periods of times required from the execution of the secure bootstrapping protocol.

Identification of a set of applications with similar device capabilities is TBD.

Modification of the protocol defined in Section 3 to define a secure bootstrapping protocol for each set is TBD.

## 6. Security Considerations

When security bootstrapping resource constraint nodes is undertaken, several attacks are possible and security bootstrapping methods described in this document do not protect the nodes against such attacks. These attacks are similar to the ones described in [RFC3971] and mainly stem from unsecured link layer. Link layer must be secured on each node before the node can begin security bootstrapping.

If a bootstrapping protocol does not rely on a pre-shared key for peer authentication, it must rely on an online or offline third-party (e.g., an authentication server, a key distribution center in Kerberos, a certification authority in PKI, a private key generator in ID-based cryptography and so on) to prevent man-in-the-middle attacks during peer authentication. Depending on use cases, a resource-constrained device may not always have access to an online third-party for peer authentication.

Depending on use cases, a bootstrapping protocol may deal with authorization separately from authentication in terms of timing and signaling path. For example, two resource-constrained devices A and B may perform mutual authentication using authentication credentials provided by an offline third-party X whereas resource-constrained device A obtains authorization for running a particular application with resource-constrained device B from an online third-party Y before or after the authentication. In some use cases, authentication and authorization are tightly coupled, e.g., successful authentication also means successful authorization. A bootstrapping protocol supports various types of authentication and authorization or different bootstrapping protocols may be used for different types of authentication and authorization.

If authorization information includes cryptographic keys, a special care must be taken for dealing with the keys, e.g., guidelines for AAA-based key management are described in [RFC4962]. A recommissioning use case may require revocation and re-installation of authentication credentials (i.e., a certificate or a shared secret and identity information, etc.) to a large number of resource-constrained devices that are already deployed. Re-installation of authentication credentials must be as secure as the initial installation regardless of whether the re-installation is done manually or automatically.

If resource-constrained devices use a multicast group key for peer authentication or message authentication or encryption, the group key must be securely distributed to the current members of the group for both initial key distribution and key update. Protocols designed for

group key management such as GSAKMP [RFC4535], GDOI [RFC3547] and MIKEY [RFC3830] may be used for group key distribution. Alternatively, key wrap attributes for securely encapsulating group key may be defined in network access authentication protocols such as PANA [RFC5191] and EAP-TTLSv0 [RFC5281]. Those protocols use an end-to-end, point-to-point communication channel with a pair-wise security association between a key distribution center and each key recipient. Further considerations may be needed for more efficient group key management to support a large number of resource-constrained devices.

## 7. IANA Considerations

This memo includes no request to IANA.

## 8. Contributors

TBD.

## 9. Acknowledgements

TBD.

## 10. References

### 10.1. Normative References

- [802.15.4] IEEE Std 802.15.4-2006, "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)", September 2006.
- [I-D.ietf-tls-oob-pubkey] Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-07 (work in progress), February 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)",



RFC 2865, June 2000.

- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowitz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [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.
- [RFC5191] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.
- [RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, March 2008.
- [RFC5548] Dohler, M., Watteyne, T., Winter, T., and D. Barthel, "Routing Requirements for Urban Low-Power and Lossy Networks", RFC 5548, May 2009.
- [RFC5673] Pister, K., Thubert, P., Dwars, S., and T. Phinney, "Industrial Routing Requirements in Low-Power and Lossy Networks", RFC 5673, October 2009.

## 10.2. Informative References

- [802.1x] IEEE Std 802.1X-2010, "IEEE 802.1X Port-Based Network Access Control", February 2010.
- [C1222] American National Standard, "Protocol Specification For Interfacing to Data Communication Networks", ANSI C12.22-2008, 2008.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.jennings-core-transitive-trust-enrollment]  
Jennings, C., "Transitive Trust Enrollment for Constrained Devices",  
draft-jennings-core-transitive-trust-enrollment-01 (work in progress), October 2012.
- [NISTIR7628VOL1]  
The Smart Grid Interoperability Panel - Cyber Security

Working Group, "Guidelines for Smart Grid Cyber Security: Vol. 1, Smart Grid Cyber Security Strategy, Architecture, and High-Level Requirements", NISTIR 7628, vol. 1, 2010.

- [RFC3547]    Baugher, M., Weis, B., Hardjono, T., and H. Harney, "The Group Domain of Interpretation", RFC 3547, July 2003.
- [RFC3830]    Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.
- [RFC3971]    Arkko, J., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, March 2005.
- [RFC3972]    Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.
- [RFC4279]    Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4347]    Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.
- [RFC4423]    Moskowitz, R. and P. Nikander, "Host Identity Protocol (HIP) Architecture", RFC 4423, May 2006.
- [RFC4535]    Harney, H., Meth, U., Colegrove, A., and G. Gross, "GSAKMP: Group Secure Association Key Management Protocol", RFC 4535, June 2006.
- [RFC4962]    Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, July 2007.
- [RFC5204]    Laganier, J. and L. Eggert, "Host Identity Protocol (HIP) Rendezvous Extension", RFC 5204, April 2008.
- [RFC5247]    Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.
- [RFC5281]    Funk, P. and S. Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, August 2008.
- [RFC5295]    Salowey, J., Dondeti, L., Narayanan, V., and M. Nakhjiri, "Specification for the Derivation of Root Keys from an

Extended Master Session Key (EMSK)", RFC 5295,  
August 2008.

- [RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen,  
"Internet Key Exchange Protocol Version 2 (IKEv2)",  
RFC 5996, September 2010.
- [RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R.,  
Levis, P., Pister, K., Struik, R., Vasseur, JP., and R.  
Alexander, "RPL: IPv6 Routing Protocol for Low-Power and  
Lossy Networks", RFC 6550, March 2012.
- [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.
- [ROMER04] Romer, K. and F. Mattern, "The design space of wireless  
sensor networks", IEEE Wireless Communications, vol. 11,  
no. 6, pp. 54-61, December 2004.
- [SE2.0] ZigBee Alliance, "Smart Energy Profile 2.0 Technical  
Requirements Document", April 2010.

Author's Address

Behcet Sarikaya  
Huawei USA  
5340 Legacy Dr.  
Plano, TX 75024

Email: sarikaya@ieee.org



CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: January 12, 2013

Z. Shelby  
Sensinode  
M. Vial  
Schneider-Electric  
July 11, 2012

CoRE Interfaces  
draft-shelby-core-interfaces-03

Abstract

This document defines well-known REST interface descriptions for Batch, Sensor, Parameter and Actuator types for use in constrained web servers using the CoRE Link Format. A short reference is provided for each type that can be efficiently included in the interface description attribute of the CoRE Link Format. These descriptions are intended to be for general use in resource designs or for inclusion in more specific interface profiles. In addition, this document defines the concepts of Function Set and Binding. The former is the basis element to create RESTful profiles and the latter helps the configuration of links between resources located on one or more endpoints.

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 12, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	3
3. Function Set . . . . .	4
3.1. Defining a Function Set . . . . .	4
3.1.1. Path template . . . . .	5
3.1.2. Resource Type . . . . .	5
3.1.3. Interface Description . . . . .	5
3.1.4. Data type . . . . .	6
3.2. Discovery . . . . .	6
3.3. Versioning . . . . .	6
4. Bindings . . . . .	6
4.1. Format . . . . .	7
4.2. Binding methods . . . . .	8
4.3. Binding table . . . . .	9
5. Interface Descriptions . . . . .	9
5.1. Link List . . . . .	11
5.2. Batch . . . . .	11
5.3. Linked Batch . . . . .	12
5.4. Sensor . . . . .	13
5.5. Parameter . . . . .	14
5.6. Read-only Parameter . . . . .	14
5.7. Actuator . . . . .	15
5.8. Binding . . . . .	15
5.9. Resource Observation . . . . .	16
5.10. Future Interfaces . . . . .	17
5.11. WADL Description . . . . .	17
6. Security Considerations . . . . .	21
7. IANA Considerations . . . . .	21
8. Acknowledgments . . . . .	21
9. Changelog . . . . .	22
10. References . . . . .	22
10.1. Normative References . . . . .	22
10.2. Informative References . . . . .	22
Appendix A. Profile example . . . . .	23
Authors' Addresses . . . . .	24

## 1. Introduction

The Constrained RESTful Environments (CoRE) working group 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 [I-D.ietf-core-link-format] and can be used by CoAP [I-D.ietf-core-coap] or HTTP servers. The CoRE Link Format defines an attribute that can be used to describe the REST interface of a resource, and may include a link to a description document. This memo describes how other specifications can combine resources with a well-known interface to create new CoRE RESTful profiles. A CoRE profile is based on the concept of Function Set, which is a group of REST resources providing a service in a distributed system. In addition, the notion of Binding is introduced in order to create a synchronization link between two resources. This document also defines well-known interface descriptions for Batch, Sensor, Parameter and Actuator types to compose new Function Sets or for standalone use in a constrained web server. A short reference is provided for each type that can be efficiently included in the interface description (if=) attribute of the CoRE Link Format.

## 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 discussed in [RFC5988] and [I-D.ietf-core-link-format]. This specification makes use of the following additional terminology:

Function Set: A group of well-known REST resources that provides a particular service.

Profile: A group of well-known Function Sets defined by a specification.

Device: An IP smart object running a web server that hosts a group of Function Set instances from a profile.

Service Discovery: The process making it possible for a web client to automatically detect devices and Function Sets offered by these devices on a CoRE network.

Resource Discovery: The process allowing a web client to identify resources being hosted on a web server.

Gradual Reveal: A REST design where resources are discovered progressively using Web Linking.

Binding: A unidirectional logical link between a source resource and a destination resource.

### 3. Function Set

This section defines how a specification can organize REST resources to create a new profile. A profile is structured into groups of resource types called Function Sets. A Function Set is similar to a function block in the sense that it consists of input, output and parameter resources and contains internal logic. A Function Set MAY have a subset of mandatory inputs, outputs and parameters to provide minimum interoperability. It MAY also be extended with manufacturer/user-specific resources. Other specifications defines the list of function sets available within a given profile. A device is composed of one or more Function Set instances. A profile specification MAY specify device profiles with mandatory function sets.

#### 3.1. Defining a Function Set

In a Function Set, types of resources are defined. Each type includes a human readable name, a path template, a Resource Type for discovery, the Interface Definition and the data type and allowed values. A Function Set definition may also include a field indicating if a sub-resource is mandatory or optional.



### 3.1.1. Path template

A Function Set is a container resource under which its sub-resources are organized. The profile defines the path to each resource of a Function Set in a path template. The template can contain either relative paths or absolute paths depending on the profile needs. An absolute Function Set SHOULD be located at its recommended root path on a web server, however it MAY be located under an alternative path if necessary (for example multi-purpose devices, gateways etc.). A relative Function Set can be instantiated as many times as needed on a web server with an arbitrary root path. However some Function Sets (e.g. device description) only make sense as singletons.

The path template includes a possible index {#} parameter, and possible fixed path segments. The index {#} allows for multiple instances of this type of resource, and can be any string. The root path and the indexes are the only variable elements in a path template. All other path segments MUST be fixed.

### 3.1.2. Resource Type

Each root resource of a Function Set is assigned a Resource Type parameter, therefore making it possible to discover it. Each sub-resource of a Function Set is also assigned a Resource Type parameter. This Resource Type is used for resource discovery and is usually necessary to discover optional resources supported on a specific device. The Resource Type of a Function Set may also be used for service discovery and MAY be exported to DNS-SD [I-D.cheshire-dnsext-dns-sd] for example.

The Resource Type parameter defines the value that MUST be included in the rt= field of the CoRE Link Format when describing a link to this resource. The value SHOULD be in the form "namespace.type" for root resources and "namespace.type.subtype" for sub-resources. This naming convention facilitates resource type filtering with the /.well-known/core resource. However a profile MAY allow mixing in foreign namespace references within a Function Set to import external references from other object models (e.g. SenML and UCUM).

### 3.1.3. Interface Description

The Interface Description parameter defines the REST interface for that type of resource. Several base interfaces are defined in Section 5 of this document. For a given profile, the Interface Description may be inferred from the Resource Type. In that case the Interface Description MAY be elided from link descriptions of resource types defined in the profile, but SHOULD be included for custom extensions to the profile.

The root resource of a Function Set should provide a list of links to its sub-resources in order to offer gradual reveal of resources. The CoRE Link List interface defined in Section 5.1 offers this functionality so a root resource SHOULD support this interface or a derived interface like CoRE Batch (See Section 5.2).

#### 3.1.4. Data type

The Data Type field defines the type of value (and possible range) that is returned in response to a GET for that resource or accepted with a PUT. The interfaces defined in Section 5 make use of plain text and SenML Media types for the actual format of this data. A profile may restrict the list of supported content types for the CoRE interfaces or define new interfaces with new content types.

#### 3.2. Discovery

A device conforming to a profile SHOULD make its resources discoverable by providing links to the resources on the path `/.well-known/core` as defined in [I-D.ietf-core-link-format]. All resources hosted on a device SHOULD be discoverable either with a direct link in `/.well-known/core` or by following successive links starting from `/.well-known/core`.

The root path of a Function Set instance SHOULD be directly referenced in `/.well-known/core` in order to offer discovery at the first discovery stage. A device with more than 10 individual resources SHOULD only expose Function Set instances in `/.well-known/core` to limit the size of this resource.

In addition, a device MAY register its resources to a Resource Directory using the registration interface defined in [I-D.shelby-core-resource-directory] if such a directory is available.

#### 3.3. Versioning

A profile should track Function Set changes to avoid incompatibility issues. Evolutions in a Function Set SHOULD be backward compatible.

#### 4. Bindings

In a M2M RESTful environment, endpoints exchange the content of their resources to operate the distributed system. Beforehand, a configuration phase is necessary to determine how the resources of the different endpoints are related to each other. This can be done either automatically using discovery mechanisms or by means of human

intervention and a so-called commissioning tool. In this document the abstract relationship between two resources is called a Binding. The configuration phase necessitates the exchange of binding information so a format recognized by all CoRE endpoints is essential. This document defines a format based on the CoRE Link-Format to represent binding information along with the rules to define a binding method which is a specialized relationship between two resources. The purpose of a binding is to synchronize the content between a source resource and a destination resource. The destination resource MAY be a group resource if the authority component of the destination URI contains a group address (either a multicast address or a name that resolves to a multicast address). Since a binding is unidirectional, the binding entry defining a relationship is present only on one endpoint. The binding entry may be located either on the source or the destination endpoint depending on the binding method. The following table gives a summary of the binding methods described in more detail in Section 4.2.

Name	Identifier	Location	Method
Polling	poll	Destination	GET
Observe	obs	Destination	GET + Observe
Push	push	Source	PUT

#### 4.1. Format

Since Binding lies in the creation of a link between two resources, Web Linking and the CoRE Link-Format are a natural way to represent binding information. This involves the creation of a new relation type, purposely named "boundto". In a Web link with this relation type, the target URI contains the location of the source resource and the context URI points to the destination resource. The Web link attributes allow a fine-grained control of the type of synchronization exchange along with the conditions that trigger an update. This specification defines the attributes below:

Attribute	Parameter	Value
Binding method	bind	xsd:string
Minimum Period (s)	pmin	xsd:integer (>0)
Maximum Period (s)	pmax	xsd:integer (>0)
Change Step	st	xsd:decimal (>0)

**Bind Method:** This is the identifier of a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

**Minimum Period:** When present, the minimum period indicates the minimum time to wait (in seconds) before sending a new synchronization message (even if it has changed). In the absence of this parameter, the minimum period is up to the notifier.

**Maximum Period:** When present, the maximum period indicates the maximum time in seconds between two consecutive synchronization messages (regardless if it has changed). In the absence of this parameter, the maximum period is up to the notifier. The maximum period **MUST** be greater than the minimum period parameter (if present).

**Change Step:** When present, the change step indicates how much the value of a resource **SHOULD** change before sending a new notification (compared to the value of the last notification). This parameter has lower priority than the period parameters, thus even if the change step has been fulfilled, the time since the last notification **SHOULD** be between pmin and pmax.

#### 4.2. Binding methods

A binding method defines the rules to generate the web-transfer exchanges that will effectively send content from the source resource to the destination resource. The description of a binding method must define the following aspects:

**Identifier:** This is value of the "bind" attribute used to identify the method.

**Location:** This information indicates whether the binding entry is stored on the source or on the destination endpoint.

**REST Method:** This is the REST method used in the Request/Response exchanges.

**Conditions:** A binding method definition must state how the condition attributes of the abstract binding definition are actually used in this specialized binding.

This specification supports 3 binding methods described below.

**Polling:** The Polling method consists of sending periodic GET requests from the destination endpoint to the source resource and copying the content to the destination resource. The binding entry for this method **MUST** be stored on the destination endpoint. The destination endpoint **MUST** ensure that the polling frequency does not exceed the limits defined by the pmin and pmax attributes of the binding entry. The copying process **MAY** filter out content from the GET requests using value-based conditions (e.g Change Step).

**Observe:** The Observe method relies on the Publish/Subscribe pattern thus an observation relationship is created between the destination endpoint and the source resource. On each notification the content from the source resource is copied to the destination resource. The creation of the observation relationship requires the CoAP Observation mechanism [I-D.ietf-core-observe] hence this method is only permitted when the resources are made available over CoAP. The binding entry for this method **MUST** be stored on the destination endpoint. The binding conditions are mapped as query string parameters (see Section 5.9).

**Push:** When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource upon change of the source resource. The source endpoint **MUST** only send a notification request if the binding conditions are met. The binding entry for this method **MUST** be stored on the source endpoint.

#### 4.3. Binding table

The binding table is a special resource that gives access to the bindings on a endpoint. A binding table resource **MUST** support the Binding interface defined in Section 5.8. A profile **SHOULD** allow only one resource table per endpoint.

### 5. Interface Descriptions

This section defines REST interfaces for Link List, Batch, Sensor, Parameter, Actuator and Binding table resources. Variants such as Linked Batch or Read-Only Parameter are also presented. Each type is described along with its Interface Description attribute value and valid methods. These are defined for each interface in the table below. These interfaces can support plain text and/or SenML Media types.

The if= column defines the Interface Description (if=) attribute

value to be used in the CoRE Link Format for a resource conforming to that interface. When this value appears in the if= attribute of a link, the resource MUST support the corresponding REST interface described in this section. The resource MAY support additional functionality, which is out of scope for this specification. Although these interface descriptions are intended to be used with the CoRE Link Format, they are applicable for use in any REST interface definition.

The Methods column defines the methods supported by that interface, which are described in more detail below.

Interface	if=	Methods
Link List	core.ll	GET
Batch	core.b	GET, PUT, POST (where applicable)
Linked Batch	core.lb	GET, PUT, POST, DELETE (where applicable)
Sensor	core.s	GET
Parameter	core.p	GET, PUT
Read-only Parameter	core.rp	GET
Actuator	core.a	GET, PUT, POST
Binding	core.bnd	GET, POST, DELETE

The following is an example of links in the CoRE Link Format using these interface descriptions. The resource hierarchy is based on a simple profile defined in Appendix A. These links are used in the subsequent examples below.

```

Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s>;rt="simple.sen";if="core.b",
</s/lt>;rt="simple.sen.lt";if="core.s",
</s/tmp>;rt="simple.sen.tmp";if="core.s";obs,
</s/hum>;rt="simple.sen.hum";if="core.s",
</a>;rt="simple.act";if="core.b",
</a/1/led>;rt="simple.act.led";if="core.a",
</a/2/led>;rt="simple.act.led";if="core.a",
</d>;rt="simple.dev";if="core.ll",
</l>;if="core.lb",

```

### 5.1. Link List

The Link List interface is used to retrieve (GET) a list of resources on a web server. The GET request SHOULD contain an Accept option with the application/link-format content type, however if the resource does not support any other form of GET methods the Accept option MAY be elided. The Accept option SHOULD only include the application/link-format content type. The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on a web server.

Link List is the base interface to provide gradual reveal of resources on a CoRE web server, hence the root resource of a Function Set SHOULD implement this interface or an extension of this interface.

The following example interacts with a Link List /d containing Parameter sub-resources /d/name, /d/model.

```
Req: GET /d (Accept:application/link-format)
Res: 2.05 Content (application/link-format)
</d/name>;rt="simple.dev.n";if="core.p",
</d/model>;rt="simple.dev.mdl";if="core.rp"
```

### 5.2. Batch

The Batch interface is used to manipulate a collection of sub-resources at the same time. The Batch interface type supports the same methods as its sub-resources, and can be used to read (GET), set (PUT) or toggle (POST) the values of those sub-resource with a single resource representation. The sub-resources of a Batch MAY be heterogeneous, a method used on the Batch only applies to sub-resources that support it. For example Sensor interfaces do not support PUT, and thus a PUT request to a Sensor member of that Batch would be ignored. A batch requires the use of SenML Media types in order to support multiple sub-resources.

In addition, The Batch interface is an extension of the Link List interface and in consequence MUST support the same methods.

The following example interacts with a Batch /s with Sensor sub-resources /s/light, /s/temp and /s/humidity.

```
Req: GET /s
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "light", "v": 123, "u": "lx" },
  { "n": "temp", "v": 27.2, "u": "degC" },
  { "n": "humidity", "v": 80, "u": "%RH" }],
}
```

### 5.3. Linked Batch

The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the web server, a Linked Batch is dynamically controlled by a web client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [RFC5988] and the CoRE Link Format [I-D.ietf-core-link-format]. A request with a POST method and a content type of application/link-format simply appends new resources to the collection. The links in the payload MUST reference a resource on the web server with an absolute path. A DELETE request empties the current collection of links. All other requests available for a basic Batch are still valid for a Linked Batch.

The following example interacts with a Linked Batch /1 and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.



```
Req: POST /1 (Content-type: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed
```

```
Req: GET /1
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "degC" },
]
```

```
Req: POST /1 (Content-type: application/link-format)
</s/humidity>
Res: 2.04 Changed
```

```
Req: GET /1 (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>
```

```
Req: GET /1
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "degC" },
  { "n": "/s/humidity", "v": 80, "u": "%RH" }],
}
```

```
Req: DELETE /1
Res: 2.04 Changed
```

#### 5.4. Sensor

The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML. Plain text MAY be used for a single measurement that does not require meta-data. For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

The following are examples of Sensor interface requests in both text/plain and application/senml+json.

```
Req: GET /s/humidity (Accept: text/plain)
Res: 2.05 Content (text/plain)
80
```

```
Req: GET /s/humidity (Accept: application/senml+json)
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "humidity", "v": 80, "u": "%RH" }],
}
```

### 5.5. Parameter

The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read (GET) or set (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading and setting a parameter.

```
Req: GET /d/name
Res: 2.05 Content (text/plain)
node5
```

```
Req: PUT /d/name (text/plain)
outdoor
Res: 2.04 Changed
```

### 5.6. Read-only Parameter

The Read-only Parameter interface allows configuration parameters to be read (GET) but not set. Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading such a parameter.

```
Req: GET /d/model
Res: 2.05 Content (text/plain)
SuperNode200
```

### 5.7. Actuator

The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or a new actuator value set (PUT). In addition, this interface defines the use of POST (with no body) to toggle an actuator between its possible values. Plain text or SenML Media types MAY be returned from this type of interface. A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to set.

The following example shows requests for reading, setting and toggling an actuator (turning on a led).

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0

Req: PUT /a/1/led (text/plain)
1
Res: 2.04 Changed

Req: POST /a/1/led (text/plain)
Res: 2.04 Changed

Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

### 5.8. Binding

The Binding interface is used to manipulate a binding table. A request with a POST method and a content type of application/link-format simply appends new bindings to the table. All links in the payload MUST have a relation type "boundTo". A GET request simply returns the current state of a binding table whereas a DELETE request empties the table.

The following example shows requests for adding, retrieving and deleting bindings in a binding table.

```

Req: POST /bnd (Content-type: application/link-format)
<coap://sensor.example.com/s/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"
Res: 2.04 Changed

```

```

Req: GET /bnd
Res: 2.05 Content (application/senml+json)
<coap://sensor.example.com/s/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"

```

```

Req: DELETE /bnd
Res: 2.04 Changed

```

### 5.9. Resource Observation

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [I-D.ietf-core-observe] MAY be used to observe any changes in a resource, and receive asynchronous notifications as a result. In addition, a set of query string parameters are defined here to allow a client to request how often a client is interested in receiving notifications and how much a resource should change for the new representation to be interesting. These query parameters are described in the following table. A resource using an interface description defined in this specification and marked as Observable in its link description SHOULD support these observation parameters. The Change Step parameter can only be supported on resources with an atomic numeric value.

Query	Parameter	Value
Minimum Period (s)	pmin	xsd:integer (>0)
Maximum Period (s)	pmax	xsd:integer (>0)
Change Step	st	xsd:decimal (>0)

**Minimum Period:** When present, the minimum period indicates the minimum time in seconds the server SHOULD wait between sending notifications. In the absence of this parameter, the minimum period is up to the server.

**Maximum Period:** When present, the maximum period indicated the maximum time in seconds the server SHOULD wait between sending notifications (regardless if it has changed). In the absence of this parameter, the maximum period is up to the server. The maximum period MUST be greater than the minimum period parameter

(if present).

**Change Step:** When present, the change step indicates how much the value of a resource SHOULD change before sending a new notification (compared to the value of the last notification). This parameter has lower priority than the period parameters, thus even if the change step has been fulfilled, the time since the last notification SHOULD be between pmin and pmax.

The following example shows an Observation request using these query parameters. Here the value of Observe indicates the number of seconds since the observation was made to show the time.

```
Req: GET Observe /s/temp?pmin=10&pmax=60&st=1
Res: 2.05 Content Observe:0 (text/plain)
23.2
```

```
Res: 2.05 Content Observe:60 (text/plain)
23.0
```

```
Res: 2.05 Content Observe:80 (text/plain)
22.0
```

```
Res: 2.05 Content Observe:140 (text/plain)
21.8
```

#### 5.10. Future Interfaces

It is expected that further interface descriptions will be defined in this and other specifications. Potential interfaces to be considered for this specifications include:

**Collection:** This resource would be a container that allows sub-resources to be added or removed.

#### 5.11. WADL Description

This section defines the formal Web Application Description Language (WADL) definition of these CoRE interface descriptions.

```
<?xml version="1.0" standalone="yes"?>

<application xmlns="http://research.sun.com/wadl/2006/10"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:senml="urn:ietf:params:xml:ns:senml">
```

```
<grammars>
  <include href="http://tools.ietf.org/html/draft-jennings-senml"/>
</grammars>

<doc title="CoRE Interfaces"/>

<resource_type id="s">
  <doc title="Sensor resource type"/>
  <method href="#read"/>
  <method href="#observe"/>
</resource_type>

<resource_type id="p">
  <doc title="Parameter resource type"/>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#update"/>
</resource_type>

<resource_type id="rp">
  <doc title="Read-only Parameter resource type"/>
  <method href="#read"/>
  <method href="#observe"/>
</resource_type>

<resource_type id="a">
  <doc title="Actuator resource type"/>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#update"/>
  <method href="#toggle"/>
</resource_type>

<resource_type id="ll">
  <doc title="Link List type"/></doc>
  <method href="#listLinks"/>
</resource_type>

<resource_type id="b">
  <doc title="Batch of sub-resources type">The methods read,
    observe, update and toggle are applied to each sub-
    resource of the requested resource that supports it. Mixed
    sub-resource types can be supported.</doc>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#update"/>
  <method href="#toggle"/>
  <method href="#listLinks"/>
```

```
</resource_type>

<resource_type id="lb">
  <doc title="Linked Batch resource type">. The methods read,
    observableRead, update and toggle are applied to each linked
    resource of the requested resource that supports it. Mixed
    linked resource types can be supported.</doc>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#update"/>
  <method href="#listLinks"/>
  <method href="#appendLinks"/>
  <method href="#clearLinks"/>
</resource_type>

<resource_type id="bnd">
  <doc title="Binding table resource type">A modifiable list of links.
    Each link MUST have the relation type "boundTo".</doc>
  <method href="#listLinks"/>
  <method href="#appendLinks"/>
  <method href="#clearLinks"/>
</resource_type>

<method id="read" name="GET">
  <doc>Retrieve the value of a sensor, an actuator or a parameter.
    Both HTTP and CoAP support this method.</doc>
  <request>
  </request>
  <response status="200">
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </response>
  <response status="2.05">
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </response>
</method>

<method id="observe" name="GET">
  <doc>Observe the value of a sensor, an actuator or a parameter.
    Only CoAP supports this method since it requires the CoRE
    Observe mechanism.</doc>
  <request>
    <param name="pmin" style="query" type="xsd:integer"/>
  </request>
</method>
```

```
<param name="pmax" style="query" type="xsd:integer"/>
<param name="st" style="query" type="xsd:decimal"/>
</request>
<response status="2.05">
  <representation mediaType="text/plain"/>
  <representation mediaType="application/senml+exi"/>
  <representation mediaType="application/senml+xml"/>
  <representation mediaType="application/senml+json"/>
</response>
</method>

<method id="update" name="PUT">
  <doc>Control the actuator or update a parameter with a new value
  or command. Both HTTP and CoAP support this method.</doc>
  <request>
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

<method id="toggle" name="POST">
  <doc>Toggle the values of actuator resources. Both HTTP and CoAP
  support this method.</doc>
  <request>
    <doc>The toggle function is only applicable if the request
    is empty.</doc>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

<method id="listLinks" name="GET">
  <doc>Retrieve the list of Web links associated to a resource.
  Both HTTP and CoAP support this method.</doc>
  <request>
    <doc>This request MUST contain an Accept option with
    application/link-format when the resource supports
    other GET methods.</doc>
  </request>
  <response status="200">
    <representation mediaType="application/link-format"/>
  </response>
  <response status="2.05">
    <representation mediaType="application/link-format"/>
```



```
    </response>
  </method>

  <method id="appendLinks" name="POST">
    <doc>Append new Web links to a resource which is a collection
    of links. Both HTTP and CoAP support this method.</doc>
    <request>
      <representation mediaType="application/link-format"/>
    </request>
    <response status="200"/>
    <response status="2.04"/>
  </method>

  <method id="clearLinks" name="DELETE">
    <doc>Clear all Web Links in a resource which is a collection
    of links. Both HTTP and CoAP support this method.</doc>
    <request>
    </request>
    <response status="200"/>
    <response status="2.04"/>
  </method>

</application>
```

## 6. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this document. A server implementing what the client thinks is a resource with one of these interface descriptions could return malformed representations and response codes either by accident or maliciously. A server sending maliciously malformed responses could attempt to take advantage of a poorly implemented client for example to crash the node or perform denial of service.

## 7. IANA Considerations

The interface description types defined require registration.

The new link relation type "boundto" requires registration.

## 8. Acknowledgments

Acknowledgement is given to colleagues from the SENSEI project who

were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface types have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this document.

## 9. Changelog

Changes from -02 to -03

- o Added Bindings
- o Updated all rt= and if= for the new Link Format IANA rules

Changes from -01 to -02

- o Defined a Function Set and its guidelines.
- o Added the Link List interface.
- o Added the Linked Batch interface.
- o Improved the WADL interface definition.
- o Added a simple profile example.

## 10. References

### 10.1. Normative References

- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format",  
draft-ietf-core-link-format-14 (work in progress),  
June 2012.
- [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.

### 10.2. Informative References

- [I-D.cheshire-dnsext-dns-sd]  
Cheshire, S. and M. Krochmal, "DNS-Based Service  
Discovery", draft-cheshire-dnsext-dns-sd-11 (work in

progress), December 2011.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-10 (work in progress), June 2012.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-05 (work in progress), March 2012.

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

Shelby, Z. and S. Krco, "CoRE Resource Directory",  
draft-shelby-core-resource-directory-03 (work in  
progress), May 2012.

## Appendix A. Profile example

The following is a short definition of simple profile. This  
simplistic profile is for use in the examples of this document.

Function Set	Root Path	RT	IF
Device Description	/d	simple.dev	core.ll
Sensors	/s	simple.sen	core.b
Actuators	/a	simple.act	core.b

### List of Function Sets

Type	Path	RT	IF	Data Type
Name	/d/name	simple.dev.n	core.p	xsd:string
Model	/d/model	simple.dev.mdl	core.rp	xsd:string

### Device Description Function Set

Type	Path	RT	IF	Data Type
Light	/s/light	simple.sen.lt	core.s	xsd:decimal (lux)
Humidity	/s/humidity	simple.sen.hum	core.s	xsd:decimal (%RH)

Temperature	/s/temp	simple.sen.tmp	core.s	xsd:decimal
				(degC)
+-----+	+-----+	+-----+	+-----+	+-----+

## Sensors Function Set

+-----+	+-----+	+-----+	+-----+	+-----+
Type	Path	RT	IF	Data Type
+-----+	+-----+	+-----+	+-----+	+-----+
LED	/a/{#}/led	simple.act.led	core.a	xsd:boolean
+-----+	+-----+	+-----+	+-----+	+-----+

## Actuators Function Set

## Authors' Addresses

Zach Shelby  
 Sensinode  
 Kidekuja 2  
 Vuokatti 88600  
 FINLAND

Phone: +358407796297  
 Email: zach@sensinode.com

Matthieu Vial  
 Schneider-Electric  
 Grenoble,  
 FRANCE

Phone: +33 (0)47657 6522  
 Email: matthieu.vial@schneider-electric.com



CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: August 29, 2013

Z. Shelby  
Sensinode  
S. Krco  
Ericsson  
C. Bormann  
Universitaet Bremen TZI  
February 25, 2013

CoRE Resource Directory  
draft-shelby-core-resource-directory-05

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 August 29, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal

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

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	3
3. Architecture and Use Cases . . . . .	4
3.1. Use Case: Cellular M2M . . . . .	5
3.2. Use Case: Home and Building Automation . . . . .	6
4. Simple Directory Discovery . . . . .	6
4.1. Finding a Directory Server . . . . .	7
5. Resource Directory Function Set . . . . .	8
5.1. Discovery . . . . .	8
5.2. Registration . . . . .	10
5.3. Update . . . . .	12
5.4. Validation . . . . .	13
5.5. Removal . . . . .	15
6. Group Function Set . . . . .	16
6.1. Register a Group . . . . .	16
6.2. Group Removal . . . . .	17
7. RD Lookup Function Set . . . . .	18
8. New Link-Format Attributes . . . . .	23
8.1. Resource Instance 'ins' attribute . . . . .	23
8.2. Export 'exp' attribute . . . . .	23
9. Security Considerations . . . . .	24
10. IANA Considerations . . . . .	24
11. Acknowledgments . . . . .	24
12. Changelog . . . . .	24
13. References . . . . .	26
13.1. Normative References . . . . .	26
13.2. Informative References . . . . .	26
Authors' Addresses . . . . .	26

## 1. Introduction

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

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

This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to registrar, maintain, lookup and remove resource descriptions. Furthermore, new link attributes useful in conjunction with a Resource Directory are defined. Although the examples in this document show the use of these interfaces with CoAP [I-D.ietf-core-coap], they may be applied in an equivalent manner to HTTP [RFC2616].

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. The term "byte" is used in its now customary sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [I-D.ietf-core-coap]. The URI Template format is used to describe the REST interfaces defined in this specification [RFC6570]. This specification makes use of the following additional terminology:



#### Resource Directory

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

#### Domain

In the context of a Resource Directory, a domain is a logical grouping of endpoints. All endpoint within a domain MUST be unique. This specification assumes that the list of Domains supported by an RD is pre-configured by that RD.

#### Group

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

#### Endpoint

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

### 3. Architecture and Use Cases

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

Endpoints are assumed to proactively register and maintain resource directory entries on the RD, which are soft state and need to be periodically refreshed. An endpoint is provided with interfaces to register, update and remove a resource directory entry. Furthermore, a mechanism to discover a RD using the CoRE Link Format is defined. It is also possible for an RD to proactively discover Web Links from endpoints and add them as resource directory entries, or to validate existing resource directory entries. A lookup interface for discovering any of the Web Links held in the RD is provided using the

CoRE Link Format.

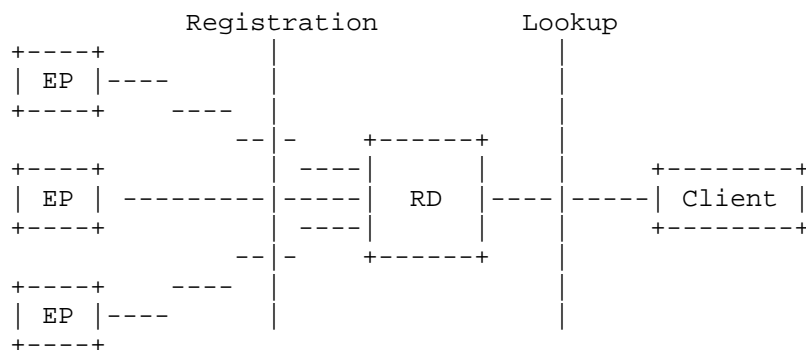


Figure 1: The resource directory architecture.

### 3.1. Use Case: Cellular M2M

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

In a typical scenario, during a boot-up procedure (and periodically afterwards), the machines (endpoints) register with a Resource Directory (for example EPs installed on vehicles enabling tracking of their position for the fleet management purposes and monitoring environment parameters) hosted by the mobile operator or somewhere else in the network, submitting a description of 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 tags, 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 a set of credentials along with the appropriate tags to the RD to look-up for EPs deployed on the vehicles the application is responsible for.

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

Home and commercial building automation systems can benefit from the use of M2M web services. The use of CoRE in home automation across multiple subnets is described in [I-D.brandt-coap-subnet-discovery] and in commercial building automation in [I-D.vanderstok-core-bc]. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices.

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

## 4. Simple Directory Discovery

Not all endpoints hosting resources are expected to know how to implement the Resource Directory Function Set and thus explicitly register with a Resource Directory (or other such directory server). Instead, simple endpoints can implement the generic Simple Directory Discovery approach described in this section. An RD implementing this specification MUST implement Simple Directory Discovery. However, there may be security reasons why this form of directory discovery would be disabled.

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

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

An endpoint that wants to make itself discoverable occasionally sends

a POST request to the /.well-known/core URI of any candidate directory server that it finds. The body of the POST request is either

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

or

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

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

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

```

EP                                     RD
|                                     |
| -- POST /.well-known/core "</sen/temp>..." ----> |
|                                     |
| <----- 2.01 Created -----> |
|                                     |

```

#### 4.1. Finding a Directory Server

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

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

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

- o DHCPv6 options that might be defined later.

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

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

## 5. Resource Directory Function Set

This section defines the REST interfaces between an RD and endpoint servers, which is called the Resource Directory Function Set. Although the examples throughout this section assume use of CoAP [I-D.ietf-core-coap], these REST interfaces can also be realized using HTTP [RFC2616]. An RD implementing this specification MUST support the discovery, registration, update, and removal interfaces defined in this section and MAY support the validation interface. For the purpose of validation, an endpoint implementing this specification SHOULD support ETag validation on /.well-known/core (which is very straightforward for static /.well-known/core link documents).

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

### 5.1. Discovery

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

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

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

The discovery request interface is specified as follows:

Interaction: EP -> RD

Method: GET

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

URI Template Variables:

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

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

The following response codes are defined for this interface:

Success: 2.05 "Content" with an `application/link-format` payload containing a matching entry for the RD resource.

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

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request"

The following example shows an endpoint discovering an RD using this interface, thus learning that the base RD resource is at `/rd`. Note that it is up to the RD to choose its base RD resource, although it is recommended to use default locations where possible.

```

      EP                                     RD
      |                                     |
      | ----- GET /.well-known/core?rt=core.rd* -----> |
      |                                     |
      | <----- 2.05 Content "</rd>; rt="core.rd" ----- |
      |                                     |

```

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

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

## 5.2. Registration

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

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

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

URI Template Variables:

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

`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. 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.

`et` := Endpoint Type (optional). The semantic type of the endpoint. The maximum length of this parameter is 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.

Content-Type: `application/link-format`

The following response codes are defined for this interface:

Success: 2.01 "Created". The Location header MUST be included with the new resource entry for the endpoint. This Location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration (update, delete).

Failure: 4.00 "Bad Request". Malformed request.

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

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



location.

```

      EP                                     RD
      |                                     |
      | --- POST /rd?ep=node1 "</sensors..." ----->
      |
      | <-- 2.01 Created Location: /rd/4521 -----
      |

```

Req: POST coap://rd.example.com/rd?ep=node1

Payload:

```
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor"
```

Res: 2.01 Created

Location: /rd/4521

### 5.3. Update

The update interface is used by an endpoint to refresh or update its registration with an RD. To use the interface, the endpoint sends a PUT request to the resource returned in the Location option in the response to the first registration. An update MAY contain registration parameters if there have been changes since the last registration or update. Parameters that have not changed SHOULD NOT be included in an update. Upon receiving an update request, the RD resets the timeout for that endpoint and stores the values of the parameters included in the update (if any).

The update request interface is specified as follows:

Interaction: EP -> RD

Method: PUT

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

URI Template Variables:

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

et := Endpoint Type (optional). The semantic type of the endpoint. The maximum length of this parameter is 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 used to register are assumed.

Content-Type: None

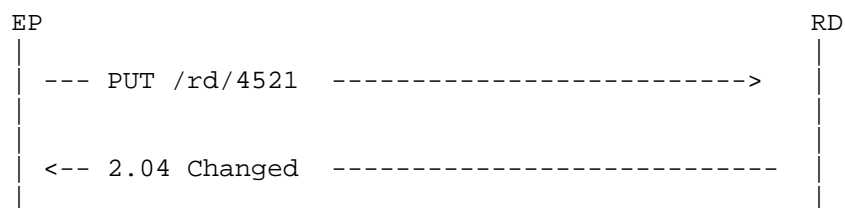
The following response codes are defined for this interface:

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

Failure: 4.00 "Bad Request". Malformed request.

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

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



Req: PUT /rd/4521

Res: 2.04 Changed

#### 5.4. Validation

In some cases, an RD may want to validate that it has the latest version of an endpoint's resources. This can be performed with a GET on the well-known interface of the CoRE Link Format including the

latest ETag stored for that endpoint. For the purpose of validation, an endpoint implementing this specification SHOULD support ETag validation on /.well-known/core.

The validation request interface is specified as follows:

Interaction: RD -> EP

Method: GET

Path: /.well-known/core

Parameters: None

ETag: The ETag option MUST be included

The following responses codes are defined for this interface:

Success: 2.03 "Valid" in case the ETag matches

Success: 2.05 "Content" in case the ETag does not match, the response MUST include the most recent resource representation (application/link-format) and its corresponding ETag.

Failure: 4.00 "Bad Request". Malformed request.

The following examples shows a successful validation.

```

EP                                     RD
|                                     |
| <--- GET /.well-known/core ETag: 0x40 -----> |
|                                     |
| --- 2.03 Valid -----> |
|                                     |

```

Req: GET /.well-known/core  
ETag: 0x40

Res: 2.03 Valid

## 5.5. Removal

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

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

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

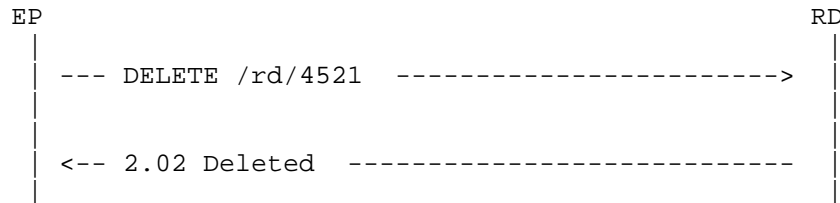
The following responses codes are defined for this interface:

Success: 2.02 "Deleted" upon successful deletion

Failure: 4.00 "Bad Request". Malformed request.

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

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



Req: DELETE /rd/4521

Res: 2.02 Deleted

## 6. Group Function Set

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

### 6.1. Register a Group

In order to create a group, a management entity used to configure groups, makes a request to the RD indicating the name of the group to create (or update), the optional domain the group belongs to, and the optional multicast address of the group. The registration message includes the list of endpoints that belong to that group. If an endpoint has already registered with the RD, the RD attempts to use the context of the endpoint from its RD endpoint entry. If the client registering the group knows the endpoint has already registered, then it MAY send a blank target URI for that endpoint link when registering the group.

The registration request interface is specified as follows:

Interaction: Manager -> RD

Method: POST

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

URI Template Variables:

`rd-group` := 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.

`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.

Content-Type: application/link-format

The following response codes are defined for this interface:

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

```
Failure: 4.00 "Bad Request". Malformed request.
```

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

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

EP	RD
	- POST /rd-group?gp=lights "<>;ep=node1..." -->
	<---- 2.01 Created Location: /rd-group/12 ----

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

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

```

EP                                     RD
|                                     |
| --- DELETE /rd-group/412 -----> |
|                                     |
| <-- 2.02 Deleted -----          |
|                                     |

```

Req: DELETE /rd-group/12

Res: 2.02 Deleted

## 7. RD Lookup Function Set

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

interfaces (e.g. supporting context or semantic based lookup).

This function set allows lookups for domains, groups, endpoints and resources using attributes defined in the RD Function Set and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) in CoRE Link Format corresponding to the type of lookup. The target of these links SHOULD be the actual location of the domain, endpoint or resource, but MAY be an intermediate proxy e.g. in the case of an HTTP lookup interface for CoAP endpoints. Multiple query parameters MAY be included in a lookup, all included parameters MUST match for a resource to be returned. The character '\*' MAY be included at the end of a parameter value as a wildcard operator.

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

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

Parameters:

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

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

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

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

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

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



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

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

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

The following responses codes are defined for this interface:

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

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

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request". Malformed request.

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

The following example shows a client performing a resource lookup:

Client	<pre> ----- GET /rd-lookup/res?rt=temperature -----&gt;  &lt;-- 2.05 Content "&lt;coap://{ip:port}/temp&gt;" ---- </pre>	RD
--------	--	----

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

Res: 2.05 Content  
<coap://{ip:port}/temp>

The following example shows a client performing an endpoint lookup:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/ep?et=power-node ----->      |
|                                                         |
|<-- 2.05 Content "<coap://{ip:port}>;ep="node5" -----|
|                                                         |

```

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

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

The following example shows a client performing a domain lookup:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/d ----->                      |
|                                                         |
|<-- 2.05 Content "</rd>;d=domain1,</rd>;d=domain2 -----|
|                                                         |

```

Req: GET /rd-lookup/d

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

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

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/gp ----->                    |
|                                                         |

```

```
<-- 2.05 Content </rd-group/12>;gp="lights1";d="domain1" --
```

```
Req: GET /rd-lookup/gp
```

```
Res: 2.05 Content
</rd-group/12>;gp="lights1";d="domain1"
```

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

```
Client | RD
| |
| |----- GET GET /rd-lookup/ep?gp=lights1----->
| |
| |<-- 2.05 Content "</rd>;d=domain1,</rd>;d=domain2 -----
| |
```

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

```
Res: 2.05 Content
<coap://host:port>;ep="node1",
<coap://host:port>;ep="node2",
```

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

```

Client                                                                    RD
|----- GET /rd-lookup/gp?ep=node1 ----->
|
|<-- 2.05 Content "</rd>;d=domain1,</rd>;d=domain2 -----
|

```

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

Res: 2.05 Content

<coap://host:port>;gp="lights1";ep="node1",

## 8. New Link-Format Attributes

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

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

### 8.1. Resource Instance 'ins' attribute

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

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

### 8.2. Export 'exp' attribute

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

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

## 9. Security Considerations

This document needs the same security considerations as described in Section 7 of [RFC5988] and Section 6 of [RFC6690]. The /.well-known/core resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [I-D.ietf-core-coap].

Access control SHOULD be performed separately for the RD Function Set and the RD Lookup Function Set, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the domain, endpoint or resource level.

## 10. IANA Considerations

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

The "exp" attribute needs to be registered when a future Web Linking attribute is created.

## 11. Acknowledgments

Szymon Sasin, Kerry Lynn, Esko Dijk, Peter van der Stok, Anders Brandt, Matthieu Vial, Sampo Ukkola and Linyi Tian have provided helpful comments, discussions and ideas to improve and shape this document. The authors would also like to thank their colleagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

## 12. Changelog

Changes from -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 & 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= paramter to be present.

Changes from -01 to -02:

- o Added a terminology section.
- o Changed the including of an ETag in registration or update to a MAY.
- o Added the concept of an RD Domain and a registration parameter for it.
- o Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- o Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

### 13. References

## 13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

## 13.2. Informative References

- [I-D.brandt-coap-subnet-discovery]  
Brandt, A., "Discovery of CoAP servers across subnets", draft-brandt-coap-subnet-discovery-00 (work in progress), March 2011.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.vanderstok-core-bc]  
Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-05 (work in progress), October 2011.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.

Authors' Addresses

Zach Shelby  
Sensinode  
Kidekuja 2  
Vuokatti 88600  
FINLAND

Phone: +358407796297  
Email: zach@sensinode.com

Srdjan Krco  
Ericsson

Phone:  
Email: srdjan.krco@ericsson.com

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

Phone: +49-421-218-63921  
Fax: +49-421-218-7000  
Email: cabo@tzi.org





CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: August 29, 2013

B. Silverajan  
Tampere University of Technology  
T. Savolainen  
Nokia  
February 25, 2013

CoAP Communication with Alternative Transports  
draft-silverajan-core-coap-alternative-transports-01

Abstract

CoAP is being standardised as an application level REST-based protocol. A single CoAP message is typically encapsulated and transmitted using UDP. This draft examines the requirements and possible solutions for conveying CoAP packets to end points over alternative transports to UDP. While UDP remains an optimal solution for use in IP-based constrained environments and nodes, M2M communication using non-IP networks, NAT and firewall traversal issues and possibly mechanisms incurring a lower overhead to CoAP/HTTP gateways provide compelling motivation for understanding how CoAP can operate in various other environments.

Status of this Memo

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

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

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

This Internet-Draft will expire on August 29, 2013.

Copyright Notice

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

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

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

## Table of Contents

1. Introduction . . . . .	3
2. Rationale and Benefits . . . . .	4
3. Use Cases . . . . .	4
4. CoAP Transport URI . . . . .	5
4.1. Transport in URI scheme name . . . . .	6
4.2. Transport in URI path or query component . . . . .	7
4.3. Expressing transport in the URI in other ways . . . . .	7
4.3.1. Transport in the URI authority component . . . . .	8
4.3.2. Transport as part of a 'service:' URL scheme . . . . .	8
4.4. Other Considerations . . . . .	8
5. Alternative Transport Analysis and Requirements . . . . .	9
6. Acknowledgements . . . . .	10
7. IANA Considerations . . . . .	10
8. Security Considerations . . . . .	11
9. Informative References . . . . .	11
Authors' Addresses . . . . .	12

## 1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is being standardised by the CoRE WG as a lightweight, HTTP-like protocol that provides 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.

CoAP's functionality and packet sizes have been specified in order to allow constrained nodes the ability to execute a simple application protocol to set and retrieve resources using a REST-based approach. To allow for very low communication overhead as well as the unreliability of constrained environments, CoAP is bound to UDP with optional reliability, to support unicast and multicast communication. Security is provided by means of the Datagram Transport Layer Security (DTLS). Interworking with web is being standardized by means of stateless HTTP mapping.

Owing to its simplicity, CoAP is an attractive option for all manner of uses. In addition to simple end-to-end communication between CoAP end-points as well as between CoAP and HTTP-based end-points, it is being used towards resource discovery and lookups, group-based communication, proxying and mirroring resources on behalf of sleeping nodes.

As the heterogeneity of interconnected networks and nodes continues increasing, alternative modes of transporting CoAP packets, in addition to UDP should be considered. This allows, for instance, retrieval of resource values and attributes of sensor nodes in non-IP networks and the ability of nodes to overcome firewall and NAT traversal issues. As the Internet of Things takes shape and begins integrating with new kinds of networks and services, it is important to understand the relevance of extending CoAP towards new transport protocols in order to have a uniform method of lightweight retrieval and modification of resources on constrained end-points and M2M communication. Not all constrained nodes might have the ability to take advantage of IP. At the same time, not all nodes with the ability to run CoAP over UDP will be confined to just one type of networking technology. As an example, the Lightweight M2M protocol being drafted by the Open Mobile Alliance uses CoAP, and as transports, specifies both UDP binding as well as Short Message Service (SMS) bindings [OMALWM2M]. The whys and hows of running CoAP over SMS, USSD and GPRS is an ongoing work, expanded upon in [I-D.becker-core-coap-sms-gprs]

This document generalizes the CoAP Unique Resource Identifier (URI), specified in [I-D.ietf-core-coap] and further expanded in

[I-D.becker-core-coap-sms-gprs]. These drafts describe CoAP using the "coap:", "coaps:" as well as "coap+tel:" URI schemes. In this draft we explore how the URI can be further extended towards specifying usable and alternative transports without imposing incompatibilities with current practices. The mechanisms introduced should remain in conformance to practices stipulated in [RFC4395].

This draft does not discuss on application QoS requirements, user policies or network adaptation, nor does it advocate replacing the current practice of UDP-based CoAP communication. The scope of this draft is limited towards a description and a requirements capture of how CoAP packets can be transmitted over alternative transports, especially how such protocols can be expressed at the CoAP layer, as well as how CoAP packets can be mapped at transport level payloads.

## 2. Rationale and Benefits

The variety of alternative transports is large. These include IETF specified transport protocols such as TCP and Websockets, Disruption Tolerant technologies such as the Bundle Protocol, non-IP transports based on Bluetooth Low Energy and Near-Field Communications (NFC). [I-D.ietf-core-coap] acknowledges that CoAP can be used in conjunction with XMPP and SIP and [I-D.becker-core-coap-sms-gprs] documents ongoing work on letting CoAP work with SMS. It is nevertheless important to understand the relevance of extending CoAP towards new transport protocols in order to have a uniform method of lightweight retrieval and modification of resources on constrained end-points by exploiting the underlying native characteristics of such networks and their transports without necessarily having to rely on an IP adaptation layer.

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 implementation reuse across new transport networks, whereby a node can continue relying on the same REST-based API changing its end point identifier and transport protocol, when for example, its network technology migrates from a non-IP transport to an IP and UDP-based transport. This might be the case in a ZigBee or BLE node having CoAP over a proprietary network layer but subsequently supporting UDP/IP adaptation.

## 3. Use Cases

CoAP [I-D.ietf-core-coap] has been designed to work on top of UDP/IP, that is, on top of transport that can lose, reorder, and duplicate

packets. UDP has been chosen as the transport protocol over IP due to its lightweight nature and connectionless characteristics allowing functions such as multicast and group communications [I-D.ietf-core-groupcomm]. As part of these design choices, CoAP uses the exponential backoff mechanism as a simple form of congestion control.

While the nature of UDP/IP transport for CoAP is well suited for constrained node communications [RFC6568], there are use cases where alternative transports would be better suited, or where UDP/IP is simply not available. In this section we discuss about a set of use cases where different transport channels could be useful.

A host with a CoAP client may reside behind a NAT or a firewall, and would like to talk to a CoAP server, possibly by using CoAP Observe-functionality [I-D.ietf-core-observe]. However, the host would wish to conserve resources, such as energy, and avoid NAT keepalives required to maintain NAT/firewall mappings. Furthermore, the application on the host may need to use HTTP for (initial) communications, but would preferably avoid use of HTTP/CoAP proxy, especially with "long polling" feature, required to be able to receive data from the CoAP server.

For the sake of simplicity, an application would like to communicate with constrained nodes using CoAP without using IP-based transport channels. For example, the application would like to use SMS [I-D.becker-core-coap-sms-gprs] or Bluetooth Low-Energy [BTCorev4.0] for communications. Furthermore, an application may be communicating via Delay-Tolerant Networks [RFC4838] using Bundle Protocol [RFC5050], and would like to transport CoAP formatted messages. In all of these cases it is not a given that UDP or IP are supported by a transport channel.

#### 4. CoAP Transport URI

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. When a local end-application supplies the URI to its own CoAP client implementation, it is parsed before the appropriate header values are encoded into the CoAP request. At the same time, the scheme specified in the URI is verified to determine whether plain UDP should be used ('coap') or

whether the CoAP client should use DTLS instead ('coaps') to initiate communication with a CoAP origin server. Secondly, the CoAP client decomposes the URI into a set of options, namely Uri-Host, Uri-Port, Uri-Path and Uri-Query that are encoded into the CoAP packet sent to the CoAP origin server to specify the target resource. An origin server receiving such a packet can reconstruct the original CoAP URI from the option values.

A COAP URI used in this way can allow an end-application to notify its CoAP implementation of the transport mechanism to be used. The CoAP URI can also be used to distinguish the transport being used between communicating CoAP entities.

How the transport protocol is identified as a distinguishable component within the URI without violating [RFC3986], whilst ensuring little or no impact to [I-D.ietf-core-coap] is the challenge. We envision several alternatives for the CoAP URI based on available existing practices, each having its strengths and limitations.

#### 4.1. Transport in URI scheme name

For a URI that is expressed generically as

```
URI = scheme ":" "://" authority path-abempty [ "?"query ]
```

The transport protocol can be expressed as part of the scheme name. According to [RFC3986] scheme names consist of a sequence of characters beginning with letter and followed by any combination of letters, digits, plus ("+"), period ((".")), or hyphen ("-"). [I-D.ietf-core-coap] uses "coaps" instead of "coap" to specify DTLS as the transport, while the preferred form used by [I-D.becker-core-coap-sms-gprs] is "coap+tel". Using alternative transports would therefore invoke new scheme names, such as "coap+sip", "coap+tcp", "coap+l2cap" and so on. The authority component of the URI would invariably then be the end point identifier specific for that transport required, be it an IP-enabled endpoint or not.

Expressing the transport this way conforms to [RFC4395]. When such a URI is provided from an end-application to its CoAP implementation, URL parsing to retrieve the transport type and endpoint identifier is trivial. It is also expected that CoAP implementations not recognising new scheme names may simply discard the request or response procedure.

As the usage of each such scheme name results in an entirely new scheme, IANA intervention is required for the registration of each scheme name. Consequently such a registration process must conform to the guidelines stipulated in [RFC4395], particularly where

permanent URI scheme registration is concerned. Care must therefore be taken to ensure the scheme is well-defined and unambiguous in the transport description.

#### 4.2. Transport in URI path or query component

For a URI that is expressed generically as

```
URI = scheme ":" "://" authority path-abempty [ "?"query ]
```

The transport protocol can alternatively be provided as a path or query component. The Diameter Base Protocol [RFC3588] is one example of a protocol that uses the "aaa" and "aaas" URI scheme names to reflect whether transport security is used, and at the same time provides the actual transport protocol to be used as a ";transport=" path component. Example valid Diameter URIs are  
aaa://host.example.com;transport=sctp and  
aaas://host.example.com:6666;transport=tcp

Adopting such a procedure for CoAP can be done in two ways. The first is to provide the transport as a path component, similar to the Diameter protocol. An example resulting URI could be  
coap://host.example.com;transport=tcp/.well-known/core?rt=core-rd  
specifying a CoAP endpoint discovering a Resource Directory and its base RD resource while using TCP as a transport instead of UDP. A URI-Path option would then be used to encode the transport used.

An alternative means of expressing the transport protocol used is to encode the transport as a query component instead. In this case, the resulting URI would then be  
coap://host.example.com/.well-known/core?rt=core-rd?tt=tcp where "tt" refers to the transport type. Such a scheme would mean that the CoAP implementation encodes two URI-query components.

Encoding the transport as part of the URI path or query provides an advantage in that IANA registration is not required, as opposed to introducing new URI scheme names. New transports can be easily introduced into the CoAP URI. As both the URI-Path and the URI-Query options fall into the "critical" class of options, caution must be exercised if an endpoint does not recognise them. In such cases, section 5.4.1 in [I-D.ietf-core-coap] provides handling guidelines.

#### 4.3. Expressing transport in the URI in other ways

Other means of indicating the transport are also possible, and while these schemes might be incompatible with existing practices, they are presented for the sake of completeness.



#### 4.3.1. Transport in the URI authority component

An application-specific, provisional resource identifier registered with IANA, has been done so by specifying the transport to be used as part of the authority [IANA-paparazzi-uri]:

```
paparazzi:[options] http:[//host>[:[port]][transport]]/
```

While the URI is used by the application to obtain a screenshot of a non-secure webpage, usage of the transport parameter is unclear and if it is at all used.

#### 4.3.2. 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.

#### 4.4. Other Considerations

This section outlines miscellaneous considerations concerning transport bindings with the CoAP URI.

1. When CoAP communication over an alternative transport is desired, a clear, unambiguous name should be used. As an example, both Bluetooth Low Energy and Classic Bluetooth carry traffic over L2CAP. A "coap+l2cap" scheme name would therefore raise ambiguity.

2. It is also conceivable that an end point may wish to register its available transports and associated end point identifiers in a CoAP resource directory, and periodically update them. A "core-transport" resource type would then need to be registered.

## 5. Alternative Transport Analysis and Requirements

In this section we take a general look at alternative protocols for CoAP and the requirements CoAP imposes on underlying layer in order to successfully support various kinds of functionality. CoAP factors lossiness, unreliability, small packet sizes and connection statelessness into its protocol logic. We discuss general transport differences and requirements to carry CoAP packets here. Note that Reqs 1, 2, and 3 are related.

REQ1: Ability to provide unique 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.

REQ2: 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.

REQ3: 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.

REQ4: Binary encoding support.

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 to exist in allowing binary payloads to be transferred over text-based transport protocols

REQ5: 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.

REQ6: MTU correlation with CoAP PDU size.

Section 4.6 of [I-D.ietf-core-coap] 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-6lowpan-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.

REQ7: 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 sleeping CoAP 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 packets is small enough not to interfere with these values for the proper operation of these functionalities.

## 6. Acknowledgements

## 7. IANA Considerations

This memo includes no request to IANA.

## 8. Security Considerations

While we envisage no new security risks 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.

## 9. Informative References

[BTCorev4.0]

BLUETOOTH Special Interest Group, "BLUETOOTH Specification Version 4.0", June 2010.

[I-D.becker-core-coap-sms-gprs]

Becker, M., Li, K., Kuladinithi, K., and T. Poetsch, "Transport of CoAP over SMS, USSD and GPRS", draft-becker-core-coap-sms-gprs-03 (work in progress), February 2013.

[I-D.ietf-6lowpan-btle]

Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "Transmission of IPv6 Packets over BLUETOOTH Low Energy", draft-ietf-6lowpan-btle-12 (work in progress), February 2013.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.

[I-D.ietf-core-groupcomm]

Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-05 (work in progress), February 2013.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-07 (work in progress), October 2012.

[IANA-paparazzi-uri]

<https://www.iana.org/assignments/uri-schemes/prov/paparazzi>, "Resource Identifier (RI) Scheme name: paparazzi", September 2012.

[OMALWM2M]

Open Mobile Alliance (OMA), "Lightweight Machine to Machine Technical Specification", 2013.

- [RFC2609] Guttman, E., Perkins, C., and J. Kempf, "Service Templates and Service: Schemes", RFC 2609, June 1999.
- [RFC3588] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.
- [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.
- [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.

#### 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

