

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).

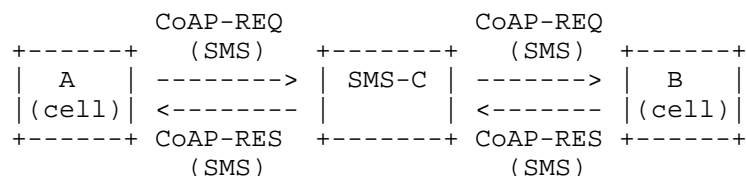


Figure 1: Cellular and Cellular Communication (only SMS-based)

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).

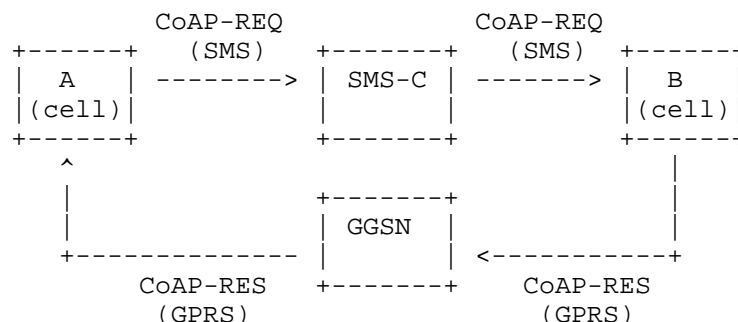


Figure 2: Cellular and Cellular Communication (SMS/GPRS-based)

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).

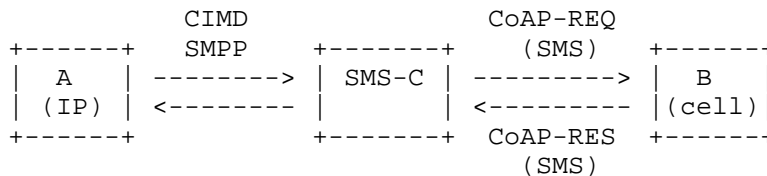


Figure 3: IP and Cellular Communication (only SMS-based)

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

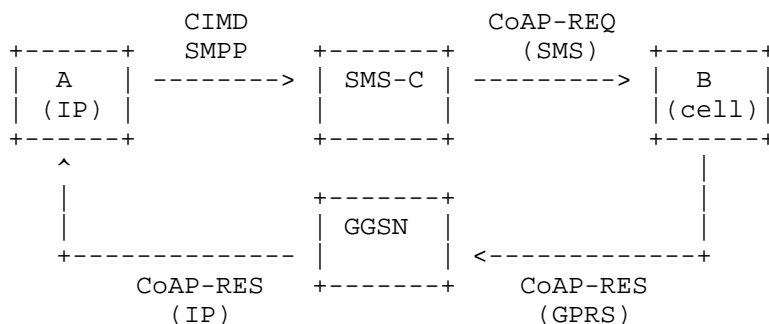


Figure 4: IP and Cellular Communication (SMS and GPRS-based)

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

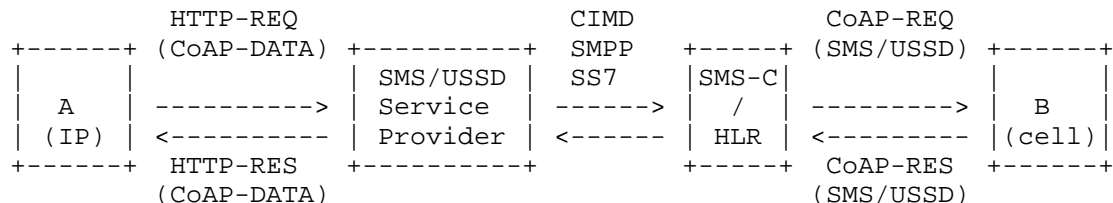


Figure 5: IP and Cellular Communication (only SMS/USSD-based, using an SMS/USSD service provider)

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).

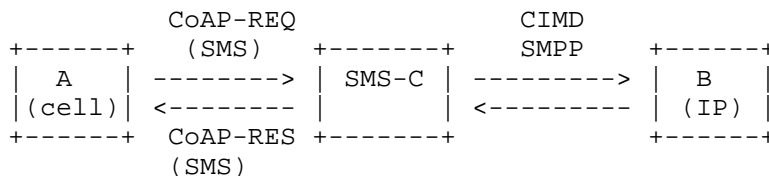


Figure 6: Cellular and IP Communication (only SMS-based)

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

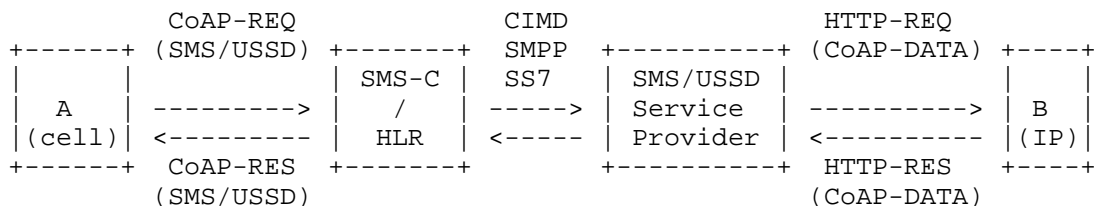


Figure 7: IP and Cellular Communication (only SMS/USSD-based, using an SMS/USSD service provider)

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).

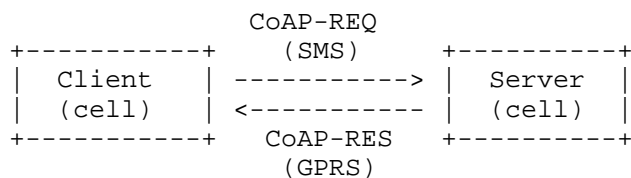


Figure 9

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

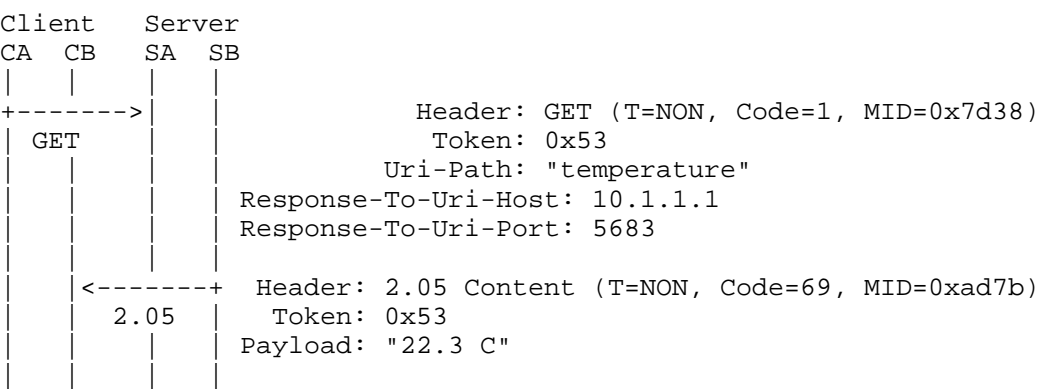


Figure 11: NON A, NON B

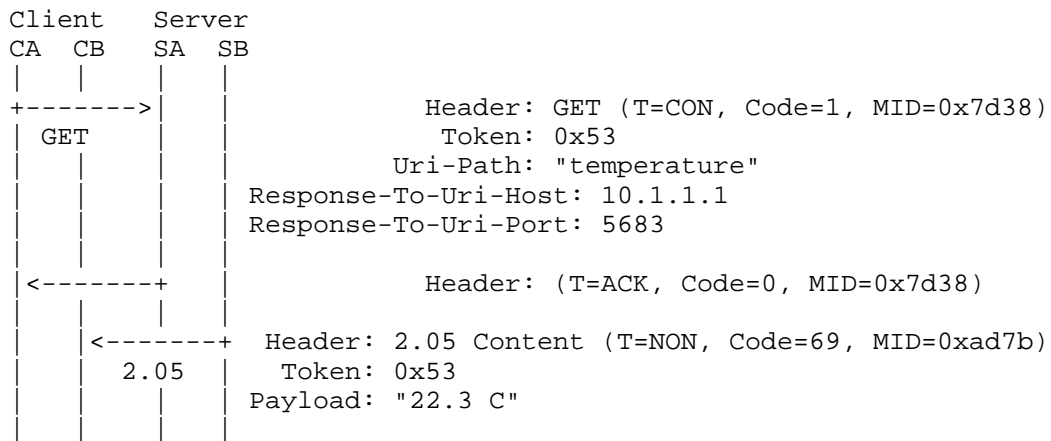


Figure 12: CON A, ACK A, NON B (separate)

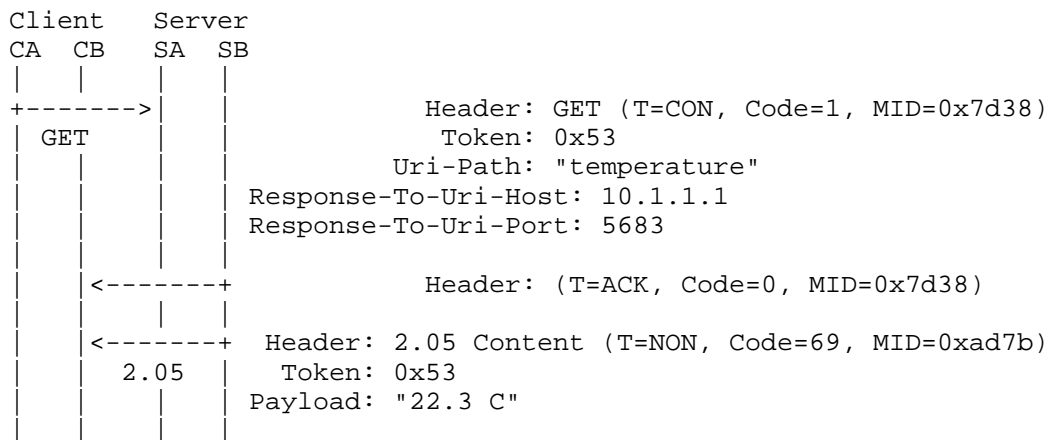


Figure 13: CON A, ACK B, NON B (separate)

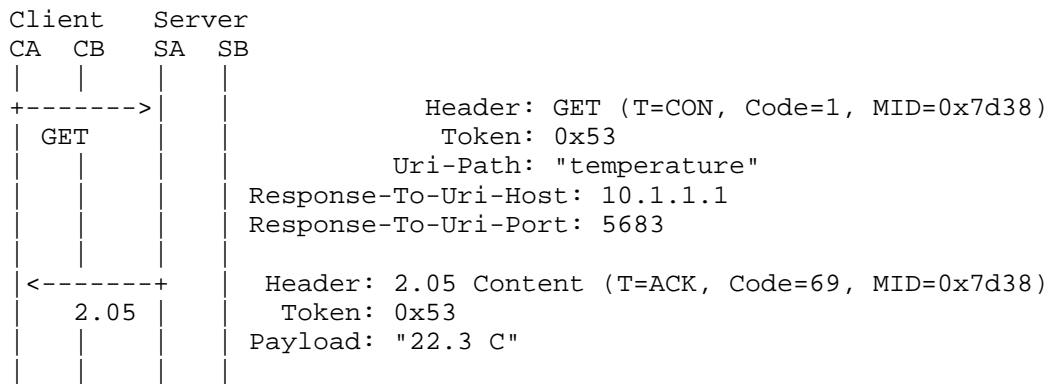


Figure 14: CON A, ACK A

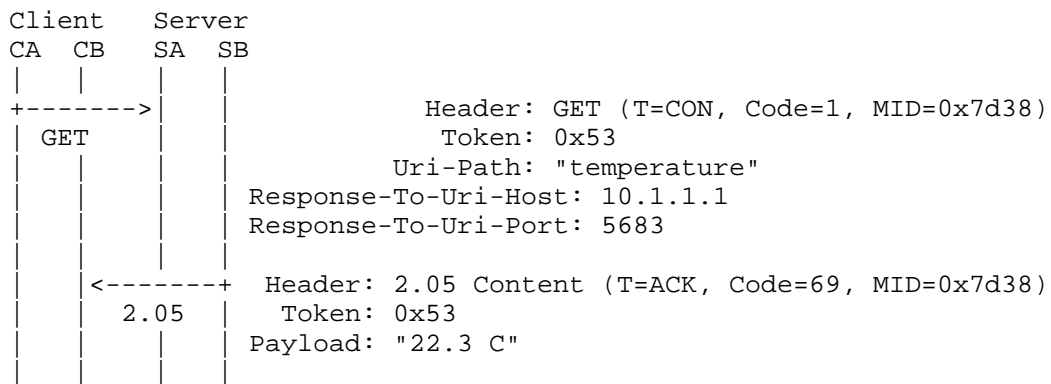


Figure 15: CON A, ACK B

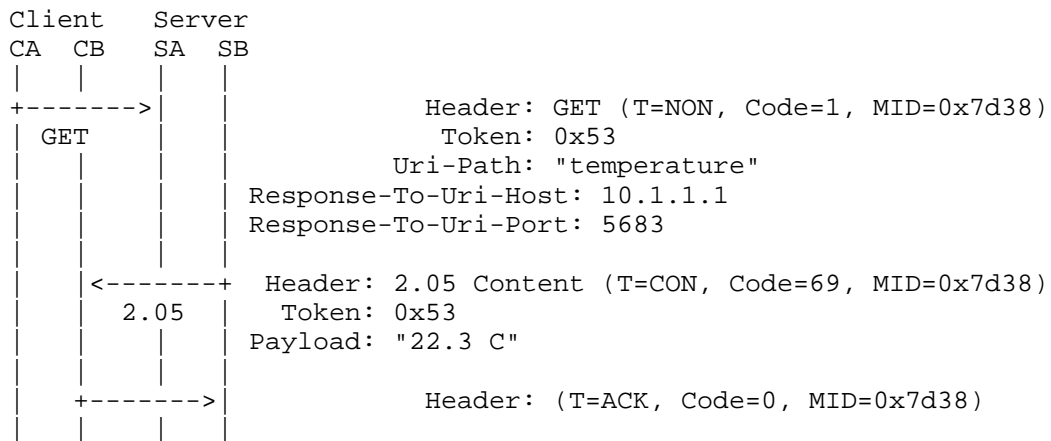


Figure 16: NON A, CON B, ACK B

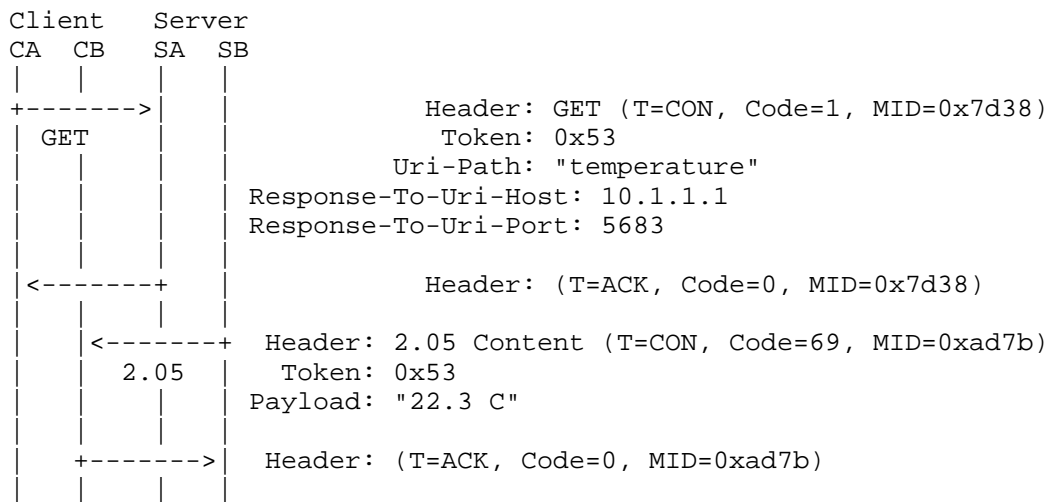


Figure 17: CON A, ACK A, CON B, ACK B (separate)

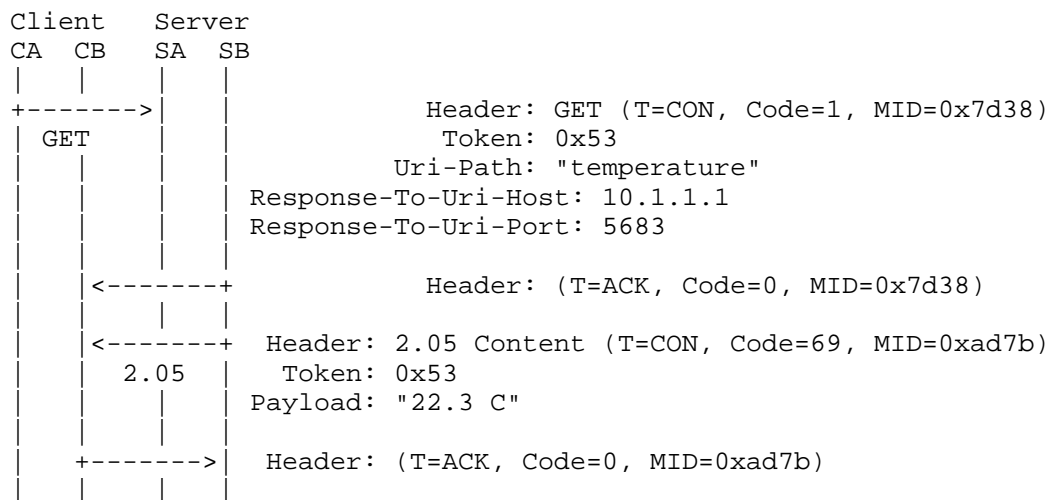


Figure 18: CON A, ACK B, CON B, ACK B (separate)

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:

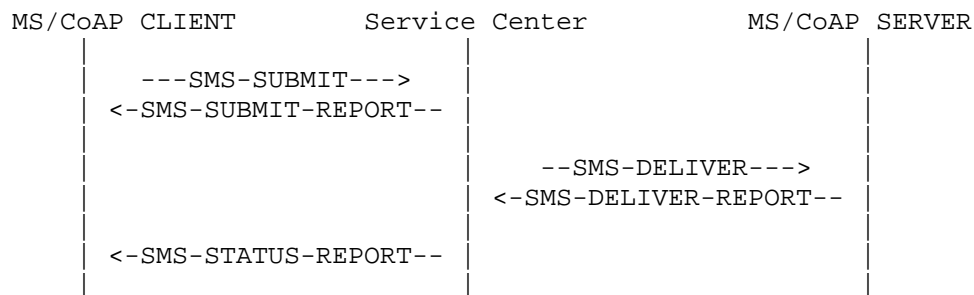


Figure 19: CoAP Messages over SMS

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.

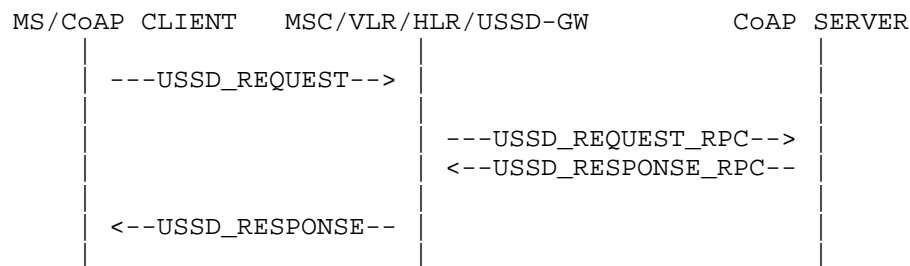


Figure 20: CoAP Messages over USSD (Mobile Originated)

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

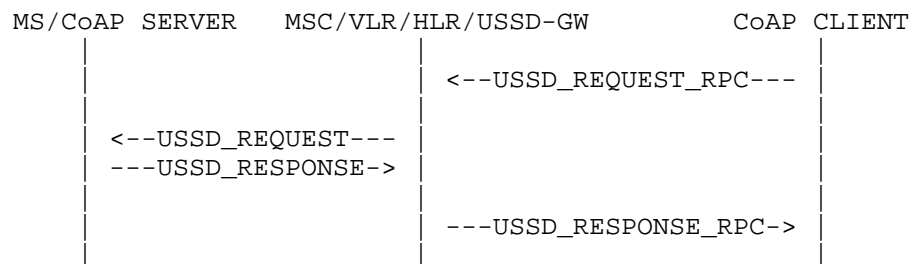


Figure 21: CoAP Messages over USSD (Mobile Terminated)

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.

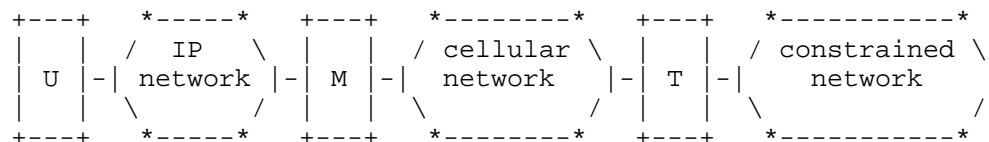


Figure 22: M2M architecture

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

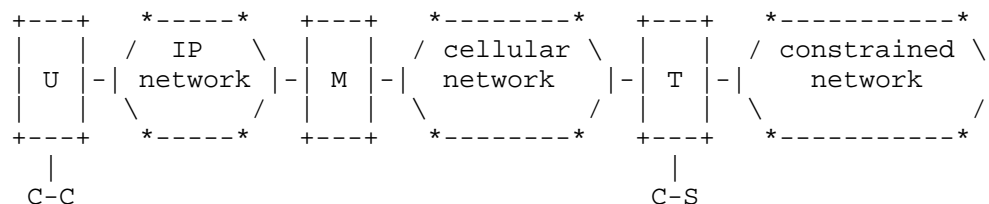


Figure 23: M2M architecture (Mobile Telematic Server) A

C-C: CoAP Client

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

C-S: CoAP Server

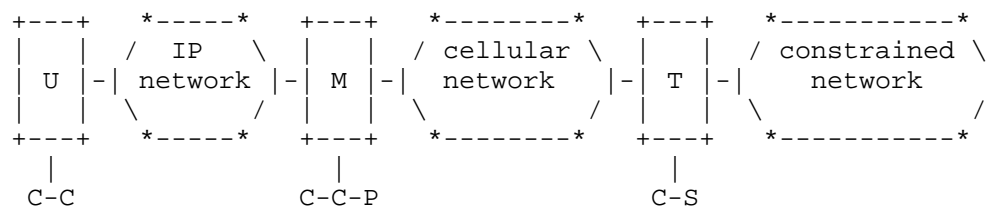


Figure 24: M2M architecture (Mobile Telematic Server) B

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

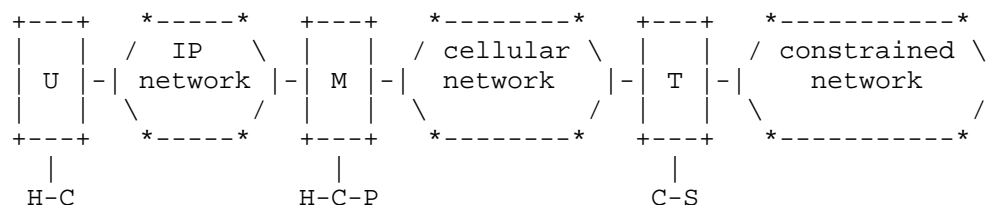


Figure 25: M2M architecture (Mobile Telematic Server) C

14.2. Mobile Telematic Client

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

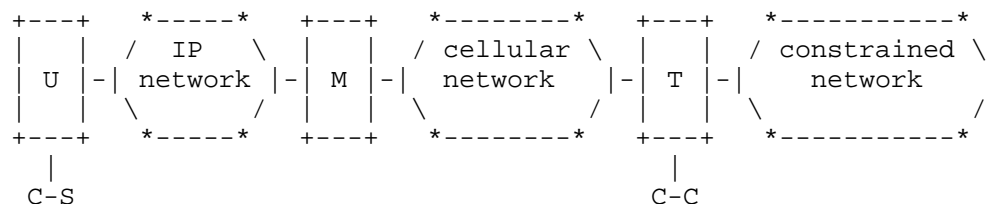


Figure 26: M2M architecture (Mobile Telematic Client) A

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

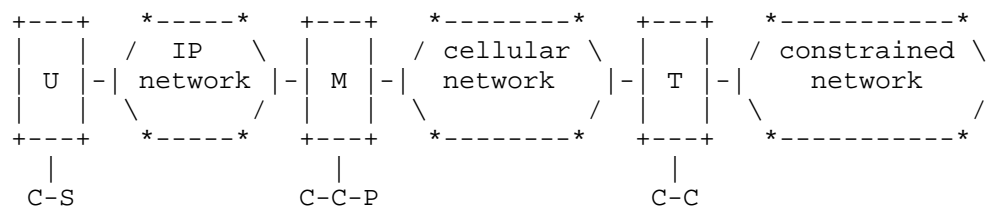


Figure 27: M2M architecture (Mobile Telematic Client) B

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

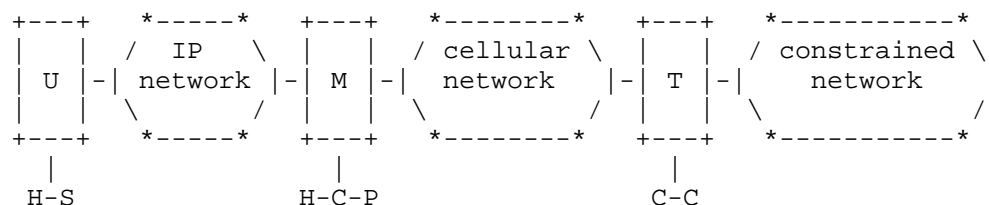


Figure 28: M2M architecture (Mobile Telematic Client) C

14.3. Mobile Server

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

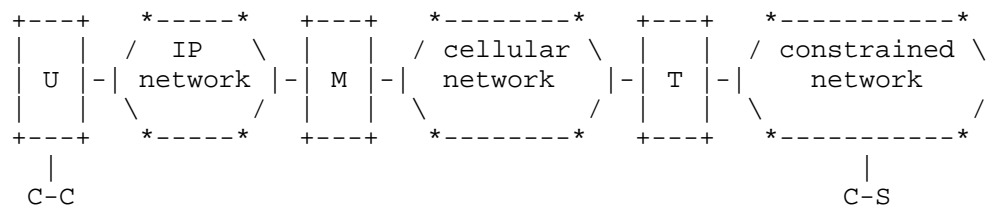


Figure 29: M2M architecture (Mobile Server) A

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

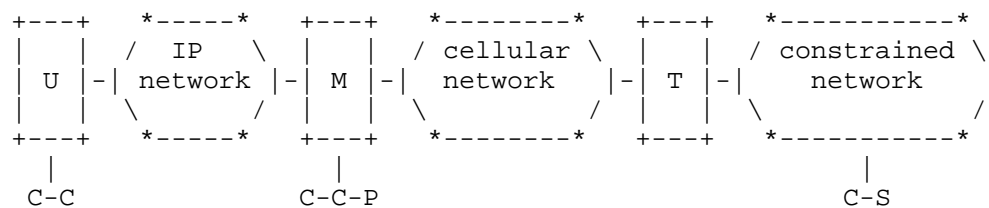


Figure 30: M2M architecture (Mobile Server) B

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

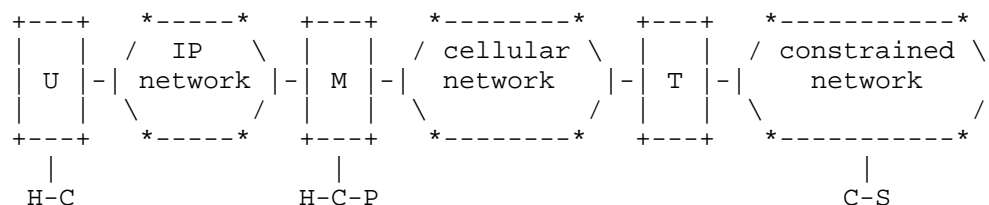


Figure 31: M2M architecture (Mobile Server) C

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)

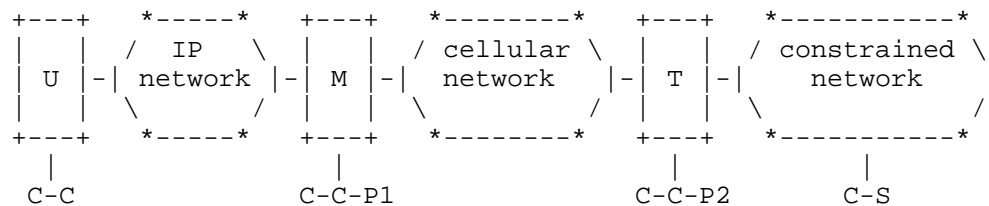


Figure 32: M2M architecture (Mobile Server) D

14.4. Mobile Client

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

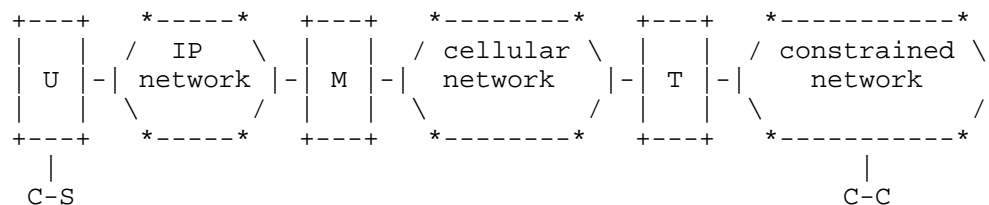


Figure 33: M2M architecture (Mobile Client) A

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

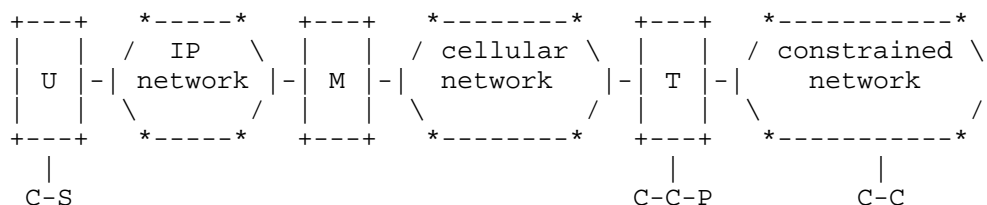


Figure 34: M2M architecture (Mobile Client) B

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)

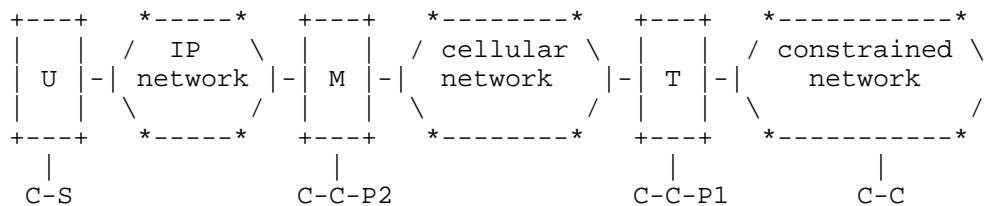


Figure 35: M2M architecture (Mobile Client) C

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

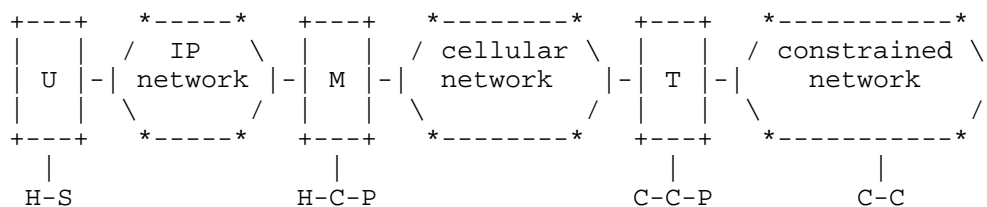


Figure 36: M2M architecture (Mobile Client) D

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
Internet-Draft
Intended status: Standards Track
Expires: January 16, 2014

M. Becker, Ed.
T. Poetsch
K. Kuladinithi
ComNets, TZI, University Bremen
July 15, 2013

Scenarios for CoAP on non-UDP Transports
draft-becker-core-transport-scenarios-00

Abstract

Several drafts in the CoRE WG are discussing the usage of CoAP on non-UDP/IP transports already. There are various use cases for non-UDP/IP transports. In order to structure the discussion, this draft introduces new terminology and transport scenarios.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 16, 2014.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
2. Terminology	3
3. Requirements Language	4
4. Scenarios	4
4.1. UI to UI	6
4.1.1. UI client to UI server	6
4.1.2. UI server to UI client	7
4.1.3. UI client/server to UI server/client	7
4.2. UI(+NUI) to UI	7
4.3. UI to UI(+NUI)	7
4.4. UI(+NUI) to UI(+NUI)	7
4.4.1. UI(+NUI) client to UI(+NUI) server	7
4.4.2. UI(+NUI) server to UI(+NUI) client	7
4.4.3. UI(+NUI) client/server to UI(+NUI) server/client	8
4.5. NUI to NUI	8
4.5.1. NUI client to NUI server	8
4.5.2. NUI server to NUI client	8
4.5.3. NUI client/server to NUI server/client	8
4.6. NUI(+UI) to NUI	8
4.7. NUI to NUI(+UI)	9
4.8. NUI(+UI) to NUI(+UI)	9
4.8.1. NUI(+UI) client to NUI(+UI) server	9
4.8.2. NUI(+UI) server to NUI(+UI) client	9
4.8.3. NUI(+UI) client/server to NUI(+UI) server/client	9
5. Proxy Scenarios	9
6. Possible non-DNS solution	10
7. Security Considerations	12
8. Acknowledgements	12
9. Normative References	12
Authors' Addresses	12

1. Introduction

Several use cases show a demand for using CoAP on non-UDP/IP transports. In order to facilitate the discussion of the use cases and the non-UDP/IP addresses, some scenarios have been assembled in this document. With non-UDP/IP transports, the following problems arise:

- o Which transport stack should be used?
- o Which protocol-specific options and parameters should be used?
- o Which addresses should be used and where are those addresses coming from? How are the addresses updated?

This information might be available in DNS, a detailed solution on how to use DNS is needed. If DNS is not available in constrained networks, a new solution is necessary.

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].

This draft distinguishes between two different stacks at the endpoints:

UDP/IP stack (UI): is an endpoint with an UDP/IP protocol stack as described in [I-D.ietf-core-coap].

non-UDP/IP stack (NUI): is an endpoint with a non-UDP/IP protocol stack (e.g. SMS, USB, BLE, TCP).

Possibly, but not necessarily, endpoints can have additionally a different transport stack, i.e. UI or NUI.

The syntax for describing the combinations of transport stacks on an endpoint are specified as follows:

Primary stack(+Optional stack): The primary stack is always available at the endpoint, while the optional stack might be enabled or disabled. Example: UI(+NUI)

The role of the endpoints are defined as follows:

Altering Endpoint: is the endpoint where a modified resource is changed first.

Altered Endpoint: is the endpoint where a modified resource is changed second.

When a client is POSTing to a server, the client is the Altering endpoint and the server is the Altered endpoint. (As well for PUT and DELETE.) When a client is GETing from a server, the client is the Altered endpoint and the server the Altering endpoint.

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 for communications with CoAP are presented first. Altering and Altered endpoints need to be defined for each transport. E.g. for Machine-to-Machine communication:

Altering endpoint		Altered endpoint
-----		-----
Device in cloud		Telematic device
Telematic device		Device in cloud
Telematic device/Device in cloud		Device in cloud/Telematic device

Figure 1: Application Role Assignment for Altering and Altered Endpoints

Altering endpoint		Altered endpoint
-----		-----
CoAP client (POST/PUT/DELETE)		-> CoAP server
CoAP server	<-	CoAP client (GET)
CoAP server/CoAP client	<- ->	CoAP client/CoAP server

Figure 2: Network Role Assignment for Altering and Altered Endpoints

Altering endpoint	Altered endpoint
UI	UI
UI(+NUI)	UI
UI	UI(+NUI)
UI(+NUI)	UI(+NUI)
NUI	NUI
NUI(+UI)	NUI
NUI	NUI(+UI)
NUI(+UI)	NUI(+UI)

Figure 3: Possible Protocol Stacks of Altering and Altered Endpoints

Altering endpoint	Altered endpoint
UI client	UI server
UI server	UI client
UI server/client	UI server/client
UI(+NUI) client	UI server
UI(+NUI) server	UI client
UI(+NUI) server/client	UI server/client
UI client	UI(+NUI) server
UI server	UI(+NUI) client
UI server/client	UI(+NUI) server/client
UI(+NUI) client	UI(+NUI) server
UI(+NUI) server	UI(+NUI) client
UI(+NUI) server/client	UI(+NUI) server/client
NUI client	NUI server
NUI server	NUI client
NUI server/client	NUI server/client
NUI(+UI) client	NUI server
NUI(+UI) server	NUI client
NUI(+UI) server/client	NUI server/client
NUI client	NUI(+UI) server
NUI server	NUI(+UI) client
NUI server/client	NUI(+UI) server/client
NUI(+UI) client	NUI(+UI) server
NUI(+UI) server	NUI(+UI) client
NUI(+UI) server/client	NUI(+UI) server/client

Figure 4: Possible Combinations of Altering and Altered Endpoints

4.1. UI to UI

This scenario works just as described in the base CoAP specification [I-D.ietf-core-coap].

4.1.1. UI client to UI server

Regular CoAP data is pushed from client to server. The client needs to know the server's IP address and UDP port. On the client side, the server's UDP/IP address information can be entered on the commandline, in a GUI textfield, preconfigured or provided in another

way.

4.1.2. UI server to UI client

An UDP/IP CoAP server hosts modified data which is pulled by the client. The client needs to know the server's UDP/IP address information. On the client side, the server's UDP/IP address information needs to be preconfigured.

4.1.3. UI client/server to UI server/client

Mixture of the above two scenarios, where both endpoints can be client and server. The first endpoint can push data to the second endpoint, which can use that data to request from the first endpoint. The first endpoint needs to know the second endpoint's UDP/IP address information.

4.2. UI(+NUI) to UI

This is a scenario where two endpoints use an UDP/IP transport, and one supports non-UDP/IP. This scenario works just as described in Section 4.1.

4.3. UI to UI(+NUI)

This is a scenario where two endpoints use an UDP/IP transport, and one supports non-UDP/IP. This scenario works just as described in Section 4.1.

4.4. UI(+NUI) to UI(+NUI)

In this scenario, both UDP/IP endpoints offer an additional non-UDP/IP stack to communicate. However, none, one or both non-UDP/IP stacks might be enabled or disabled.

4.4.1. UI(+NUI) client to UI(+NUI) server

Regular CoAP data is pushed from client to server. The client needs to know the server's IP address and UDP port. On the client side, the server's UDP/IP address information can be entered on the commandline, in a GUI textfield, preconfigured or provided in another way. A way to decide which stack to use needs to be available, if multiple stacks are enabled on the endpoints.

4.4.2. UI(+NUI) server to UI(+NUI) client

An UDP/IP CoAP server hosts modified data which is pulled by the client. The client needs to know the server's UDP/IP address

information. On the client side, the server's UDP/IP address information needs to be preconfigured. A way to decide which stack to use needs to be available, if multiple stacks are enabled on the endpoints.

4.4.3. UI(+NUI) client/server to UI(+NUI) server/client

Mixture of the above two scenarios, where both endpoints can be client and/or server. The first endpoint can push data to the second endpoint, which can use that data to request from the first endpoint. The first endpoint needs to know the second endpoint's UDP/IP address information. A way to decide which stack to use needs to be available, if multiple stacks are enabled on the endpoint.

4.5. NUI to NUI

This is a scenario where two endpoints use a non-UDP/IP transport.

4.5.1. NUI client to NUI server

CoAP data is pushed from client to server. The client needs to know the server's non-UDP/IP address information. On the client side, non-UDP/IP address information can be entered on the commandline, in a GUI textfield, preconfigured or in a similar way. A representation of the non-UDP/IP address information needs to be defined.

4.5.2. NUI server to NUI client

A non-UDP/IP CoAP server hosts modified data which is pulled by the client. The client needs to know the server's non-UDP/IP address information. On the client side, the server's non-UDP/IP address information needs to be preconfigured. A representation of the non-UDP/IP address information needs to be defined.

4.5.3. NUI client/server to NUI server/client

Mixture of the above two scenarios, where both endpoints can be client and/or server. The first endpoint can push data to the second endpoint, which can use that data to request from the first endpoint. The first endpoint needs to know the second endpoint's non-UDP/IP address information. A representation of the non-UDP/IP address information needs to be defined.

4.6. NUI(+UI) to NUI

This is a scenario where two endpoints use a non-UDP/IP transport, and one endpoint supports UDP/IP. This scenario works just as described in Section 4.5.

4.7. NUI to NUI(+UI)

This is a scenario where two endpoints use a non-UDP/IP transport, and one endpoint supports UDP/IP. This scenario works just as described in Section 4.5.

4.8. NUI(+UI) to NUI(+UI)

In this scenario, both non-UDP/IP endpoints offer an additional UDP/IP stack to communicate. However, none, one or both UDP/IP stacks might be enabled or disabled.

4.8.1. NUI(+UI) client to NUI(+UI) server

CoAP data is pushed from client to server. The client needs to know the server's non-UDP/IP address information. On the client side, non-UDP/IP address information can be entered on the commandline, in a GUI textfield, preconfigured or in a similar way. A representation of the non-UDP/IP address information needs to be defined. A way to decide which stack to use needs to be available, if multiple stacks are enabled on the endpoint.

4.8.2. NUI(+UI) server to NUI(+UI) client

The non-UDP/IP CoAP server hosts modified data which is pulled by the client. The client needs to know the server's non-UDP/IP address information. On the client side, the server's non-UDP/IP address information needs to be preconfigured. A representation of the non-UDP/IP address information needs to be defined. A way to decide which stack to use needs to be available, if multiple stacks are enabled on the endpoint.

4.8.3. NUI(+UI) client/server to NUI(+UI) server/client

Mixture of the above two scenarios, where both endpoints can be client and/or server. The first endpoint can push data to the second endpoint, which can use that data to request from the first endpoint. The first endpoint needs to know the second endpoint's non-UDP/IP address information. A representation of the non-UDP/IP address information needs to be defined. A way to decide which stack to use needs to be available, if multiple stacks are enabled on the endpoint.

5. Proxy Scenarios

TODO

6. Possible non-DNS solution

POST to e.g. `.well-known/tconf` with JSON or link-format? link-format:

```
<coap://[2001:DB8::1]:8080>;preference=1  
<coap+sms://+49....>;preference=2
```

Figure 5: Link-Format

JSON:


```

{
  "protocoloptions": [
    {
      "preference": "1",
      "protocolstack": [
        {
          "protocol": "coap",
          "config": {
            "ACK_TIMEOUT": "2",
            "MAX_RETRANSMIT": "4"
          }
        },
        {
          "protocol": "udp",
          "config": {
            "port": "8080"
          }
        },
        {
          "protocol": "ip",
          "config": {
            "dest": "[2001:DB8::1]"
          }
        }
      ]
    },
    {
      "preference": "2",
      "protocolstack": [
        {
          "protocol": "coap",
          "config": {
            "ACK_TIMEOUT": "10",
            "MAX_RETRANSMIT": "1"
          }
        },
        {
          "protocol": "sms",
          "config": {
            "address": "+49 172..."
          }
        }
      ]
    }
  ]
}

```

Figure 6: JSON

7. Security Considerations

TODO

8. 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.

9. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

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

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

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

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

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

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

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

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on January 2, 2014.

Copyright Notice

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

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

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

Table of Contents

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

1. Terminology and Conventions

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

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

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

At least two different kinds of HC proxies exist:

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

2. Introduction

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

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

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

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

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

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

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

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

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

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

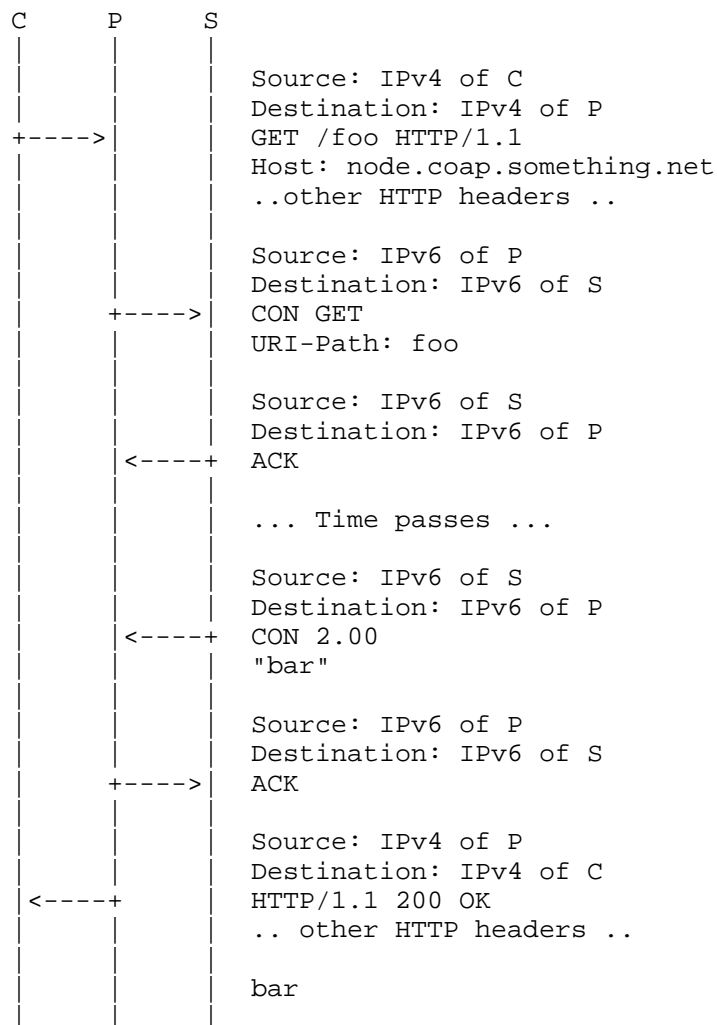


Figure 1: HTTP/IPv4 to CoAP/IPv6 Mapping

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

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

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

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

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

4. Multiple Message Exchanges Mapping

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

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

4.1. Relevant Features of Existing Standards

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

4.1.1. Multipart Messages

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

4.1.2. Immediate Message Delivery

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

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

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

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

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

4.1.3. Detailing Source Information

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

4.2. Multicast Mapping

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

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

4.2.1. URI Identification and Mapping

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

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

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

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

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

4.2.2. Request Handling

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

Multicast request handling consists of the following steps:

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

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

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

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

4.2.3. Examples

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

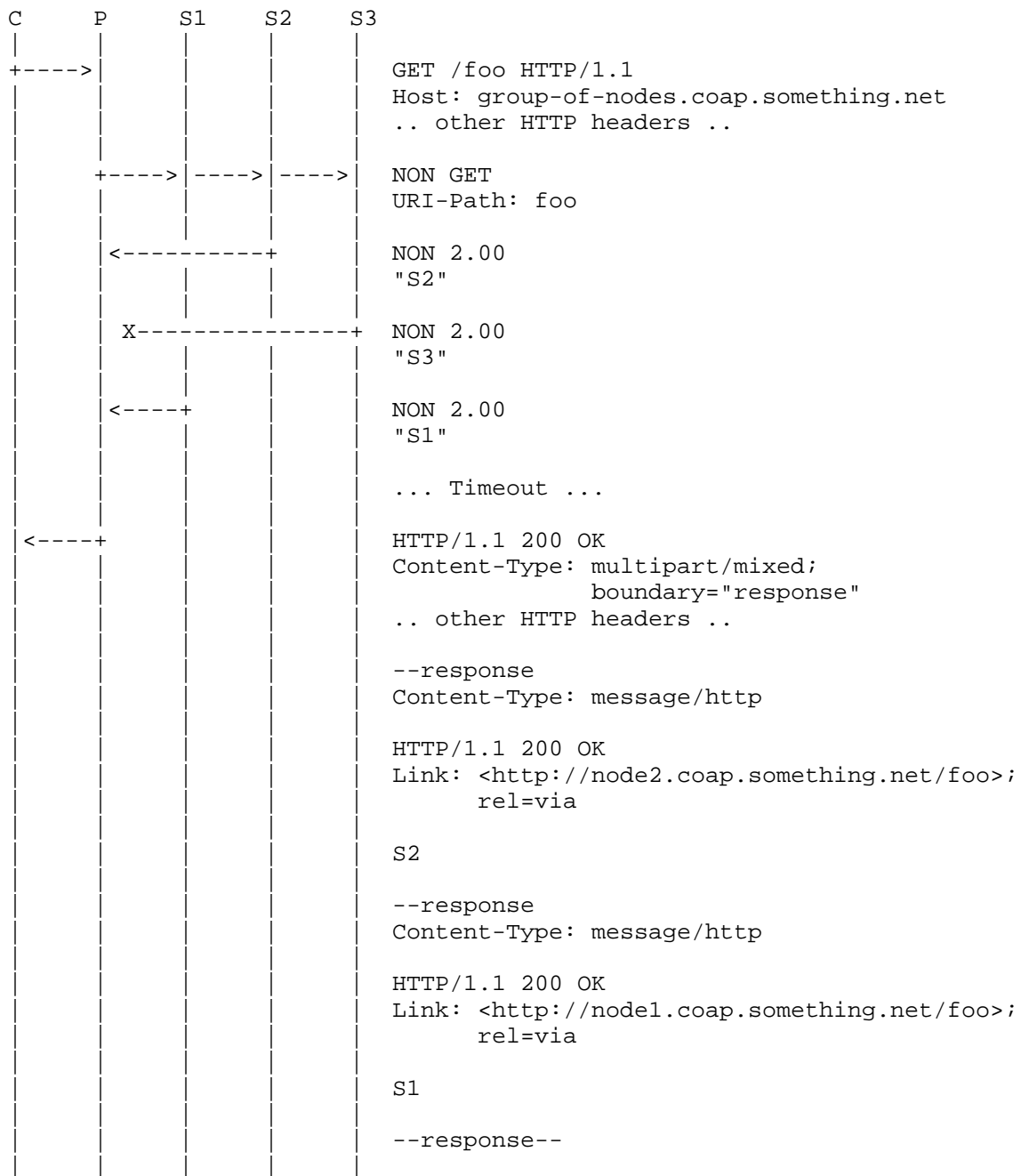


Figure 2: Unicast HTTP to Multicast CoAP Mapping

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

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

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

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

4.3. Multicast Response Caching

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

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

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

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

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

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

4.4. Observe Mapping

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

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

4.4.1. Identification

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

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

4.4.1.1. Observable URI Mapping

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

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

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

4.4.1.1.1. Discovery

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

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

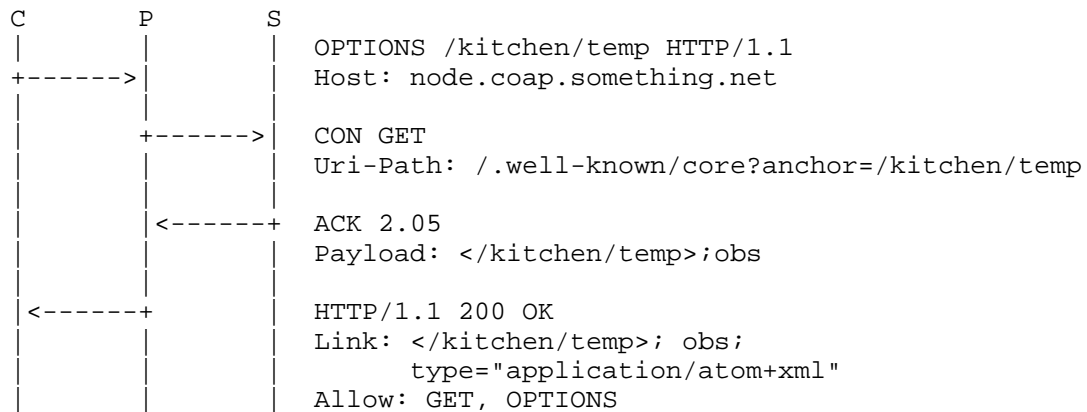


Figure 3: Discover Observability with HTTP OPTIONS

4.4.1.2. Differentiation Using HTTP Header

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

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

4.4.1.2.1. Expect Header

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

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

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

4.4.1.2.2. Prefer Header

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

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

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

4.4.2. Notification(s) Mapping

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

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

4.4.2.1. Multipart Messaging

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

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

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

4.4.2.2. Using ATOM Feeds

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

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

4.4.3. Examples

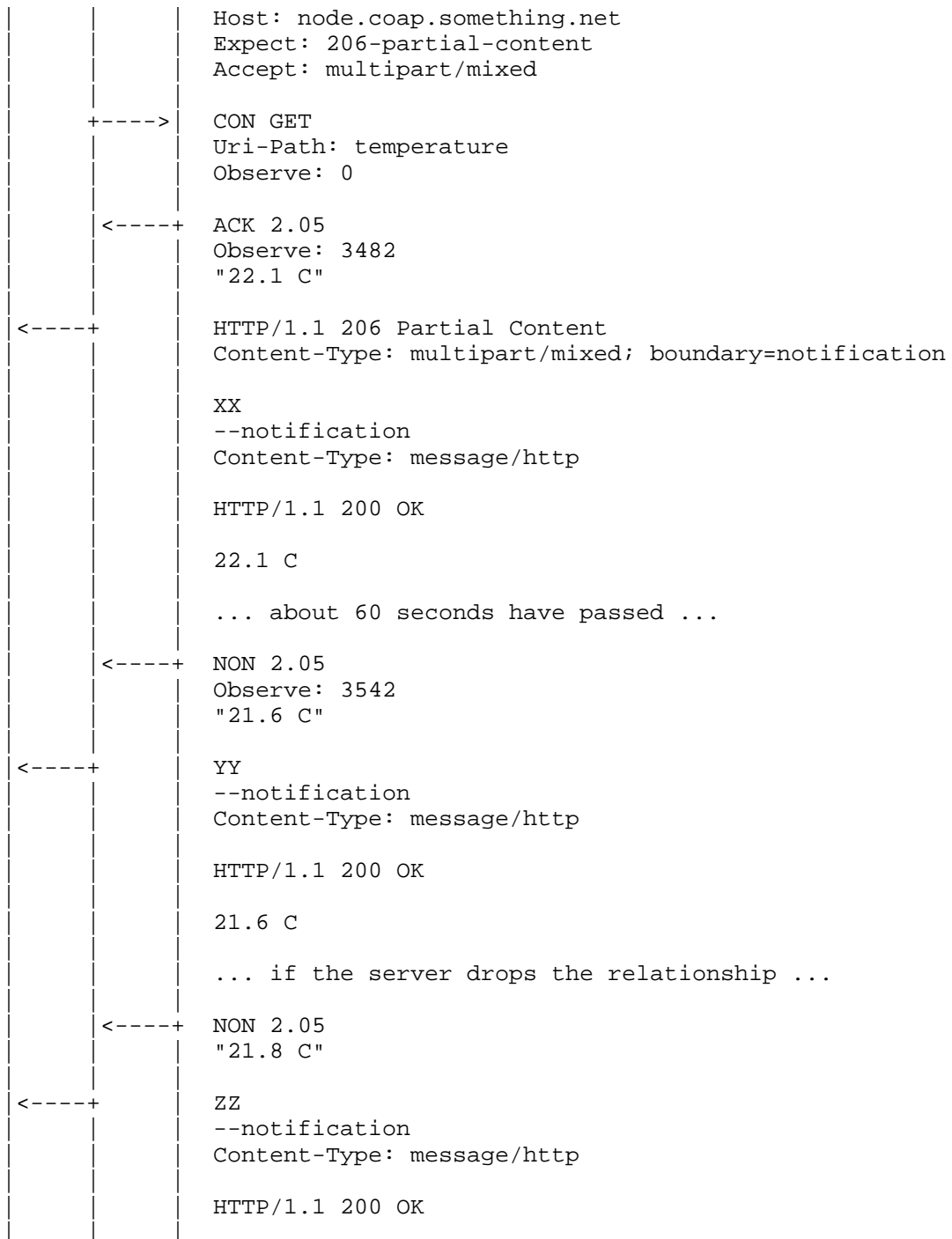
Figure 4 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

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

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

```

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.

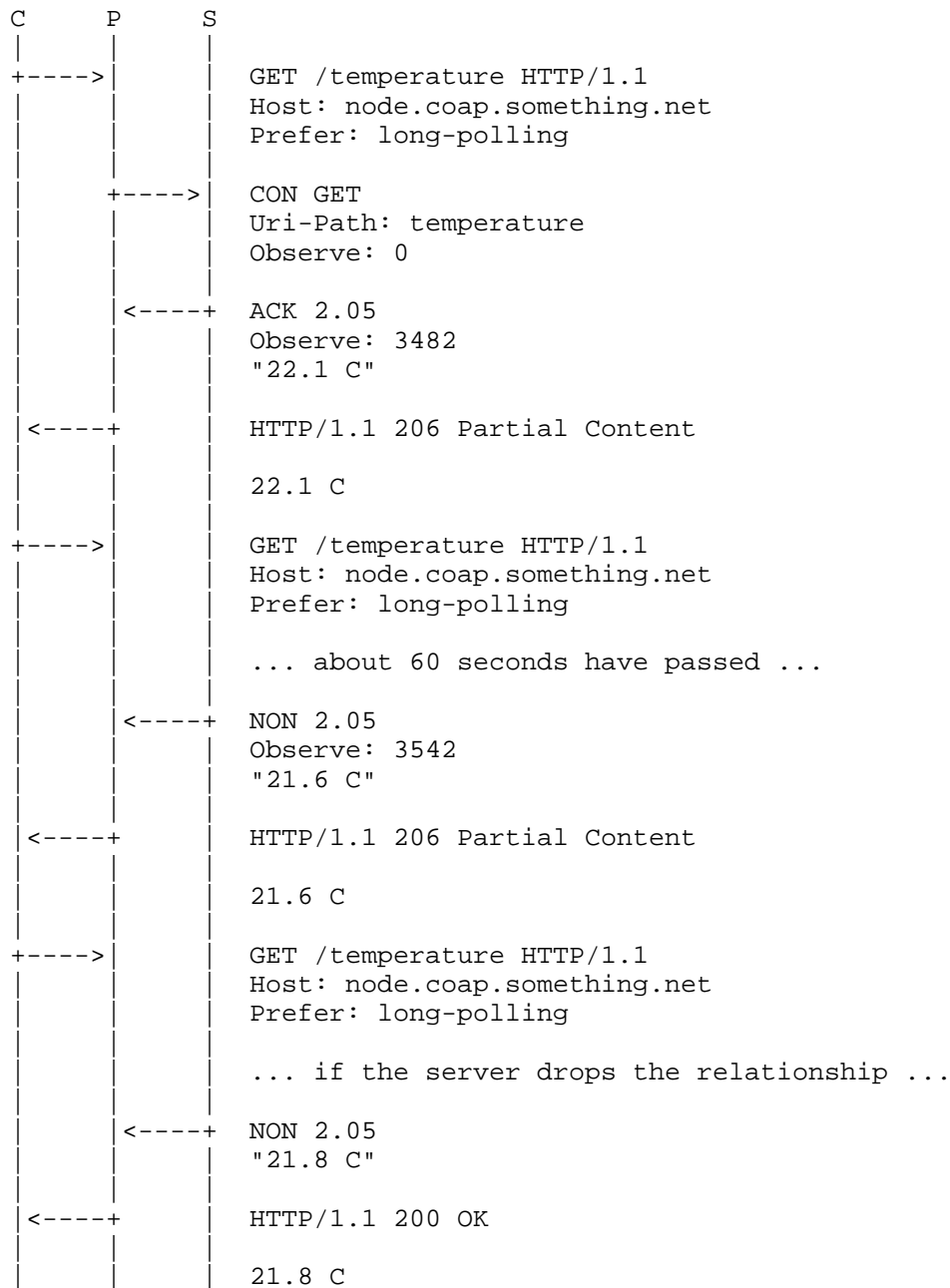


Figure 5: HTTP Long Polling to CoAP Observe

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.

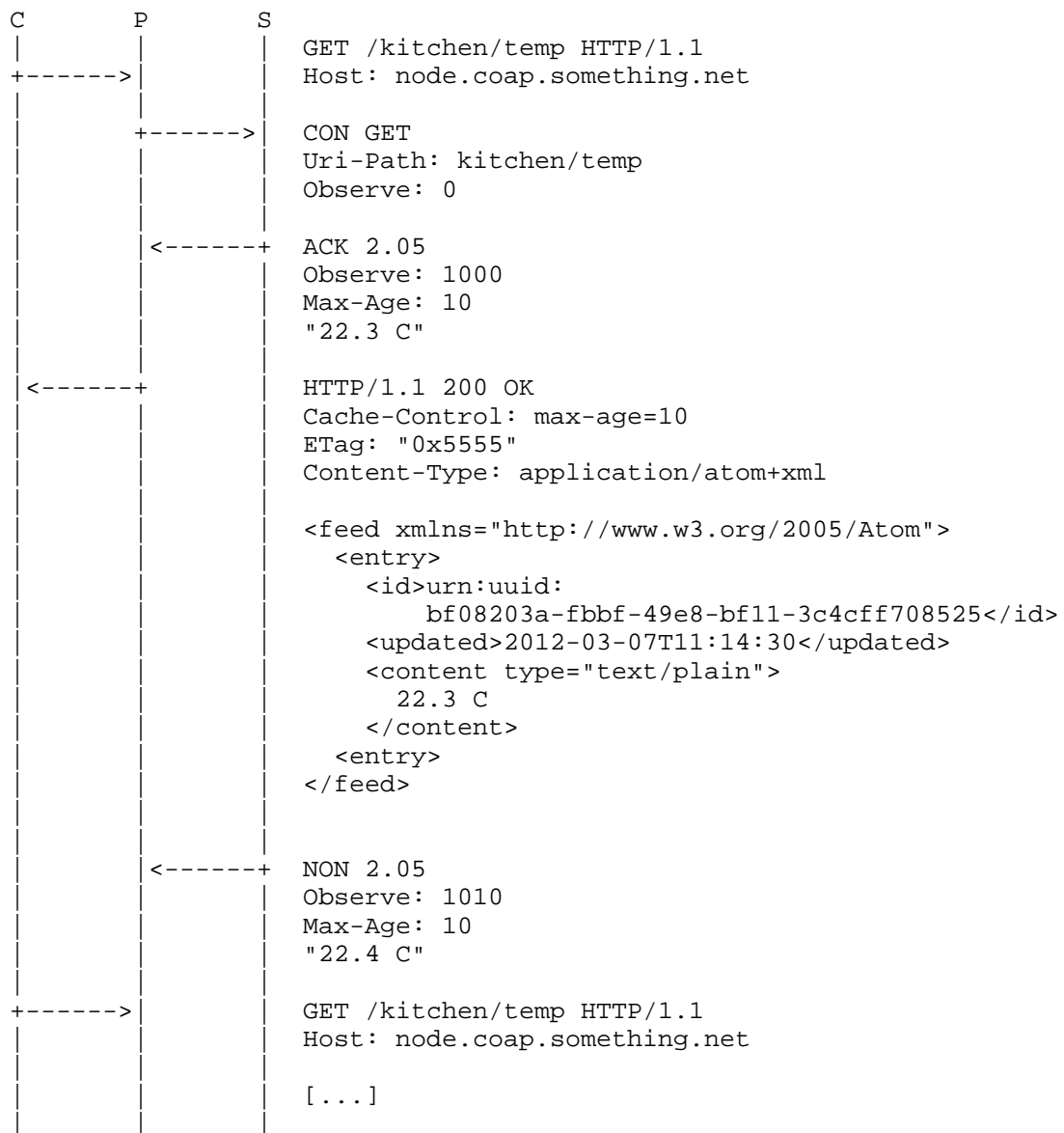


Figure 6: Observation via Atom feeds

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.

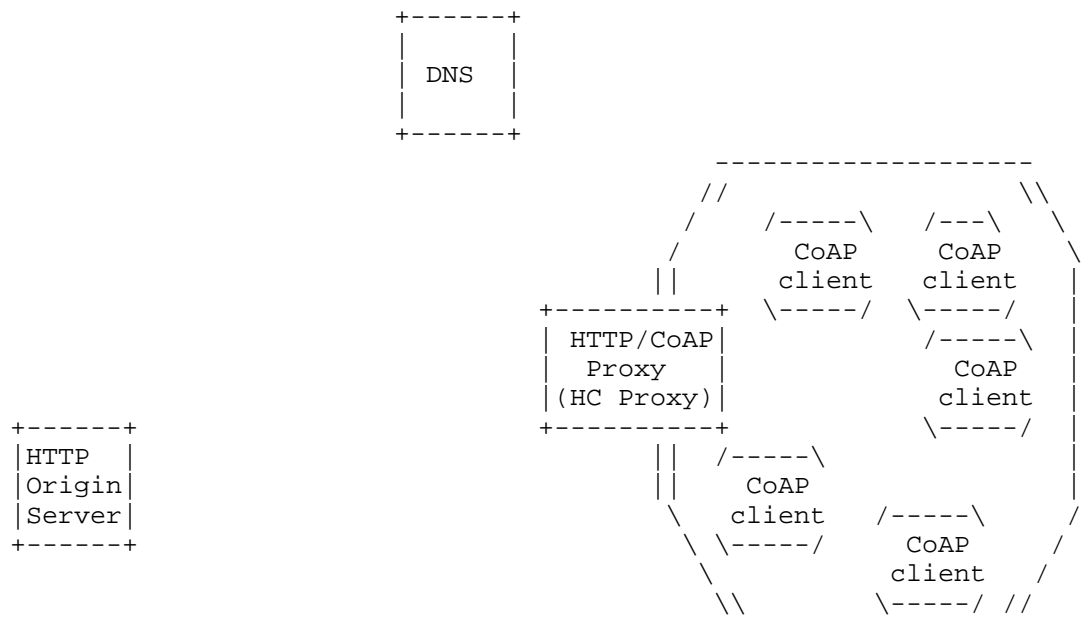


Figure 7: Client-side HC Proxy Deployment Scenario

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.

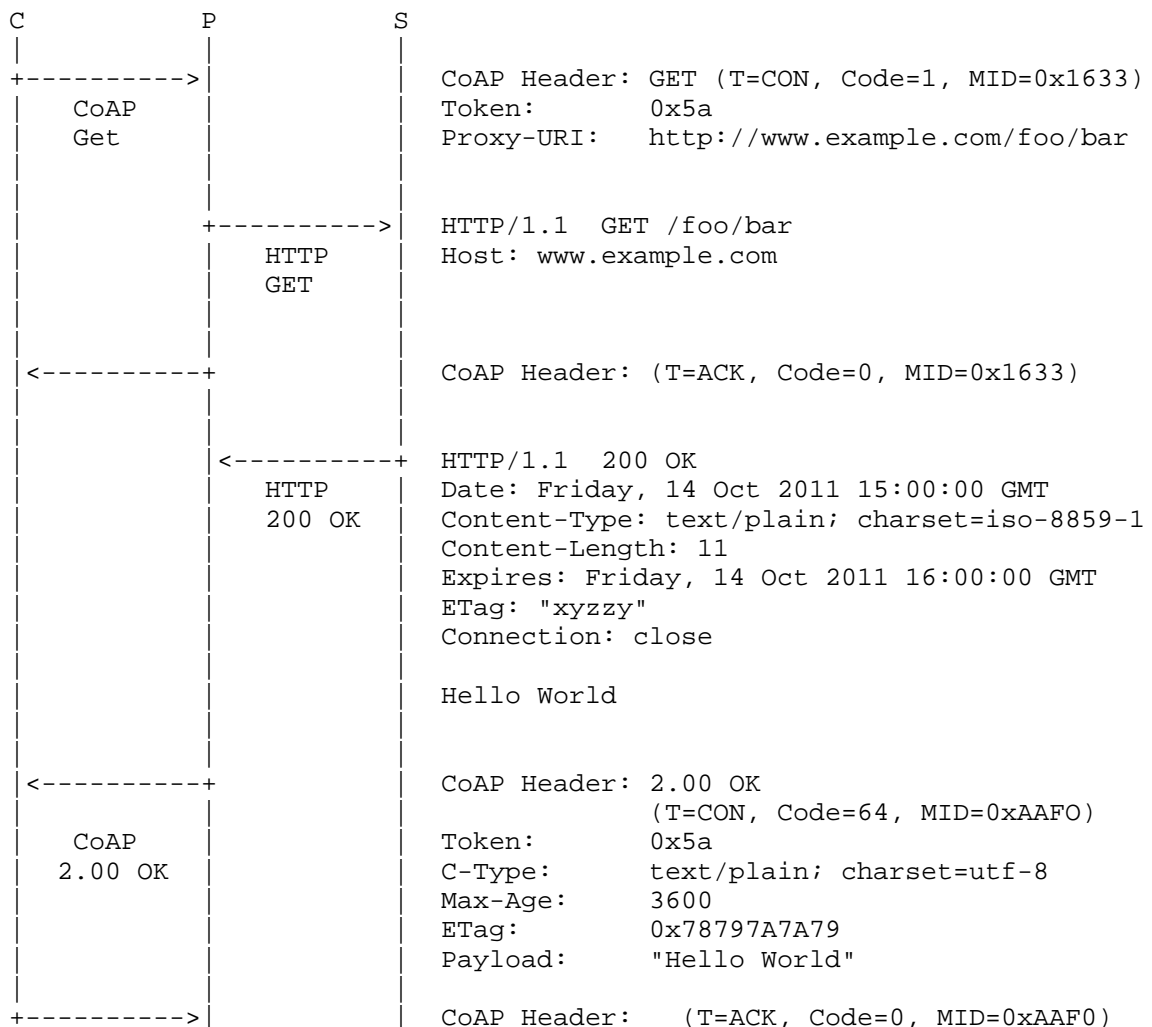


Figure 8: A Basic CoAP-HTTP GET Request

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.

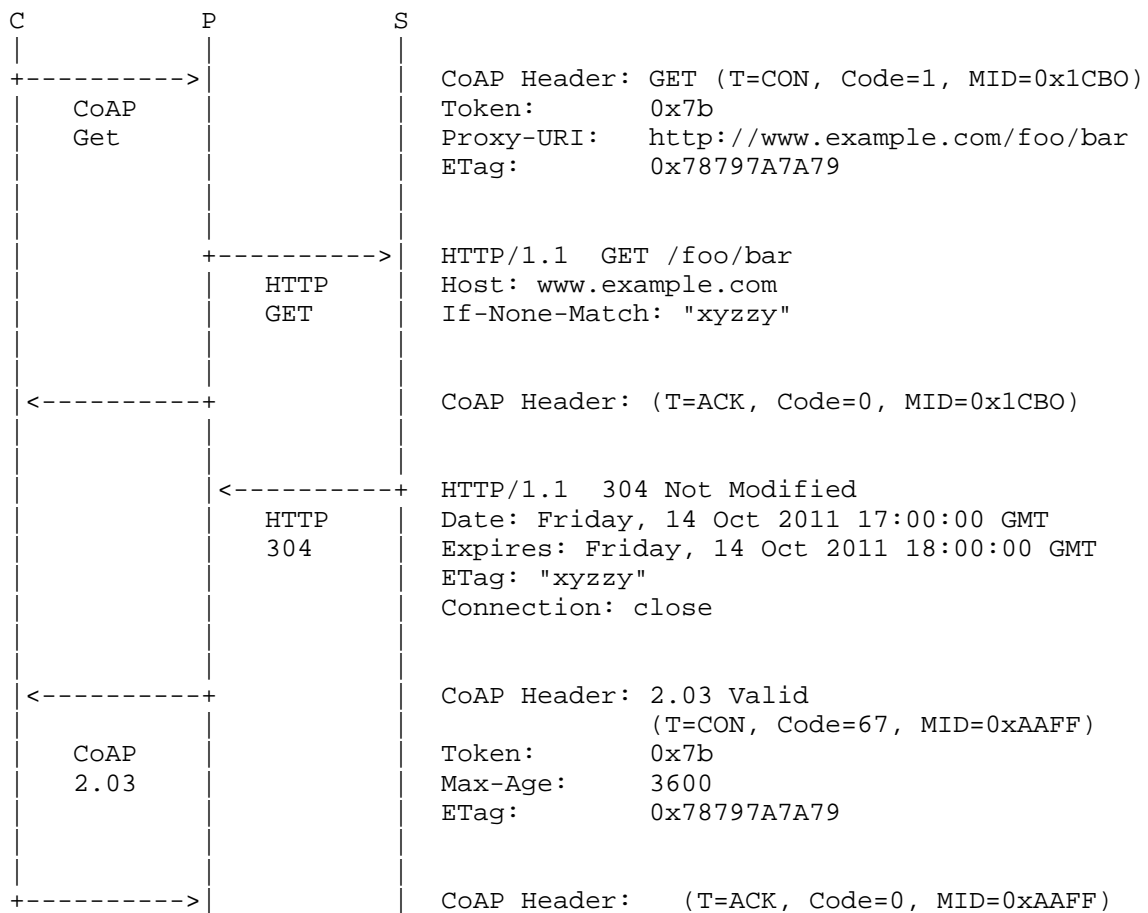


Figure 9: A CoAP-HTTP GET Request with an ETag Option

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

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

[I-D.ietf-core-observe]

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

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

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

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

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

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

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

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

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

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

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

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

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

11.2. Informative References

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

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

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

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

independent mapping source.

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

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

A.1. URL Map Algorithm

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

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

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

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

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

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

INPUT
* requested URL

OUTPUT
* target URL

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

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

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

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

A.2. Security Policy Map Algorithm

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

INPUT

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

OUTPUT

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

SYNTAX

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

EXAMPLE

<TBD>

A.3. Content-Type Map Algorithm

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

INPUT

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

OUTPUT

- * mapped content-type

SYNTAX

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

EXAMPLE

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

Authors' Addresses

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

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

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

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

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

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

Esko Dijk
Philips Research

Email: esko.dijk@philips.com

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

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

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on December 19, 2013.

Copyright Notice

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

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

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

Table of Contents

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

1. Introduction

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

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

2. Potential Solutions for Group Communication

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

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

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

3. Use Cases

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

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

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

4. Requirements

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

4.1. Background

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

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

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

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

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

4.2. General Requirements

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

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

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

4.3. Security Requirements

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

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

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

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

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

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

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

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

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

5. Group Communication Solutions

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

5.1. IP Multicast Transmission Methods

5.1.1. Serial unicast

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

5.1.2. Unreliable IP Multicast

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

5.1.3. Reliable IP Multicast

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

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

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

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

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

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

There are various approaches to achieve reliability, such as

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

5.2. Overlay Multicast

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

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

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

5.3. CoAP Application Layer Group Management

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

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

The following group management primitives are in general useful:

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

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

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

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

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

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

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

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

Figure 1: CoAP Header Options for Group Management

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

0	1	2	3
0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1
Ver	T	OC	Code
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://proxy1.bld2.example.com/groupmgt?j=group1&l=group2
```

6. DNS-SD Based Group Resource Manipulation

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

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

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

7. Group Discovery and Member Discovery

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

7.1. DNS-SD

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

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

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

7.2. CoRE Resource Directory

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

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

8. Deployment Guidelines

8.1. Overview

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

8.2. Implementation in Target Network Topologies

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

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

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

8.2.1. Single LLN Topology

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

8.2.1.1. Mesh-Under Multicast Routing

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

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

8.2.1.2. RPL Multicast Routing

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

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

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

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

8.2.1.3. RPL Routers with Non-RPL Hosts

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

8.2.1.4. Trickle Multicast Forwarding

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

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

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

8.2.1.5. Other Route-Over Methods

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

8.2.2. Single LLN with Backbone Topology

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

8.2.2.1. Mesh-Under Multicast Routing

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

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

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

8.2.2.2. RPL Multicast Routing

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

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

8.2.2.3. RPL Routers with Non-RPL Hosts

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

8.2.2.4. Trickle Multicast Forwarding

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

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

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

8.2.2.5. Other Route-Over Methods

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

8.2.3. Multiple LLNs with Backbone Topology

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

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

8.2.4. LLN(s) with Multiple 6LBRs

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

8.2.5. Conclusions

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

8.3. Implementation Considerations

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

8.3.1. MLD Implementation on LLNs and MLD alternatives

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

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

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

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

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

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

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

8.3.2. 6LBR Implementation

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

8.3.3. Backbone IP Multicast Infrastructure

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

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

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

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

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

9. Miscellaneous Topics

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

9.1. CoAP Multicast and HTTP Unicast Interworking

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

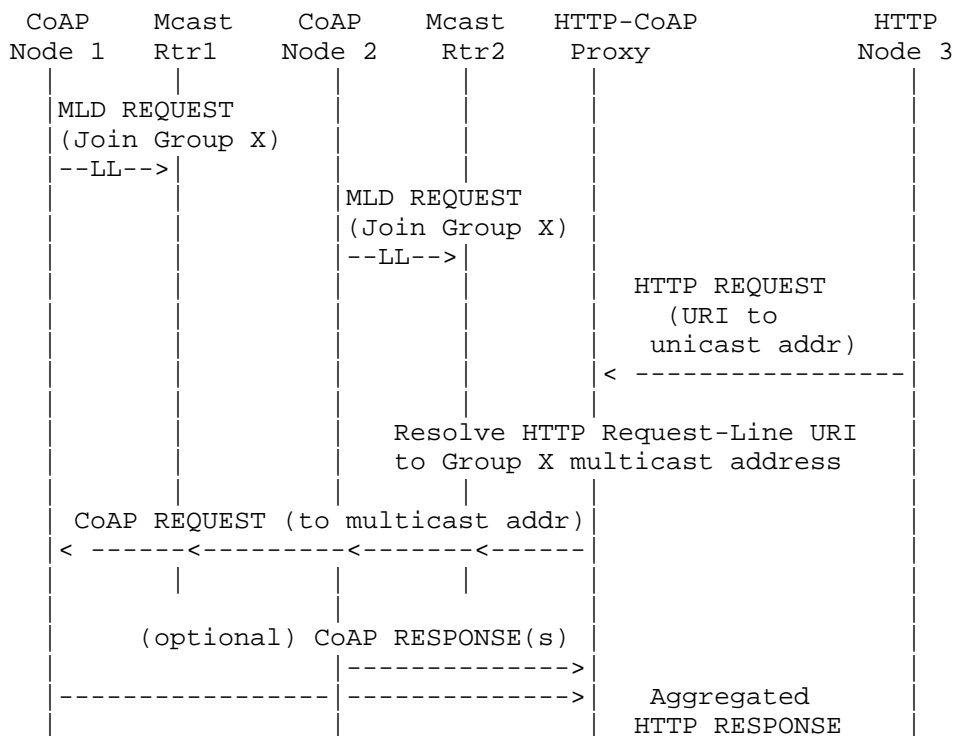




Figure 3: CoAP Multicast and HTTP Unicast Interworking

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

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

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

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

10. Acknowledgements

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

11. IANA Considerations

This memo includes no request to IANA.

12. Security Considerations

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

13. References

13.1. Normative References

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

- [RFC5374] Weis, B., Gross, G., and D. Ignjatic, "Multicast Extensions to the Security Architecture for the Internet Protocol", RFC 5374, November 2008.
- [RFC5867] Martocci, J., De Mil, P., Riou, N., and W. Vermeylen, "Building Automation Routing Requirements in Low-Power and Lossy Networks", RFC 5867, June 2010.
- [RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, March 2012.
- [RFC6636] Asaeda, H., Liu, H., and Q. Wu, "Tuning the Behavior of the Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) for Routers in Mobile and Wireless Networks", RFC 6636, May 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.

13.2. Informative References

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

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

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

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

[I-D.ietf-core-block]

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

[I-D.ietf-core-groupcomm]

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

[I-D.ietf-core-observe]

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

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

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

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

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

[I-D.rahman-core-groupcomm]

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

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

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

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

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

[I-D.vanderstok-core-bc]

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

[I-D.vanderstok-core-dna]

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

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

Appendix A. Multicast Listener Discovery (MLD)

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

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

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

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

Appendix B. CoAP-Observe Alternative to Group Communication

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

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

Authors' Addresses

Esko Dijk (editor)
Philips Research

Email: esko.dijk@philips.com

Akbar Rahman (editor)
InterDigital Communications, LLC

Email: Akbar.Rahman@InterDigital.com

CoRE
Internet-Draft
Intended status: Informational
Expires: February 9, 2014

Y. Doi
TOSHIBA Corporation
K. Lynn
Consultant
August 8, 2013

CoAP Content-Type Parameter Option
draft-doi-core-parameter-option-03

Abstract

Content-Types may have 'parameter' to make fine-grained description of contents. The CoAP Accept Content-Type Parameter Option (Accept-CT-Parameter Option) is CoAP options to add a parameter to a content type specified in CoAP Accept Options.

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 9, 2014.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
2. Use Cases of Option Parameter in CoAP	3
2.1. Clarification on URI, Resource, and Representation	3
2.2. Parameter Coordination	3
2.3. Schema Negotiation of Schema-Informed EXI Communication	4
3. Accept Content-Type Parameter Option	4
3.1. Requirements	4
3.2. Accept-CT-Parameter Option	5
3.2.1. Attribute ID	6
4. Security Considerations	6
5. IANA Considerations	6
6. Normative References	7
Appendix A. Consideration on Compact Encodings of Content-Type Parameter	8
Appendix B. ChangeLog	8
Authors' Addresses	9

1. Introduction

Content-Type field[RFC2045] have 'parameter' to make fine-grained description of contents. The document proposes the CoAP Content-Type Parameter Option that enables wide range of parameter description over content types used in CoAP.

2. Use Cases of Option Parameter in CoAP

2.1. Clarification on URI, Resource, and Representation

In CoAP, a resource is specified by a CoAP URI. However, in some use cases described in followings, an URI may correspond to multiple versions of the resource, or a resource may have multiple representations. As best practices, there are several ways to identify a version and a representation on a resource pointed by an URI. Some of discussions are given in [W3C.Finding.alternatives-discovery].

One of the approaches commonly used is to parameterize contents with content-type parameter and make a server-side content negotiation.

Basic specification of CoAP[I-D.ietf-core-coap] does not cover such content negotiation and this draft is to propose a way to mimic server-side content negotiation by making room for content type parameters with a new option.

2.2. Parameter Coordination

If a resource has wide range of representations, a client may try to specify what representation of the resource is requested by a GET message. In HTTP, Accept: header and content type (media type) parameter is used to coordinate parameters between clients and servers. audio/rtp-midi[RFC6295] is an example of a content-type with various parameters.

The audio use case seems not to be widely used in CoAP so far. However, the same use case is applicable for sensor data. Sensor data is time series data and it is possible to define a content type with preferred sensing interval to avoid over/underquality of sampling. In such cases, parameters with wildcard (rate=*) or range (rate=10-20) is useful.

Similar parameter coordination is also proposed in [I-D.wilde-atom-profile]. In the draft, several representations can exist on a resource defined with a URI, and a client can negotiate representation of the resource.

2.3. Schema Negotiation of Schema-Informed EXI Communication

In some use case of Schema-Informed EXI [W3C.REC-exi-20110310], a server and a client need to coordinate a XML schema to encode a message. If there are some versions of XML schemas in an application, a sender (may be server or client) needs to know schemas a receiver has.

There are two choices. First choice is to define a content-type for each version of XML schema. However, there are two problems. First, the Content-type ID space is a global space and not suitable to describe every minor revision. Second, Content-type ID per schema cannot describe relation between a lineage of schemas. XML schema could be backward compatible and newer schema version can be applied on older document validation and EXI encoding. Common practice on this is to use (major).(minor) style versioning.

However, content-ID or other class of ID cannot describe which version is compatible and which version is not compatible.

The other choice is to make use of content-type parameters. It seems to be more acceptable because parameter is local to each content-type. For example, an application may define 'application/example-exi' as a content-type for the application. The application may use 'sv' parameter as acceptable schema version. If the application use backward-compatible approach, 'Accept: application/example-exi;sv=1.4' from a client means the client can receive schema version 1.0, 1.1, 1.2, 1.3, and 1.4.

Detailed discussion on EXI schema negotiation can be found in [I-D.doi-exi-messaging-requirements].

3. Accept Content-Type Parameter Option

3.1. Requirements

In general, a content-type parameter has following notations.

```
parameter := ";" attribute "=" value
attribute := token ; case insensitive
value := token / quoted-string
```

In CoAP, a content-type parameter should have similar ability of expression with regards to use cases. At the same time, a CoAP option should be compact enough to fit in constrained environments.

As CoAP options do not have room for parameters, the content-type

parameter option is designed to be an independent option that gives additional description on a content-type in a CoAP message.

An attribute of CoAP option parameter should be fit in relatively smaller set. The authors consider the attribute part could be described in short integer (16 bits). On the other hand, the value part should have higher degree of freedom for applications including wildcards and range description. The author believes it is simple and safe to have a string value in option parameter option.

3.2. Accept-CT-Parameter Option

To enable server-side content type negotiation, an option to describe content type parameter is required. This document defines Accept-CT-Parameter option for the purpose.

Table 1: List of options. U: proxy-Unsafe, C: Critical, R: Repeatable

No	C	U	N	R	Name	Format	Length	Default
.								
TB D				x	Accept-CT-Parameter	(see below)	3-270B	(none)

The Accept-CT-Parameter option is used to attach a parameter on an Accept option on the same CoAP message. The Accept-CT-parameter options are proxy safe, elective.

An Accept-CT-Parameter option has two fields: attribute ID(aid), and value.

```
|<--- option length ---->|
+-----+-----+....-+
|   aid   | value   |
+-----+-----+....-+
|<2 Bytes>|<- optlen-2 ->|
```

:Figure 2: Structure of Accept-CT-Parameter Option

Attribute ID (aid) is a two-byte integer that describes the attribute name (key) of the parameter. Details are described in later section (see Table 2).

A value is opaque octets (Unicode string in most cases) with the length of the option length minus two (2) octets. A value may be empty. Meanings of the values should be defined by the content-type authority.

3.2.1. Attribute ID

Attribute ID is a 2-byte integer. An attribute is described in 2-byte integer as shown in the following table.

Table 2: List of Attribute IDs

ID	Name	Reference
0	(reserved)	
1	charset	RFC2045
2	version	RFC2045,RFC2046
3	boundary	RFC2045
4	type	RFC2046
5	padding	RFC2046
6	msgtype	RFC2616
7	filename	RFC2616
8	level	RFC2616
0xf000-0xffff	(reserved)	

Other attribute ID may be managed and added by IANA, based on first-come-first-serve basis for parameters defined in RFCs. Parameters described in an internet draft or in proprietary extensions may be added upon approval of core WG experts.

4. Security Considerations

Applications on CoAP servers should check the validity of parameters before use. It may contain arbitrary string value.

5. IANA Considerations

This document requests a CoAP option ID assigned to Accept-CT-Parameter option.

Attribute ID registry policy should be lined up with IANA considerations of ()[#I-D.ietf-core-coap].

6. Normative References

- [I-D.doi-exi-messaging-requirements]
Doi, D., "EXI Messaging Requirements",
draft-doi-exi-messaging-requirements (work in progress),
October 2012.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained
Application Protocol (CoAP)", draft-ietf-core-coap-18
(work in progress), June 2013.
- [I-D.wilde-atom-profile]
Wilde, E., "Profile Support for the Atom Syndication
Format", draft-wilde-atom-profile-01 (work in progress),
April 2013.
- [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.
- [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.
- [RFC6295] Lazzaro, J. and J. Wawrzynek, "RTP Payload Format for
MIDI", RFC 6295, June 2011.
- [W3C.Finding.alternatives-discovery]
Raman, T., "On Linking Alternative Representations To
Enable Discovery And Publishing", World Wide Web
Consortium Finding Finding-alternatives-discovery,
November 2006, <[http://www.w3.org/2001/tag/doc/
alternatives-discovery.html](http://www.w3.org/2001/tag/doc/alternatives-discovery.html)>.
- [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>>.

Appendix A. Consideration on Compact Encodings of Content-Type Parameter

The use of 'value' part of parameter is up to the content-type. Some content-type may use non-string integer or other format to describe values in compact format, e.g. TLV, fixed-length integers, etc.

The specification that defines a content-type may define ASCII/UTF-8 notation for HTTP use and binary compact notation for CoAP. Anyway, CoAP software/library will not need to understand content-type parameter. The parameter should be transferred from/to application without modification.

Appendix B. ChangeLog

- o from draft-doi-core-parameter-option-01 to 02
 - * Removed content-type parameter of message content, and this draft is now for content type parameter for Accept option
 - * Added description on relation between resource and representation on RESTful architecture
 - * Added even some more use cases
- o from draft-doi-core-parameter-option-00 to 01
 - * Added more use cases
 - * Parameter format has changed and now have clearly different format for content-type and accept-content-type
- o from draft-doi-core-option-parameter-option-00 to draft-doi-core-ct-parameter-option-00
 - * Effect of the option is limited to Content-Type parameter (i.e. Content-Type and Accept option).
 - * Name of the option has changed to 'Content-Type Parameter Option'
 - * Removed Accept: preference use case (CoAP already defines accept option order as preference)
 - * Removed Option Parameter Option 2 and 3.

- * Option Parameter Option is replaced with content-type parameter option and accept content-type parameter option.
- * Options are now considered to be proxy-safe (is it the right assumption?)
- * Some other unnecessary descriptions on option parameters (such as option order constraint) are removed

Authors' Addresses

Yusuke Doi
TOSHIBA Corporation
Komukai Toshiba Cho 1
Saiwai-Ku
Kawasaki, Kanagawa 2128582
JAPAN

Phone: +81-45-342-7230
Email: yusuke.doi@toshiba.co.jp

Kerry Lynn
Consultant

Phone: +1 978 460 4253
Email: kerlyn@ieee.org

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 16, 2014

S. Gerdes
O. Bergmann
C. Bormann
Universitaet Bremen TZI
July 15, 2013

Delegated CoAP Authorization Function (DCAF)
draft-gerdes-core-dcaf-authorize-00

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on January 16, 2014.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	4
1.1. Features	4
1.2. Terminology	4
1.2.1. Roles	4
2. System Overview	6
3. Protocol	7
3.1. Overview	7
3.2. Unauthorized Resource Request Message	8
3.3. AS(RS) Information Message	9
3.4. Authorization Request	9
3.5. Ticket Request Message	11
3.6. Ticket Grant Message	11
3.7. Ticket Transmission Message	13
3.8. DTLS Channel Setup Between C and RS	13
3.9. Authorized Resource Request Message	13
4. Ticket	15
4.1. Face	15
4.2. Verifier	15
4.3. Revocation	16
4.3.1. Lifetime	16
4.3.2. Revocation Messages	16
5. Payload Format and Encoding (application/dcaf+json)	17
5.1. Examples	18
6. DTLS PSK Generation Methods	21
6.1. DTLS PSK Transfer	21
6.2. Distributed Key Derivation	21
7. Authorization Configuration	23
8. Trust Relationships	24
9. Listing Authorization Server Information in a Resource Directory	25
10. Examples	26
10.1. Access Granted	26
10.2. Access Denied	28
10.3. Access Restricted	28
11. Security Considerations	30
12. IANA Considerations	31
12.1. dcaf+json Media Type Registration	31
12.2. CoAP Content Format Registration	32
13. References	33
13.1. Normative References	33
13.2. Informative References	33
Authors' Addresses	35

1. Introduction

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

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

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

1.1. Features

- o Providing means for DTLS communication with pre-shared keys.
- o Authenticated exchange of authorization information.
- o Using only symmetric encryption on constrained nodes.

1.2. Terminology

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

1.2.1. Roles

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

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

Client (C): A device which wants to access a resource on the Resource Server.

Resource Owner: The subject who owns the resource and controls its access permissions.

2. System Overview

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

To gain access to a specific resource on a Resource Server (RS), a client (C) has to request an authorization ticket from RS' Authorization Server (AS(RS)) either directly or, if it is a constrained device using its own Authorization Server (AS(C)).

If AS(RS) decides that C is allowed to access the resource, it generates a DTLS pre-shared key (PSK) for the communication between C and RS and wraps it together with the access permissions into an authorization ticket. After presenting the ticket to the RS, C and RS can communicate securely.

To be able to provide for the authentication and authorization services, the Authorization Servers have to fulfill several requirements. They MUST have enough disk space to store a sufficient number of credentials (matching the number of clients and Resource Servers) and MUST possess means for user interaction, for example input/output devices like keyboard and display to allow for configuration of authorization information by the Resource Owner. Additionally they MUST have enough processing power to handle the authorization requests for all devices they are responsible for.

3. Protocol

The DCAF protocol comprises three parts:

1. Transmission of authorization information between C and RS,
2. transmission of authorization requests and access grants between C and AS(C), and
3. transmission of authorization requests and access grants between AS(C) and AS(RS).

3.1. Overview

In Figure 1, a protocol flow with an Authentication Server for RS is depicted (messages in square brackets are optional):

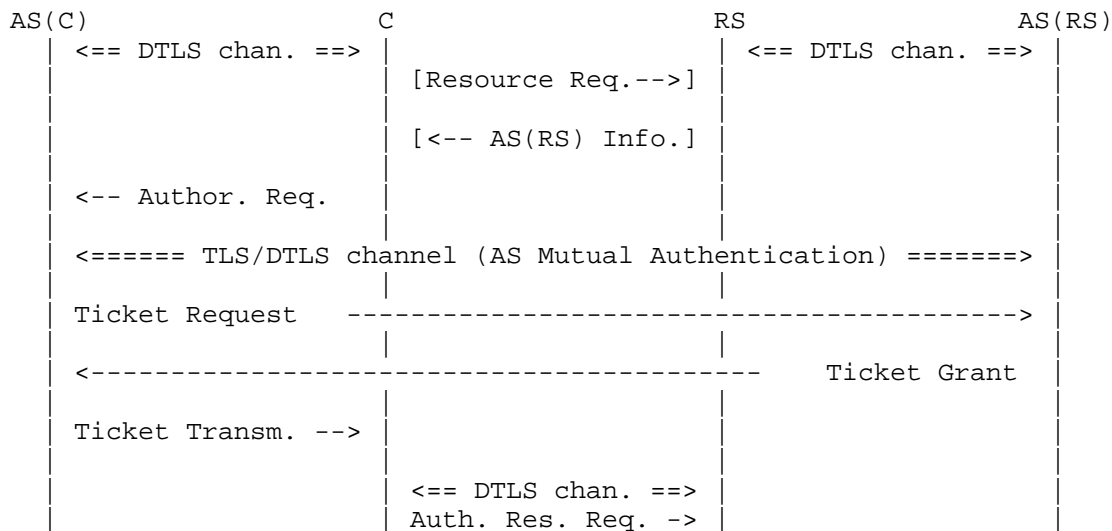


Figure 1: Protocol Overview

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

Instead of the initial Unauthorized Resource Request message, C MAY lookup the desired resource in a resource directory (cf. [I-D.ietf-core-resource-directory]) that lists RS's resources as

discussed in Section 9.

Once C knows AS(RS)'s address, it can send a request for authorization to AS(RS) using its own Authorization Server (AS(C)). After authenticating AS(C), AS(RS) decides if C is allowed to access the requested resource and in this case generates an access ticket for C. The ticket contains a representation of the permissions C has for the resource as well as keying material for the establishment of a secure channel. C keeps one part of the access ticket and presents the other part to RS to prove its right to access the resource. With their respective parts of the ticket, C and RS are able to establish a secure channel.

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

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

3.2. Unauthorized Resource Request Message

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

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

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

Unauthorized Resource Request messages MUST be rejected with a 4.01 (Unauthorized) response. In this response, the Resource Server MUST provide proper AS(RS) Information to enable the Client to request an access ticket from RS's Authorization Server as described in

Section 3.3.

3.3. AS(RS) Information Message

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

The message MAY also contain a timestamp generated by RS.

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

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

Figure 2: AS(RS) Information Payload Example

In this example, the attribute AS points the receiver of this message to the URI "coaps://as-rs.example.com/authorize" to request access permissions. The originator of the AS(RS) Information payload (i.e. RS) uses a local clock that is loosely synchronized with wall clock time. Therefore, it has included a time stamp on its own time scale that is used as a nonce for replay attack prevention. Refer to Section 4.1 for more details concerning the usage of time stamps to ensure freshness of authorization tickets.

3.4. Authorization Request

To retrieve an authorization ticket for the resource that C wants to access, C sends an Authorization Request to its authorization server AS(C). The Authorization Request is constructed as follows:

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

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

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

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

1. Contact information for the AS(RS) to use.
2. An identifier of C that is recognized by AS(RS).
3. An absolute URI of the resource that C wants to access.
4. The actions that C wants to perform on the resource.
5. Any time stamp generated by RS.

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

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

Figure 3: Authorization Message Example

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

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

message from RS.

The response to an Authorization Request is delivered by AS(C) back to C in a Ticket Transmission message.

3.5. Ticket Request Message

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

When the payload is correct AS(C) MUST create a Ticket Request message from the Authorization Request received from C as follows:

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

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

3.6. Ticket Grant Message

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

AS(RS) decides whether or not access is granted to the requested resource and then creates a Ticket Grant message that reflects the result. If AS(RS) grants access to the requested resource it has to create an access ticket comprised of a Face and a Verifier as described in Section 4.1.

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

The Ticket Grant message MAY provide cache-control options to enable intermediaries to cache the response. The message MAY be cached according to the rules defined in [I-D.ietf-core-coap] to facilitate ticket retrieval when C has crashed and wants to recover the DTLS session with RS.

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

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

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

Figure 4: Example Ticket Grant Message

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

2.05 Content

Content-Format: application/dcaf+json

Figure 5: Example Ticket Grant Message With Reject

3.7. Ticket Transmission Message

A Ticket Transmission message delivers the authorization information sent by AS(RS) in a Ticket Grant message to the requesting client C. The Ticket Transmission message **MUST** be the first and only response to the corresponding Authorization Request message sent from C to AS(C) and include any authorization information from AS(RS) contained in the Ticket Grant message.

3.8. DTLS Channel Setup Between C and RS

Using the information contained in a positive response to its Authorization Request (i.e. a Ticket Transmission message that contains a Face and a Verifier), C can initiate establishment of a new DTLS channel with RS. To use DTLS with pre-shared keys, C follows the PSK key exchange algorithm specified in Section 2 of [RFC4279], with the following additional requirements:

1. C **MUST** set the `psk_identity` field of the ClientKeyExchange message to the ticket Face received in the Ticket Transmission message.
2. C **MUST** use the ticket Verifier as PSK when constructing the premaster secret.

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

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

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

3.9. Authorized Resource Request Message

Successful establishment of the DTLS channel between C and RS ties the authorization information contained in the `psk_identity` field to this channel. Any request that RS receives on this channel is checked against these authorization rules. Incoming CoAP requests

that are not Authorized Resource Requests MUST be rejected by RS with 4.01 response as described in Section 3.2.

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

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

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

4. Ticket

A DCAF ticket consists of two parts, the Face and the Verifier.

4.1. Face

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

- o Authorization Information
- o Client Identifier
- o A timestamp generated by AS(RS)

Optionally, Face MAY also contain:

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

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

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

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

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

4.2. Verifier

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

4.3. Revocation

4.3.1. Lifetime

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

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

4.3.2. Revocation Messages

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

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

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

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

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

The following attributes are defined:

AS: Authentication Server. This attribute denotes the authorization server that is in charge of the resource specified in attribute R. The attribute's value is a string that contains an absolute URI according to Section 4.3 of [RFC3986].

AI: Authorization Information. A data structure used to convey authorization information from AS(RS) to RS (see below).

CI: Client Identity. A string that identifies the initiator of the authorization request. This label MAY be a fully qualified domain name, an IP address, or any other character literal that is used by the Authorization Server to decide whether or not access is granted to the requesting entity.

E: Encrypted Ticket Face. A string containing an encrypted ticket Face encoded as base64 according to Section 4 of [RFC4648].

K: Key. A string that identifies the shared key between RS and AS(RS) that can be used to decrypt the contents of E. If the attribute E is present and no attribute K has been specified, the default is to use the current session key for the secured channel between RS and AS(RS).

M: Methods. The list of actions to be performed on the resource R, encoded as an array of strings. In an authorization request, this list contains the actions that the initiator of the request wants to perform. In an authorization ticket, this attribute denotes the actions that are permitted.

R: Resource. A string that denotes the absolute URI of the resource to be accessed. As the access ticket is requested in order to establish a DTLS connection with the server that hosts this resource, the URI scheme typically is "coaps".

TS: Time Stamp. An optional time stamp that indicates the instant when the access ticket request was formed. This attribute can be used by the resource server in an AS(RS) Information message to convey a time stamp in its local time scale (e.g. when it does not have a real time clock with synchronized global time). When the

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

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

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

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

5.1. Examples

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

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

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

The following example shows a ticket for the distributed key

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

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

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

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

```
{
  "F": {
    "AI": { "Role" : 3 },
    "CI": "2001:db8:ab9:1234:7920:3133:ae5f:87",
    "TS": 2938749,
    "L": 3600,
    "G": "hmac_sha256"
  },
  "V": "zrPOuc6xzr/Pjc+Bz4TOuSDOvM6lIM68zq3Ou865Cg=="
}
```

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

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

```
{
  "E": "rjtolfjyX9q7Emxgsnz+nf0xTQhe1MjzZBRoIEW4vmSVlyJdW4KDgVtW
        LyBnQSVX0lmVpxUYbdNuk/5PkCOJBeex0obiEBC1UmKoJfJfjy7bLQhq
        k9HuJD7cvjHNOVZtNZf5qrxt7xJSoZFe6j/SJuxGNH/72SPDrdMQeXJI
        pX6vCJB698FcRDOXh/ipi9KT8YWeo/ljUMgJc+LI",
  "K": "key0",
  "V": "zrPOuc6x zr/Pjc+Bz4TOuSDOvM6lIM68zq3Ou865Cg=="
}
```

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

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

6. DTLS PSK Generation Methods

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

6.1. DTLS PSK Transfer

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

The DTLS PSK transfer method is defined as follows:

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

6.2. Distributed Key Derivation

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

The distributed key derivation method is defined as follows:

- o AS(RS) and RS both generate the DTLS PSK using the information included in Face. They use an HMAC algorithm on Face with a shared key. The result serves as the DTLS PSK. How AS(RS) and RS negotiate the used HMAC algorithm is not in the scope of this document. They MAY however use the HMAC algorithm they use for their DTLS connection.
- o AS(RS) MUST include a representation of the DTLS PSK in the Verifier.

- o As AS(RS) and C do not have a shared secret, the Verifier MUST be transmitted to C using encrypted channels.
- o AS(RS) MUST NOT include a representation of the DTLS PSK in Face.
- o AS(RS) MUST NOT encrypt Face.

7. Authorization Configuration

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

8. Trust Relationships

C and RS trust their respective Authorization Servers and vice versa. How this trust is established, is not in the scope of this document. It may be achieved by using a bootstrapping mechanism similar to [bergmann12].

Additionally, AS(RS) and AS(C) already have a trust relationship. Its establishment is also not in the scope of this document. It fulfills the following conditions:

1. AS(RS) has means to identify AS(C) (e.g. it has a certificate of AS(C) or a PKI in which AS(C) is included) and vice versa
2. AS(RS) knows that AS(C) is responsible for C
3. AS(RS) can be sure that AS(C) does not transmit tickets that have been generated for C to another client

AS(RS) trusts C implicitly because it trusts AS(C). The DCAF Protocol does not provide any means for AS(RS) to validate that a resource requests stems from C.

C trusts AS(RS) because AS(C) trusts AS(RS).

AS(C) trusts RS implicitly because it trusts AS(RS)

C implicitly trusts RS because it trusts AS(C) and because RS can proof that it shares a key with AS(RS).

AS(C) <-----> AS(RS)

```

  /|\          /|\
  |            |
  \|/          \|/

```

C RS

9. Listing Authorization Server Information in a Resource Directory

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

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

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

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

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

The resource directory that hosts the resource descriptions of RS could list the following description. In this example, the URI "ep/nodel38/a/switch2941" is relative to the resource context "coaps://as-rs.example.com/", i.e. the authorization server AS(RS).

```
<ep/nodel38/a/switch2941>;rt="auth-request";ct=TBD1;ep="nodel38"
;title="Request Client Authorization"
;anchor="coaps://as-rs.example.com/"
```

10. Examples

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

10.1. Access Granted

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

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

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

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

AS(C) --> AS(RS) [Mid=23146]
POST ep/node138/a/switch2941
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
 "CI": "2001:DB8::c",
 "M": [ "PUT" ],
 "R": "coaps://[2001:DB8::dcaf:1234]/a/switch2941"
}

AS(C) <-- AS(RS)
```

```

2.05 Content [Mid=23146]
Content-Format: application/dcaf+json
{ "F": { "AI": { "R" : "a/switch2941", "M" : [ "GET", "PUT" ] },
        "CI": "2001:DB8::c",
        "TS": "2013-07-04T20:17:38.002",
        "G": "hmac_sha256"
      },
  "V": "yVLYZZ5Nssbn0by3fqe19WK6jHdoYyNej2d/kSuBLw="
}

C <-- AS(C)
2.05 Content [Mid=1235,Token="tok"]
Content-Format: application/dcaf+json
{ "F": { "AI": { "R" : "a/switch2941", "M" : [ "GET", "PUT" ] },
        "CI": "2001:DB8::c",
        "TS": "2013-07-04T20:17:38.002",
        "G": "hmac_sha256"
      },
  "V": "MR5TMrNngbSEAkF10akmsdbmzF0gqxGI/d3KjwT8GxI="
}

C --> RS
ClientHello (TLS_PSK_WITH_AES_128_CCM_8)

C <-- RS
ServerHello (TLS_PSK_WITH_AES_128_CCM_8)
ServerHelloDone

C --> RS
ClientKeyExchange
  psk_identity="{\"F\":{\"AI\":{\"R\":\"a/switch2941\",\"M\":[\"GET\",\"PUT\"]},
                    \"CI\": \"2001:DB8::c\",
                    \"TS\": \"2013-07-04T20:17:38.002\",
                    \"G\": \"hmac_sha256\"}}"

(C decodes the contents of V and uses the result as PSK)
ChangeCipherSpec
Finished

(RS calculates PSK from AI, CI, TS and its session key
 HMAC_sha256("{\"R\":\"a/switch2941\",\"M\":[\"GET\",\"PUT\"]}"+
              "2001:DB8::c"+"2013-07-04T20:17:38.002",
              "secret")
= 311e5332b36781b484024165d1a926b1d6e6cc5d20ab1188fdddca8f04fc1b12
)

C <-- RS
ChangeCipherSpec

```

Finished

10.2. Access Denied

This example shows a denied Authorization request for the DELETE operation.

```
C --> RS
DELETE a/switch2941
```

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

```
C --> AS(C)
POST client-authorize
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
 "CI": "2001:DB8::c",
 "M": [ "DELETE" ],
 "R": "coaps://[2001:DB8::dcaf:1234]/a/switch2941"
}
```

```
AS(C) --> AS(RS)
POST ep/node138/a/switch2941
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
 "CI": "2001:DB8::c",
 "M": [ "DELETE" ],
 "R": "coaps://[2001:DB8::dcaf:1234]/a/switch2941"
}
```

```
AS(C) <-- AS(RS)
2.05 Content
Content-Format: application/dcaf+json
```

```
C <-- AS(C)
2.05 Content
Content-Format: application/dcaf+json
```

10.3. Access Restricted

This example shows a denied Authorization request for the operations GET, PUT, and DELETE. AS(RS) grants access for PUT only.

```
AS(C) --> AS(RS)
POST ep/node138/a/switch2941
Content-Format: application/dcaf+json
{"AS": "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
 "CI": "2001:DB8::c",
 "M": [ "GET", "PUT", "DELETE" ],
 "R": "coaps://[2001:DB8::dcaf:1234]/a/switch2941"
}
```

```
AS(C) <-- AS(RS)
2.05 Content
Content-Format: application/dcaf+json
{ "F": { "AI": { "R" : "a/switch2941", "M" : [ "GET", "PUT" ] },
        "CI": "2001:DB8::c",
        "TS": "2013-07-04T21:33:11.930",
        "G": "hmac_sha256"
      },
  "V": "NZ8Q3o8P4eHOzkoscaUpoRvrn5d74Cscw/aXAiNmC/k="
}
```

11. Security Considerations

As this protocol builds on transitive trust between authorization servers as mentioned in Section 8, AS(RS) has no means to validate that a resource request originates from C. It has to trust AS(C) that it is responsible for C and that it does not give authorization tickets meant for C to another client nor disclose the contained session key.

The Authentication Server also constitutes a single point of failure. If the Authentication Server fails, the resources on all Resource Servers it is responsible for cannot be accessed any more. Thus, it is crucial for large networks to use Authorization Servers in a failsafe setup.

12. IANA Considerations

The following registrations are done following the procedure specified in [RFC6838].

Note to RFC Editor: Please replace all occurrences of "[RFC-XXXX]" with the RFC number of this specification.

12.1. dcaf+json Media Type Registration

Type name: application

Subtype name: dcaf+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC4627]. Specifically, only the primitive data types String and Number are allowed. The type Number is restricted to int (i.e., no negative numbers, fractions or exponents are allowed). Encoding MUST be UTF-8. These restrictions simplify implementations on devices that have very limited memory capacity.

Security considerations: TBD

Interoperability considerations: TBD

Published specification: [RFC-XXXX]

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): dcaf

Macintosh file type code(s): none

Person & email address to contact for further information: TBD

Intended usage: COMMON

Restrictions on usage: None

Author: TBD

Change controller: IESG

12.2. CoAP Content Format Registration

This document specifies a new media type `application/dcaf+json` (cf. Section 12.1). For use with CoAP, a numeric Content-Format identifier is to be registered in the "CoAP Content-Formats" sub-registry within the "CoRE Parameters" registry.

Note to RFC Editor: Please replace all occurrences of "RFC-XXXX" with the RFC number of this specification.

Media type	Encoding	Id.	Reference
<code>application/dcaf+json</code>	<code>utf-8</code>	TBD1	[RFC-XXXX]

13. References

13.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013.

13.2. Informative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-12 (work in progress), June 2013.
- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-09 (work in progress), July 2013.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-ietf-core-resource-directory-00 (work in progress), June 2013.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [bergmann12] Bergmann, O., Gerdes, S., Schaefer, S., Junge, F., and C. Bormann, "Secure Bootstrapping of Nodes in a CoAP Network", IEEE Wireless Communications and Networking Conference Workshops (WCNCW), April 2012.

Authors' Addresses

Stefanie Gerdes
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63906
Email: gerdes@tzi.org

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63904
Email: bergmann@tzi.org

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

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 16, 2014

K. Hartke
Universitaet Bremen TZI
July 15, 2013

Liveliness and Cancellation of Separate Responses and Observations
draft-hartke-core-coap-liveliness-00

Abstract

This document extends the Constrained Application Protocol (CoAP) with a simple mechanism for a client to determine the "liveliness" of a request for which it is still expecting a response or a notification, and with a mechanism to request the cancellation of such a request.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 16, 2014.

Copyright Notice

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

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

Table of Contents

1. Requirements Notation	3
2. Liveliness	3
3. Cancellation	5
4. Security Considerations	6
5. IANA Considerations	6
6. Acknowledgements	6
7. Normative References	6
Author's Address	6

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 RFC 2119 [RFC2119].

2. Liveliness

When a CoAP server [I-D.ietf-core-coap] needs longer to process a request than it has time before sending back an acknowledgement, it can acknowledge the request message first and return the response at a later time (Separate Response). However, a client that does not receive any response for some time (despite the acknowledgement, which effectively is the server's promise to send a response) cannot know if the server simply hasn't finished processing the request yet or, for example, if the server had to reboot and thus couldn't finish the task.

Similarly, when a client observes a resource [I-D.ietf-core-observe] but hasn't received a notification for some time, it cannot know if the resource state simply did not change or if the server forgot the the client's interest in the resource. As long as the client still has a fresh representation of the target resource, this is not a problem. However, eventually the client must decide whether it needs to register its interest in the resource again or not (provided that it's still interested in the resource). If the client at this point makes the wrong decision, it ends up being in the list of observers of the resource either twice or not at all, both of which is not desirable.

This document extends CoAP with a simple mechanism for a client to determine the "liveliness" of a pending request. The liveliness check consists of the exchange of two CoAP messages: a "ping" and a "pong".

Ping: An Empty, Confirmable message that specifies the Token of a request for which the client is still expecting a response or a notification.

Pong: An Empty Acknowledgement message that is returned by the server if it recognises the token and wishes to reinforce the promise to send a response or further notifications; otherwise, an Empty Reset message.

The mechanism is thus just a small extension of the "CoAP ping" defined in [I-D.ietf-core-coap]. A short example is shown in Figure 1 below.

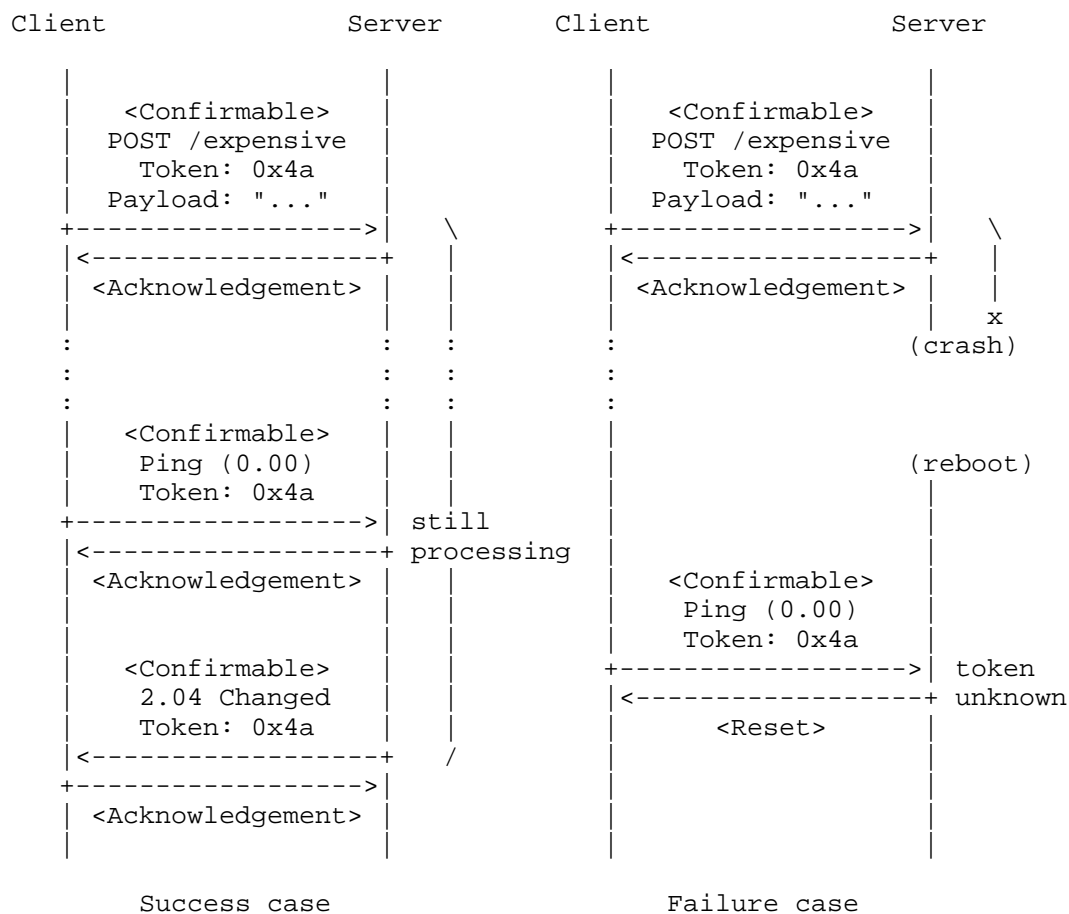


Figure 1: Liveliness check of a request with token 0x4a

A client that is expecting a response or notification SHALL NOT assume that the associated token is no longer in use until it has sent a Ping and received a matching Pong.

Furthermore, the client SHOULD NOT make a new resource-consuming request to the server until it has sent a Ping and received a matching Pong, such as a POST request taking significant processing time or a GET request with an Observe Option. The client MAY significantly increase the number of retransmit attempts (MAX_RETRANSMIT) for a Ping if necessary, and MAY truncate the retransmission timeout to a maximum of no less than 60 seconds.

Support for this mechanism is REQUIRED.

3. Cancellation

A CoAP client that is no longer interested in receiving a Separate Response or further notifications while observing a resource, can simply "forget" the pending request. When the server then sends the response or a notification, the client will not recognize the Token in the message and thus return a Reset message. This signals to the server the lost interest of the client and, in the case of an resource observation, will cause the server to remove the client from the list of observers. The resources allocated by the server to the request are effectively "garbage collected" by the server.

In some circumstances it may be desirable to cancel a pending request and release the resources allocated by the server more eagerly. This document extends CoAP with a mechanism for a client to request cancellation of a pending request.

A client MAY request the cancellation of pending request by sending a Confirmable or Non-Confirmable CoAP message with the Code field set to 7.31 and the Token field set to the token of the request to be cancelled.

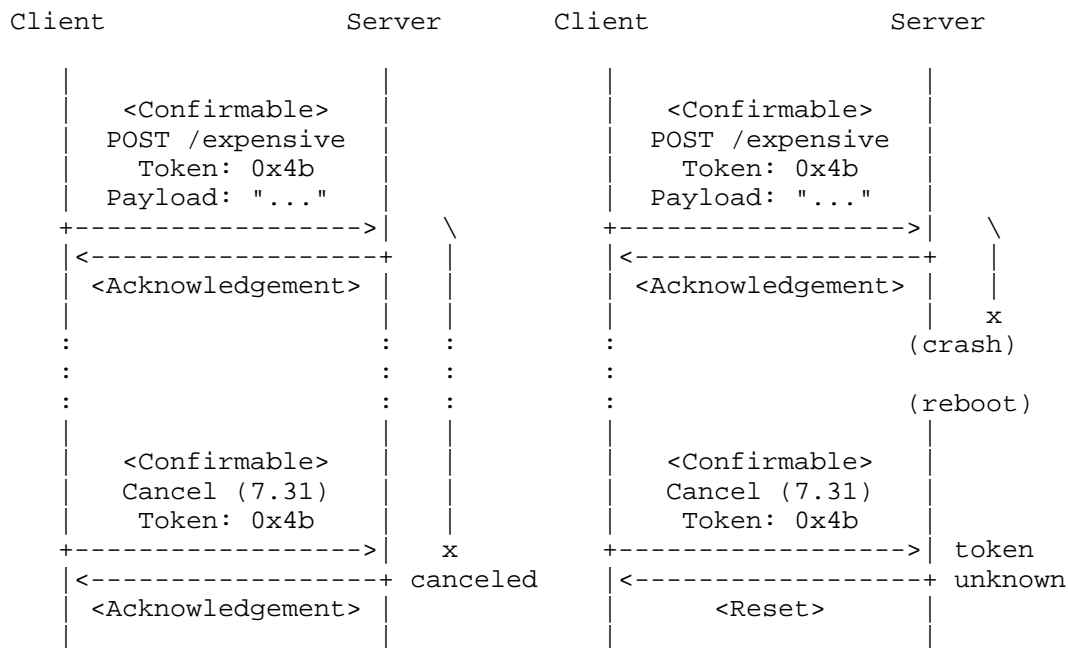


Figure 2: Cancellation of a request with token 0x4b

A server SHOULD NOT send any response or further notification in reply to the specified request after receiving such a message, and MUST remove the client from the list of observers of the target resource of the request.

The server SHOULD abort any ongoing tasks related to the request. However, if it is not possible to abort the tasks without leaving the system in an inconsistent state, it MAY continue to execute the tasks and just suppress the return of the resulting response.

An example is shown in Figure 2 above.

Support for this mechanism is REQUIRED.

4. Security Considerations

(TODO.)

5. IANA Considerations

This document makes no request to IANA.

6. Acknowledgements

Thanks to Matthias Kovatsch for helpful comments and discussions that have shaped the document.

7. Normative References

[I-D.ietf-core-coap]

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

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-09 (work in progress), July 2013.

[RFC2119]

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

Author's Address

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

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

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

C. Bormann
Universitaet Bremen TZI
Z. Shelby, Ed.
Sensinode
June 27, 2013

Blockwise transfers in CoAP
draft-ietf-core-block-12

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.

The present revision -11 fixes one example and adds the text and examples about the Block/Observe interaction, taken from -observe. It also adds a couple of formatting bugs from the new xml2rfc. The "grand rewrite" is next.

Status of this Memo

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

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

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

This Internet-Draft will expire on December 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	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 Block2 Option	11
2.5. Using the Block1 Option	12
2.6. Combining Blockwise Transfers with the Observe Option	13
2.7. Block2 and Initiative	14
3. Examples	15
3.1. Block2 Examples	15
3.2. Block1 Examples	18
3.3. Combining Block1 and Block2	20
3.4. Combining Observe and Block2	22
4. The Size Options	26
5. HTTP Mapping Considerations	28
6. IANA Considerations	30
7. Security Considerations	31
7.1. Mitigating Resource Exhaustion Attacks	31
7.2. Mitigating Amplification Attacks	32
8. Acknowledgements	33
9. References	34
9.1. Normative References	34
9.2. Informative References	34
Authors' Addresses	35

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. The Content-Format option applies to the body, not to the payload, in particular the boundaries between the blocks may in places that are not whole units in terms of the structure, encoding, or content-coding used by the Content-Format.

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

2.1. The 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 (Block1 or Block2) option:

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

The value of the Block Option is a variable-size (0 to 3 byte) unsigned integer (uint, see Appendix A of [I-D.ietf-core-coap]). This integer value encodes these three fields, see Figure 1. (Due to the CoAP uint encoding rules, when all of NUM, M, and SZX happen to be zero, a zero-byte integer will be sent.)

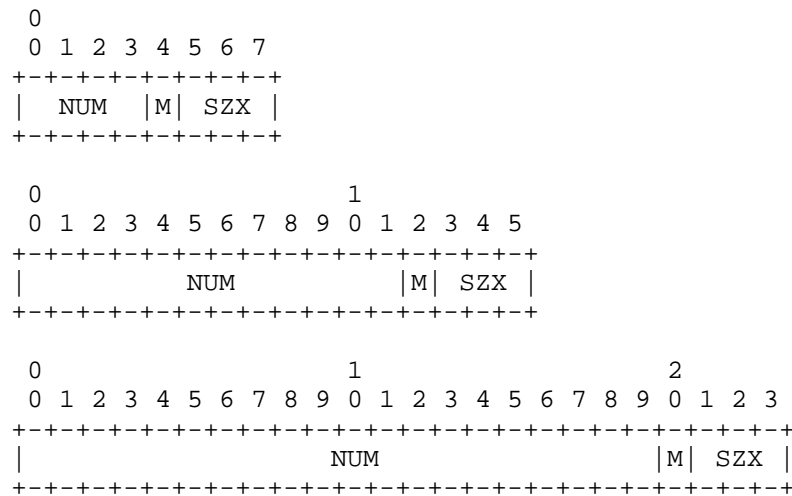


Figure 1: Block option value

The block size is encoded using a three-bit unsigned integer (0 for $2^{**}4$ to 6 for $2^{**}10$ bytes), which we call the "SZX" ("size exponent"); the actual block size is then " $2^{**}(\text{SZX} + 4)$ ". SZX is transferred in the three least significant bits of the option value (i.e., " $\text{val} \& 7$ " where "val" is the value of the option).

The fourth least significant bit, the M or "more" bit (" $\text{val} \& 8$ "), indicates whether more blocks are following or the current block-wise transfer is the last block being transferred.

The option value divided by sixteen (the NUM field) is the sequence number of the block currently being transferred, starting from zero. The current transfer is therefore about the "size" bytes starting at byte " $\text{NUM} \ll (\text{SZX} + 4)$ ".

Implementation note: As an implementation convenience, " $(\text{val} \& \sim 0xF) \ll (\text{val} \& 7)$ ", i.e., the option value with the last 4 bits masked out, shifted to the left by the value of SZX, gives the byte position of the block being transferred.

More specifically, within the option value of a Block1 or Block2 Option, the meaning of the option fields is defined as follows:

NUM: Block Number, indicating the block number being requested or provided. Block number 0 indicates the first block of a body (i.e., starting with the first byte of the body).

M: More Flag (not last block). For descriptive usage, this flag, if unset, indicates that the payload in this message is the last block in the body; when set it indicates that there are one or more additional blocks available. When a Block2 Option is used in a request to retrieve a specific block number ("control usage"), the M bit MUST be sent as zero and ignored on reception. (In a Block1 Option in a response, the M flag is used to indicate atomicity, see below.)

SZX: Block Size. The block size is represented as three-bit unsigned integer indicating the size of a block to the power of two. Thus block size = $2^{(SZX + 4)}$. The allowed values of SZX are 0 to 6, i.e., the minimum block size is $2^{(0+4)} = 16$ and the maximum is $2^{(6+4)} = 1024$. The value 7 for SZX (which would indicate a block size of 2048) is reserved, i.e. MUST NOT be sent and MUST lead to a 4.00 Bad Request response code upon reception in a request.

There is no default value for the Block1 and Block2 Options. Absence of one of these options is equivalent to an option value of 0 with respect to the value of NUM and M that could be given in the option, i.e. it indicates that the current block is the first and only block of the transfer (block number 0, M bit not set). However, in contrast to the explicit value 0, which would indicate an SZX of 0 and thus a size value of 16 bytes, there is no specific explicit size implied by the absence of the option -- the size is left unspecified. (As for any uint, the explicit value 0 is efficiently indicated by a zero-length option; this, therefore, is different in semantics from the absence of the option.)

2.3. Block Options in Requests and Responses

The Block options are used in one of three roles:

- o In descriptive usage, i.e., a Block2 Option in a response (such as a 2.05 response for GET), or a Block1 Option in a request (a PUT or POST):
 - * The NUM field in the option value describes what block number is contained in the payload of this message.
 - * The M bit indicates whether further blocks need to be transferred to complete the transfer of that body.
 - * The block size given by SZX MUST match the size of the payload in bytes, if the M bit is set. (SZX does not govern the payload size if M is unset). For Block2, if the request suggested a larger value of SZX, the next request MUST move SZX

down to the size given in the response. (The effect is that, if the server uses the smaller of (1) its preferred block size and (2) the block size requested, all blocks for a body use the same block size.)

- o A Block2 Option in control usage in a request (e.g., GET):
 - * The NUM field in the Block2 Option gives the block number of the payload that is being requested to be returned in the response.
 - * In this case, the M bit has no function and MUST be set to zero.
 - * The block size given (SZX) suggests a block size (in the case of block number 0) or repeats the block size of previous blocks received (in the case of a non-zero block number).
- o A Block1 Option in control usage in a response (e.g., a 2.xx response for a PUT or POST request):
 - * The NUM field of the Block1 Option indicates what block number is being acknowledged.
 - * If the M bit was set in the request, the server can choose whether to act on each block separately, with no memory, or whether to handle the request for the entire body atomically, or any mix of the two.
 - + If the M bit is also set in the response, it indicates that this response does not carry the final response code to the request, i.e. the server collects further blocks from the same endpoint and plans to implement the request atomically (e.g., acts only upon reception of the last block of payload). In this case, the response MUST NOT carry a Block2 option.
 - + Conversely, if the M bit is unset even though it was set in the request, it indicates the block-wise request was enacted now specifically for this block, and the response carries the final response to this request (and to any previous ones with the M bit set in the response's Block1 Option in this sequence of block-wise transfers); the client is still expected to continue sending further blocks, the request method for which may or may not also be enacted per-block.

- * Finally, the SZX block size given in a control Block1 Option indicates the largest block size preferred by the server for transfers toward the resource that is the same or smaller than the one used in the initial exchange; the client SHOULD use this block size or a smaller one in all further requests in the transfer sequence, even if that means changing the block size (and possibly scaling the block number accordingly) from now on.

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

2.4. Using the Block2 Option

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

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

Once the Block2 Option is used by the requester and a first response has been received with a possibly adjusted block size, all further requests in a single block-wise transfer SHOULD ultimately use the same size, except that there may not be enough content to fill the last block (the one returned with the M bit not set). (Note that the client may start using the Block2 Option in a second request after a first request without a Block2 Option resulted in a Block2 option in the response.) The server SHOULD use the block size indicated in the request option or a smaller size, but the requester MUST take note of the actual block size used in the response it receives to its initial request and proceed to use it in subsequent requests. The server behavior MUST ensure that this client behavior results in the same block size for all responses in a sequence (except for the last one with the M bit not set, and possibly the first one if the initial request did not contain a Block2 Option).

Block-wise transfers can be used to GET resources the representations

of which are entirely static (not changing over time at all, such as in a schema describing a device), or for dynamically changing resources. In the latter case, the Block2 Option SHOULD be used in conjunction with the ETag Option, to ensure that the blocks being reassembled are from the same version of the representation: The server SHOULD include an ETag option in each response. If an ETag option is available, the client's reassembler, when reassembling the representation from the blocks being exchanged, MUST compare ETag Options. If the ETag Options do not match in a GET transfer, the requester has the option of attempting to retrieve fresh values for the blocks it retrieved first. To minimize the resulting inefficiency, the server MAY cache the current value of a representation for an ongoing sequence of requests. (The server may identify the sequence by the combination of the requesting end-point and the URI being the same in each block-wise request.) Note well that this specification makes no requirement for the server to establish any state; however, servers that offer quickly changing resources may thereby make it impossible for a client to ever retrieve a consistent set of blocks.

2.5. Using the Block1 Option

In a request with a request payload (e.g., PUT or POST), the Block1 Option refers to the payload in the request (descriptive usage).

In response to a request with a payload (e.g., a PUT or POST transfer), the block size given in the Block1 Option indicates the block size preference of the server for this resource (control usage). Obviously, at this point the first block has already been transferred by the client without benefit of this knowledge. Still, the client SHOULD heed the preference 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 its Block1 option than requested is a hint to try a smaller SZX.)

The Block1 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise request payload transfer (e.g., PUT or POST) operations to the same resource. Starting a new block-wise sequence of requests to the same resource (before an old sequence from the same endpoint was finished) simply overwrites the context the server may still be keeping. (This is probably exactly what one wants in this case - the client may simply have restarted and lost its knowledge of the previous sequence.)

2.6. Combining Blockwise Transfers with the Observe Option

The Observe Option provides a way for a client to be notified about changes over time of a resource [I-D.ietf-core-observe]. Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. The following rules apply to the combination of blockwise transfers with notifications.

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 and apply it as a maximum size to all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the 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 Section 3.4 for examples.

2.7. Block2 and Initiative

In a basic block-wise GET request, it is the job of the client to initiate each further block transfer. We say that the "initiative" is with the client. If no buffering of a snapshot of the resource is required, the server can stay entirely stateless. This is particularly useful for very simple servers for which all resources that are big enough to merit block-wise transfer are static (such as the links in `"/.well-known/core"`).

However, when Block2 is combined with Observe or Block1, this simple approach no longer works very well. Therefore, the presence of an Observe or Block1 option in combination with a Block2 option is said to reverse the initiative: From then on, it is the job of the server to provide additional responses that complete the blockwise transfer of the notification (Observe) or response to a block-wise PUT or POST transfer (Block1). As all these additional responses are in response to the single request that caused them, they all carry the token of this request: The GET with an Observe option, or the PUT/POST with a Block1 option.

(For the request side of block-wise transfers that use the Block1 option, it is of course always the initiative of the client to send the next block - which is quite natural, as the client has to generate them and therefore knows when it is time to send the next block.)

3. Examples

This section gives a number of short examples with message flows for a block-wise GET, and for a PUT or POST. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and examples for the operation of the block size negotiation.

In all these examples, a Block option is shown in a decomposed way indicating the kind of Block option (1 or 2) followed by a colon, and then the block number (NUM), more bit (M), and block size exponent ($2^{*(SZX+4)}$) separated by slashes. E.g., a Block2 Option value of 33 would be shown as 2:2/0/32), or a Block1 Option value of 59 would be shown as 1:3/1/128.

3.1. Block2 Examples

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

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

Figure 2: Simple blockwise GET

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

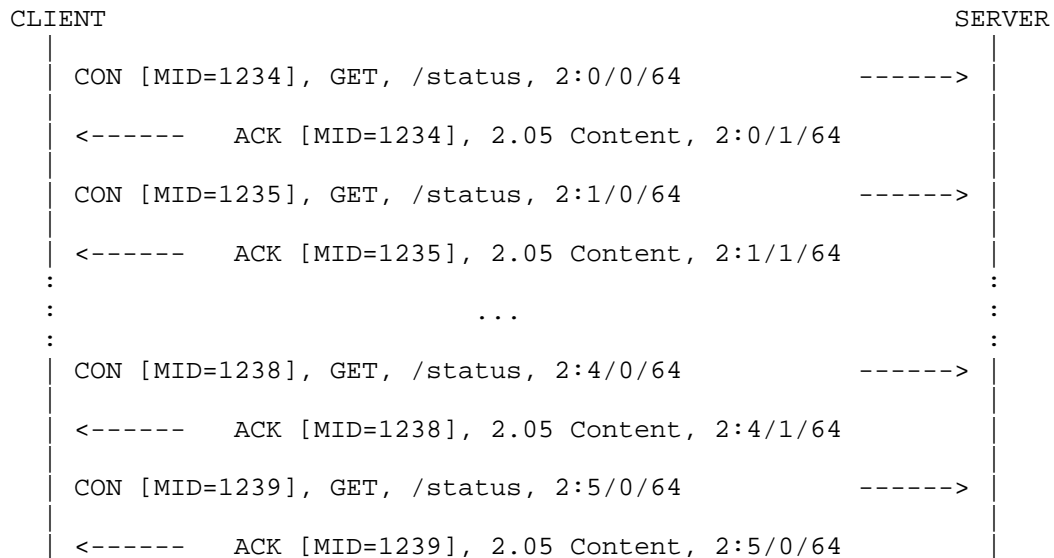


Figure 3: Blockwise GET with early negotiation

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

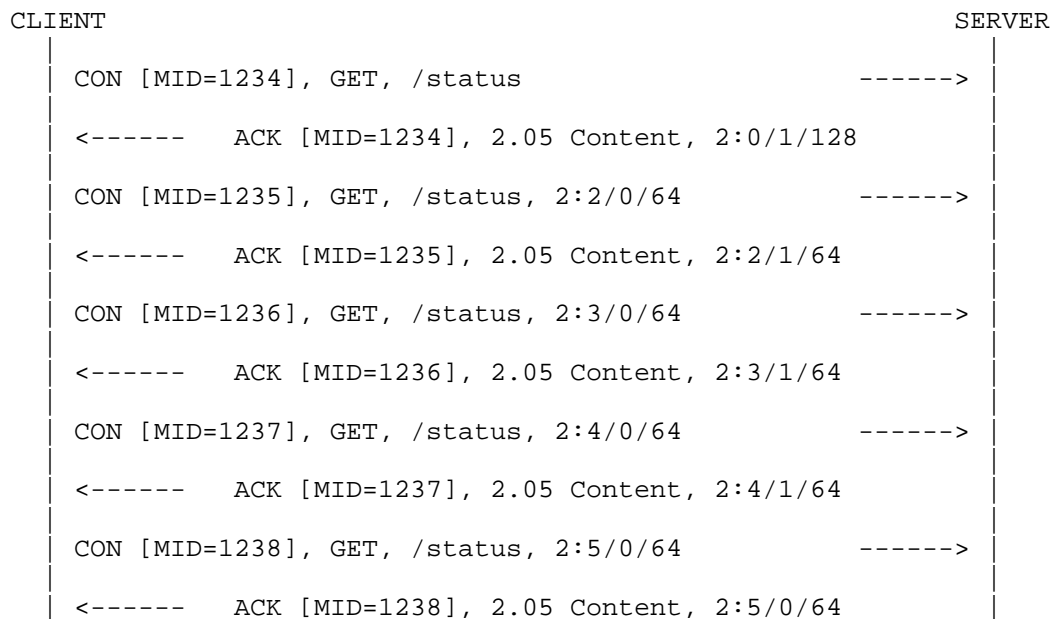


Figure 4: Blockwise GET with late negotiation

In all these (and the following) cases, retransmissions are handled by the CoAP message exchange layer, so they don't influence the block operations (Figure 5, Figure 6).

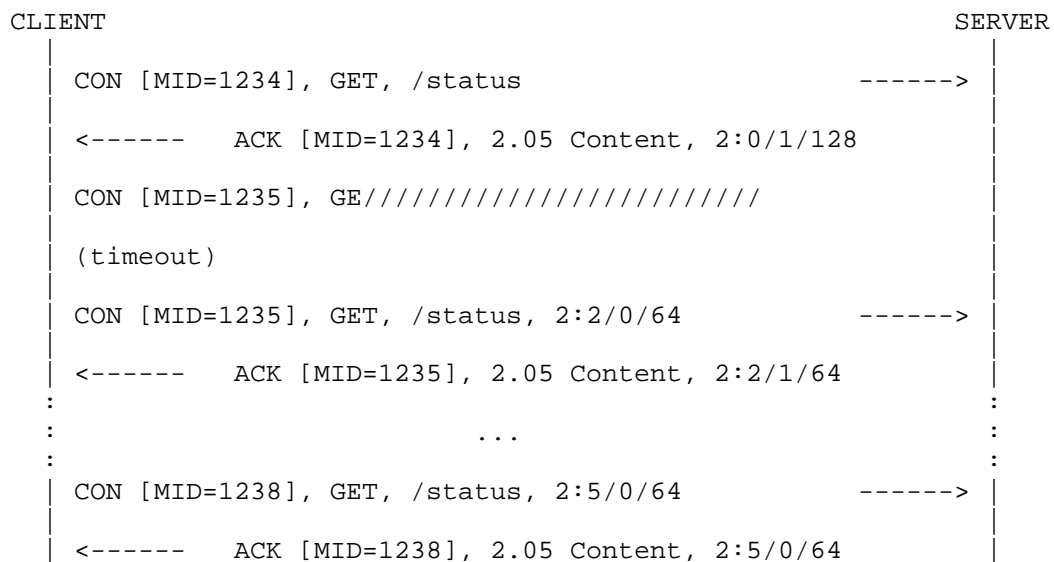


Figure 5: Blockwise GET with late negotiation and lost CON

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

3.2. Block1 Examples

The following examples demonstrate a PUT exchange; a POST exchange looks the same, with different requirements on atomicity/idempotence. Note that, similar to GET, the responses to the requests that have a more bit in the request Block1 Option are provisional; only the final response tells the client that the PUT succeeded.

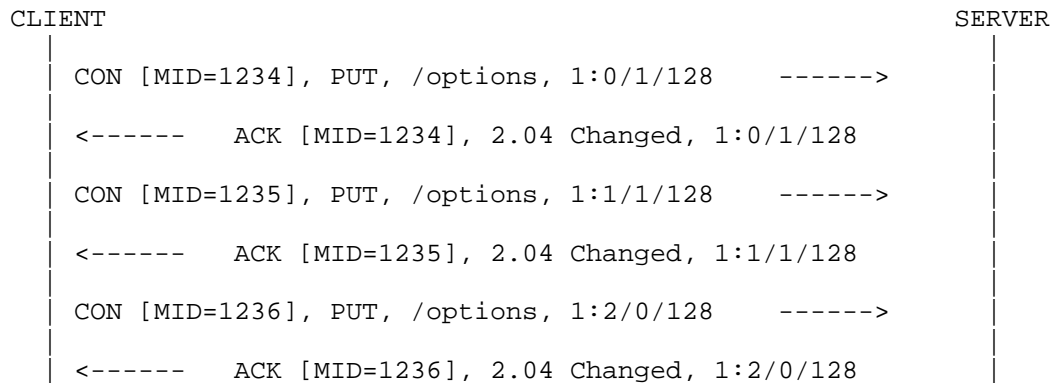


Figure 7: Simple atomic blockwise PUT

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

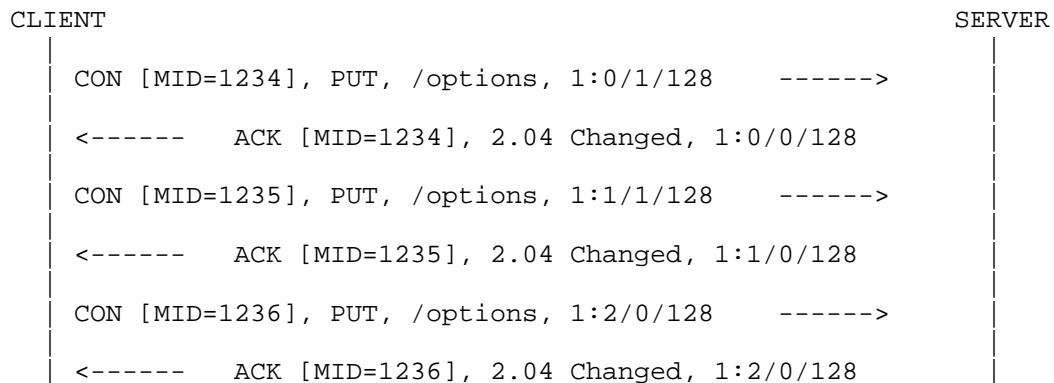


Figure 8: Simple stateless blockwise PUT

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

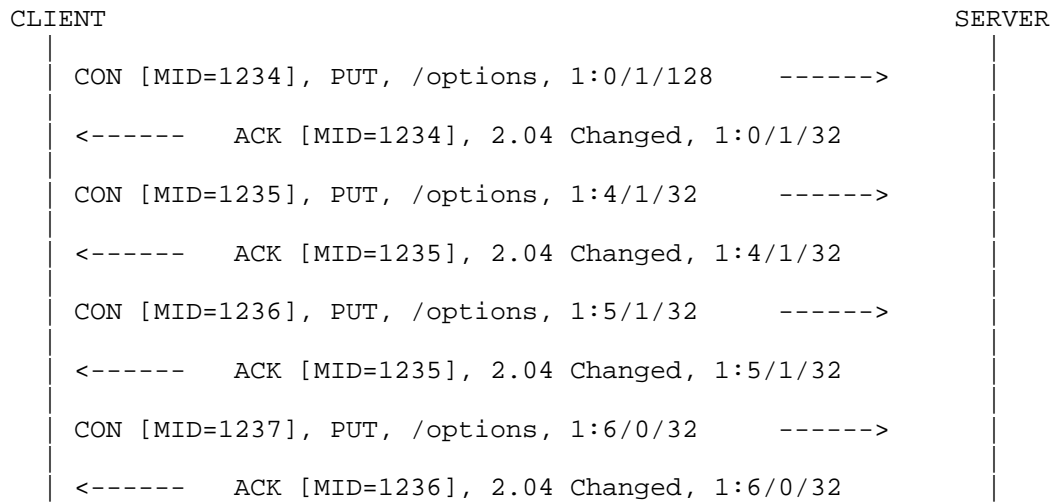


Figure 9: Simple atomic blockwise PUT with negotiation

3.3. Combining Block1 and Block2

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

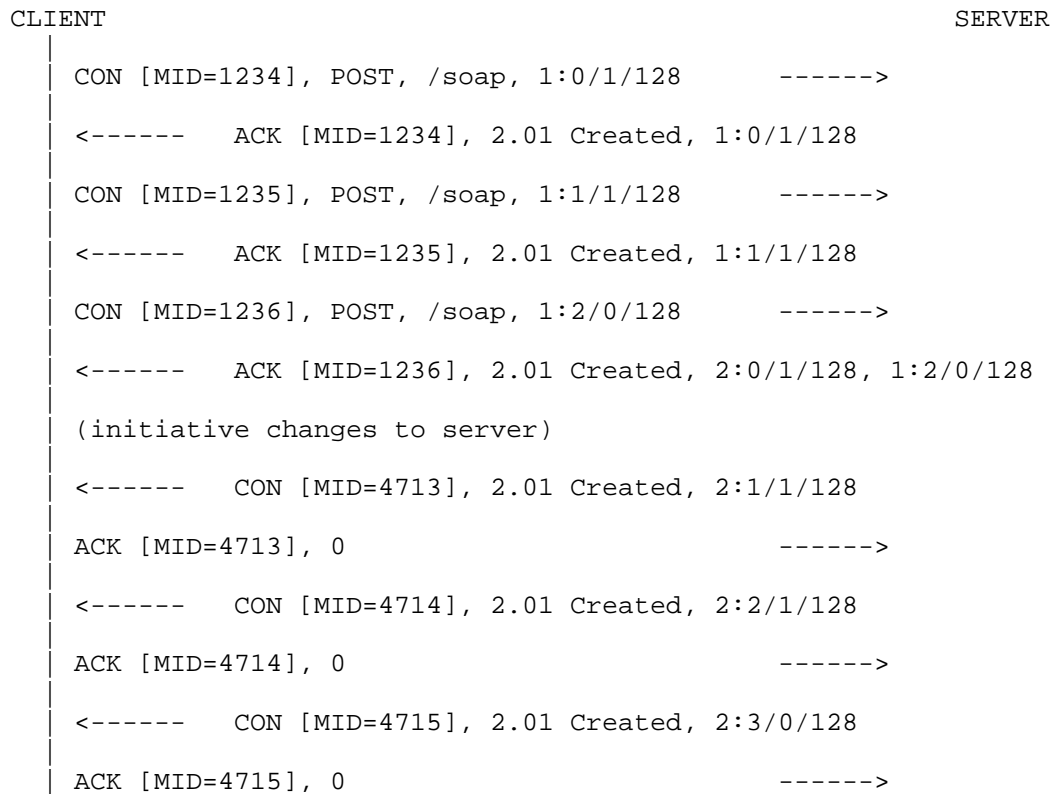


Figure 10: Atomic blockwise POST with separate blockwise response

This model does provide for early negotiation input to the Block2 blockwise transfer, as shown below. (However, there is no way to provide late negotiation with server initiative.)

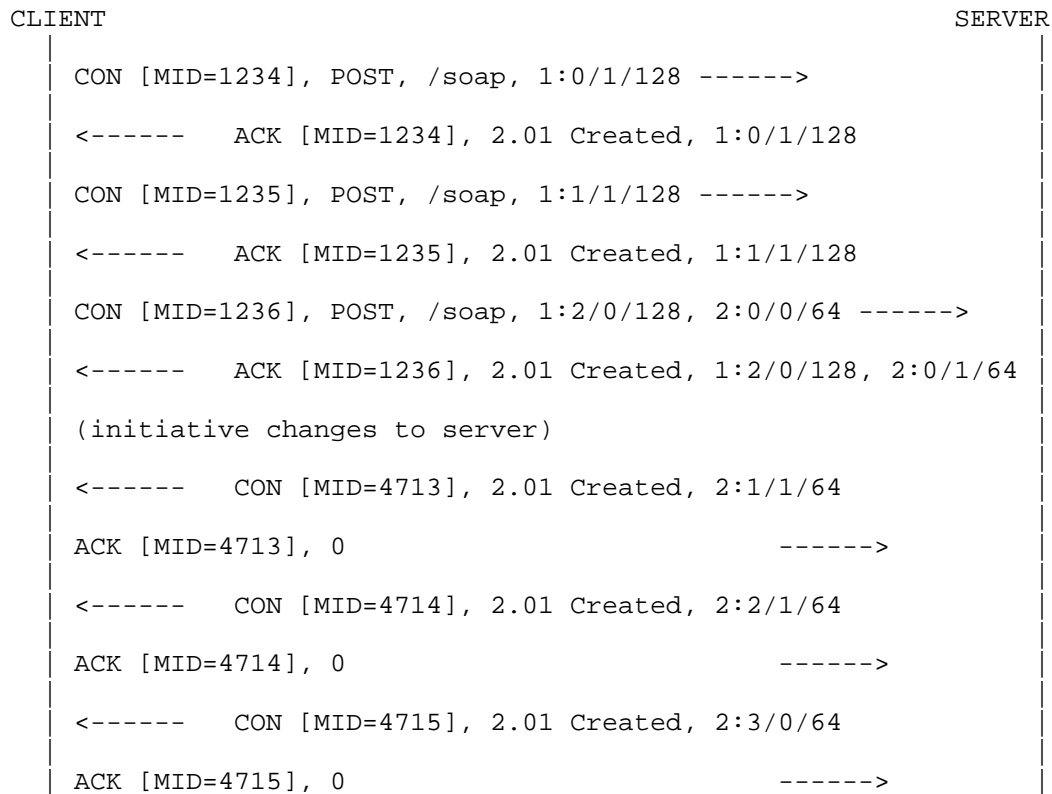


Figure 11: Atomic blockwise POST with separate blockwise response,
early negotiation

3.4. Combining Observe and Block2

In the following example, the server sends two notifications of two blocks each. The first notification is a direct response to the GET request; the first block therefore can be sent piggy-backed in the ACK. The Observe Option indicates that the initiative has switched to the server.

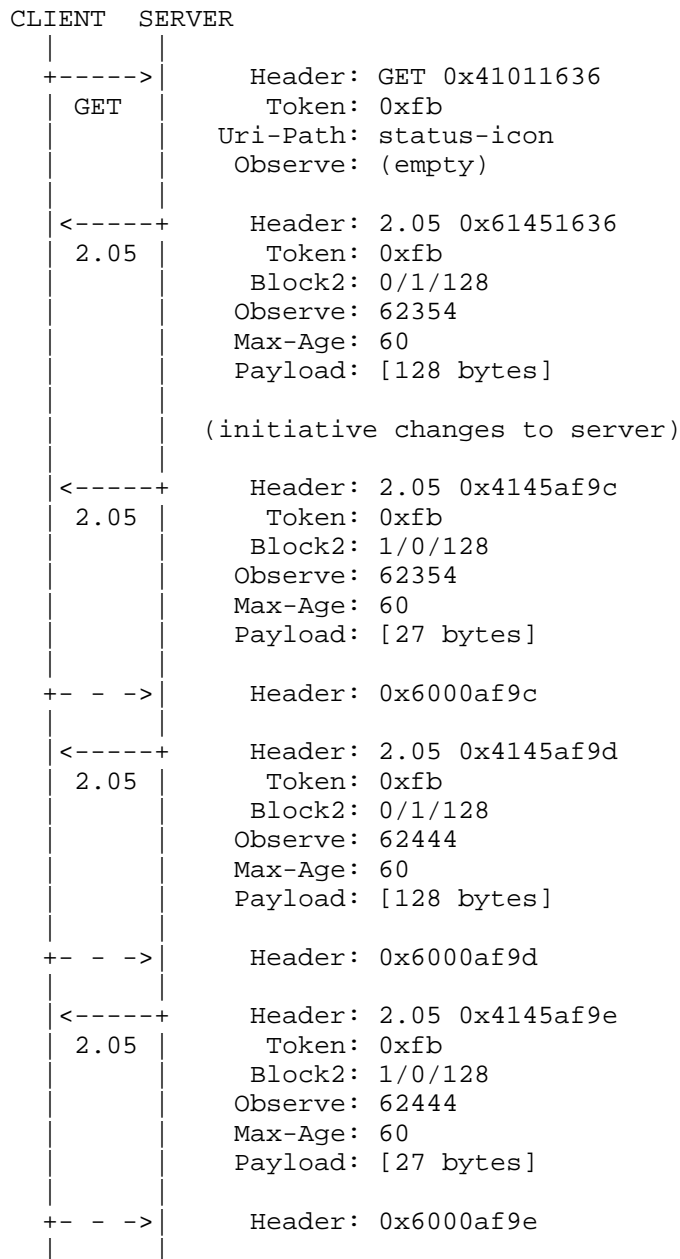


Figure 12: Observe sequence with blockwise response

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

CLIENT	SERVER
+----->	Header: GET 0x41011636
GET	Token: 0xfb
	Uri-Path: status-icon
	Observe: (empty)
	Block2: 0/0/64
<-----+	Header: 2.05 0x61451636
2.05	Token: 0xfb
	Block2: 0/1/64
	Observe: 62354
	Max-Age: 60
	Payload: [64 bytes]
	(initiative changes to server)
<-----+	Header: 2.05 0x4145af9c
2.05	Token: 0xfb
	Block2: 1/1/64
	Observe: 62354
	Max-Age: 60
	Payload: [64 bytes]
+ - - ->	Header: 0x6000af9c
<-----+	Header: 2.05 0x4145af9d
2.05	Token: 0xfb
	Block2: 2/0/64
	Observe: 62354
	Max-Age: 60
	Payload: [27 bytes]
+ - - ->	Header: 0x6000af9d
<-----+	Header: 2.05 0x4145af9e
2.05	Token: 0xfb
	Block2: 0/1/64
	Observe: 62444
	Max-Age: 60
	Payload: [128 bytes]
+ - - ->	Header: 0x6000af9e
<-----+	Header: 2.05 0x4145af9f
2.05	Token: 0xfb
	Block2: 1/1/64
	Observe: 62444

```
|
|
|
+- - ->|
|
|<-----+
| 2.05   |
|
|
+- - ->|
|
```

Max-Age: 60
Payload: [128 bytes]

Header: 0x6000af9f

Header: 2.05 0x4145afa0
Token: 0xfb
Block2: 2/0/64
Observe: 62444
Max-Age: 60
Payload: [27 bytes]

Header: 0x6000afa0

Figure 13: Observe sequence with early negotiation

4. The Size Options

In many cases when transferring a large resource representation block by block, it is advantageous to know the total size early in the process. Some indication may be available from the maximum size estimate attribute "sz" provided in a resource description [RFC6690]. However, the size may vary dynamically, so a more up-to-date indication may be useful.

This specification defines two CoAP Options, Size1 for indicating the size of the representation transferred in requests, and Size2 for indicating the size of the representation transferred in responses.

The Size2 Option may be used for two purposes:

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

Similarly, the Size1 Option may be used for two purposes:

- o in a request carrying a Block1 Option, to indicate the current estimate the client has of the total size of the resource representation, measured in bytes ("size indication").
- o in a 4.13 response, to indicate the maximum size that would have been acceptable [I-D.ietf-core-coap], measured in bytes.

Apart from conveying/asking for size information, the Size options have no other effect on the processing of the request or response. If the client wants to minimize the size of the payload in the resulting response, it should add a Block2 option to the request with a small block size (e.g., setting SZX=0).

The Size Options are "elective", i.e., a client MUST be prepared for the server to ignore the size estimate request. The Size Options MUST NOT occur more than once.

Type	C	U	N	R	Name	Format	Length	Default
60			x		Size1	uint	0-4 B	(none)
28			x		Size2	uint	0-4 B	(none)

Table 2: Size Option Numbers

Implementation Notes:

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

5. HTTP Mapping Considerations

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

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

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

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

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

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

6. IANA Considerations

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

Number	Name	Reference
23	Block2	[RFCXXXX]
27	Block1	[RFCXXXX]
28	Size2	[RFCXXXX]
60	Size1	[RFCXXXX]

Table 3: CoAP Option Numbers

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

Code	Description	Reference
136	4.08 Request Entity Incomplete	[RFCXXXX]

Table 4: CoAP Response Codes

7. Security Considerations

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

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

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

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

7.1. Mitigating Resource Exhaustion Attacks

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

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

Performing segmentation at the application layer is almost always better in this respect than at the transport layer or lower (IP fragmentation, adaptation layer fragmentation), e.g. because there is application layer semantics that can be used for mitigation or because lower layers provide security associations that can prevent attacks. However, it is less common to apply timeouts and keepalive mechanisms at the application layer than at lower layers. Servers MAY want to clean up accumulated state by timing it out (cf. response code 4.08), and clients SHOULD be prepared to run blockwise transfers in an expedient way to minimize the likelihood of running into such a timeout.

7.2. Mitigating Amplification Attacks

[I-D.ietf-core-coap] discusses the susceptibility of CoAP end-points for use in amplification attacks.

A CoAP server can reduce the amount of amplification it provides to an attacker by offering large resource representations only in relatively small blocks. With this, e.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

8. Acknowledgements

Much of the content of this draft is the result of discussions with the [I-D.ietf-core-coap] authors, and via many CoRE WG discussions.

Charles Palmer provided extensive editorial comments to a previous version of this draft, some of which the authors hope to have covered in this version. Esko Dijk reviewed a more recent version, leading to a number of further editorial improvements 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. Klaus Hartke wrote some of the text describing the interaction of Block2 with Observe.

9. References

9.1. Normative References

- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-17 (work in progress), May 2013.
- [I-D.ietf-core-observe] Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

9.2. Informative References

- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

Authors' Addresses

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

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

Zach Shelby (editor)
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2013

Z. Shelby
Sensinode
K. Hartke
C. Bormann
Universitaet Bremen TZI
June 28, 2013

Constrained Application Protocol (CoAP)
draft-ietf-core-coap-18

Abstract

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as 6LoWPAN often have high packet error rates and a typical throughput of 10s of kbit/s. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 30, 2013.

Copyright Notice

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

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

Table of Contents

1. Introduction	5
1.1. Features	5
1.2. Terminology	6
2. Constrained Application Protocol	9
2.1. Messaging Model	10
2.2. Request/Response Model	12
2.3. Intermediaries and Caching	14
2.4. Resource Discovery	15
3. Message Format	15
3.1. Option Format	17
3.2. Option Value Formats	19
4. Message Transmission	20
4.1. Messages and Endpoints	20
4.2. Messages Transmitted Reliably	20
4.3. Messages Transmitted Without Reliability	22
4.4. Message Correlation	23
4.5. Message Deduplication	24
4.6. Message Size	24
4.7. Congestion Control	25
4.8. Transmission Parameters	26
4.8.1. Changing The Parameters	27
4.8.2. Time Values derived from Transmission Parameters	28
5. Request/Response Semantics	30
5.1. Requests	30
5.2. Responses	30
5.2.1. Piggy-backed	32
5.2.2. Separate	32
5.2.3. Non-confirmable	33
5.3. Request/Response Matching	33
5.3.1. Token	34
5.3.2. Request/Response Matching Rules	35

5.4.	Options	35
5.4.1.	Critical/Elective	36
5.4.2.	Proxy Unsafe/Safe-to-Forward and NoCacheKey	37
5.4.3.	Length	38
5.4.4.	Default Values	38
5.4.5.	Repeatable Options	38
5.4.6.	Option Numbers	38
5.5.	Payloads and Representations	39
5.5.1.	Representation	39
5.5.2.	Diagnostic Payload	40
5.5.3.	Selected Representation	40
5.5.4.	Content Negotiation	40
5.6.	Caching	41
5.6.1.	Freshness Model	42
5.6.2.	Validation Model	42
5.7.	Proxying	43
5.7.1.	Proxy Operation	43
5.7.2.	Forward-Proxies	45
5.7.3.	Reverse-Proxies	45
5.8.	Method Definitions	46
5.8.1.	GET	46
5.8.2.	POST	46
5.8.3.	PUT	46
5.8.4.	DELETE	47
5.9.	Response Code Definitions	47
5.9.1.	Success 2.xx	47
5.9.2.	Client Error 4.xx	49
5.9.3.	Server Error 5.xx	50
5.10.	Option Definitions	51
5.10.1.	Uri-Host, Uri-Port, Uri-Path and Uri-Query	52
5.10.2.	Proxy-Uri and Proxy-Scheme	53
5.10.3.	Content-Format	53
5.10.4.	Accept	54
5.10.5.	Max-Age	54
5.10.6.	ETag	54
5.10.7.	Location-Path and Location-Query	55
5.10.8.	Conditional Request Options	56
5.10.9.	Size1 Option	57
6.	CoAP URIs	57
6.1.	coap URI Scheme	58
6.2.	coaps URI Scheme	59
6.3.	Normalization and Comparison Rules	59
6.4.	Decomposing URIs into Options	60
6.5.	Composing URIs from Options	61
7.	Discovery	62
7.1.	Service Discovery	62
7.2.	Resource Discovery	63
7.2.1.	'ct' Attribute	63

8. Multicast CoAP	64
8.1. Messaging Layer	64
8.2. Request/Response Layer	65
8.2.1. Caching	66
8.2.2. Proxying	66
9. Securing CoAP	66
9.1. DTLS-secured CoAP	68
9.1.1. Messaging Layer	69
9.1.2. Request/Response Layer	69
9.1.3. Endpoint Identity	70
10. Cross-Protocol Proxying between CoAP and HTTP	73
10.1. CoAP-HTTP Proxying	74
10.1.1. GET	74
10.1.2. PUT	75
10.1.3. DELETE	75
10.1.4. POST	75
10.2. HTTP-CoAP Proxying	76
10.2.1. OPTIONS and TRACE	76
10.2.2. GET	76
10.2.3. HEAD	77
10.2.4. POST	77
10.2.5. PUT	78
10.2.6. DELETE	78
10.2.7. CONNECT	78
11. Security Considerations	78
11.1. Protocol Parsing, Processing URIs	78
11.2. Proxying and Caching	79
11.3. Risk of amplification	80
11.4. IP Address Spoofing Attacks	81
11.5. Cross-Protocol Attacks	82
11.6. Constrained node considerations	84
12. IANA Considerations	84
12.1. CoAP Code Registries	84
12.1.1. Method Codes	85
12.1.2. Response Codes	85
12.2. Option Number Registry	87
12.3. Content-Format Registry	89
12.4. URI Scheme Registration	90
12.5. Secure URI Scheme Registration	91
12.6. Service Name and Port Number Registration	92
12.7. Secure Service Name and Port Number Registration	93
12.8. Multicast Address Registration	94
13. Acknowledgements	94
14. References	95
14.1. Normative References	95
14.2. Informative References	97
Appendix A. Examples	100
Appendix B. URI Examples	105

Appendix C. Changelog	107
Authors' Addresses	117

1. Introduction

The use of web services (web APIs) on the Internet has become ubiquitous in most applications, and depends on the fundamental Representational State Transfer [REST] architecture of the web.

The Constrained RESTful Environments (CoRE) work aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN, [RFC4944]). Constrained networks such as 6LoWPAN support the fragmentation of IPv6 packets into small link-layer frames, however incurring significant reduction in packet delivery probability. One design goal of CoAP has been to keep message overhead small, thus limiting the need for fragmentation.

One of the main goals of CoAP is to design a generic web protocol for the special requirements of this constrained environment, especially considering energy, building automation and other machine-to-machine (M2M) applications. The goal of CoAP is not to blindly compress HTTP [RFC2616], but rather to realize a subset of REST common with HTTP but optimized for M2M applications. Although CoAP could be used for refashioning simple HTTP interfaces into a more compact protocol, it more importantly also offers features for M2M such as built-in discovery, multicast support and asynchronous message exchanges.

This document specifies the Constrained Application Protocol (CoAP), which easily translates to HTTP for integration with the existing web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments and M2M applications.

1.1. Features

CoAP has the following main features:

- o Constrained web protocol fulfilling M2M requirements.
- o UDP [RFC0768] binding with optional reliability supporting unicast and multicast requests.
- o Asynchronous message exchanges.
- o Low header overhead and parsing complexity.
- o URI and Content-type support.

- o Simple proxy and caching capabilities.
- o A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.
- o Security binding to Datagram Transport Layer Security (DTLS) [RFC6347].

1.2. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC2616], including "resource", "representation", "cache", and "fresh". In addition, this specification defines the following terminology:

Endpoint

An entity participating in the CoAP protocol. Colloquially, an endpoint lives on a "Node", although "Host" would be more consistent with Internet standards usage, and is further identified by transport layer multiplexing information that can include a UDP port number and a security association (Section 4.1).

Sender

The originating endpoint of a message. When the aspect of identification of the specific sender is in focus, also "source endpoint".

Recipient

The destination endpoint of a message. When the aspect of identification of the specific recipient is in focus, also "destination endpoint".

Client

The originating endpoint of a request; the destination endpoint of a response.

Server

The destination endpoint of a request; the originating endpoint of a response.

Origin Server

The server on which a given resource resides or is to be created.

Intermediary

A CoAP endpoint that acts both as a server and as a client towards (possibly via further intermediaries) an origin server. A common form of an intermediary is a proxy; several classes of such proxies are discussed in this specification.

Proxy

An intermediary that mainly is concerned with forwarding requests and relaying back responses, possibly performing caching, namespace translation, or protocol translation in the process. As opposed to intermediaries in the general sense, proxies generally do not implement specific application semantics. Based on the position in the overall structure of the request forwarding, there are two common forms of proxy: forward-proxy and reverse-proxy. In some cases, a single endpoint might act as an origin server, forward-proxy, or reverse-proxy, switching behavior based on the nature of each request.

Forward-Proxy

A "forward-proxy" is an endpoint selected by a client, usually via local configuration rules, to perform requests on behalf of the client, doing any necessary translations. Some translations are minimal, such as for proxy requests for "coap" URIs, whereas other requests might require translation to and from entirely different application-layer protocols.

Reverse-Proxy

A "reverse-proxy" is an endpoint that stands in for one or more other server(s) and satisfies requests on behalf of these, doing any necessary translations. Unlike a forward-proxy, the client may not be aware that it is communicating with a reverse-proxy; a reverse-proxy receives requests as if it was the origin server for the target resource.

CoAP-to-CoAP Proxy

A proxy that maps from a CoAP request to a CoAP request, i.e. uses the CoAP protocol both on the server and the client side. Contrast to cross-proxy.

Cross-Proxy

A cross-protocol proxy, or "cross-proxy" for short, is a proxy that translates between different protocols, such as a CoAP-to-HTTP proxy or an HTTP-to-CoAP proxy. While this specification makes very specific demands of CoAP-to-CoAP proxies, there is more variation possible in cross-proxies.

Confirmable Message

Some messages require an acknowledgement. These messages are called "Confirmable". When no packets are lost, each Confirmable message elicits exactly one return message of type Acknowledgement or type Reset.

Non-confirmable Message

Some other messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor.

Acknowledgement Message

An Acknowledgement message acknowledges that a specific Confirmable message arrived. By itself, an Acknowledgement message does not indicate success or failure of any request encapsulated in the Confirmable message, but the Acknowledgement message may also carry a Piggy-Backed Response (q.v.).

Reset Message

A Reset message indicates that a specific message (Confirmable or Non-confirmable) was received, but some context is missing to properly process it. This condition is usually caused when the receiving node has rebooted and has forgotten some state that would be required to interpret the message. Provoking a Reset message (e.g., by sending an Empty Confirmable message) is also useful as an inexpensive check of the liveness of an endpoint ("CoAP ping").

Piggy-backed Response

A Piggy-backed Response is included right in a CoAP Acknowledgement (ACK) message that is sent to acknowledge receipt of the Request for this Response (Section 5.2.1).

Separate Response

When a Confirmable message carrying a Request is acknowledged with an Empty message (e.g., because the server doesn't have the answer right away), a Separate Response is sent in a separate message exchange (Section 5.2.2).

Empty Message

A message with a Code of 0.00; neither a request nor a response. An Empty message only contains the four-byte header.

Critical Option

An option that would need to be understood by the endpoint ultimately receiving the message in order to properly process the message (Section 5.4.1). Note that the implementation of critical options is, as the name "Option" implies, generally optional:

unsupported critical options lead to an error response or summary rejection of the message.

Elective Option

An option that is intended to be ignored by an endpoint that does not understand it. Processing the message even without understanding the option is acceptable (Section 5.4.1).

Unsafe Option

An option that would need to be understood by a proxy receiving the message in order to safely forward the message (Section 5.4.2). Not every critical option is an unsafe option.

Safe-to-Forward Option

An option that is intended to be safe for forwarding by a proxy that does not understand it. Forwarding the message even without understanding the option is acceptable (Section 5.4.2).

Resource Discovery

The process where a CoAP client queries a server for its list of hosted resources (i.e., links, Section 7).

Content-Format

The combination of an Internet media type, potentially with specific parameters given, and a content-coding (which is often the identity content-coding), identified by a numeric identifier defined by the CoAP Content-Format Registry. When the focus is less on the numeric identifier than on the combination of these characteristics of a resource representation, this is also called "representation format".

Additional terminology for constrained nodes and constrained node networks can be found in [I-D.ietf-lwig-terminology].

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

All multi-byte integers in this protocol are interpreted in network byte order.

Where arithmetic is used, this specification uses the notation familiar from the programming language C, except that the operator "***" stands for exponentiation.

2. Constrained Application Protocol

The interaction model of CoAP is similar to the client/server model of HTTP. However, machine-to-machine interactions typically result

in a CoAP implementation acting in both client and server roles. A CoAP request is equivalent to that of HTTP, and is sent by a client to request an action (using a method code) on a resource (identified by a URI) on a server. The server then sends a response with a response code; this response may include a resource representation.

Unlike HTTP, CoAP deals with these interchanges asynchronously over a datagram-oriented transport such as UDP. This is done logically using a layer of messages that supports optional reliability (with exponential back-off). CoAP defines four types of messages: Confirmable, Non-confirmable, Acknowledgement, Reset; method codes and response codes included in some of these messages make them carry requests or responses. The basic exchanges of the four types of messages are somewhat orthogonal to the request/response interactions; requests can be carried in Confirmable and Non-confirmable messages, and responses can be carried in these as well as piggy-backed in Acknowledgement messages.

One could think of CoAP logically as using a two-layer approach, a CoAP messaging layer used to deal with UDP and the asynchronous nature of the interactions, and the request/response interactions using Method and Response codes (see Figure 1). CoAP is however a single protocol, with messaging and request/response just features of the CoAP header.

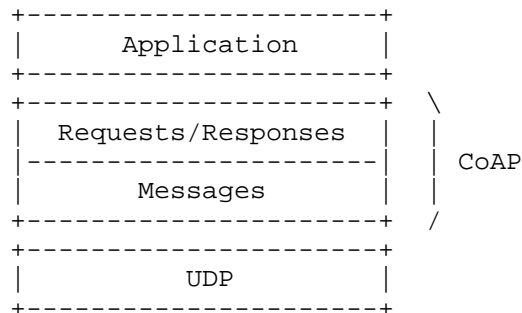


Figure 1: Abstract layering of CoAP

2.1. Messaging Model

The CoAP messaging model is based on the exchange of messages over UDP between endpoints.

CoAP uses a short fixed-length binary header (4 bytes) that may be followed by compact binary options and a payload. This message format is shared by requests and responses. The CoAP message format is specified in Section 3. Each message contains a Message ID used

to detect duplicates and for optional reliability. (The Message ID is compact; its 16-bit size enables up to about 250 messages per second from one endpoint to another with default protocol parameters.)

Reliability is provided by marking a message as Confirmable (CON). A Confirmable message is retransmitted using a default timeout and exponential back-off between retransmissions, until the recipient sends an Acknowledgement message (ACK) with the same Message ID (in this example, 0x7d34) from the corresponding endpoint; see Figure 2. When a recipient is not at all able to process a Confirmable message (i.e., not even able to provide a suitable error response), it replies with a Reset message (RST) instead of an Acknowledgement (ACK).

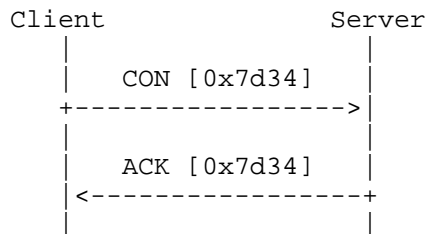


Figure 2: Reliable message transmission

A message that does not require reliable transmission, for example each single measurement out of a stream of sensor data, can be sent as a Non-confirmable message (NON). These are not acknowledged, but still have a Message ID for duplicate detection (in this example, 0x01a0); see Figure 3. When a recipient is not able to process a Non-confirmable message, it may reply with a Reset message (RST).

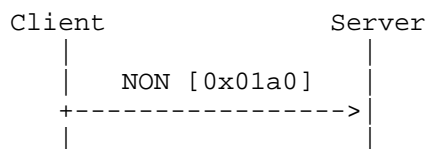


Figure 3: Unreliable message transmission

See Section 4 for details of CoAP messages.

As CoAP runs over UDP, it also supports the use of multicast IP destination addresses, enabling multicast CoAP requests. Section 8 discusses the proper use of CoAP messages with multicast addresses and precautions for avoiding response congestion.

Several security modes are defined for CoAP in Section 9 ranging from no security to certificate-based security. This document specifies a binding to DTLS for securing the protocol; the use of IPsec with CoAP is discussed in [I-D.bormann-core-ipsec-for-coap].

2.2. Request/Response Model

CoAP request and response semantics are carried in CoAP messages, which include either a Method code or Response code, respectively. Optional (or default) request and response information, such as the URI and payload media type are carried as CoAP options. A Token is used to match responses to requests independently from the underlying messages (Section 5.3). (Note that the Token is a concept separate from the Message ID.)

A request is carried in a Confirmable (CON) or Non-confirmable (NON) message, and if immediately available, the response to a request carried in a Confirmable message is carried in the resulting Acknowledgement (ACK) message. This is called a piggy-backed response, detailed in Section 5.2.1. (There is no need for separately acknowledging a piggy-backed response, as the client will retransmit the request if the Acknowledgement message carrying the piggy-backed response is lost.) Two examples for a basic GET request with piggy-backed response are shown in Figure 4, one successful, one resulting in a 4.04 (Not Found) response.

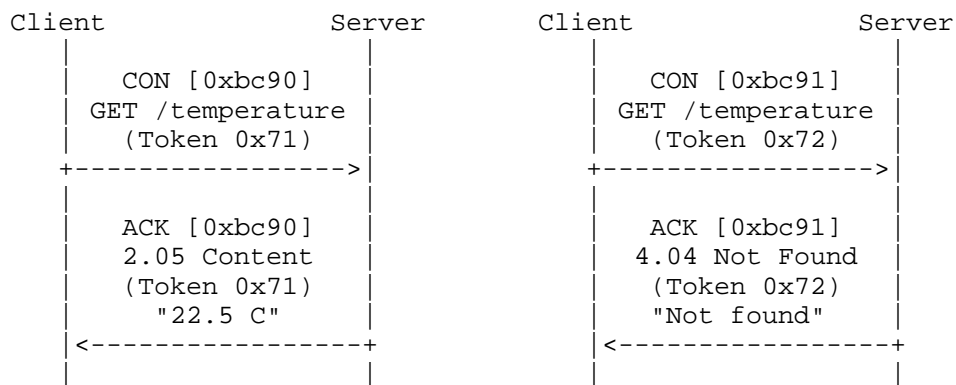


Figure 4: Two GET requests with piggy-backed responses

If the server is not able to respond immediately to a request carried in a Confirmable message, it simply responds with an Empty Acknowledgement message so that the client can stop retransmitting the request. When the response is ready, the server sends it in a new Confirmable message (which then in turn needs to be acknowledged by the client). This is called a separate response, as illustrated in Figure 5 and described in more detail in Section 5.2.2.

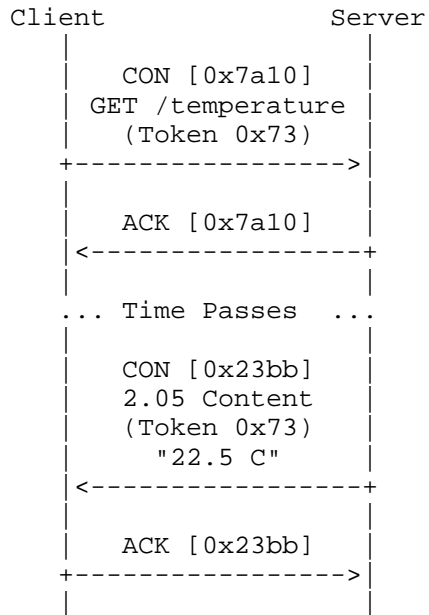
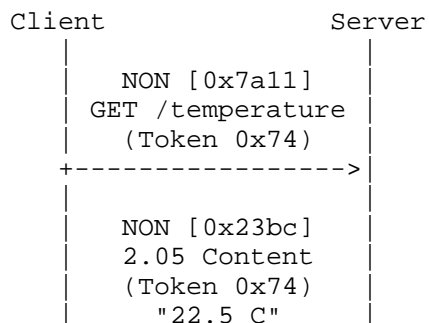


Figure 5: A GET request with a separate response

If a request is sent in a Non-confirmable message, then the response is sent using a new Non-confirmable message, although the server may instead send a Confirmable message. This type of exchange is illustrated in Figure 6.



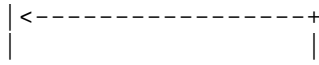


Figure 6: A NON request and response

CoAP makes use of GET, PUT, POST and DELETE methods in a similar manner to HTTP, with the semantics specified in Section 5.8. (Note that the detailed semantics of CoAP methods are "almost, but not entirely unlike" [HHGTTG] those of HTTP methods: Intuition taken from HTTP experience generally does apply well, but there are enough differences that make it worthwhile to actually read the present specification.)

Methods beyond the basic four can be added to CoAP in separate specifications. New methods do not necessarily have to use requests and responses in pairs. Even for existing methods, a single request may yield multiple responses, e.g. for a multicast request (Section 8) or with the Observe option [I-D.ietf-core-observe].

URI support in a server is simplified as the client already parses the URI and splits it into host, port, path and query components, making use of default values for efficiency. Response codes relate to a small subset of HTTP response codes with a few CoAP specific codes added, as defined in Section 5.9.

2.3. Intermediaries and Caching

The protocol supports the caching of responses in order to efficiently fulfill requests. Simple caching is enabled using freshness and validity information carried with CoAP responses. A cache could be located in an endpoint or an intermediary. Caching functionality is specified in Section 5.6.

Proxying is useful in constrained networks for several reasons, including network traffic limiting, to improve performance, to access resources of sleeping devices or for security reasons. The proxying of requests on behalf of another CoAP endpoint is supported in the protocol. When using a proxy, the URI of the resource to request is included in the request, while the destination IP address is set to the address of the proxy. See Section 5.7 for more information on proxy functionality.

As CoAP was designed according to the REST architecture [REST] and thus exhibits functionality similar to that of the HTTP protocol, it is quite straightforward to map from CoAP to HTTP and from HTTP to CoAP. Such a mapping may be used to realize an HTTP REST interface using CoAP, or for converting between HTTP and CoAP. This conversion can be carried out by a cross-protocol proxy ("cross-proxy"), which

converts the method or response code, media type, and options to the corresponding HTTP feature. Section 10 provides more detail about HTTP mapping.

2.4. Resource Discovery

Resource discovery is important for machine-to-machine interactions, and is supported using the CoRE Link Format [RFC6690] as discussed in Section 7.

3. Message Format

CoAP is based on the exchange of compact messages which, by default, are transported over UDP (i.e. each CoAP message occupies the data section of one UDP datagram). CoAP may also be used over Datagram Transport Layer Security (DTLS) (see Section 9.1). It could also be used over other transports such as SMS, TCP or SCTP, the specification of which is out of this document's scope. (UDP-lite [RFC3828] and UDP zero checksum [RFC6936] are not supported by CoAP.)

CoAP messages are encoded in a simple binary format. The message format starts with a fixed-size 4-byte header. This is followed by a variable-length Token value which can be between 0 and 8 bytes long. Following the Token value comes a sequence of zero or more CoAP Options in Type-Length-Value (TLV) format, optionally followed by a payload which takes up the rest of the datagram.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|Ver| T |  TKL  |      Code      |      Message ID      |
+-----+-----+-----+-----+-----+-----+-----+
|  Token (if any, TKL bytes) ...
+-----+-----+-----+-----+-----+-----+-----+
|  Options (if any) ...
+-----+-----+-----+-----+-----+-----+-----+
|1 1 1 1 1 1 1 1|      Payload (if any) ...
+-----+-----+-----+-----+-----+-----+-----+

```


Figure 7: Message Format

The fields in the header are defined as follows:

Version (Ver): 2-bit unsigned integer. Indicates the CoAP version number. Implementations of this specification MUST set this field to 1 (01 binary). Other values are reserved for future versions. Messages with unknown version numbers MUST be silently ignored.

Type (T): 2-bit unsigned integer. Indicates if this message is of type Confirmable (0), Non-confirmable (1), Acknowledgement (2) or Reset (3). The semantics of these message types are defined in Section 4.

Token Length (TKL): 4-bit unsigned integer. Indicates the length of the variable-length Token field (0-8 bytes). Lengths 9-15 are reserved, MUST NOT be sent, and MUST be processed as a message format error.

Code: 8-bit unsigned integer, split into a 3-bit class (most significant bits) and a 5-bit detail (least significant bits), documented as c.dd where c is a digit from 0 to 7 for the 3-bit subfield and dd are two digits from 00 to 31 for the 5-bit subfield. The class can indicate a request (0), a success response (2), a client error response (4), or a server error response (5). (All other class values are reserved.) As a special case, Code 0.00 indicates an Empty message. In case of a request, the Code field indicates the Request Method; in case of a response a Response Code. Possible values are maintained in the CoAP Code Registries (Section 12.1). The semantics of requests and responses are defined in Section 5.

Message ID: 16-bit unsigned integer in network byte order. Used for the detection of message duplication, and to match messages of type Acknowledgement/Reset to messages of type Confirmable/Non-confirmable. The rules for generating a Message ID and matching messages are defined in Section 4.

The header is followed by the Token value, which may be 0 to 8 bytes, as given by the Token Length field. The Token value is used to correlate requests and responses. The rules for generating a Token and correlating requests and responses are defined in Section 5.3.1.

Header and Token are followed by zero or more Options (Section 3.1). An Option can be followed by the end of the message, by another Option, or by the Payload Marker and the payload.

Following the header, token, and options, if any, comes the optional payload. If present and of non-zero length, it is prefixed by a fixed, one-byte Payload Marker (0xFF) which indicates the end of options and the start of the payload. The payload data extends from after the marker to the end of the UDP datagram, i.e., the Payload Length is calculated from the datagram size. The absence of the Payload Marker denotes a zero-length payload. The presence of a marker followed by a zero-length payload MUST be processed as a message format error.

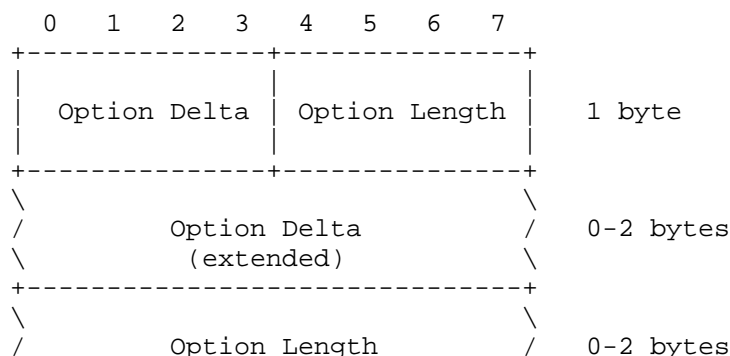
Implementation Note: The byte value 0xFF may also occur within an option length or value, so simple byte-wise scanning for 0xFF is not a viable technique for finding the payload marker. The byte 0xFF has the meaning of a payload marker only where the beginning of another option could occur.

3.1. Option Format

CoAP defines a number of options which can be included in a message. Each option instance in a message specifies the Option Number of the defined CoAP option, the length of the Option Value and the Option Value itself.

Instead of specifying the Option Number directly, the instances MUST appear in order of their Option Numbers and a delta encoding is used between them: The Option Number for each instance is calculated as the sum of its delta and the Option Number of the preceding instance in the message. For the first instance in a message, a preceding option instance with Option Number zero is assumed. Multiple instances of the same option can be included by using a delta of zero.

Option Numbers are maintained in the CoAP Option Number Registry (Section 12.2). See Section 5.4 for the semantics of the options defined in this document.



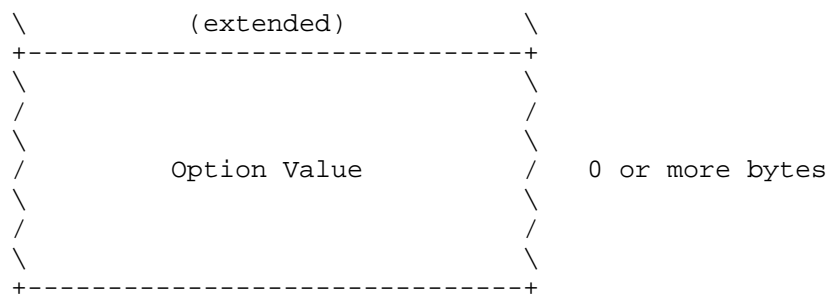


Figure 8: Option Format

The fields in an option are defined as follows:

Option Delta: 4-bit unsigned integer. A value between 0 and 12 indicates the Option Delta. Three values are reserved for special constructs:

- 13: An 8-bit unsigned integer follows the initial byte and indicates the Option Delta minus 13.
- 14: A 16-bit unsigned integer in network byte order follows the initial byte and indicates the Option Delta minus 269.
- 15: Reserved for the Payload Marker. If the field is set to this value but the entire byte is not the payload marker, this MUST be processed as a message format error.

The resulting Option Delta is used as the difference between the Option Number of this option and that of the previous option (or zero for the first option). In other words, the Option Number is calculated by simply summing the Option Delta values of this and all previous options before it.

Option Length: 4-bit unsigned integer. A value between 0 and 12 indicates the length of the Option Value, in bytes. Three values are reserved for special constructs:

- 13: An 8-bit unsigned integer precedes the Option Value and indicates the Option Length minus 13.
- 14: A 16-bit unsigned integer in network byte order precedes the Option Value and indicates the Option Length minus 269.
- 15: Reserved for future use. If the field is set to this value, it MUST be processed as a message format error.

Value: A sequence of exactly Option Length bytes. The length and format of the Option Value depend on the respective option, which MAY define variable length values. See Section 3.2 for the formats used in this document; options defined in other documents MAY make use of other option value formats.

3.2. Option Value Formats

The options defined in this document make use of the following option value formats.

empty: A zero-length sequence of bytes.

opaque: An opaque sequence of bytes.

uint: A non-negative integer which is represented in network byte order using the number of bytes given by the Option Length field.

An option definition may specify a range of permissible numbers of bytes; if it has a choice, a sender SHOULD represent the integer with as few bytes as possible, i.e., without leading zero bytes. For example, the number 0 is represented with an empty option value (a zero-length sequence of bytes), and the number 1 by a single byte with the numerical value of 1 (bit combination 00000001 in most significant bit first notation). A recipient MUST be prepared to process values with leading zero bytes.

Implementation Note: The exceptional behavior permitted for the sender is intended for highly constrained, templated implementations (e.g., hardware implementations) that use fixed size options in the templates.

string: A Unicode string which is encoded using UTF-8 [RFC3629] in Net-Unicode form [RFC5198].

Note that here and in all other places where UTF-8 encoding is used in the CoAP protocol, the intention is that the encoded strings can be directly used and compared as opaque byte strings by CoAP protocol implementations. There is no expectation and no need to perform normalization within a CoAP implementation (except where Unicode strings that are not known to be normalized are imported from sources outside the CoAP protocol). Note also that ASCII strings (that do not make use of special control characters) are always valid UTF-8 Net-Unicode strings.

4. Message Transmission

CoAP messages are exchanged asynchronously between CoAP endpoints. They are used to transport CoAP requests and responses, the semantics of which are defined in Section 5.

As CoAP is bound to non-reliable transports such as UDP, CoAP messages may arrive out of order, appear duplicated, or go missing without notice. For this reason, CoAP implements a lightweight reliability mechanism, without trying to re-create the full feature set of a transport like TCP. It has the following features:

- o Simple stop-and-wait retransmission reliability with exponential back-off for Confirmable messages.
- o Duplicate detection for both Confirmable and Non-confirmable messages.

4.1. Messages and Endpoints

A CoAP endpoint is the source or destination of a CoAP message. The specific definition of an endpoint depends on the transport being used for CoAP. For the transports defined in this specification, the endpoint is identified depending on the security mode used (see Section 9): With no security, the endpoint is solely identified by an IP address and a UDP port number. With other security modes, the endpoint is identified as defined by the security mode.

There are different types of messages. The type of a message is specified by the Type field of the CoAP Header.

Separate from the message type, a message may carry a request, a response, or be Empty. This is signaled by the Request/Response Code field in the CoAP Header and is relevant to the request/response model. Possible values for the field are maintained in the CoAP Code Registries (Section 12.1).

An Empty message has the Code field set to 0.00. The Token Length field MUST be set to 0 and bytes of data MUST NOT be present after the Message ID field. If there are any bytes, they MUST be processed as a message format error.

4.2. Messages Transmitted Reliably

The reliable transmission of a message is initiated by marking the message as Confirmable in the CoAP header. A Confirmable message always carries either a request or response, unless it is used only to elicit a Reset message in which case it is Empty. A recipient

MUST acknowledge a Confirmable message with an Acknowledgement message or, if it lacks context to process the message properly (including the case where the message is Empty, uses a code with a reserved class (1, 6 or 7), or has a message format error), MUST reject it; rejecting a Confirmable message is effected by sending a matching Reset message and otherwise ignoring it. The Acknowledgement message MUST echo the Message ID of the Confirmable message, and MUST carry a response or be Empty (see Section 5.2.1 and Section 5.2.2). The Reset message MUST echo the Message ID of the Confirmable message, and MUST be Empty. Rejecting an Acknowledgement or Reset message (including the case where the Acknowledgement carries a request or a code with a reserved class, or the Reset message is not Empty) is effected by silently ignoring it. More generally, recipients of Acknowledgement and Reset messages MUST NOT respond with either Acknowledgement or Reset messages.

The sender retransmits the Confirmable message at exponentially increasing intervals, until it receives an acknowledgement (or Reset message), or runs out of attempts.

Retransmission is controlled by two things that a CoAP endpoint MUST keep track of for each Confirmable message it sends while waiting for an acknowledgement (or reset): a timeout and a retransmission counter. For a new Confirmable message, the initial timeout is set to a random duration (often not an integral number of seconds) between `ACK_TIMEOUT` and `(ACK_TIMEOUT * ACK_RANDOM_FACTOR)` (see Section 4.8), and the retransmission counter is set to 0. When the timeout is triggered and the retransmission counter is less than `MAX_RETRANSMIT`, the message is retransmitted, the retransmission counter is incremented, and the timeout is doubled. If the retransmission counter reaches `MAX_RETRANSMIT` on a timeout, or if the endpoint receives a Reset message, then the attempt to transmit the message is canceled and the application process informed of failure. On the other hand, if the endpoint receives an acknowledgement in time, transmission is considered successful.

This specification makes no strong requirements on the accuracy of the clocks used to implement the above binary exponential backoff algorithm. In particular, an endpoint may be late for a specific retransmission due to its sleep schedule, and maybe catch up on the next one. However, the minimum spacing before another retransmission is `ACK_TIMEOUT`, and the entire sequence of (re-)transmissions MUST stay in the envelope of `MAX_TRANSMIT_SPAN` (see Section 4.8.2), even if that means a sender may miss an opportunity to transmit.

A CoAP endpoint that sent a Confirmable message MAY give up in attempting to obtain an ACK even before the `MAX_RETRANSMIT` counter value is reached: E.g., the application has canceled the request as

it no longer needs a response, or there is some other indication that the CON message did arrive. In particular, a CoAP request message may have elicited a separate response, in which case it is clear to the requester that only the ACK was lost and a retransmission of the request would serve no purpose. However, a responder **MUST NOT** in turn rely on this cross-layer behavior from a requester, i.e. it **MUST** retain the state to create the ACK for the request, if needed, even if a Confirmable response was already acknowledged by the requester.

Another reason for giving up retransmission **MAY** be the receipt of ICMP errors. If it is desired to take account of ICMP errors, to mitigate potential spoofing attacks, implementations **SHOULD** take care to check the information about the original datagram in the ICMP message, including port numbers and CoAP header information such as message type and code, Message ID, and Token; if this is not possible due to limitations of the UDP service API, ICMP errors **SHOULD** be ignored. Packet Too Big errors [RFC4443] ("fragmentation needed and DF set" for IPv4 [RFC0792]) cannot properly occur and **SHOULD** be ignored if the implementation note in Section 4.6 is followed; otherwise, they **SHOULD** feed into a path MTU discovery algorithm [RFC4821]. Source Quench and Time Exceeded ICMP messages **SHOULD** be ignored. Host, network, port or protocol unreachable errors, or parameter problem errors **MAY**, after appropriate vetting, be used to inform the application of a failure in sending.

4.3. Messages Transmitted Without Reliability

Some messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual success is sufficient.

As a more lightweight alternative, a message can be transmitted less reliably by marking the message as Non-confirmable. A Non-confirmable message always carries either a request or response and **MUST NOT** be Empty. A Non-confirmable message **MUST NOT** be acknowledged by the recipient. If a recipient lacks context to process the message properly (including the case where the message is Empty, uses a code with a reserved class (1, 6 or 7), or has a message format error), it **MUST** reject the message; rejecting a Non-confirmable message **MAY** involve sending a matching Reset message, and apart from the Reset message the rejected message **MUST** be silently ignored.

At the CoAP level, there is no way for the sender to detect if a Non-confirmable message was received or not. A sender **MAY** choose to transmit multiple copies of a Non-confirmable message within

MAX_TRANSMIT_SPAN (limited by the provisions of Section 4.7, in particular by PROBING_RATE if no response is received), or the network may duplicate the message in transit. To enable the receiver to act only once on the message, Non-confirmable messages specify a Message ID as well. (This Message ID is drawn from the same number space as the Message IDs for Confirmable messages.)

Summarizing Section 4.2 and Section 4.3, the four message types can be used as in Table 1. "*" means that the combination is not used in normal operation, but only to elicit a Reset message ("CoAP ping").

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

Table 1: Usage of message types

4.4. Message Correlation

An Acknowledgement or Reset message is related to a Confirmable message or Non-confirmable message by means of a Message ID along with additional address information of the corresponding endpoint. The Message ID is a 16-bit unsigned integer that is generated by the sender of a Confirmable or Non-confirmable message and included in the CoAP header. The Message ID MUST be echoed in the Acknowledgement or Reset message by the recipient.

The same Message ID MUST NOT be re-used (in communicating with the same endpoint) within the EXCHANGE_LIFETIME (Section 4.8.2).

Implementation Note: Several implementation strategies can be employed for generating Message IDs. In the simplest case a CoAP endpoint generates Message IDs by keeping a single Message ID variable, which is changed each time a new Confirmable or Non-confirmable message is sent regardless of the destination address or port. Endpoints dealing with large numbers of transactions could keep multiple Message ID variables, for example per prefix or destination address (note that some receiving endpoints may not be able to distinguish unicast and multicast packets addressed to it, so endpoints generating Message IDs need to make sure these do not overlap). It is strongly recommended that the initial value of the variable (e.g., on startup) be randomized, in order to make successful off-path attacks on the protocol less likely.

For an Acknowledgement or Reset message to match a Confirmable or Non-confirmable message, the Message ID and source endpoint of the Acknowledgement or Reset message MUST match the Message ID and destination endpoint of the Confirmable or Non-confirmable message.

4.5. Message Deduplication

A recipient might receive the same Confirmable message (as indicated by the Message ID and source endpoint) multiple times within the EXCHANGE_LIFETIME (Section 4.8.2), for example, when its Acknowledgement went missing or didn't reach the original sender before the first timeout. The recipient SHOULD acknowledge each duplicate copy of a Confirmable message using the same Acknowledgement or Reset message, but SHOULD process any request or response in the message only once. This rule MAY be relaxed in case the Confirmable message transports a request that is idempotent (see Section 5.1) or can be handled in an idempotent fashion. Examples for relaxed message deduplication:

- o A server might relax the requirement to answer all retransmissions of an idempotent request with the same response (Section 4.2), so that it does not have to maintain state for Message IDs. For example, an implementation might want to process duplicate transmissions of a GET, PUT or DELETE request as separate requests if the effort incurred by duplicate processing is less expensive than keeping track of previous responses would be.
- o A constrained server might even want to relax this requirement for certain non-idempotent requests if the application semantics make this trade-off favorable. For example, if the result of a POST request is just the creation of some short-lived state at the server, it may be less expensive to incur this effort multiple times for a request than keeping track of whether a previous transmission of the same request already was processed.

A recipient might receive the same Non-confirmable message (as indicated by the Message ID and source endpoint) multiple times within NON_LIFETIME (Section 4.8.2). As a general rule that MAY be relaxed based on the specific semantics of a message, the recipient SHOULD silently ignore any duplicated Non-confirmable message, and SHOULD process any request or response in the message only once.

4.6. Message Size

While specific link layers make it beneficial to keep CoAP messages small enough to fit into their link layer packets (see Section 1), this is a matter of implementation quality. The CoAP specification itself provides only an upper bound to the message size. Messages

larger than an IP packet result in undesirable packet fragmentation. A CoAP message, appropriately encapsulated, SHOULD fit within a single IP packet (i.e., avoid IP fragmentation) and (by fitting into one UDP payload) obviously needs to fit within a single IP datagram. If the Path MTU is not known for a destination, an IP MTU of 1280 bytes SHOULD be assumed; if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size.

Implementation Note: CoAP's choice of message size parameters works well with IPv6 and with most of today's IPv4 paths. (However, with IPv4, it is harder to absolutely ensure that there is no IP fragmentation. If IPv4 support on unusual networks is a consideration, implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes; worse, the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload. Implementations extremely focused on this problem set might also set the IPv4 DF bit and perform some form of path MTU discovery [RFC4821]; this should generally be unnecessary in most realistic use cases for CoAP, however.) A more important kind of fragmentation in many constrained networks is that on the adaptation layer (e.g., 6LoWPAN L2 packets are limited to 127 bytes including various overheads); this may motivate implementations to be frugal in their packet sizes and to move to block-wise transfers [I-D.ietf-core-block] when approaching three-digit message sizes.

Message sizes are also of considerable importance to implementations on constrained nodes. Many implementations will need to allocate a buffer for incoming messages. If an implementation is too constrained to allow for allocating the above-mentioned upper bound, it could apply the following implementation strategy for messages not using DTLS security: Implementations receiving a datagram into a buffer that is too small are usually able to determine if the trailing portion of a datagram was discarded and to retrieve the initial portion. So, if not all of the payload, at least the CoAP header and options are likely to fit within the buffer. A server can thus fully interpret a request and return a 4.13 (Request Entity Too Large, see Section 5.9.2.9) response code if the payload was truncated. A client sending an idempotent request and receiving a response larger than would fit in the buffer can repeat the request with a suitable value for the Block Option [I-D.ietf-core-block].

4.7. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.2.

In order not to cause congestion, Clients (including proxies) MUST strictly limit the number of simultaneous outstanding interactions that they maintain to a given server (including proxies) to NSTART. An outstanding interaction is either a CON for which an ACK has not yet been received but is still expected (message layer) or a request for which neither a response nor an Acknowledgment message has yet been received but is still expected (which may both occur at the same time, counting as one outstanding interaction). The default value of NSTART for this specification is 1.

Further congestion control optimizations and considerations are expected in the future, which may for example provide automatic initialization of the CoAP transmission parameters defined in Section 4.8, and thus may allow a value for NSTART greater than one.

A client stops expecting a response to a Confirmable request for which no acknowledgment message was received, after EXCHANGE_LIFETIME. The specific algorithm by which a client stops to "expect" a response to a Confirmable request that was acknowledged, or to a Non-confirmable request, is not defined. Unless this is modified by additional congestion control optimizations, it MUST be chosen in such a way that an endpoint does not exceed an average data rate of PROBING_RATE in sending to another endpoint that does not respond.

Note: CoAP places the onus of congestion control mostly on the clients. However, clients may malfunction or actually be attackers, e.g. to perform amplification attacks (Section 11.3). To limit the damage (to the network and to its own energy resources), a server SHOULD implement some rate limiting for its response transmission based on reasonable assumptions about application requirements. This is most helpful if the rate limit can be made effective for the misbehaving endpoints, only.

4.8. Transmission Parameters

Message transmission is controlled by the following parameters:

name	default value
ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1

DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 Byte/second

Table 2: CoAP Protocol Parameters

4.8.1. Changing The Parameters

The values for `ACK_TIMEOUT`, `ACK_RANDOM_FACTOR`, `MAX_RETRANSMIT`, `NSTART`, `DEFAULT_LEISURE` (Section 8.2), and `PROBING_RATE` may be configured to values specific to the application environment (including dynamically adjusted values), however the configuration method is out of scope of this document. It is RECOMMENDED that an application environment use consistent values for these parameters; the specific effects of operating with inconsistent values in an application environment are outside the scope of the present specification.

The transmission parameters have been chosen to achieve a behavior in the presence of congestion that is safe in the Internet. If a configuration desires to use different values, the onus is on the configuration to ensure these congestion control properties are not violated. In particular, a decrease of `ACK_TIMEOUT` below 1 second would violate the guidelines of [RFC5405].

([I-D.allman-tcpm-rto-consider] provides some additional background.) CoAP was designed to enable implementations that do not maintain round-trip-time (RTT) measurements. However, where it is desired to decrease the `ACK_TIMEOUT` significantly or increase `NSTART`, this can only be done safely when maintaining such measurements. Configurations MUST NOT decrease `ACK_TIMEOUT` or increase `NSTART` without using mechanisms that ensure congestion control safety, either defined in the configuration or in future standards documents.

`ACK_RANDOM_FACTOR` MUST NOT be decreased below 1.0, and it SHOULD have a value that is sufficiently different from 1.0 to provide some protection from synchronization effects.

`MAX_RETRANSMIT` can be freely adjusted, but a too small value will reduce the probability that a Confirmable message is actually received, while a larger value than given here will require further adjustments in the time values (see Section 4.8.2).

If the choice of transmission parameters leads to an increase of derived time values (see Section 4.8.2), the configuration mechanism MUST ensure the adjusted value is also available to all the endpoints that these adjusted values are to be used to communicate with.

4.8.2. Time Values derived from Transmission Parameters

The combination of `ACK_TIMEOUT`, `ACK_RANDOM_FACTOR` and `MAX_RETRANSMIT` influences the timing of retransmissions, which in turn influences how long certain information items need to be kept by an implementation. To be able to unambiguously reference these derived time values, we give them names as follows:

- o `MAX_TRANSMIT_SPAN` is the maximum time from the first transmission of a Confirmable message to its last retransmission. For the default transmission parameters, the value is $(2+4+8+16)*1.5 = 45$ seconds, or more generally:

$$\text{ACK_TIMEOUT} * ((2 ** \text{MAX_RETRANSMIT}) - 1) * \text{ACK_RANDOM_FACTOR}$$

- o `MAX_TRANSMIT_WAIT` is the maximum time from the first transmission of a Confirmable message to the time when the sender gives up on receiving an acknowledgement or reset. For the default transmission parameters, the value is $(2+4+8+16+32)*1.5 = 93$ seconds, or more generally:

$$\text{ACK_TIMEOUT} * ((2 ** (\text{MAX_RETRANSMIT} + 1)) - 1) * \text{ACK_RANDOM_FACTOR}$$

In addition, some assumptions need to be made on the characteristics of the network and the nodes.

- o `MAX_LATENCY` is the maximum time a datagram is expected to take from the start of its transmission to the completion of its reception. This constant is related to the MSL (Maximum Segment Lifetime) of [RFC0793], which is "arbitrarily defined to be 2 minutes" ([RFC0793] glossary, page 81). Note that this is not necessarily smaller than `MAX_TRANSMIT_WAIT`, as `MAX_LATENCY` is not intended to describe a situation when the protocol works well, but the worst case situation against which the protocol has to guard. We, also arbitrarily, define `MAX_LATENCY` to be 100 seconds. Apart from being reasonably realistic for the bulk of configurations as well as close to the historic choice for TCP, this value also allows Message ID lifetime timers to be represented in 8 bits (when measured in seconds). In these calculations, there is no assumption that the direction of the transmission is irrelevant (i.e. that the network is symmetric), just that the same value can reasonably be used as a maximum value for both directions. If that is not the case, the following calculations become only slightly more complex.
- o `PROCESSING_DELAY` is the time a node takes to turn around a Confirmable message into an acknowledgement. We assume the node

will attempt to send an ACK before having the sender time out, so as a conservative assumption we set it equal to ACK_TIMEOUT.

- o MAX_RTT is the maximum round-trip time, or:

$$(2 * MAX_LATENCY) + PROCESSING_DELAY$$

From these values, we can derive the following values relevant to the protocol operation:

- o EXCHANGE_LIFETIME is the time from starting to send a Confirmable message to the time when an acknowledgement is no longer expected, i.e. message layer information about the message exchange can be purged. EXCHANGE_LIFETIME includes a MAX_TRANSMIT_SPAN, a MAX_LATENCY forward, PROCESSING_DELAY, and a MAX_LATENCY for the way back. Note that there is no need to consider MAX_TRANSMIT_WAIT if the configuration is chosen such that the last waiting period ($ACK_TIMEOUT * (2 * MAX_RETRANSMIT)$) or the difference between MAX_TRANSMIT_SPAN and MAX_TRANSMIT_WAIT is less than MAX_LATENCY -- which is a likely choice, as MAX_LATENCY is a worst case value unlikely to be met in the real world. In this case, EXCHANGE_LIFETIME simplifies to:

$$MAX_TRANSMIT_SPAN + (2 * MAX_LATENCY) + PROCESSING_DELAY$$

or 247 seconds with the default transmission parameters.

- o NON_LIFETIME is the time from sending a Non-confirmable message to the time its Message ID can be safely reused. If multiple transmission of a NON message is not used, its value is MAX_LATENCY, or 100 seconds. However, a CoAP sender might send a NON message multiple times, in particular for multicast applications. While the period of re-use is not bounded by the specification, an expectation of reliable detection of duplication at the receiver is in the timescales of MAX_TRANSMIT_SPAN. Therefore, for this purpose, it is safer to use the value:

$$MAX_TRANSMIT_SPAN + MAX_LATENCY$$

or 145 seconds with the default transmission parameters; however, an implementation that just wants to use a single timeout value for retiring Message IDs can safely use the larger value for EXCHANGE_LIFETIME.

Table 3 summarizes the derived parameters introduced in this subsection with their default values.

name	default value
MAX_TRANSMIT_SPAN	45 s
MAX_TRANSMIT_WAIT	93 s
MAX_LATENCY	100 s
PROCESSING_DELAY	2 s
MAX_RTT	202 s
EXCHANGE_LIFETIME	247 s
NON_LIFETIME	145 s

Table 3: Derived Protocol Parameters

5. Request/Response Semantics

CoAP operates under a similar request/response model as HTTP: a CoAP endpoint in the role of a "client" sends one or more CoAP requests to a "server", which services the requests by sending CoAP responses. Unlike HTTP, requests and responses are not sent over a previously established connection, but exchanged asynchronously over CoAP messages.

5.1. Requests

A CoAP request consists of the method to be applied to the resource, the identifier of the resource, a payload and Internet media type (if any), and optional meta-data about the request.

CoAP supports the basic methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP. They have the same properties of safe (only retrieval) and idempotent (you can invoke it multiple times with the same effects) as HTTP (see Section 9.1 of [RFC2616]). The GET method is safe, therefore it MUST NOT take any other action on a resource other than retrieval. The GET, PUT and DELETE methods MUST be performed in such a way that they are idempotent. POST is not idempotent, because its effect is determined by the origin server and dependent on the target resource; it usually results in a new resource being created or the target resource being updated.

A request is initiated by setting the Code field in the CoAP header of a Confirmable or a Non-confirmable message to a Method Code and including request information.

The methods used in requests are described in detail in Section 5.8.

5.2. Responses

After receiving and interpreting a request, a server responds with a CoAP response, which is matched to the request by means of a client-generated token (Section 5.3, note that this is different from the Message ID that matches a Confirmable message to its Acknowledgement).

A response is identified by the Code field in the CoAP header being set to a Response Code. Similar to the HTTP Status Code, the CoAP Response Code indicates the result of the attempt to understand and satisfy the request. These codes are fully defined in Section 5.9. The Response Code numbers to be set in the Code field of the CoAP header are maintained in the CoAP Response Code Registry (Section 12.1.2).

```

0
0 1 2 3 4 5 6 7
+---+---+---+---+
|class|  detail |
+---+---+---+---+

```

Figure 9: Structure of a Response Code

The upper three bits of the 8-bit Response Code number define the class of response. The lower five bits do not have any categorization role; they give additional detail to the overall class (Figure 9).

As a human readable notation for specifications and protocol diagnostics, CoAP code numbers including the response code are documented in the format "c.dd", where "c" is the class in decimal, and "dd" is the detail as a two-digit decimal. For example, "Forbidden" is written as 4.03 -- indicating an 8-bit code value of hexadecimal 0x83 (4*0x20+3) or decimal 131 (4*32+3).

There are 3 classes of response codes:

- 2 - Success: The request was successfully received, understood, and accepted.
- 4 - Client Error: The request contains bad syntax or cannot be fulfilled.
- 5 - Server Error: The server failed to fulfill an apparently valid request.

The response codes are designed to be extensible: Response Codes in the Client Error and Server Error class that are unrecognized by an endpoint are treated as being equivalent to the generic Response Code

of that class (4.00 and 5.00, respectively). However, there is no generic Response Code indicating success, so a Response Code in the Success class that is unrecognized by an endpoint can only be used to determine that the request was successful without any further details.

The possible response codes are described in detail in Section 5.9.

Responses can be sent in multiple ways, which are defined in the following subsections.

5.2.1. Piggy-backed

In the most basic case, the response is carried directly in the Acknowledgement message that acknowledges the request (which requires that the request was carried in a Confirmable message). This is called a "Piggy-backed" Response.

The response is returned in the Acknowledgement message independent of whether the response indicates success or failure. In effect, the response is piggy-backed on the Acknowledgement message, and no separate message is required to return the response.

Implementation Note: The protocol leaves the decision whether to piggy-back a response or not (i.e., send a separate response) to the server. The client **MUST** be prepared to receive either. On the quality of implementation level, there is a strong expectation that servers will implement code to piggy-back whenever possible -- saving resources in the network and both at the client and at the server.

5.2.2. Separate

It may not be possible to return a piggy-backed response in all cases. For example, a server might need longer to obtain the representation of the resource requested than it can wait sending back the Acknowledgement message, without risking the client to repeatedly retransmit the request message (see also the discussion of `PROCESSING_DELAY` in Section 4.8.2). The Response to a request carried in a Non-confirmable message is always sent separately (as there is no Acknowledgement message).

One way to implement this in a server is to initiate the attempt to obtain the resource representation and, while that is in progress, time out an acknowledgement timer. A server may also immediately send an acknowledgement knowing in advance that there will be no piggy-backed response. In both cases, the acknowledgement effectively is a promise that the request will be acted upon later.

When the server finally has obtained the resource representation, it sends the response. When it is desired that this message is not lost, it is sent as a Confirmable message from the server to the client and answered by the client with an Acknowledgement, echoing the new Message ID chosen by the server. (It may also be sent as a Non-confirmable message; see Section 5.2.3.)

When the server chooses to use a separate response, it sends the Acknowledgement to the Confirmable request as an Empty message. Once the server sends back an Empty Acknowledgement, it MUST NOT send back the response in another Acknowledgement, even if the client retransmits another identical request. If a retransmitted request is received (perhaps because the original Acknowledgement was delayed), another Empty Acknowledgement is sent and any response MUST be sent as a separate response.

If the server then sends a Confirmable response, the client's Acknowledgement to that response MUST also be an Empty message (one that carries neither a request nor a response). The server MUST stop retransmitting its response on any matching Acknowledgement (silently ignoring any response code or payload) or Reset message.

Implementation Notes: Note that, as the underlying datagram transport may not be sequence-preserving, the Confirmable message carrying the response may actually arrive before or after the Acknowledgement message for the request; for the purposes of terminating the retransmission sequence, this also serves as an acknowledgement. Note also that, while the CoAP protocol itself does not make any specific demands here, there is an expectation that the response will come within a time frame that is reasonable from an application point of view; as there is no underlying transport protocol that could be instructed to run a keep-alive mechanism, the requester may want to set up a timeout that is unrelated to CoAP's retransmission timers in case the server is destroyed or otherwise unable to send the response.)

5.2.3. Non-confirmable

If the request message is Non-confirmable, then the response SHOULD be returned in a Non-confirmable message as well. However, an endpoint MUST be prepared to receive a Non-confirmable response (preceded or followed by an Empty Acknowledgement message) in reply to a Confirmable request, or a Confirmable response in reply to a Non-confirmable request.

5.3. Request/Response Matching

Regardless of how a response is sent, it is matched to the request by means of a token that is included by the client in the request, along with additional address information of the corresponding endpoint.

5.3.1. Token

The Token is used to match a response with a request. The token value is a sequence of 0 to 8 bytes. (Note that every message carries a token, even if it is of zero length.) Every request carries a client-generated token, which the server **MUST** echo in any resulting response without modification.

A token is intended for use as a client-local identifier for differentiating between concurrent requests (see Section 5.3); it could have been called a "request ID".

The client **SHOULD** generate tokens in such a way that tokens currently in use for a given source/destination endpoint pair are unique. (Note that a client implementation can use the same token for any request if it uses a different endpoint each time, e.g. a different source port number.) An empty token value is appropriate e.g. when no other tokens are in use to a destination, or when requests are made serially per destination and receive piggy-backed responses. There are however multiple possible implementation strategies to fulfill this.

A client sending a request without using transport layer security (Section 9) **SHOULD** use a non-trivial, randomized token to guard against spoofing of responses (Section 11.4). This protective use of tokens is the reason they are allowed to be up to 8 bytes in size. The actual size of the random component to be used for the Token depends on the security requirements of the client and the level of threat posed by spoofing of responses. A client that is connected to the general Internet **SHOULD** use at least 32 bits of randomness; keeping in mind that not being directly connected to the Internet is not necessarily sufficient protection against spoofing. (Note that the Message ID adds little in protection as it is usually sequentially assigned, i.e. guessable, and can be circumvented by spoofing a separate response.) Clients that want to optimize the Token length may further want to detect the level of ongoing attacks (e.g., by tallying recent Token mismatches in incoming messages) and adjust the Token length upwards appropriately. [RFC4086] discusses randomness requirements for security.

An endpoint receiving a token it did not generate **MUST** treat it as opaque and make no assumptions about its content or structure.

5.3.2. Request/Response Matching Rules

The exact rules for matching a response to a request are as follows:

1. The source endpoint of the response MUST be the same as the destination endpoint of the original request.
2. In a piggy-backed response, both the Message ID of the Confirmable request and the Acknowledgement, and the token of the response and original request MUST match. In a separate response, just the token of the response and original request MUST match.

In case a message carrying a response is unexpected (the client is not waiting for a response from the identified endpoint, at the endpoint addressed, and/or with the given token), the response is rejected (Section 4.2, Section 4.3).

Implementation Note: A client that receives a response in a CON message may want to clean up the message state right after sending the ACK. If that ACK is lost and the server retransmits the CON, the client may no longer have any state to correlate this response to, making the retransmission an unexpected message; the client will likely send a Reset message so it does not receive any more retransmissions. This behavior is normal and not an indication of an error. (Clients that are not aggressively optimized in their state memory usage will still have message state that will identify the second CON as a retransmission. Clients that actually expect more messages from the server [I-D.ietf-core-observe] will have to keep state in any case.)

5.4. Options

Both requests and responses may include a list of one or more options. For example, the URI in a request is transported in several options, and meta-data that would be carried in an HTTP header in HTTP is supplied as options as well.

CoAP defines a single set of options that are used in both requests and responses:

- o Content-Format
- o ETag
- o Location-Path
- o Location-Query

- o Max-Age
- o Proxy-Uri
- o Proxy-Scheme
- o Uri-Host
- o Uri-Path
- o Uri-Port
- o Uri-Query
- o Accept
- o If-Match
- o If-None-Match
- o Size1

The semantics of these options along with their properties are defined in detail in Section 5.10.

Not all options are defined for use with all methods and response codes. The possible options for methods and response codes are defined in Section 5.8 and Section 5.9 respectively. In case an option is not defined for a method or response code, it **MUST NOT** be included by a sender and **MUST** be treated like an unrecognized option by a recipient.

5.4.1. Critical/Elective

Options fall into one of two classes: "critical" or "elective". The difference between these is how an option unrecognized by an endpoint is handled:

- o Upon reception, unrecognized options of class "elective" **MUST** be silently ignored.
- o Unrecognized options of class "critical" that occur in a Confirmable request **MUST** cause the return of a 4.02 (Bad Option) response. This response **SHOULD** include a diagnostic payload describing the unrecognized option(s) (see Section 5.5.2).

- o Unrecognized options of class "critical" that occur in a Confirmable response, or piggy-backed in an Acknowledgement, MUST cause the response to be rejected (Section 4.2).
- o Unrecognized options of class "critical" that occur in a Non-confirmable message MUST cause the message to be rejected (Section 4.3).

Note that, whether critical or elective, an option is never "mandatory" (it is always optional): These rules are defined in order to enable implementations to stop processing options they do not understand or implement.

Critical/Elective rules apply to non-proxying endpoints. A proxy processes options based on Unsafe/Safe-to-Forward classes as defined in Section 5.7.

5.4.2. Proxy Unsafe/Safe-to-Forward and NoCacheKey

In addition to an option being marked as Critical or Elective, options are also classified based on how a proxy is to deal with the option if it does not recognize it. For this purpose, an option can either be considered Unsafe to Forward (Unsafe is set) or Safe-to-Forward (Unsafe is clear).

In addition, for an option that is marked Safe-to-Forward, the option number indicates whether it is intended to be part of the Cache-Key (Section 5.6) in a request or not; if some of the NoCacheKey bits are 0, it is, if all NoCacheKey bits are 1, it is not (see Section 5.4.6).

Note: The Cache-Key indication is relevant only for proxies that do not implement the given option as a request option and instead rely on the Unsafe/Safe-to-Forward indication only. E.g., for ETag, actually using the request option as a part of the Cache-Key is grossly inefficient, but it is the best thing one can do if ETag is not implemented by a proxy, as the response is going to differ based on the presence of the request option. A more useful proxy that does implement the ETag request option is not using ETag as a part of the Cache-Key.

NoCacheKey is indicated in three bits so that only one out of eight codepoints is qualified as NoCacheKey, assuming this is the less likely case.

Proxy behavior with regard to these classes is defined in Section 5.7.

5.4.3. Length

Option values are defined to have a specific length, often in the form of an upper and lower bound. If the length of an option value in a request is outside the defined range, that option **MUST** be treated like an unrecognized option (see Section 5.4.1).

5.4.4. Default Values

Options may be defined to have a default value. If the value of option is intended to be this default value, the option **SHOULD NOT** be included in the message. If the option is not present, the default value **MUST** be assumed.

Where a critical option has a default value, this is chosen in such a way that the absence of the option in a message can be processed properly both by implementations unaware of the critical option and by implementations that interpret this absence as the presence of the default value for the option.

5.4.5. Repeatable Options

The definition of some options specifies that those options are repeatable. An option that is repeatable **MAY** be included one or more times in a message. An option that is not repeatable **MUST NOT** be included more than once in a message.

If a message includes an option with more occurrences than the option is defined for, each supernumerary option occurrence that appears subsequently in the message **MUST** be treated like an unrecognized option (see Section 5.4.1).

5.4.6. Option Numbers

An Option is identified by an option number, which also provides some additional semantics information: e.g., odd numbers indicate a critical option, while even numbers indicate an elective option. Note that this is not just a convention, it is a feature of the protocol: Whether an option is elective or critical is entirely determined by whether its option number is even or odd.

More generally speaking, an Option number is constructed with a bit mask to indicate if an option is Critical/Elective, Unsafe/Safe-to-Forward and in the case of Safe-to-Forward, also a Cache-Key indication as shown by the following figure. In the following text, the bit mask is expressed as a single byte that is applied to the least significant byte of the option number in unsigned integer representation. When bit 7 (the least significant bit) is 1, an

option is Critical (and likewise Elective when 0). When bit 6 is 1, an option is Unsafe (and likewise Safe-to-Forward when 0). When bit 6 is 0, i.e., the option is not Unsafe, it is not a Cache-Key (NoCacheKey) if and only if bits 3-5 are all set to 1; all other bit combinations mean that it indeed is a Cache-Key. These classes of options are explained in the next sections.

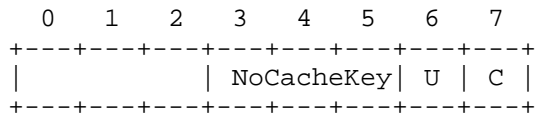


Figure 10: Option Number Mask (Least Significant Byte)

An endpoint may use an equivalent of the C code in Figure 11 to derive the characteristics of an option number "onum".

```

Critical = (onum & 1);
UnSafe = (onum & 2);
NoCacheKey = ((onum & 0x1e) == 0x1c);

```

Figure 11: Determining Characteristics from an Option Number

The option numbers for the options defined in this document are listed in the CoAP Option Number Registry (Section 12.2).

5.5. Payloads and Representations

Both requests and responses may include a payload, depending on the method or response code respectively. If a method or response code is not defined to have a payload, then a sender MUST NOT include one, and a recipient MUST ignore it.

5.5.1. Representation

The payload of requests or of responses indicating success is typically a representation of a resource ("resource representation") or the result of the requested action ("action result"). Its format is specified by the Internet media type and content coding given by the Content-Format Option. In the absence of this option, no default value is assumed and the format will need to be inferred by the application (e.g., from the application context). Payload "sniffing" SHOULD only be attempted if no content type is given.

Implementation Note: On a quality of implementation level, there is a strong expectation that a Content-Format indication will be provided with resource representations whenever possible. This is not a "SHOULD"-level requirement solely because it is not a

protocol requirement, and it also would be difficult to outline exactly in what cases this expectation can be violated.

For responses indicating a client or server error, the payload is considered a representation of the result of the requested action only if a Content-Format Option is given. In the absence of this option, the payload is a Diagnostic Payload (Section 5.5.2).

5.5.2. Diagnostic Payload

If no Content-Format option is given, the payload of responses indicating a client or server error is a brief human-readable diagnostic message, explaining the error situation. This diagnostic message **MUST** be encoded using UTF-8 [RFC3629], more specifically using Net-Unicode form [RFC5198].

The message is similar to the Reason-Phrase on an HTTP status line. It is not intended for end-users but for software engineers that during debugging need to interpret it in the context of the present, English-language specification; therefore no mechanism for language tagging is needed or provided. In contrast to what is usual in HTTP, the payload **SHOULD** be empty if there is no additional information beyond the response code.

5.5.3. Selected Representation

Not all responses carry a payload that provides a representation of the resource addressed by the request. It is, however, sometimes useful to be able to refer to such a representation in relation to a response, independent of whether it actually was enclosed.

We use the term "selected representation" to refer to the current representation of a target resource that would have been selected in a successful response if the corresponding request had used the method GET and excluded any conditional request options (Section 5.10.8).

Certain response options provide metadata about the selected representation, which might differ from the representation included in the message for responses to some state-changing methods. Of the response options defined in this specification, only the ETag response option (Section 5.10.6) is defined as selected representation metadata.

5.5.4. Content Negotiation

A server may be able to supply a representation for a resource in one of multiple representation formats. Without further information from

the client, it will provide the representation in the format it prefers.

By using the Accept Option (Section 5.10.4) in a request, the client can indicate which content-format it prefers to receive.

5.6. Caching

CoAP endpoints MAY cache responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests.

The goal of caching in CoAP is to reuse a prior response message to satisfy a current request. In some cases, a stored response can be reused without the need for a network request, reducing latency and network round-trips; a "freshness" mechanism is used for this purpose (see Section 5.6.1). Even when a new request is required, it is often possible to reuse the payload of a prior response to satisfy the request, thereby reducing network bandwidth usage; a "validation" mechanism is used for this purpose (see Section 5.6.2).

Unlike HTTP, the cacheability of CoAP responses does not depend on the request method, but the Response Code. The cacheability of each Response Code is defined along the Response Code definitions in Section 5.9. Response Codes that indicate success and are unrecognized by an endpoint MUST NOT be cached.

For a presented request, a CoAP endpoint MUST NOT use a stored response, unless:

- o the presented request method and that used to obtain the stored response match,
- o all options match between those in the presented request and those of the request used to obtain the stored response (which includes the request URI), except that there is no need for a match of any request options marked as NoCacheKey (Section 5.4) or recognized by the Cache and fully interpreted with respect to its specified cache behavior (such as the ETag request option, Section 5.10.6, see also Section 5.4.2), and
- o the stored response is either fresh or successfully validated as defined below.

The set of request options that is used for matching the cache entry is also collectively referred to as the "Cache-Key". For URI schemes other than coap and coaps, matching of those options that constitute the request URI may be performed under rules specific to the URI scheme.

5.6.1. Freshness Model

When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.

The mechanism for determining freshness is for an origin server to provide an explicit expiration time in the future, using the Max-Age Option (see Section 5.10.5). The Max-Age Option indicates that the response is to be considered not fresh after its age is greater than the specified number of seconds.

The Max-Age Option defaults to a value of 60. Thus, if it is not present in a cacheable response, then the response is considered not fresh after its age is greater than 60 seconds. If an origin server wishes to prevent caching, it **MUST** explicitly include a Max-Age Option with a value of zero seconds.

If a client has a fresh stored response and makes a new request matching the request for that stored response, the new response invalidates the old response.

5.6.2. Validation Model

When an endpoint has one or more stored responses for a GET request, but cannot use any of them (e.g., because they are not fresh), it can use the ETag Option (Section 5.10.6) in the GET request to give the origin server an opportunity to both select a stored response to be used, and to update its freshness. This process is known as "validating" or "revalidating" the stored response.

When sending such a request, the endpoint **SHOULD** add an ETag Option specifying the entity-tag of each stored response that is applicable.

A 2.03 (Valid) response indicates the stored response identified by the entity-tag given in the response's ETag Option can be reused, after updating it as described in Section 5.9.1.3.

Any other response code indicates that none of the stored responses nominated in the request is suitable. Instead, the response **SHOULD** be used to satisfy the request and **MAY** replace the stored response.

5.7. Proxying

A proxy is a CoAP endpoint that can be tasked by CoAP clients to perform requests on their behalf. This may be useful, for example, when the request could otherwise not be made, or to service the response from a cache in order to reduce response time and network bandwidth or energy consumption.

In an overall architecture for a Constrained RESTful Environment, proxies can serve quite different purposes. Proxies can be explicitly selected by clients, a role that we term "forward-proxy". Proxies can also be inserted to stand in for origin servers, a role that we term "reverse-proxy". Orthogonal to this distinction, a proxy can map from a CoAP request to a CoAP request (CoAP-to-CoAP proxy) or translate from or to a different protocol ("cross-proxy"). Full definitions of these terms are provided in Section 1.2.

Notes: The terminology in this specification has been selected to be culturally compatible with the terminology used in the wider Web application environments, without necessarily matching it in every detail (which may not even be relevant to Constrained RESTful Environments). Not too much semantics should be ascribed to the components of the terms (such as "forward", "reverse", or "cross").

HTTP proxies, besides acting as HTTP proxies, often offer a transport protocol proxying function ("CONNECT") to enable end-to-end transport layer security through the proxy. No such function is defined for CoAP-to-CoAP proxies in this specification, as forwarding of UDP packets is unlikely to be of much value in Constrained RESTful environments. See also Section 10.2.7 for the cross-proxy case.

When a client uses a proxy to make a request that will use a secure URI scheme (e.g., coaps or https), the request towards the proxy SHOULD be sent using DTLS security except where equivalent lower layer security is used for the leg between the client and the proxy.

5.7.1. Proxy Operation

A proxy generally needs a way to determine potential request parameters for a request to a destination based on the request it received. This way is fully specified for a forward-proxy, but may depend on the specific configuration for a reverse-proxy. In particular, the client of a reverse-proxy generally does not indicate a locator for the destination, necessitating some form of namespace translation in the reverse-proxy. However, some aspects of the operation of proxies are common to all its forms.

If a proxy does not employ a cache, then it simply forwards the translated request to the determined destination. Otherwise, if it does employ a cache but does not have a stored response that matches the translated request and is considered fresh, then it needs to refresh its cache according to Section 5.6. For options in the request that the proxy recognizes, it knows whether the option is intended to act as part of the key used in looking up the cached value or not. E.g., since requests for different Uri-Path values address different resources, Uri-Path values are always part of the Cache-Key, while, e.g., Token values are never part of the Cache-Key. For options that the proxy does not recognize but that are marked Safe-to-Forward in the option number, the option also indicates whether it is to be included in the Cache-Key (NoCacheKey is not all set) or not (NoCacheKey is all set). (Options that are unrecognized and marked Unsafe lead to 4.02 Bad Option.)

If the request to the destination times out, then a 5.04 (Gateway Timeout) response MUST be returned. If the request to the destination returns a response that cannot be processed by the proxy (e.g, due to unrecognized critical options, message format errors), then a 5.02 (Bad Gateway) response MUST be returned. Otherwise, the proxy returns the response to the client.

If a response is generated out of a cache, the generated (or implied) Max-Age Option MUST NOT extend the max-age originally set by the server, considering the time the resource representation spent in the cache. E.g., the Max-Age Option could be adjusted by the proxy for each response using the formula:

$$\text{proxy-max-age} = \text{original-max-age} - \text{cache-age}$$

For example if a request is made to a proxied resource that was refreshed 20 seconds ago and had an original Max-Age of 60 seconds, then that resource's proxied max-age is now 40 seconds. Considering potential network delays on the way from the origin server, a proxy should be conservative in the max-age values offered.

All options present in a proxy request MUST be processed at the proxy. Unsafe options in a request that are not recognized by the proxy MUST lead to a 4.02 (Bad Option) response being returned by the proxy. A CoAP-to-CoAP proxy MUST forward to the origin server all Safe-to-Forward options that it does not recognize. Similarly, Unsafe options in a response that are not recognized by the CoAP-to-CoAP proxy server MUST lead to a 5.02 (Bad Gateway) response. Again, Safe-to-Forward options that are not recognized MUST be forwarded.

Additional considerations for cross-protocol proxying between CoAP and HTTP are discussed in Section 10.

5.7.2. Forward-Proxies

CoAP distinguishes between requests made (as if) to an origin server and a request made through a forward-proxy. CoAP requests to a forward-proxy are made as normal Confirmable or Non-confirmable requests to the forward-proxy endpoint, but specify the request URI in a different way: The request URI in a proxy request is specified as a string in the Proxy-Uri Option (see Section 5.10.2), while the request URI in a request to an origin server is split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options (see Section 5.10.1); alternatively the URI in a proxy request can be assembled from a Proxy-Scheme option and the split options mentioned.

When a proxy request is made to an endpoint and the endpoint is unwilling or unable to act as proxy for the request URI, it MUST return a 5.05 (Proxying Not Supported) response. If the authority (host and port) is recognized as identifying the proxy endpoint itself (see Section 5.10.2), then the request MUST be treated as a local (non-proxied) request.

Unless a proxy is configured to forward the proxy request to another proxy, it MUST translate the request as follows: The scheme of the request URI defines the outgoing protocol and its details (e.g., CoAP is used over UDP for the "coap" scheme and over DTLS for the "coaps" scheme.) For a CoAP-to-CoAP proxy, the origin server's IP address and port are determined by the authority component of the request URI, and the request URI is decoded and split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options. This consumes the Proxy-Uri or Proxy-Scheme option, which is therefore not forwarded to the origin server.

5.7.3. Reverse-Proxies

Reverse-proxies do not make use of the Proxy-Uri or Proxy-Scheme options, but need to determine the destination (next hop) of a request from information in the request and information in their configuration. E.g., a reverse-proxy might offer various resources the existence of which it has learned through resource discovery as if they were its own resources. The reverse-proxy is free to build a namespace for the URIs that identify these resources. A reverse-proxy may also build a namespace that gives the client more control over where the request goes, e.g. by embedding host identifiers and port numbers into the URI path of the resources offered.

In processing the response, a reverse-proxy has to be careful that ETag option values from different sources are not mixed up on one resource offered to its clients. In many cases, the ETag can be forwarded unchanged. If the mapping from a resource offered by the

reverse-proxy to resources offered by its various origin servers is not unique, the reverse-proxy may need to generate a new ETag, making sure the semantics of this option are properly preserved.

5.8. Method Definitions

In this section each method is defined along with its behavior. A request with an unrecognized or unsupported Method Code MUST generate a 4.05 (Method Not Allowed) piggy-backed response.

5.8.1. GET

The GET method retrieves a representation for the information that currently corresponds to the resource identified by the request URI. If the request includes an Accept Option, that indicates the preferred content-format of a response. If the request includes an ETag Option, the GET method requests that ETag be validated and that the representation be transferred only if validation failed. Upon success a 2.05 (Content) or 2.03 (Valid) response code SHOULD be present in the response.

The GET method is safe and idempotent.

5.8.2. POST

The POST method requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and dependent on the target resource. It usually results in a new resource being created or the target resource being updated.

If a resource has been created on the server, the response returned by the server SHOULD have a 2.01 (Created) response code and SHOULD include the URI of the new resource in a sequence of one or more Location-Path and/or Location-Query Options (Section 5.10.7). If the POST succeeds but does not result in a new resource being created on the server, the response SHOULD have a 2.04 (Changed) response code. If the POST succeeds and results in the target resource being deleted, the response SHOULD have a 2.02 (Deleted) response code.

POST is neither safe nor idempotent.

5.8.3. PUT

The PUT method requests that the resource identified by the request URI be updated or created with the enclosed representation. The representation format is specified by the media type and content coding given in the Content-Format Option, if provided.

If a resource exists at the request URI the enclosed representation SHOULD be considered a modified version of that resource, and a 2.04 (Changed) response code SHOULD be returned. If no resource exists then the server MAY create a new resource with that URI, resulting in a 2.01 (Created) response code. If the resource could not be created or modified, then an appropriate error response code SHOULD be sent.

Further restrictions to a PUT can be made by including the If-Match (see Section 5.10.8.1) or If-None-Match (see Section 5.10.8.2) options in the request.

PUT is not safe, but is idempotent.

5.8.4. DELETE

The DELETE method requests that the resource identified by the request URI be deleted. A 2.02 (Deleted) response code SHOULD be used on success or in case the resource did not exist before the request.

DELETE is not safe, but is idempotent.

5.9. Response Code Definitions

Each response code is described below, including any options required in the response. Where appropriate, some of the codes will be specified in regards to related response codes in HTTP [RFC2616]; this does not mean that any such relationship modifies the HTTP mapping specified in Section 10.

5.9.1. Success 2.xx

This class of status code indicates that the clients request was successfully received, understood, and accepted.

5.9.1.1. 2.01 Created

Like HTTP 201 "Created", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

If the response includes one or more Location-Path and/or Location-Query Options, the values of these options specify the location at which the resource was created. Otherwise, the resource was created at the request URI. A cache receiving this response MUST mark any stored response for the created resource as not fresh.

This response is not cacheable.

5.9.1.2. 2.02 Deleted

Like HTTP 204 "No Content", but only used in response to requests that cause the resource to cease being available, such as DELETE and in certain circumstances POST. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache MUST mark any stored response for the deleted resource as not fresh.

5.9.1.3. 2.03 Valid

Related to HTTP 304 "Not Modified", but only used to indicate that the response identified by the entity-tag identified by the included ETag Option is valid. Accordingly, the response MUST include an ETag Option, and MUST NOT include a payload.

When a cache that recognizes and processes the ETag response option receives a 2.03 (Valid) response, it MUST update the stored response with the value of the Max-Age Option included in the response (explicitly, or implicitly as a default value; see also Section 5.6.2). For each type of Safe-to-Forward option present in the response, the (possibly empty) set of options of this type that are present in the stored response MUST be replaced with the set of options of this type in the response received. (Unsafe options may trigger similar option specific processing as defined by the option.)

5.9.1.4. 2.04 Changed

Like HTTP 204 "No Content", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache MUST mark any stored response for the changed resource as not fresh.

5.9.1.5. 2.05 Content

Like HTTP 200 "OK", but only used in response to GET requests.

The payload returned with the response is a representation of the target resource.

This response is cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1) and (if present) the ETag Option for validation (see Section 5.6.2).

5.9.2. Client Error 4.xx

This class of response code is intended for cases in which the client seems to have erred. These response codes are applicable to any request method.

The server SHOULD include a diagnostic payload under the conditions detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

5.9.2.1. 4.00 Bad Request

Like HTTP 400 "Bad Request".

5.9.2.2. 4.01 Unauthorized

The client is not authorized to perform the requested action. The client SHOULD NOT repeat the request without first improving its authentication status to the server. Which specific mechanism can be used for this is outside this document's scope; see also Section 9.

5.9.2.3. 4.02 Bad Option

The request could not be understood by the server due to one or more unrecognized or malformed options. The client SHOULD NOT repeat the request without modification.

5.9.2.4. 4.03 Forbidden

Like HTTP 403 "Forbidden".

5.9.2.5. 4.04 Not Found

Like HTTP 404 "Not Found".

5.9.2.6. 4.05 Method Not Allowed

Like HTTP 405 "Method Not Allowed", but with no parallel to the "Allow" header field.

5.9.2.7. 4.06 Not Acceptable

Like HTTP 406 "Not Acceptable", but with no response entity.

5.9.2.8. 4.12 Precondition Failed

Like HTTP 412 "Precondition Failed".

5.9.2.9. 4.13 Request Entity Too Large

Like HTTP 413 "Request Entity Too Large".

The response SHOULD include a Size1 Option (Section 5.10.9) to indicate the maximum size of request entity the server is able and willing to handle, unless the server is not in a position to make this information available.

5.9.2.10. 4.15 Unsupported Content-Format

Like HTTP 415 "Unsupported Media Type".

5.9.3. Server Error 5.xx

This class of response code indicates cases in which the server is aware that it has erred or is incapable of performing the request. These response codes are applicable to any request method.

The server SHOULD include a diagnostic payload under the conditions detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

5.9.3.1. 5.00 Internal Server Error

Like HTTP 500 "Internal Server Error".

5.9.3.2. 5.01 Not Implemented

Like HTTP 501 "Not Implemented".

5.9.3.3. 5.02 Bad Gateway

Like HTTP 502 "Bad Gateway".

5.9.3.4. 5.03 Service Unavailable

Like HTTP 503 "Service Unavailable", but using the Max-Age Option in place of the "Retry-After" header field to indicate the number of seconds after which to retry.

5.9.3.5. 5.04 Gateway Timeout

Like HTTP 504 "Gateway Timeout".

5.9.3.6. 5.05 Proxying Not Supported

The server is unable or unwilling to act as a forward-proxy for the URI specified in the Proxy-Uri Option or using Proxy-Scheme (see Section 5.10.2).

5.10. Option Definitions

The individual CoAP options are summarized in Table 4 and explained in the subsections of this section.

In this table, the C, U, and N columns indicate the properties, Critical, UnSafe, and NoCacheKey, respectively. Since NoCacheKey only has a meaning for options that are Safe-to-Forward (not marked UnSafe), the column is filled with a dash for UnSafe options. (The present specification does not define any NoCacheKey options, but the format of the table is intended to be useful for additional specifications.)

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 4: Options

5.10.1. Uri-Host, Uri-Port, Uri-Path and Uri-Query

The Uri-Host, Uri-Port, Uri-Path and Uri-Query Options are used to specify the target resource of a request to a CoAP origin server. The options encode the different components of the request URI in a way that no percent-encoding is visible in the option values and that the full URI can be reconstructed at any involved endpoint. The syntax of CoAP URIs is defined in Section 6.

The steps for parsing URIs into options is defined in Section 6.4. These steps result in zero or more Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in a request, where each option holds the following values:

- o the Uri-Host Option specifies the Internet host of the resource being requested,
- o the Uri-Port Option specifies the transport layer port number of the resource,
- o each Uri-Path Option specifies one segment of the absolute path to the resource, and
- o each Uri-Query Option specifies one argument parameterizing the resource.

Note: Fragments ([RFC3986], Section 3.5) are not part of the request URI and thus will not be transmitted in a CoAP request.

The default value of the Uri-Host Option is the IP literal representing the destination IP address of the request message. Likewise, the default value of the Uri-Port Option is the destination UDP port. The default values for the Uri-Host and Uri-Port Options are sufficient for requests to most servers. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers.

The Uri-Path and Uri-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Uri-Path Option MUST NOT be "." or ".." (as the request URI must be resolved before parsing it into options).

The steps for constructing the request URI from the options are defined in Section 6.5. Note that an implementation does not

necessarily have to construct the URI; it can simply look up the target resource by looking at the individual options.

Examples can be found in Appendix B.

5.10.2. Proxy-Uri and Proxy-Scheme

The Proxy-Uri Option is used to make a request to a forward-proxy (see Section 5.7). The forward-proxy is requested to forward the request or service it from a valid cache, and return the response.

The option value is an absolute-URI ([RFC3986], Section 4.3).

Note that the forward-proxy MAY forward the request on to another proxy or directly to the server specified by the absolute-URI. In order to avoid request loops, a proxy MUST be able to recognize all of its server names, including any aliases, local variations, and the numeric IP addresses.

An endpoint receiving a request with a Proxy-Uri Option that is unable or unwilling to act as a forward-proxy for the request MUST cause the return of a 5.05 (Proxying Not Supported) response.

The Proxy-Uri Option MUST take precedence over any of the Uri-Host, Uri-Port, Uri-Path or Uri-Query options (which MUST NOT be included at the same time in a request containing the Proxy-Uri Option).

As a special case to simplify many proxy clients, the absolute-URI can be constructed from the Uri-* options. When a Proxy-Scheme Option is present, the absolute-URI is constructed as follows: A CoAP URI is constructed from the Uri-* options as defined in Section 6.5. In the resulting URI, the initial scheme up to, but not including the following colon is then replaced by the content of the Proxy-Scheme Option. Note that this case is only applicable if the components of the desired URI other than the scheme component actually can be expressed using Uri-* options; e.g., to represent a URI with a userinfo component in the authority, only Proxy-Uri can be used.

5.10.3. Content-Format

The Content-Format Option indicates the representation format of the message payload. The representation format is given as a numeric content format identifier that is defined in the CoAP Content Format Registry (Section 12.3). In the absence of the option, no default value is assumed, i.e. the representation format of any representation message payload is indeterminate (Section 5.5).

5.10.4. Accept

The CoAP Accept option can be used to indicate which Content-Format is acceptable to the client. The representation format is given as a numeric Content-Format identifier that is defined in the CoAP Content-Format Registry (Section 12.3). If no Accept option is given, the client does not express a preference (thus no default value is assumed). The client prefers the representation returned by the server to be in the Content-Format indicated. The server returns the preferred Content-Format if available. If the preferred Content-Format cannot be returned, then a 4.06 "Not Acceptable" MUST be sent as a response, unless another error code takes precedence for this response.

5.10.5. Max-Age

The Max-Age Option indicates the maximum time a response may be cached before it is considered not fresh (see Section 5.6.1).

The option value is an integer number of seconds between 0 and $2^{32}-1$ inclusive (about 136.1 years). A default value of 60 seconds is assumed in the absence of the option in a response.

The value is intended to be current at the time of transmission. Servers that provide resources with strict tolerances on the value of Max-Age SHOULD update the value before each retransmission. (See also Section 5.7.1.)

5.10.6. ETag

An entity-tag is intended for use as a resource-local identifier for differentiating between representations of the same resource that vary over time. It is generated by the server providing the resource, which may generate it in any number of ways including a version, checksum, hash or time. An endpoint receiving an entity-tag MUST treat it as opaque and make no assumptions about its content or structure. (Endpoints that generate an entity-tag are encouraged to use the most compact representation possible, in particular in regards to clients and intermediaries that may want to store multiple ETag values.)

5.10.6.1. ETag as a Response Option

The ETag Option in a response provides the current value (i.e., after the request was processed) of the entity-tag for the "tagged representation". If no Location-* options are present, the tagged representation is the selected representation (Section 5.5.3) of the target resource. If one or more Location-* options are present and

thus a location URI is indicated (Section 5.10.7), the tagged representation is the representation that would be retrieved by a GET request to the location URI.

An ETag response option can be included with any response for which there is a tagged representation (e.g., it would not be meaningful in a 4.04 or 4.00 response). The ETag Option MUST NOT occur more than once in a response.

There is no default value for the ETag Option; if it is not present in a response, the server makes no statement about the entity-tag for the tagged representation.

5.10.6.2. ETag as a Request Option

In a GET request, an endpoint that has one or more representations previously obtained from the resource, and has obtained ETag response options with these, can specify an instance of the ETag Option for one or more of these stored responses.

A server can issue a 2.03 Valid response (Section 5.9.1.3) in place of a 2.05 Content response if one of the ETags given is the entity-tag for the current representation, i.e. is valid; the 2.03 Valid response then echoes this specific ETag in a response option.

In effect, a client can determine if any of the stored representations is current (see Section 5.6.2) without needing to transfer them again.

The ETag Option MAY occur zero, one or more times in a request.

5.10.7. Location-Path and Location-Query

The Location-Path and Location-Query Options together indicate a relative URI that consists either of an absolute path, a query string or both. A combination of these options is included in a 2.01 (Created) response to indicate the location of the resource created as the result of a POST request (see Section 5.8.2). The location is resolved relative to the request URI.

If a response with one or more Location-Path and/or Location-Query Options passes through a cache that interprets these options and the implied URI identifies one or more currently stored responses, those entries MUST be marked as not fresh.

Each Location-Path Option specifies one segment of the absolute path to the resource, and each Location-Query Option specifies one argument parameterizing the resource. The Location-Path and

Location-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Location-Path Option MUST NOT be "." or "..".

The steps for constructing the location URI from the options are analogous to Section 6.5, except that the first five steps are skipped and the result is a relative URI-reference, which is then interpreted relative to the request URI. Note that the relative URI-reference constructed this way always includes an absolute-path (e.g., leaving out Location-Path but supplying Location-Query means the path component in the URI is "/").

The options that are used to compute the relative URI-reference are collectively called Location-* options. Beyond Location-Path and Location-Query, more Location-* options may be defined in the future, and have been reserved option numbers 128, 132, 136, and 140. If any of these reserved option numbers occurs in addition to Location-Path and/or Location-Query and are not supported, then a 4.02 (Bad Option) error MUST be returned.

5.10.8. Conditional Request Options

Conditional request options enable a client to ask the server to perform the request only if certain conditions specified by the option are fulfilled.

For each of these options, if the condition given is not fulfilled, then the server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

If the condition is fulfilled, the server performs the request method as if the conditional request options were not present.

If the request would, without the conditional request options, result in anything other than a 2.xx or 4.12 response code, then any conditional request options MAY be ignored.

5.10.8.1. If-Match

The If-Match Option MAY be used to make a request conditional on the current existence or value of an ETag for one or more representations of the target resource. If-Match is generally useful for resource update requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource (i.e., the "lost update" problem).

The value of an If-Match option is either an ETag or the empty string. An If-Match option with an ETag matches a representation with that exact ETag. An If-Match option with an empty value matches any existing representation (i.e., it places the precondition on the existence of any current representation for the target resource).

The If-Match Option can occur multiple times. If any of the options match, then the condition is fulfilled.

If there is one or more If-Match Option, but none of the options match, then the condition is not fulfilled.

5.10.8.2. If-None-Match

The If-None-Match Option MAY be used to make a request conditional on the non-existence of the target resource. If-None-Match is useful for resource creation requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource. The If-None-Match Option carries no value.

If the target resource does exist, then the condition is not fulfilled.

(It is not very useful to combine If-Match and If-None-Match options in one request, because the condition will then never be fulfilled.)

5.10.9. Size1 Option

The Size1 option provides size information about the resource representation in a request. The option value is an integer number of bytes. Its main use is with block-wise transfers [I-D.ietf-core-block]. In the present specification, it is used in 4.13 responses (Section 5.9.2.9) to indicate the maximum size of request entity that the server is able and willing to handle.

6. CoAP URIs

CoAP uses the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. Resources are organized hierarchically and governed by a potential CoAP origin server listening for CoAP requests ("coap") or DTLS-secured CoAP requests ("coaps") on a given UDP port. The CoAP server is identified via the generic syntax's authority component, which includes a host component and optional UDP port number. The remainder of the URI is considered to be identifying a resource which can be operated on by the methods defined by the CoAP protocol. The "coap" and "coaps" URI schemes can thus be compared to the "http" and "https" URI schemes respectively.

The syntax of the "coap" and "coaps" URI schemes is specified in this section in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", "query", "segment", "IP-literal", "IPv4address" and "reg-name" are adopted from [RFC3986].

Implementation Note: Unfortunately, over time the URI format has acquired significant complexity. Implementers are encouraged to examine [RFC3986] closely. E.g., the ABNF for IPv6 addresses is more complicated than maybe expected. Also, implementers should take care to perform the processing of percent decoding/encoding exactly once on the way from a URI to its decoded components or back. Percent encoding is crucial for data transparency, but may lead to unusual results such as a slash in a path component.

6.1. coap URI Scheme

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

If the host component is provided as an IP-literal or IPv4address, then the CoAP server can be reached at that IP address. If host is a registered name, then that name is considered an indirect identifier and the endpoint might use a name resolution service, such as DNS, to find the address of that host. The host MUST NOT be empty; if a URI is received with a missing authority or an empty host, then it MUST be considered invalid. The port subcomponent indicates the UDP port at which the CoAP server is located. If it is empty or not given, then the default port 5683 is assumed.

The path identifies a resource within the scope of the host and port. It consists of a sequence of path segments separated by a slash character (U+002F SOLIDUS "/").

The query serves to further parameterize the resource. It consists of a sequence of arguments separated by an ampersand character

(U+0026 AMPERSAND "&"). An argument is often in the form of a "key=value" pair.

The "coap" URI scheme supports the path prefix `"/.well-known/"` defined by [RFC5785] for "well-known locations" in the name-space of a host. This enables discovery of policy or other information about a host ("site-wide metadata"), such as hosted resources (see Section 7).

Application designers are encouraged to make use of short, but descriptive URIs. As the environments that CoAP is used in are usually constrained for bandwidth and energy, the trade-off between these two qualities should lean towards the shortness, without ignoring descriptiveness.

6.2. coaps URI Scheme

```
coaps-URI = "coaps:" "/" host [ ":" port ] path-abempty
           [ "?" query ]
```

All of the requirements listed above for the "coap" scheme are also requirements for the "coaps" scheme, except that a default UDP port of [IANA_TBD_PORT] is assumed if the port subcomponent is empty or not given, and the UDP datagrams MUST be secured through the use of DTLS as described in Section 9.1.

Considerations for caching of responses to "coaps" identified requests are discussed in Section 11.2.

Resources made available via the "coaps" scheme have no shared identity with the "coap" scheme even if their resource identifiers indicate the same authority (the same host listening to the same UDP port). They are distinct name spaces and are considered to be distinct origin servers.

6.3. Normalization and Comparison Rules

Since the "coap" and "coaps" schemes conform to the URI generic syntax, such URIs are normalized and compared according to the algorithm defined in [RFC3986], Section 6, using the defaults described above for each scheme.

If the port is equal to the default port for a scheme, the normal form is to elide the port subcomponent. Likewise, an empty path component is equivalent to an absolute path of `"/"`, so the normal form is to provide a path of `"/"` instead. The scheme and host are case-insensitive and normally provided in lowercase; IP-literals are

in recommended form [RFC5952]; all other components are compared in a case-sensitive manner. Characters other than those in the "reserved" set are equivalent to their percent-encoded bytes (see [RFC3986], Section 2.1): the normal form is to not encode them.

For example, the following three URIs are equivalent, and cause the same options and option values to appear in the CoAP messages:

```
coap://example.com:5683/~sensors/temp.xml
coap://EXAMPLE.com/%7Eensors/temp.xml
coap://EXAMPLE.com:/%7esensors/temp.xml
```

6.4. Decomposing URIs into Options

The steps to parse a request's options from a string `|url|` are as follows. These steps either result in zero or more of the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in the request, or they fail.

1. If the `|url|` string is not an absolute URI ([RFC3986]), then fail this algorithm.
2. Resolve the `|url|` string using the process of reference resolution defined by [RFC3986]. At this stage the URL is in ASCII encoding [RFC0020], even though the decoded components will be interpreted in UTF-8 [RFC3629] after step 5, 8 and 9.

NOTE: It doesn't matter what it is resolved relative to, since we already know it is an absolute URL at this point.

3. If `|url|` does not have a `<scheme>` component whose value, when converted to ASCII lowercase, is "coap" or "coaps", then fail this algorithm.
4. If `|url|` has a `<fragment>` component, then fail this algorithm.
5. If the `<host>` component of `|url|` does not represent the request's destination IP address as an IP-literal or IPv4address, include a Uri-Host Option and let that option's value be the value of the `<host>` component of `|url|`, converted to ASCII lowercase, and then converting all percent-encodings ("% followed by two hexadecimal digits) to the corresponding characters.

NOTE: In the usual case where the request's destination IP address is derived from the host part, this ensures that a Uri-Host Option is only used for a `<host>` component of the form reg-name.

6. If `|url|` has a `<port>` component, then let `|port|` be that component's value interpreted as a decimal integer; otherwise, let `|port|` be the default port for the scheme.
7. If `|port|` does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be `|port|`.
8. If the value of the `<path>` component of `|url|` is empty or consists of a single slash character (U+002F SOLIDUS "/"), then move to the next step.

Otherwise, for each segment in the `<path>` component, include a Uri-Path Option and let that option's value be the segment (not including the delimiting slash characters) after converting each percent-encoding ("% followed by two hexadecimal digits) to the corresponding byte.

9. If `|url|` has a `<query>` component, then, for each argument in the `<query>` component, include a Uri-Query Option and let that option's value be the argument (not including the question mark and the delimiting ampersand characters) after converting each percent-encoding to the corresponding byte.

Note that these rules completely resolve any percent-encoding.

6.5. Composing URIs from Options

The steps to construct a URI from a request's options are as follows. These steps either result in a URI, or they fail. In these steps, percent-encoding a character means replacing each of its (UTF-8 encoded) bytes by a "%" character followed by two hexadecimal digits representing the byte, where the digits A-F are in upper case (as defined in [RFC3986] Section 2.1; to reduce variability, the hexadecimal notation for percent-encoding in CoAP URIs MUST use uppercase letters). The definitions of "unreserved" and "sub-delims" are adopted from [RFC3986].

1. If the request is secured using DTLS, let `|url|` be the string "coaps://". Otherwise, let `|url|` be the string "coap://".
2. If the request includes a Uri-Host Option, let `|host|` be that option's value, where any non-ASCII characters are replaced by their corresponding percent-encoding. If `|host|` is not a valid reg-name or IP-literal or IPv4address, fail the algorithm. If the request does not include a Uri-Host Option, let `|host|` be the IP-literal (making use of the conventions of [RFC5952]) or IPv4address representing the request's destination IP address.

3. Append |host| to |url|.
4. If the request includes a Uri-Port Option, let |port| be that option's value. Otherwise, let |port| be the request's destination UDP port.
5. If |port| is not the default port for the scheme, then append a single U+003A COLON character (:) followed by the decimal representation of |port| to |url|.
6. Let |resource name| be the empty string. For each Uri-Path Option in the request, append a single character U+002F SOLIDUS (/) followed by the option's value to |resource name|, after converting any character that is not either in the "unreserved" set, "sub-delims" set, a U+003A COLON (:) or U+0040 COMMERCIAL AT (@) character, to its percent-encoded form.
7. If |resource name| is the empty string, set it to a single character U+002F SOLIDUS (/).
8. For each Uri-Query Option in the request, append a single character U+003F QUESTION MARK (?) (first option) or U+0026 AMPERSAND (&) (subsequent options) followed by the option's value to |resource name|, after converting any character that is not either in the "unreserved" set, "sub-delims" set (except U+0026 AMPERSAND (&)), a U+003A COLON (:), U+0040 COMMERCIAL AT (@), U+002F SOLIDUS (/) or U+003F QUESTION MARK (?) character, to its percent-encoded form.
9. Append |resource name| to |url|.
10. Return |url|.

Note that these steps have been designed to lead to a URI in normal form (see Section 6.3).

7. Discovery

7.1. Service Discovery

As a part of discovering the services offered by a CoAP server, a client has to learn about the endpoint used by a server.

A server is discovered by a client by the client (knowing or) learning a URI that references a resource in the namespace of the server. Alternatively, clients can use Multicast CoAP (see Section 8) and the "All CoAP Nodes" multicast address to find CoAP servers.

Unless the port subcomponent in a "coap" or "coaps" URI indicates the UDP port at which the CoAP server is located, the server is assumed to be reachable at the default port.

The CoAP default port number 5683 MUST be supported by a server that offers resources for resource discovery (see Section 7.2 below) and SHOULD be supported for providing access to other resources. The default port number [IANA_TBD_PORT] for DTLS-secured CoAP MAY be supported by a server for resource discovery and for providing access to other resources. In addition other endpoints may be hosted at other ports, e.g. in the dynamic port space.

Implementation Note: When a CoAP server is hosted by a 6LoWPAN node, header compression efficiency is improved when it also supports a port number in the 61616-61631 compressed UDP port space defined in [RFC4944] (note that, as its UDP port differs from the default port, it is a different endpoint from the server at the default port).

7.2. Resource Discovery

The discovery of resources offered by a CoAP endpoint is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. To maximize interoperability in a CoRE environment, a CoAP endpoint SHOULD support the CoRE Link Format of discoverable resources as described in [RFC6690], except where fully manual configuration is desired. It is up to the server which resources are made discoverable (if any).

7.2.1. 'ct' Attribute

This section defines a new Web Linking [RFC5988] attribute for use with [RFC6690]. The Content-Format code "ct" attribute provides a hint about the Content-Formats this resource returns. Note that this is only a hint, and does not override the Content-Format Option of a CoAP response obtained by actually requesting the representation of the resource. The value is in the CoAP identifier code format as a decimal ASCII integer and MUST be in the range of 0-65535 (16-bit unsigned integer). For example application/xml would be indicated as "ct=41". If no Content-Format code attribute is present then nothing about the type can be assumed. The Content-Format code attribute MAY include a space-separated sequence of Content-Format codes, indicating that multiple content-formats are available. The syntax of the attribute value is summarized in the production ct-value in Figure 12, where cardinal, SP and DQUOTE are defined as in [RFC6690].


```
ct-value = cardinal
          / DQUOTE cardinal *( 1*SP cardinal ) DQUOTE
```

Figure 12

8. Multicast CoAP

CoAP supports making requests to a IP multicast group. This is defined by a series of deltas to Unicast CoAP. A more general discussion of group communication with CoAP is in [I-D.ietf-core-groupcomm].

CoAP endpoints that offer services that they want other endpoints to be able to find using multicast service discovery, join one or more of the appropriate all-CoAP-nodes multicast addresses (Section 12.8) and listen on the default CoAP port. Note that an endpoint might receive multicast requests on other multicast addresses, including the all-nodes IPv6 address (or via broadcast on IPv4); an endpoint MUST therefore be prepared to receive such messages but MAY ignore them if multicast service discovery is not desired.

8.1. Messaging Layer

A multicast request is characterized by being transported in a CoAP message that is addressed to an IP multicast address instead of a CoAP endpoint. Such multicast requests MUST be Non-confirmable.

A server SHOULD be aware that a request arrived via multicast, e.g. by making use of modern APIs such as IPV6_RECVPKTINFO [RFC3542], if available.

To avoid an implosion of error responses, when a server is aware that a request arrived via multicast, it MUST NOT return a RST in reply to NON. If it is not aware, it MAY return a RST in reply to NON as usual. Because such a Reset message will look identical to an RST for a unicast message from the sender, the sender MUST avoid using a Message ID that is also still active from this endpoint with any unicast endpoint that might receive the multicast message.

At the time of writing, multicast messages can only be carried in UDP, not in DTLS. This means that the security modes defined for CoAP in this document are not applicable to multicast.

8.2. Request/Response Layer

When a server is aware that a request arrived via multicast, the server MAY always ignore the request, in particular if it doesn't have anything useful to respond (e.g., if it only has an empty payload or an error response). The decision for this may depend on the application. (For example, in [RFC6690] query filtering, a server should not respond to a multicast request if the filter does not match. More examples are in [I-D.ietf-core-groupcomm].)

If a server does decide to respond to a multicast request, it should not respond immediately. Instead, it should pick a duration for the period of time during which it intends to respond. For purposes of this exposition, we call the length of this period the *Leisure*. The specific value of this *Leisure* may depend on the application, or MAY be derived as described below. The server SHOULD then pick a random point of time within the chosen *Leisure* period to send back the unicast response to the multicast request. If further responses need to be sent based on the same multicast address membership, a new *leisure* period starts at the earliest after the previous one finishes.

To compute a value for *Leisure*, the server should have a group size estimate *G*, a target data transfer rate *R* (which both should be chosen conservatively) and an estimated response size *S*; a rough lower bound for *Leisure* can then be computed as

$$lb_Leisure = S * G / R$$

E.g., for a multicast request with link-local scope on an 2.4 GHz IEEE 802.15.4 (6LoWPAN) network, *G* could be (relatively conservatively) set to 100, *S* to 100 bytes, and the target rate to 8 kbit/s = 1 kB/s. The resulting lower bound for the *Leisure* is 10 seconds.

If a CoAP endpoint does not have suitable data to compute a value for *Leisure*, it MAY resort to `DEFAULT_LEISURE`.

When matching a response to a multicast request, only the token MUST match; the source endpoint of the response does not need to (and will not) be the same as the destination endpoint of the original request.

For the purposes of interpreting the `Location-*` options and any links embedded in the representation and, the request URI (base URI) relative to which the response is interpreted, is formed by replacing the multicast address in the Host component of the original request URI by the literal IP address of the endpoint actually responding.

8.2.1. Caching

When a client makes a multicast request, it always makes a new request to the multicast group (since there may be new group members that joined meanwhile or ones that did not get the previous request). It MAY update a cache with the received responses. Then it uses both cached-still-fresh and 'new' responses as the result of the request.

A response received in reply to a GET request to a multicast group MAY be used to satisfy a subsequent request on the related unicast request URI. The unicast request URI is obtained by replacing the authority part of the request URI with the transport layer source address of the response message.

A cache MAY revalidate a response by making a GET request on the related unicast request URI.

A GET request to a multicast group MUST NOT contain an ETag option. A mechanism to suppress responses the client already has is left for further study.

8.2.2. Proxying

When a forward-proxy receives a request with a Proxy-Uri or URI constructed from Proxy-Scheme that indicates a multicast address, the proxy obtains a set of responses as described above and sends all responses (both cached-still-fresh and new) back to the original client.

This specification does not provide a way to indicate the unicast-modified request URI (base URI) in responses thus forwarded. Proxying multicast requests is discussed in more detail in [I-D.ietf-core-groupcomm]; one proposal to address the base URI issue can be found in section 3 of [I-D.bormann-coap-misc].

9. Securing CoAP

This section defines the DTLS binding for CoAP.

During the provisioning phase, a CoAP device is provided with the security information that it needs, including keying materials and access control lists. This specification defines provisioning for the RawPublicKey mode in Section 9.1.3.2.1. At the end of the provisioning phase, the device will be in one of four security modes with the following information for the given mode. The NoSec and RawPublicKey modes are mandatory to implement for this specification.

NoSec: There is no protocol level security (DTLS is disabled).

Alternative techniques to provide lower layer security SHOULD be used when appropriate. The use of IPsec is discussed in [I-D.bormann-core-ipsec-for-coap]. Certain link layers in use with constrained nodes also provide link layer security, which may be appropriate with proper key management.

PreSharedKey: DTLS is enabled and there is a list of pre-shared keys [RFC4279] and each key includes a list of which nodes it can be used to communicate with as described in Section 9.1.3.1. At the extreme there may be one key for each node this CoAP node needs to communicate with (1:1 node/key ratio). Conversely, if more than two entities share a specific pre-shared key, this key only enables the entities to authenticate as a member of that group and not as a specific peer.

RawPublicKey: DTLS is enabled and the device has an asymmetric key pair without a certificate (a raw public key) that is validated using an out-of-band mechanism [I-D.ietf-tls-oob-pubkey] as described in Section 9.1.3.2. The device also has an identity calculated from the public key and a list of identities of the nodes it can communicate with.

Certificate: DTLS is enabled and the device has an asymmetric key pair with an X.509 certificate [RFC5280] that binds it to its Authority Name and is signed by some common trust root as described in Section 9.1.3.3. The device also has a list of root trust anchors that can be used for validating a certificate.

In the "NoSec" mode, the system simply sends the packets over normal UDP over IP and is indicated by the "coap" scheme and the CoAP default port. The system is secured only by keeping attackers from being able to send or receive packets from the network with the CoAP nodes; see Section 11.5 for an additional complication with this approach.

The other three security modes are achieved using DTLS and are indicated by the "coaps" scheme and DTLS-secured CoAP default port. The result is a security association that can be used to authenticate (within the limits of the security model) and, based on this authentication, authorize the communication partner. CoAP itself does not provide protocol primitives for authentication or authorization; where this is required, it can either be provided by communication security (i.e., IPsec or DTLS) or by object security (within the payload). Devices that require authorization for certain operations are expected to require one of these two forms of security. Necessarily, where an intermediary is involved, communication security only works when that intermediary is part of

the trust relationships; CoAP does not provide a way to forward different levels of authorization that clients may have with an intermediary to further intermediaries or origin servers -- it therefore may be required to perform all authorization at the first intermediary.

9.1. DTLS-secured CoAP

Just as HTTP is secured using Transport Layer Security (TLS) over TCP, CoAP is secured using Datagram TLS (DTLS) [RFC6347] over UDP (see Figure 13). This section defines the CoAP binding to DTLS, along with the minimal mandatory-to-implement configurations appropriate for constrained environments. The binding is defined by a series of deltas to Unicast CoAP. DTLS is in practice TLS with added features to deal with the unreliable nature of the UDP transport.

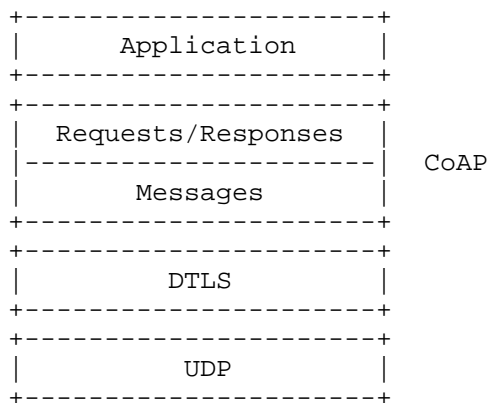


Figure 13: Abstract layering of DTLS-secured CoAP

In some constrained nodes (limited flash and/or RAM) and networks (limited bandwidth or high scalability requirements), and depending on the specific cipher suites in use, all modes of DTLS may not be applicable. Some DTLS cipher suites can add significant implementation complexity as well as some initial handshake overhead needed when setting up the security association. Once the initial handshake is completed, DTLS adds a limited per-datagram overhead of approximately 13 bytes, not including any initialization vectors/nonces (e.g., 8 bytes with `TLS_PSK_WITH_AES_128_CCM_8` [RFC6655]), integrity check values (e.g., 8 bytes with `TLS_PSK_WITH_AES_128_CCM_8` [RFC6655]) and padding required by the cipher suite. Whether and which mode of using DTLS is applicable for a CoAP-based application should be carefully weighed considering the specific cipher suites that may be applicable, and whether the session maintenance makes it

compatible with application flows and sufficient resources are available on the constrained nodes and for the added network overhead. (For some modes of using DTLS, this specification identifies a mandatory to implement cipher suite. This is an implementation requirement to maximize interoperability in those cases where these cipher suites are indeed appropriate. The specific security policies of an application may determine the actual (set of) cipher suites that can be used.) DTLS is not applicable to group keying (multicast communication); however, it may be a component in a future group key management protocol.

9.1.1. Messaging Layer

The endpoint acting as the CoAP client should also act as the DTLS client. It should initiate a session to the server on the appropriate port. When the DTLS handshake has finished, the client may initiate the first CoAP request. All CoAP messages MUST be sent as DTLS "application data".

The following rules are added for matching an ACK or RST to a CON message or a RST to a NON message: The DTLS session MUST be the same and the epoch MUST be the same.

A message is the same when it is sent within the same DTLS session and same epoch and has the same Message ID.

Note: When a Confirmable message is retransmitted, a new DTLS sequence_number is used for each attempt, even though the CoAP Message ID stays the same. So a recipient still has to perform deduplication as described in Section 4.5. Retransmissions MUST NOT be performed across epochs.

DTLS connections in RawPublicKey and Certificate mode are set up using mutual authentication so they can remain up and be reused for future message exchanges in either direction. Devices can close a DTLS connection when they need to recover resources but in general they should keep the connection up for as long as possible. Closing the DTLS connection after every CoAP message exchange is very inefficient.

9.1.2. Request/Response Layer

The following rules are added for matching a response to a request: The DTLS session MUST be the same and the epoch MUST be the same.

This means the response to a DTLS secured request MUST always be DTLS secured using the same security session and epoch. Any attempt to supply a NoSec response to a DTLS request simply does not match the

request and (unless it does match an unrelated NoSec request) therefore MUST be rejected.

9.1.3. Endpoint Identity

Devices SHOULD support the Server Name Indication (SNI) to indicate their Authority Name in the SNI HostName field as defined in Section 3 of [RFC6066]. This is needed so that when a host that acts as a virtual server for multiple Authorities receives a new DTLS connection, it knows which keys to use for the DTLS session.

9.1.3.1. Pre-Shared Keys

When forming a connection to a new node, the system selects an appropriate key based on which nodes it is trying to reach and then forms a DTLS session using a PSK (Pre-Shared Key) mode of DTLS. Implementations in these modes MUST support the mandatory to implement cipher suite TLS_PSK_WITH_AES_128_CCM_8 as specified in [RFC6655].

Depending on the commissioning model, applications may need to define an application profile for identity hints as required and detailed in [RFC4279] (Section 5.2) to enable the use of PSK identity hints.

The security considerations of [RFC4279] (Section 7) apply. In particular, applications should carefully weigh whether they need Perfect Forward Secrecy (PFS) or not and select an appropriate cipher suite (7.1). The entropy of the PSK must be sufficient to mitigate against brute-force and (where the PSK is not chosen randomly but by a human) dictionary attacks (7.2). The cleartext communication of client identities may leak data or compromise privacy (7.3).

9.1.3.2. Raw Public Key Certificates

In this mode the device has an asymmetric key pair but without an X.509 certificate (called a raw public key); e.g., the asymmetric key pair is generated by the manufacturer and installed on the device (see also Section 11.6). A device MAY be configured with multiple raw public keys. The type and length of the raw public key depends on the cipher suite used. Implementations in RawPublicKey mode MUST support the mandatory to implement cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as specified in [I-D.mcgrew-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. The key used MUST be ECDSA-capable. The curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. Implementations MUST use the Supported Elliptic Curves Extension and Supported Point Format extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090]

can be used as an implementation method. Some guidance relevant to the implementation of this cipher suite can be found in [W3CXMLSEC]. The mechanism for using raw public keys with TLS is specified in [I-D.ietf-tls-oob-pubkey].

Implementation Note: Specifically, this means the extensions listed in Figure 14 with at least the values listed will be present in the DTLS handshake.

Extension: elliptic_curves
Type: elliptic_curves (0x000a)
Length: 4
Elliptic Curves Length: 2
Elliptic curves (1 curve)
Elliptic curve: secp256r1 (0x0017)

Extension: ec_point_formats
Type: ec_point_formats (0x000b)
Length: 2
EC point formats Length: 1
Elliptic curves point formats (1)
EC point format: uncompressed (0)

Extension: signature_algorithms
Type: signature_algorithms (0x000d)
Length: 4
Data (4 bytes): 00 02 04 03
HashAlgorithm: sha256 (4)
SignatureAlgorithm: ecdsa (3)

Figure 14: DTLS extensions present for
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

9.1.3.2.1. Provisioning

The RawPublicKey mode was designed to be easily provisioned in M2M deployments. It is assumed that each device has an appropriate asymmetric public key pair installed. An identifier is calculated by the endpoint from the public key as described in Section 2 of [RFC6920]. All implementations that support checking RawPublicKey identities MUST support at least the sha-256-120 mode (SHA-256 truncated to 120 bits). Implementations SHOULD support also longer length identifiers and MAY support shorter lengths. Note that the shorter lengths provide less security against attacks and their use is NOT RECOMMENDED.

Depending on how identifiers are given to the system that verifies them, support for URI, binary, and/or human-speakable format

[RFC6920] needs to be implemented. All implementations SHOULD support the binary mode and implementations that have a user interface SHOULD also support the human-speakable format.

During provisioning, the identifier of each node is collected, for example by reading a barcode on the outside of the device or by obtaining a pre-compiled list of the identifiers. These identifiers are then installed in the corresponding endpoint, for example an M2M data collection server. The identifier is used for two purposes, to associate the endpoint with further device information and to perform access control. During (initial and ongoing) provisioning, an access control list of identifiers the device may start DTLS sessions with SHOULD also be installed and maintained.

9.1.3.3. X.509 Certificates

Implementations in Certificate Mode MUST support the mandatory to implement cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as specified in [I-D.mcgregor-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. Namely, the certificate includes a SubjectPublicKeyInfo that indicates an algorithm of id-ecPublicKey with namedCurves secp256r1 [RFC5480]; the public key format is uncompressed [RFC5480]; the hash algorithm is SHA-256; if included the key usage extension indicates digitalSignature. Certificates MUST be signed with ECDSA using secp256r1, and the signature MUST use SHA-256. The key used MUST be ECDSA-capable. The curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. Implementations MUST use the Supported Elliptic Curves Extension and Supported Point Format extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090] can be used as an implementation method.

The Authority Name in the certificate would be built out of a long term unique identifier for the device such as the EUI-64 [EUI64]. The Authority Name could also be based on the FQDN that was used as the Host part of the CoAP URI. However, the device's IP address should not typically be used as the Authority name as it would change over time. The discovery process used in the system would build up the mapping between IP addresses of the given devices and the Authority Name for each device. Some devices could have more than one Authority and would need more than a single certificate.

When a new connection is formed, the certificate from the remote device needs to be verified. If the CoAP node has a source of absolute time, then the node SHOULD check that the validity dates of the certificate are within range. The certificate MUST be validated as appropriate for the security requirements, using functionality equivalent to the algorithm specified in [RFC5280] section 6. If the

certificate contains a SubjectAltName, then the Authority Name MUST match at least one of the authority names of any CoAP URI found in a field of URI type in the SubjectAltName set. If there is no SubjectAltName in the certificate, then the Authoritative Name MUST match the CN found in the certificate using the matching rules defined in [RFC2818] with the exception that certificates with wildcards are not allowed.

CoRE support for certificate status checking requires further study. As a mapping of OCSP [RFC2560] onto CoAP is not currently defined and OCSP may also not be easily applicable in all environments, an alternative approach may be using the TLS Certificate Status Request extension ([RFC6066] section 8, also known as "OCSP stapling") or preferably the Multiple Certificate Status Extension ([I-D.ietf-tls-multiple-cert-status-extension]), if available.

If the system has a shared key in addition to the certificate, then a cipher suite that includes the shared key such as TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA [RFC5489] SHOULD be used.

10. Cross-Protocol Proxying between CoAP and HTTP

CoAP supports a limited subset of HTTP functionality, and thus cross-protocol proxying to HTTP is straightforward. There might be several reasons for proxying between CoAP and HTTP, for example when designing a web interface for use over either protocol or when realizing a CoAP-HTTP proxy. Likewise, CoAP could equally be proxied to other protocols such as XMPP [RFC6120] or SIP [RFC3264]; the definition of these mechanisms is out of scope of this specification.

There are two possible directions to access a resource via a forward-proxy:

CoAP-HTTP Proxying: Enables CoAP clients to access resources on HTTP servers through an intermediary. This is initiated by including the Proxy-Uri or Proxy-Scheme Option with an "http" or "https" URI in a CoAP request to a CoAP-HTTP proxy.

HTTP-CoAP Proxying: Enables HTTP clients to access resources on CoAP servers through an intermediary. This is initiated by specifying a "coap" or "coaps" URI in the Request-Line of an HTTP request to an HTTP-CoAP proxy.

Either way, only the Request/Response model of CoAP is mapped to HTTP. The underlying model of Confirmable or Non-confirmable messages, etc., is invisible and MUST have no effect on a proxy function. The following sections describe the handling of requests to a forward-proxy. Reverse proxies are not specified as the proxy

function is transparent to the client with the proxy acting as if it was the origin server. However, similar considerations apply to reverse-proxies as to forward-proxies, and there generally will be an expectation that reverse-proxies operate in a similar way forward-proxies would. As an implementation note, HTTP client libraries may make it hard to operate an HTTP-CoAP forward proxy by not providing a way to put a CoAP URI on the HTTP Request-Line; reverse-proxying may therefore lead to wider applicability of a proxy. A separate specification may define a convention for URIs operating such a HTTP-CoAP reverse proxy [I-D.castellani-core-http-mapping].

10.1. CoAP-HTTP Proxying

If a request contains a Proxy-Uri or Proxy-Scheme Option with an 'http' or 'https' URI [RFC2616], then the receiving CoAP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated HTTP resource and return the result to the client. (See also Section 5.7 for how the request to the proxy is formulated, including security requirements.)

This section specifies for any CoAP request the CoAP response that the proxy should return to the client. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to an HTTP origin server.

Since HTTP and CoAP share the basic set of request methods, performing a CoAP request on an HTTP resource is not so different from performing it on a CoAP resource. The meanings of the individual CoAP methods when performed on HTTP resources are explained in the subsections of this section.

If the proxy is unable or unwilling to service a request with an HTTP URI, a 5.05 (Proxying Not Supported) response is returned to the client. If the proxy services the request by interacting with a third party (such as the HTTP origin server) and is unable to obtain a result within a reasonable time frame, a 5.04 (Gateway Timeout) response is returned; if a result can be obtained but is not understood, a 5.02 (Bad Gateway) response is returned.

10.1.1. GET

The GET method requests the proxy to return a representation of the HTTP resource identified by the request URI.

Upon success, a 2.05 (Content) response code SHOULD be returned. The payload of the response MUST be a representation of the target HTTP resource, and the Content-Format Option be set accordingly. The

response MUST indicate a Max-Age value that is no greater than the remaining time the representation can be considered fresh. If the HTTP entity has an entity tag, the proxy SHOULD include an ETag Option in the response and process ETag Options in requests as described below.

A client can influence the processing of a GET request by including the following option:

Accept: The request MAY include an Accept Option, identifying the preferred response content-format.

ETag: The request MAY include one or more ETag Options, identifying responses that the client has stored. This requests the proxy to send a 2.03 (Valid) response whenever it would send a 2.05 (Content) response with an entity tag in the requested set otherwise. Note that CoAP ETags are always strong ETags in the HTTP sense; CoAP does not have the equivalent of HTTP weak ETags, and there is no good way to make use of these in a cross-proxy.

10.1.2. PUT

The PUT method requests the proxy to update or create the HTTP resource identified by the request URI with the enclosed representation.

If a new resource is created at the request URI, a 2.01 (Created) response MUST be returned to the client. If an existing resource is modified, a 2.04 (Changed) response MUST be returned to indicate successful completion of the request.

10.1.3. DELETE

The DELETE method requests the proxy to delete the HTTP resource identified by the request URI at the HTTP origin server.

A 2.02 (Deleted) response MUST be returned to client upon success or if the resource does not exist at the time of the request.

10.1.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the HTTP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 2.04 (Changed) response MUST be returned to the client. If a resource has been created on the origin server, a 2.01 (Created) response MUST be returned.

10.2. HTTP-CoAP Proxying

If an HTTP request contains a Request-URI with a 'coap' or 'coaps' URI, then the receiving HTTP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated CoAP resource and return the result to the client.

This section specifies for any HTTP request the HTTP response that the proxy should return to the client. Unless otherwise specified all the statements made are RECOMMENDED behavior; some highly constrained implementations may need to resort to shortcuts. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to a CoAP origin server. The meanings of the individual HTTP methods when performed on CoAP resources are explained in the subsections of this section.

If the proxy is unable or unwilling to service a request with a CoAP URI, a 501 (Not Implemented) response is returned to the client. If the proxy services the request by interacting with a third party (such as the CoAP origin server) and is unable to obtain a result within a reasonable time frame, a 504 (Gateway Timeout) response is returned; if a result can be obtained but is not understood, a 502 (Bad Gateway) response is returned.

10.2.1. OPTIONS and TRACE

As the OPTIONS and TRACE methods are not supported in CoAP a 501 (Not Implemented) error MUST be returned to the client.

10.2.2. GET

The GET method requests the proxy to return a representation of the CoAP resource identified by the Request-URI.

Upon success, a 200 (OK) response is returned. The payload of the response MUST be a representation of the target CoAP resource, and the Content-Type and Content-Encoding header fields be set accordingly. The response MUST indicate a max-age directive that indicates a value no greater than the remaining time the representation can be considered fresh. If the CoAP response has an ETag option, the proxy should include an ETag header field in the response.

A client can influence the processing of a GET request by including the following options:

Accept: The most preferred Media-type of the HTTP Accept header field in a request is mapped to a CoAP Accept option. HTTP Accept Media-type ranges, parameters and extensions are not supported by the CoAP Accept option. If the proxy cannot send a response which is acceptable according to the combined Accept field value, then the proxy sends a 406 (not acceptable) response. The proxy MAY then retry the request with further Media-types from the HTTP Accept header field.

Conditional GETs: Conditional HTTP GET requests that include an "If-Match" or "If-None-Match" request-header field can be mapped to a corresponding CoAP request. The "If-Modified-Since" and "If-Unmodified-Since" request-header fields are not directly supported by CoAP, but are implemented locally by a caching proxy.

10.2.3. HEAD

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response.

Although there is no direct equivalent of HTTP's HEAD method in CoAP, an HTTP-CoAP proxy responds to HEAD requests for CoAP resources, and the HTTP headers are returned without a message-body.

Implementation Note: An HTTP-CoAP proxy may want to try using a block-wise transfer [I-D.ietf-core-block] option to minimize the amount of data actually transferred, but needs to be prepared for the case that the origin server does not support block-wise transfers.

10.2.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the CoAP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 200 (OK) or 204 (No Content) response MUST be returned to the client. If a resource has been created on the origin server, a 201 (Created) response MUST be returned.

If any of the Location-* Options are present in the CoAP response, a Location header field constructed from the values of these options is returned.

10.2.5. PUT

The PUT method requests the proxy to update or create the CoAP resource identified by the Request-URI with the enclosed representation.

If a new resource is created at the Request-URI, a 201 (Created) response is returned to the client. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes is sent to indicate successful completion of the request.

10.2.6. DELETE

The DELETE method requests the proxy to delete the CoAP resource identified by the Request-URI at the CoAP origin server.

A successful response is 200 (OK) if the response includes an entity describing the status or 204 (No Content) if the action has been enacted but the response does not include an entity.

10.2.7. CONNECT

This method can not currently be satisfied by an HTTP-CoAP proxy function as TLS to DTLS tunneling has not yet been specified. For now, a 501 (Not Implemented) error is returned to the client.

11. Security Considerations

This section analyzes the possible threats to the protocol. It is meant to inform protocol and application developers about the security limitations of CoAP as described in this document. As CoAP realizes a subset of the features in HTTP/1.1, the security considerations in Section 15 of [RFC2616] are also pertinent to CoAP. This section concentrates on describing limitations specific to CoAP.

11.1. Protocol Parsing, Processing URIs

A network-facing application can exhibit vulnerabilities in its processing logic for incoming packets. Complex parsers are well-known as a likely source of such vulnerabilities, such as the ability to remotely crash a node, or even remotely execute arbitrary code on it. CoAP attempts to narrow the opportunities for introducing such vulnerabilities by reducing parser complexity, by giving the entire range of encodable values a meaning where possible, and by

aggressively reducing complexity that is often caused by unnecessary choice between multiple representations that mean the same thing. Much of the URI processing has been moved to the clients, further reducing the opportunities for introducing vulnerabilities into the servers. Even so, the URI processing code in CoAP implementations is likely to be a large source of remaining vulnerabilities and should be implemented with special care. CoAP access control implementations need to ensure they don't introduce vulnerabilities through discrepancies between the code deriving access control decisions from a URI and the code finally serving up the resource addressed by the URI. The most complex parser remaining could be the one for the CoRE Link Format, although this also has been designed with a goal of reduced implementation complexity [RFC6690]. (See also section 15.2 of [RFC2616].)

11.2. Proxying and Caching

As mentioned in 15.7 of [RFC2616], proxies are by their very nature men-in-the-middle, breaking any IPsec or DTLS protection that a direct CoAP message exchange might have. They are therefore interesting targets for breaking confidentiality or integrity of CoAP message exchanges. As noted in [RFC2616], they are also interesting targets for breaking availability.

The threat to confidentiality and integrity of request/response data is amplified where proxies also cache. Note that CoAP does not define any of the cache-suppressing Cache-Control options that HTTP/1.1 provides to better protect sensitive data.

For a caching implementation, any access control considerations that would apply to making the request that generated the cache entry also need to be applied to the value in the cache. This is relevant for clients that implement multiple security domains, as well as for proxies that may serve multiple clients. Also, a caching proxy **MUST NOT** make cached values available to requests that have lesser transport security properties than to which it would make available the process of forwarding the request in the first place.

Unlike the "coap" scheme, responses to "coaps" identified requests are never "public" and thus **MUST NOT** be reused for shared caching unless the cache is able to make equivalent access control decisions to the ones that led to the cached entry. They can, however, be reused in a private cache if the message is cacheable by default in CoAP.

Finally, a proxy that fans out Separate Responses (as opposed to Piggy-backed Responses) to multiple original requesters may provide additional amplification (see Section 11.3).

11.3. Risk of amplification

CoAP servers generally reply to a request packet with a response packet. This response packet may be significantly larger than the request packet. An attacker might use CoAP nodes to turn a small attack packet into a larger attack packet, an approach known as amplification. There is therefore a danger that CoAP nodes could become implicated in denial of service (DoS) attacks by using the amplifying properties of the protocol: An attacker that is attempting to overload a victim but is limited in the amount of traffic it can generate, can use amplification to generate a larger amount of traffic.

This is particularly a problem in nodes that enable NoSec access, that are accessible from an attacker and can access potential victims (e.g. on the general Internet), as the UDP protocol provides no way to verify the source address given in the request packet. An attacker need only place the IP address of the victim in the source address of a suitable request packet to generate a larger packet directed at the victim.

As a mitigating factor, many constrained networks will only be able to generate a small amount of traffic, which may make CoAP nodes less attractive for this attack. However, the limited capacity of the constrained network makes the network itself a likely victim of an amplification attack.

Therefore, large amplification factors SHOULD NOT be provided in the response if the request is not authenticated. A CoAP server can reduce the amount of amplification it provides to an attacker by using slicing/blocking modes of CoAP [I-D.ietf-core-block] and offering large resource representations only in relatively small slices. E.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

CoAP also supports the use of multicast IP addresses in requests, an important requirement for M2M. Multicast CoAP requests may be the source of accidental or deliberate denial of service attacks, especially over constrained networks. This specification attempts to reduce the amplification effects of multicast requests by limiting when a response is returned. To limit the possibility of malicious use, CoAP servers SHOULD NOT accept multicast requests that can not be authenticated in some way, cryptographically or by some multicast boundary limiting the potential sources. If possible a CoAP server SHOULD limit the support for multicast requests to the specific resources where the feature is required.

On some general purpose operating systems providing a Posix-style API, it is not straightforward to find out whether a packet received was addressed to a multicast address. While many implementations will know whether they have joined a multicast group, this creates a problem for packets addressed to multicast addresses of the form FF0x::1, which are received by every IPv6 node. Implementations SHOULD make use of modern APIs such as IPV6_RECVPKTINFO [RFC3542], if available, to make this determination.

11.4. IP Address Spoofing Attacks

Due to the lack of a handshake in UDP, a rogue endpoint which is free to read and write messages carried by the constrained network (i.e. NoSec or PreSharedKey deployments with nodes/key ratio > 1:1), may easily attack a single endpoint, a group of endpoints, as well as the whole network e.g. by:

1. spoofing RST in response to a CON or NON message, thus making an endpoint "deaf"; or
2. spoofing an ACK in response to a CON message, thus potentially preventing the sender of the CON message from retransmitting, and drowning out the actual response; or
3. spoofing the entire response with forged payload/options (this has different levels of impact: from single response disruption, to much bolder attacks on the supporting infrastructure, e.g. poisoning proxy caches, or tricking validation / lookup interfaces in resource directories and, more generally, any component that stores global network state and uses CoAP as the messaging facility to handle state set/update's is a potential target.); or
4. spoofing a multicast request for a target node which may result in both network congestion/collapse and victim DoS'ing / forced wakeup from sleeping; or
5. spoofing observe messages, etc.

Response spoofing by off-path attackers can be detected and mitigated even without transport layer security by choosing a non-trivial, randomized token in the request (Section 5.3.1). [RFC4086] discusses randomness requirements for security.

In principle, other kinds of spoofing can be detected by CoAP only in case CON semantics is used, because of unexpected ACK/RSTs coming from the deceived endpoint. But this imposes keeping track of the used Message IDs which is not always possible, and moreover detection

becomes available usually after the damage is already done. This kind of attack can be prevented using security modes other than NoSec.

With or without source address spoofing, a client can attempt to overload a server by sending requests, preferably complex ones, to a server; address spoofing makes tracing back, and blocking, this attack harder. Given that the cost of a CON request is small, this attack can easily be executed. Under this attack, a constrained node with limited total energy available may exhaust that energy much more quickly than planned (battery depletion attack). Also, if the client uses a Confirmable message and the server responds with a Confirmable separate response to a (possibly spoofed) address that does not respond, the server will have to allocate buffer and retransmission logic for each response up to the exhaustion of MAX_TRANSMIT_SPAN, making it more likely that it runs out of resources for processing legitimate traffic. The latter problem can be mitigated somewhat by limiting the rate of responses as discussed in Section 4.7. An attacker could also spoof the address of a legitimate client, which, if the server uses separate responses, might block legitimate responses to that client because of NSTART=1. All these attacks can be prevented using a security mode other than NoSec, leaving only attacks on the security protocol.

11.5. Cross-Protocol Attacks

The ability to incite a CoAP endpoint to send packets to a fake source address can be used not only for amplification, but also for cross-protocol attacks against a victim listening to UDP packets at a given address (IP address and port):

- o the attacker sends a message to a CoAP endpoint with the given address as the fake source address,
- o the CoAP endpoint replies with a message to the given source address,
- o the victim at the given address receives a UDP packet that it interprets according to the rules of a different protocol.

This may be used to circumvent firewall rules that prevent direct communication from the attacker to the victim, but happen to allow communication from the CoAP endpoint (which may also host a valid role in the other protocol) to the victim.

Also, CoAP endpoints may be the victim of a cross-protocol attack generated through an endpoint of another UDP-based protocol such as DNS. In both cases, attacks are possible if the security properties

of the endpoints rely on checking IP addresses (and firewalling off direct attacks sent from outside using fake IP addresses). In general, because of their lack of context, UDP-based protocols are relatively easy targets for cross-protocol attacks.

Finally, CoAP URIs transported by other means could be used to incite clients to send messages to endpoints of other protocols.

One mitigation against cross-protocol attacks is strict checking of the syntax of packets received, combined with sufficient difference in syntax. As an example, it might help if it were difficult to incite a DNS server to send a DNS response that would pass the checks of a CoAP endpoint. Unfortunately, the first two bytes of a DNS reply are an ID that can be chosen by the attacker, which map into the interesting part of the CoAP header, and the next two bytes are then interpreted as CoAP's Message ID (i.e., any value is acceptable). The DNS count words may be interpreted as multiple instances of a (non-existent, but elective) CoAP option 0, or possibly as a Token. The echoed query finally may be manufactured by the attacker to achieve a desired effect on the CoAP endpoint; the response added by the server (if any) might then just be interpreted as added payload.

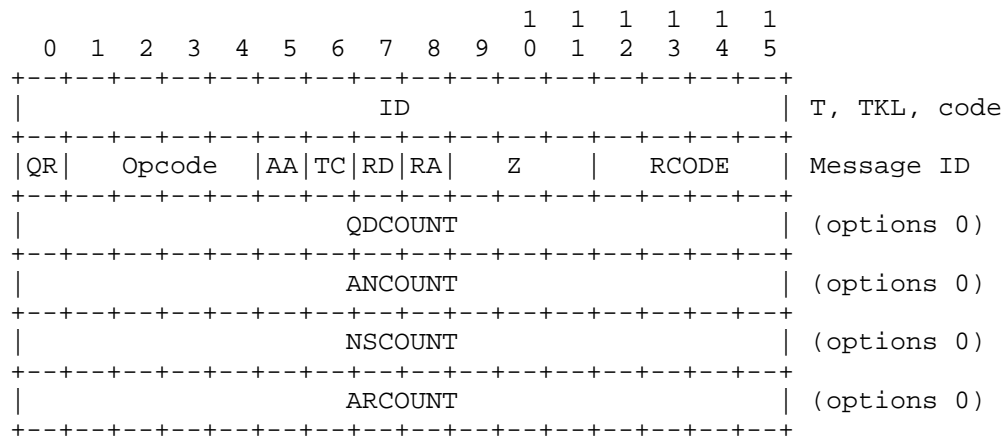


Figure 15: DNS Header vs. CoAP Message

In general, for any pair of protocols, one of the protocols can very well have been designed in a way that enables an attacker to cause the generation of replies that look like messages of the other protocol. It is often much harder to ensure or prove the absence of viable attacks than to generate examples that may not yet completely enable an attack but might be further developed by more creative minds. Cross-protocol attacks can therefore only be completely

mitigated if endpoints don't authorize actions desired by an attacker just based on trusting the source IP address of a packet. Conversely, a NoSec environment that completely relies on a firewall for CoAP security not only needs to firewall off the CoAP endpoints but also all other endpoints that might be incited to send UDP messages to CoAP endpoints using some other UDP-based protocol.

In addition to the considerations above, the security considerations for DTLS with respect to cross-protocol attacks apply. E.g., if the same DTLS security association ("connection") is used to carry data of multiple protocols, DTLS no longer provides protection against cross-protocol attacks between these protocols.

11.6. Constrained node considerations

Implementers on constrained nodes often find themselves without a good source of entropy [RFC4086]. If that is the case, the node **MUST** NOT be used for processes that require good entropy, such as key generation. Instead, keys should be generated externally and added to the device during manufacturing or commissioning.

Due to their low processing power, constrained nodes are particularly susceptible to timing attacks. Special care must be taken in implementation of cryptographic primitives.

Large numbers of constrained nodes will be installed in exposed environments and will have little resistance to tampering, including recovery of keying materials. This needs to be considered when defining the scope of credentials assigned to them. In particular, assigning a shared key to a group of nodes may make any single constrained node a target for subverting the entire group.

12. IANA Considerations

12.1. CoAP Code Registries

This document defines two sub-registries for the values of the Code field in the CoAP header within the Constrained RESTful Environments (CoRE) Parameters ("CoRE Parameters") registry.

Values in the two sub-registries are eight-bit values notated as three decimal digits c.dd separated by a period between the first and the second digit; the first digit c is between 0 and 7 and denotes the code class; the second and third digit dd denote a decimal number between 00 and 31 for the detail.

All Code values are assigned by sub-registries according to the following ranges:

0.00 Indicates an Empty message (see Section 4.1).

0.01-0.31 Indicates a request. Values in this range are assigned by the "CoAP Method Codes" sub-registry (see Section 12.1.1).

1.00-1.31 Reserved

2.00-5.31 Indicates a response. Values in this range are assigned by the "CoAP Response Codes" sub-registry (see Section 12.1.2).

6.00-7.31 Reserved

12.1.1. Method Codes

The name of the sub-registry is "CoAP Method Codes".

Each entry in the sub-registry must include the Method Code in the range 0.01-0.31, the name of the method, and a reference to the method's documentation.

Initial entries in this sub-registry are as follows:

Code	Name	Reference
0.01	GET	[RFCXXXX]
0.02	POST	[RFCXXXX]
0.03	PUT	[RFCXXXX]
0.04	DELETE	[RFCXXXX]

Table 5: CoAP Method Codes

All other Method Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG approval" as described in [RFC5226].

The documentation of a method code should specify the semantics of a request with that code, including the following properties:

- o The response codes the method returns in the success case.
- o Whether the method is idempotent, safe, or both.

12.1.2. Response Codes

The name of the sub-registry is "CoAP Response Codes".

Each entry in the sub-registry must include the Response Code in the range 2.00-5.31, a description of the Response Code, and a reference to the Response Code's documentation.

Initial entries in this sub-registry are as follows:

Code	Description	Reference
2.01	Created	[RFCXXXX]
2.02	Deleted	[RFCXXXX]
2.03	Valid	[RFCXXXX]
2.04	Changed	[RFCXXXX]
2.05	Content	[RFCXXXX]
4.00	Bad Request	[RFCXXXX]
4.01	Unauthorized	[RFCXXXX]
4.02	Bad Option	[RFCXXXX]
4.03	Forbidden	[RFCXXXX]
4.04	Not Found	[RFCXXXX]
4.05	Method Not Allowed	[RFCXXXX]
4.06	Not Acceptable	[RFCXXXX]
4.12	Precondition Failed	[RFCXXXX]
4.13	Request Entity Too Large	[RFCXXXX]
4.15	Unsupported Content-Format	[RFCXXXX]
5.00	Internal Server Error	[RFCXXXX]
5.01	Not Implemented	[RFCXXXX]
5.02	Bad Gateway	[RFCXXXX]
5.03	Service Unavailable	[RFCXXXX]
5.04	Gateway Timeout	[RFCXXXX]
5.05	Proxying Not Supported	[RFCXXXX]

Table 6: CoAP Response Codes

The Response Codes 3.00-3.31 are Reserved for future use. All other Response Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG approval" as described in [RFC5226].

The documentation of a response code should specify the semantics of a response with that code, including the following properties:

- o The methods the response code applies to.
- o Whether payload is required, optional or not allowed.

- o The semantics of the payload. For example, the payload of a 2.05 (Content) response is a representation of the target resource; the payload in an error response is a human-readable diagnostic payload.
- o The format of the payload. For example, the format in a 2.05 (Content) response is indicated by the Content-Format Option; the format of the payload in an error response is always Net-Unicode text.
- o Whether the response is cacheable according to the freshness model.
- o Whether the response is validatable according to the validation model.
- o Whether the response causes a cache to mark responses stored for the request URI as not fresh.

12.2. Option Number Registry

This document defines a sub-registry for the Option Numbers used in CoAP options within the "CoRE Parameters" registry. The name of the sub-registry is "CoAP Option Numbers".

Each entry in the sub-registry must include the Option Number, the name of the option and a reference to the option's documentation.

Initial entries in this sub-registry are as follows:

Number	Name	Reference
0	(Reserved)	[RFCXXXX]
1	If-Match	[RFCXXXX]
3	Uri-Host	[RFCXXXX]
4	ETag	[RFCXXXX]
5	If-None-Match	[RFCXXXX]
7	Uri-Port	[RFCXXXX]
8	Location-Path	[RFCXXXX]
11	Uri-Path	[RFCXXXX]
12	Content-Format	[RFCXXXX]
14	Max-Age	[RFCXXXX]
15	Uri-Query	[RFCXXXX]
17	Accept	[RFCXXXX]
20	Location-Query	[RFCXXXX]
35	Proxy-Uri	[RFCXXXX]
39	Proxy-Scheme	[RFCXXXX]

60	Size1	[RFCXXXX]
128	(Reserved)	[RFCXXXX]
132	(Reserved)	[RFCXXXX]
136	(Reserved)	[RFCXXXX]
140	(Reserved)	[RFCXXXX]

Table 7: CoAP Option Numbers

The IANA policy for future additions to this sub-registry is split into three tiers as follows. The range of 0..255 is reserved for options defined by the IETF (IETF Review or IESG approval). The range of 256..2047 is reserved for commonly used options with public specifications (Specification Required). The range of 2048..64999 is for all other options including private or vendor specific ones, which undergo a Designated Expert review to help ensure that the option semantics are defined correctly. The option numbers between 65000 and 65535 inclusive are reserved for experiments. They are not meant for vendor specific use of any kind and MUST NOT be used in operational deployments.

Option Number	Policy [RFC5226]
0..255	IETF Review or IESG approval
256..2047	Specification Required
2048..64999	Designated Expert
65000..65535	Reserved for experiments

Table 8: CoAP Option Number Registry Policy

The documentation of an Option Number should specify the semantics of an option with that number, including the following properties:

- o The meaning of the option in a request.
- o The meaning of the option in a response.
- o Whether the option is critical or elective, as determined by the Option Number.
- o Whether the option is Safe-to-Forward, and, if yes, whether it is part of the Cache-Key, as determined by the Option Number (see Section 5.4.2).
- o The format and length of the option's value.

- o Whether the option must occur at most once or whether it can occur multiple times.
- o The default value, if any. For a critical option with a default value, a discussion on how the default value enables processing by implementations not implementing the critical option (Section 5.4.4).

12.3. Content-Format Registry

Internet media types are identified by a string, such as "application/xml" [RFC2046]. In order to minimize the overhead of using these media types to indicate the format of payloads, this document defines a sub-registry for a subset of Internet media types to be used in CoAP and assigns each, in combination with a content-coding, a numeric identifier. The name of the sub-registry is "CoAP Content-Formats", within the "CoRE Parameters" registry.

Each entry in the sub-registry must include the media type registered with IANA, the numeric identifier in the range 0-65535 to be used for that media type in CoAP, the content-coding associated with this identifier, and a reference to a document describing what a payload with that media type means semantically.

CoAP does not include a separate way to convey content-encoding information with a request or response, and for that reason the content-encoding is also specified for each identifier (if any). If multiple content-encodings will be used with a media type, then a separate Content-Format identifier for each is to be registered. Similarly, other parameters related to an Internet media type, such as level, can be defined for a CoAP Content-Format entry.

Initial entries in this sub-registry are as follows:

Media type	Encoding	Id.	Reference
text/plain; charset=utf-8	-	0	[RFC2046][RFC3676][RFC5147]
application/ link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/ octet-stream	-	42	[RFC2045][RFC2046]
application/exi	-	47	[EXIMIME]
application/json	-	50	[RFC4627]

Table 9: CoAP Content-Formats

The identifiers between 65000 and 65535 inclusive are reserved for experiments. They are not meant for vendor specific use of any kind and MUST NOT be used in operational deployments. The identifiers between 256 and 9999 are reserved for future use in IETF specifications (IETF review or IESG approval). All other identifiers are Unassigned.

Because the name space of single-byte identifiers is so small, the IANA policy for future additions in the range 0-255 inclusive to the sub-registry is "Expert Review" as described in [RFC5226]. The IANA policy for additions in the range 10000-64999 inclusive is "First Come First Served" as described in [RFC5226].

In machine to machine applications, it is not expected that generic Internet media types such as text/plain, application/xml or application/octet-stream are useful for real applications in the long term. It is recommended that M2M applications making use of CoAP will request new Internet media types from IANA indicating semantic information about how to create or parse a payload. For example, a Smart Energy application payload carried as XML might request a more specific type like application/se+xml or application/se-exi.

12.4. URI Scheme Registration

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

URI scheme name.
coap

Status.

Permanent.

URI scheme syntax.

Defined in Section 6.1 of [RFCXXXX].

URI scheme semantics.

The "coap" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP). The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "http" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.

The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.

The scheme is used by CoAP endpoints to access CoAP resources.

Interoperability considerations.

None.

Security considerations.

See Section 11.1 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

References.

[RFCXXXX]

12.5. Secure URI Scheme Registration

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

URI scheme name.

coaps

Status.

Permanent.

URI scheme syntax.

Defined in Section 6.2 of [RFCXXXX].

URI scheme semantics.

The "coaps" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using Datagram Transport Layer Security (DTLS) for transport security. The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "https" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.

The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.

The scheme is used by CoAP endpoints to access CoAP resources using DTLS.

Interoperability considerations.

None.

Security considerations.

See Section 11.1 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

References.

[RFCXXXX]

12.6. Service Name and Port Number Registration

One of the functions of CoAP is resource discovery: a CoAP client can ask a CoAP server about the resources offered by it (see Section 7). To enable resource discovery just based on the knowledge of an IP address, the CoAP port for resource discovery needs to be standardized.

IANA has assigned the port number 5683 and the service name "coap", in accordance with [RFC6335].

Besides unicast, CoAP can be used with both multicast and anycast.

Service Name.

coap

Transport Protocol.

UDP

Assignee.

IESG <iesg@ietf.org>

Contact.

IETF Chair <chair@ietf.org>

Description.

Constrained Application Protocol (CoAP)

Reference.

[RFCXXXX]

Port Number.

5683

12.7. Secure Service Name and Port Number Registration

CoAP resource discovery may also be provided using the DTLS-secured CoAP "coaps" scheme. Thus the CoAP port for secure resource discovery needs to be standardized.

This document requests the assignment of the port number [IANA_TBD_PORT] and the service name "coaps", in accordance with [RFC6335].

Besides unicast, DTLS-secured CoAP can be used with anycast.

Service Name.

coaps

Transport Protocol.

UDP

Assignee.

IESG <iesg@ietf.org>

Contact.

IETF Chair <chair@ietf.org>

Description.

DTLS-secured CoAP

Reference.

[RFCXXXX]

Port Number.

[IANA_TBD_PORT]

12.8. Multicast Address Registration

Section 8, "Multicast CoAP", defines the use of multicast. This document requests the assignment of the following multicast addresses for use by CoAP nodes:

IPv4 -- "All CoAP Nodes" address [TBD1], from the IPv4 Multicast Address Space Registry. As the address is used for discovery that may span beyond a single network, it should come from the Internetwork Control Block (224.0.1.x, RFC 5771).

IPv6 -- "All CoAP Nodes" address [TBD2], from the IPv6 Multicast Address Space Registry, in the Variable Scope Multicast Addresses space (RFC3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only. The address should be of the form FF0x::nn, where nn is a single byte, to ensure good compression of the local-scope address with [RFC6282].

[The explanatory text to be removed upon allocation of the addresses, except for the note about the distinct multicast addresses.]

13. Acknowledgements

Brian Frank was a contributor to and co-author of previous drafts of this specification.

Special thanks to Peter Bigot, Esko Dijk and Cullen Jennings for substantial contributions to the ideas and text in the document, along with countless detailed reviews and discussions.

Thanks to Ed Beroaset, Angelo P. Castellani, Gilbert Clark, Robert Cragie, Esko Dijk, Lisa Dusseault, Mehmet Ersue, Thomas Fossati, Tom Herbst, Richard Kelsey, Ari Keranen, Matthias Kovatsch, Salvatore Loreto, Kerry Lynn, Alexey Melnikov, Guido Moritz, Petri Mutka, Colin O'Flynn, Charles Palmer, Adriano Pezzuto, Robert Quattlebaum, Akbar Rahman, Eric Rescorla, Dan Romascanu, David Ryan, Szymon Sasin, Michael Scharf, Dale Seed, Robby Simpson, Peter van der Stok, Michael Stuber, Linyi Tian, Gilman Tolle, Matthieu Vial and Alper Yegin for helpful comments and discussions that have shaped the document.

Special thanks also to the IESG reviewers, Adrian Farrel, Martin Stiernerling, Pete Resnick, Richard Barnes, Sean Turner, Spencer Dawkins, Stephen Farrell, and Ted Lemon, who contributed in-depth reviews.

Some of the text has been borrowed from the working documents of the IETF httpbis working group.

14. References

14.1. Normative References

- [I-D.ietf-tls-oob-pubkey]
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-07 (work in progress), February 2013.
- [I-D.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-06 (work in progress), February 2013.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.
- [RFC2045] Freed, N. and N.S. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

- [RFC3676] Gellens, R., "The Text/Plain Format and DelSp Parameters", RFC 3676, February 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.
- [RFC5147] Wilde, E. and M. Duerst, "URI Fragment Identifiers for the text/plain Media Type", RFC 5147, April 2008.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, April 2013.

14.2. Informative References

- [EUI64] , "GUIDELINES FOR 64-BIT GLOBAL IDENTIFIER (EUI-64) REGISTRATION AUTHORITY", April 2010, <<http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>>.
- [EXIMIME] , "Efficient XML Interchange (EXI) Format 1.0", December 2009, <<http://www.w3.org/TR/2009/CR-exi-20091208/#mediaTypeRegistration>>.
- [HHGTTG] Adams, D., "The Hitchhiker's Guide to the Galaxy", October 1979.
- [I-D.allman-tcpm-rto-consider]
Allman, M., "Retransmission Timeout Considerations", draft-allman-tcpm-rto-consider-01 (work in progress), May 2012.
- [I-D.bormann-coap-misc]
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-22 (work in progress), December 2012.
- [I-D.bormann-core-ipsec-for-coap]
Bormann, C., "Using CoAP with IPsec", draft-bormann-core-ipsec-for-coap-00 (work in progress), December 2012.
- [I-D.castellani-core-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-http-mapping-07 (work in progress), February 2013.
- [I-D.ietf-core-block]

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

[I-D.ietf-core-groupcomm]

Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-06 (work in progress), April 2013.

[I-D.ietf-core-observe]

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

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and A. Keraenen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-04 (work in progress), April 2013.

[I-D.ietf-tls-multiple-cert-status-extension]

Pettersen, Y., "The TLS Multiple Certificate Status Request Extension", draft-ietf-tls-multiple-cert-status-extension-08 (work in progress), April 2013.

[REST]

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

[RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

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

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

- [RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.
- [RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, July 2004.
- [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.
- [RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.
- [RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.
- [RFC5489] Badra, M. and I. Hajjeh, "ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)", RFC 5489, March 2009.
- [RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, September 2011.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA)

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

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July 2012.

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, April 2013.

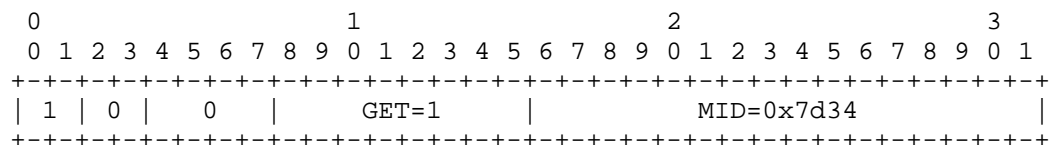
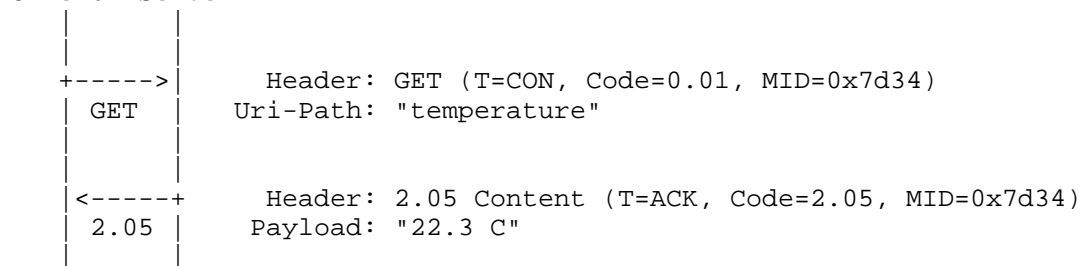
[W3CXMLESEC] Wenning, R., "Report of the XML Security PAG", October 2012, <<http://www.w3.org/2011/xmlsec-pag/pagreport.html>>.

Appendix A. Examples

This section gives a number of short examples with message flows for GET requests. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and multicast.

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

Client Server



```

| 11 | 11 | "temperature" (11 B) ...
+-----+
0      1      2      3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+
| 1 | 2 | 0 | 2.05=69 | MID=0x7d34 |
+-----+
| 1 1 1 1 1 1 1 | "22.3 C" (6 B) ...
+-----+

```

Figure 16: Confirmable request; piggy-backed response

Figure 17 shows a similar example, but with the inclusion of a non-empty Token (Value 0x20) in the request and the response, increasing the sizes to 17 and 12 bytes, respectively.

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

```

0      1      2      3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+
| 1 | 0 | 1 | GET=1 | MID=0x7d35 |
+-----+
| 0x20 |
+-----+
| 11 | 11 | "temperature" (11 B) ...
+-----+

```

```

0      1      2      3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+
| 1 | 2 | 1 | 2.05=69 | MID=0x7d35 |
+-----+

```

```

+-----+
|      0x20      |
+-----+
| 1 1 1 1 1 1 1 | "22.3 C" (6 B) ...
+-----+

```

Figure 17: Confirmable request; piggy-backed response

In Figure 18, the Confirmable GET request is lost. After ACK_TIMEOUT seconds, the client retransmits the request, resulting in a piggy-backed response as in the previous example.

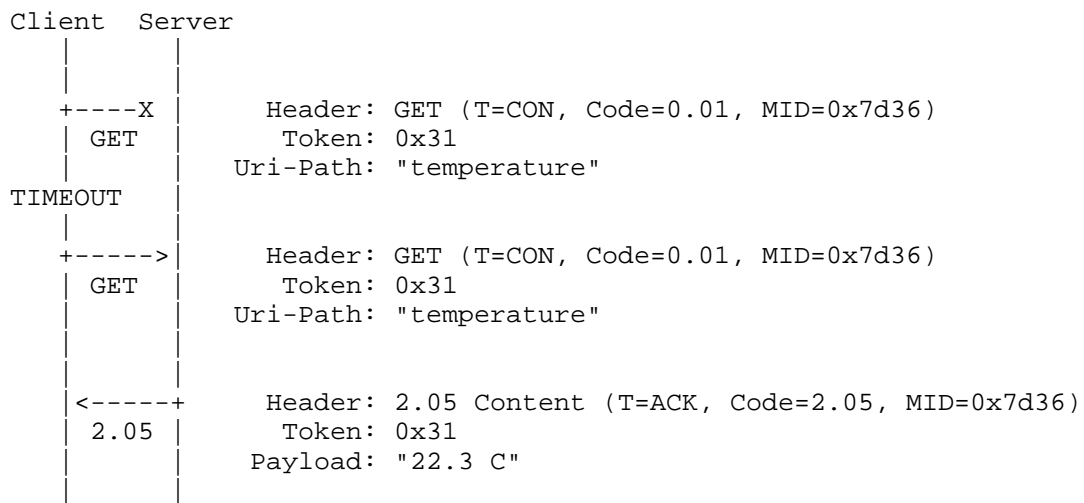
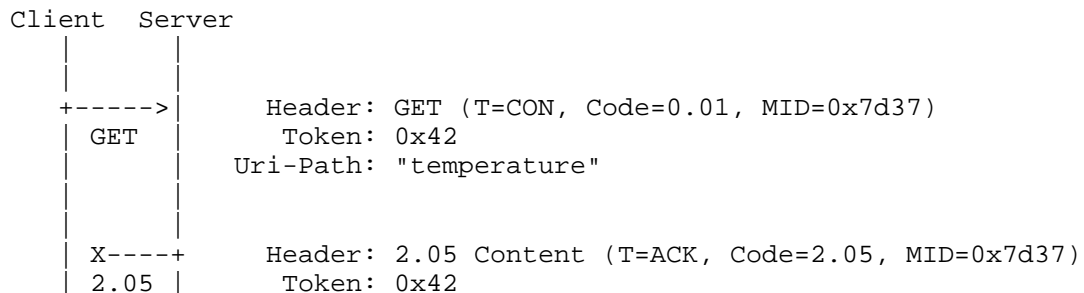


Figure 18: Confirmable request (retransmitted); piggy-backed response

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



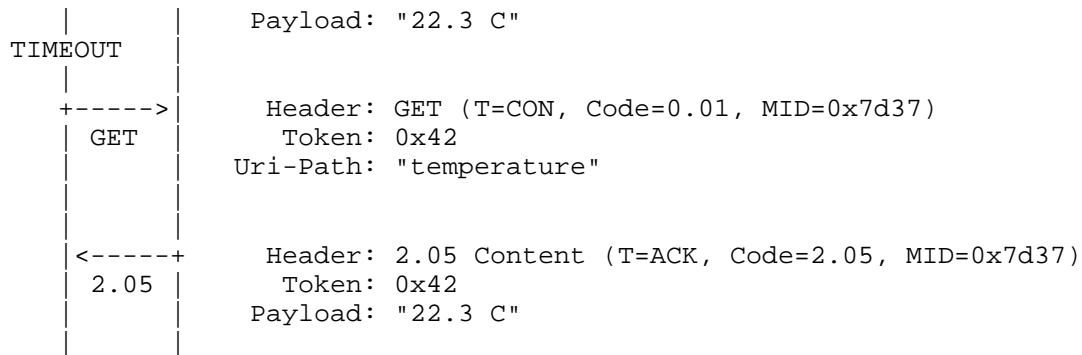


Figure 19: Confirmable request; piggy-backed response (retransmitted)

In Figure 20, the server acknowledges the Confirmable request and sends a 2.05 (Content) response separately in a Confirmable message. Note that the Acknowledgement message and the Confirmable response do not necessarily arrive in the same order as they were sent. The client acknowledges the Confirmable response.

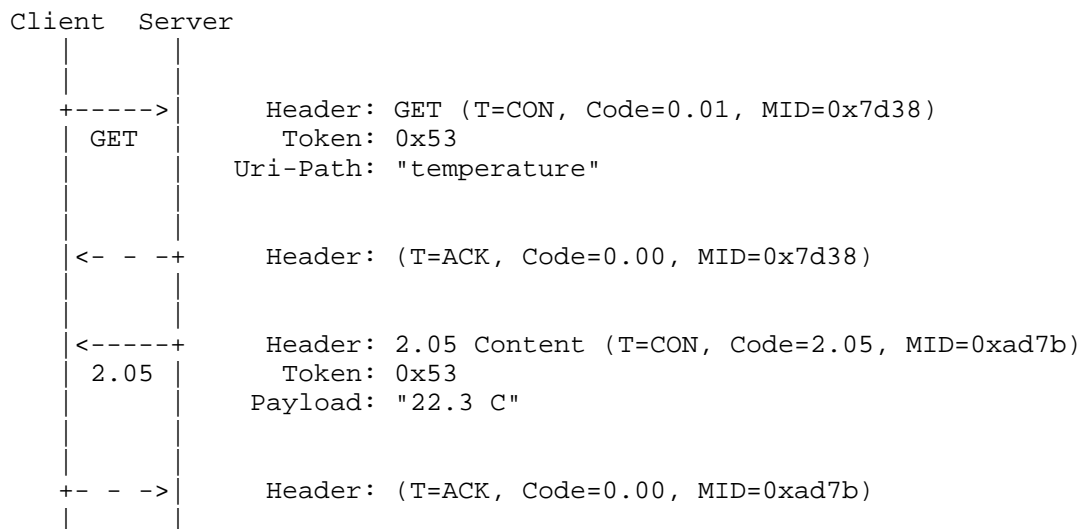


Figure 20: Confirmable request; separate response

Figure 21 shows an example where the client loses its state (e.g., crashes and is rebooted) right after sending a Confirmable request, so the separate response arriving some time later comes unexpected. In this case, the client rejects the Confirmable response with a Reset message. Note that the unexpected ACK is silently ignored.

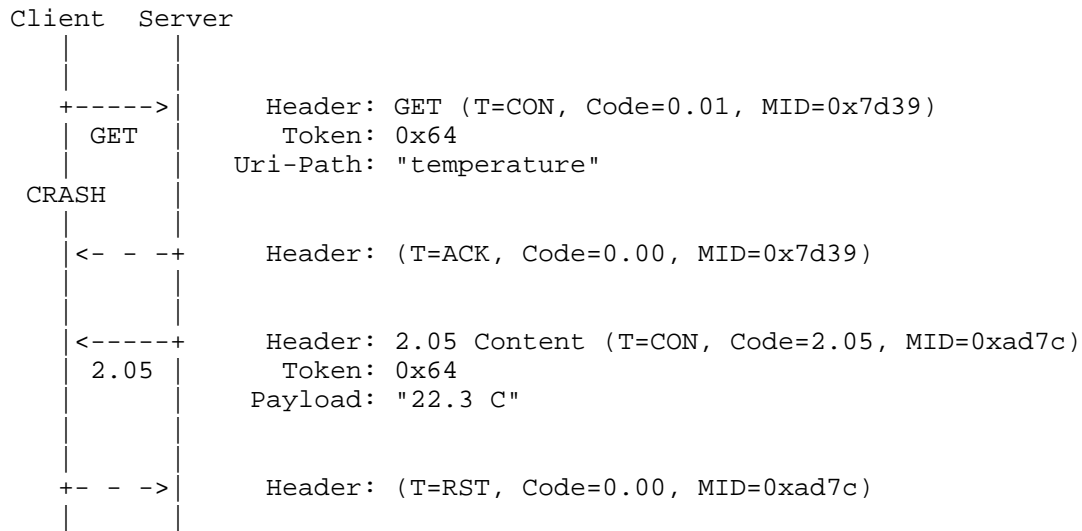


Figure 21: Confirmable request; separate response (unexpected)

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

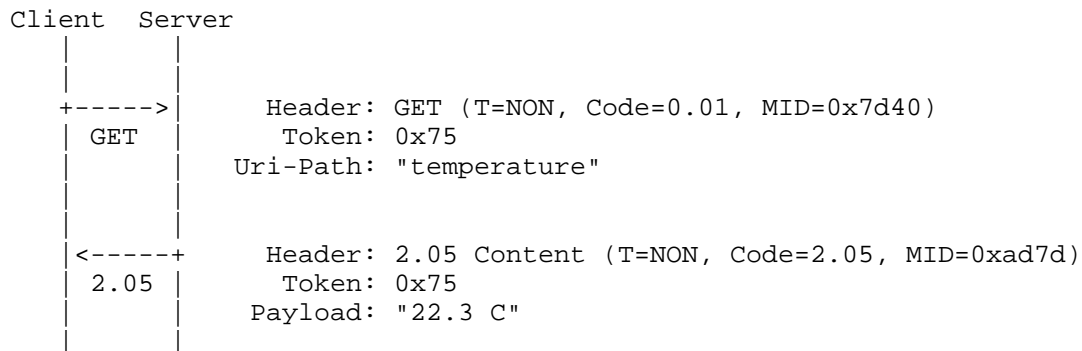


Figure 22: Non-confirmable request; Non-confirmable response

In Figure 23, the client sends a Non-confirmable GET request to a multicast address: all nodes in link-local scope. There are 3 servers on the link: A, B and C. Servers A and B have a matching resource, therefore they send back a Non-confirmable 2.05 (Content) response. The response sent by B is lost. C does not have matching response, therefore it sends a Non-confirmable 4.04 (Not Found) response.

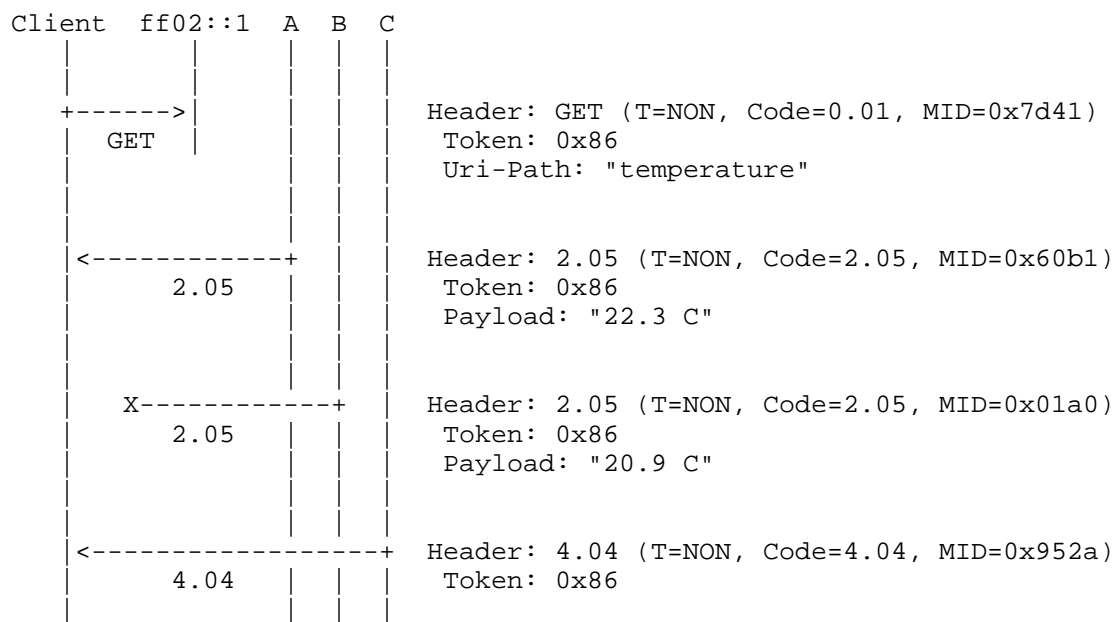


Figure 23: Non-confirmable request (multicast); Non-confirmable response

Appendix B. URI Examples

The following examples demonstrate different sets of Uri options, and the result after constructing an URI from them. In addition to the options, Section 6.5 refers to the destination IP address and port, but not all paths of the algorithm cause the destination IP address and port to be included in the URI.

o Input:

Destination IP Address = [2001:db8::2:1]
Destination UDP Port = 5683

Output:

coap://[2001:db8::2:1]/

o Input:

Destination IP Address = [2001:db8::2:1]
Destination UDP Port = 5683
Uri-Host = "example.net"

Output:

```
coap://example.net/
```

o Input:

```
Destination IP Address = [2001:db8::2:1]
Destination UDP Port = 5683
Uri-Host = "example.net"
Uri-Path = ".well-known"
Uri-Path = "core"
```

Output:

```
coap://example.net/.well-known/core
```

o Input:

```
Destination IP Address = [2001:db8::2:1]
Destination UDP Port = 5683
Uri-Host = "xn--18j4d.example"
Uri-Path = the string composed of the Unicode characters U+3053
U+3093 U+306b U+3061 U+306f, usually represented in UTF-8 as
E38193E38293E381ABE381A1E381AF hexadecimal
```

Output:

```
coap://xn--18j4d.example/
%E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF
```

(The line break has been inserted for readability; it is not part of the URI.)

o Input:

```
Destination IP Address = 198.51.100.1
Destination UDP Port = 61616
Uri-Path = ""
Uri-Path = "/"
Uri-Path = ""
Uri-Path = ""
Uri-Query = "//"
Uri-Query = "?&"
```

Output:

```
coap://198.51.100.1:61616//%2F//?%2F%2F&?%26
```

Appendix C. Changelog

(To be removed by RFC editor before publication.)

Changes from ietf-17 to ietf-18: Address comments from the IESG reviews.

- o Accept is now critical.
- o Add Size1 option for 4.13 responses.

Changes from ietf-15 to ietf-16: Address comments from the IESG reviews. These should not impact interoperability.

- o Clarify that once there has been an empty ACK, all further ACKs to the same message also must be empty (#301).
- o Define Cache-key properly (#302).
- o Clarify that ACKs don't get retransmitted, the CONs do (#303).
- o Clarify: NON is like separate for CON (#304).
- o Don't use decimal response codes, keep the 3+5 structure throughout (#305).
- o RFC 2119 usage in 4.5 (#306) and 8.2 (#307).
- o Ensure all protocol reactions to reserved or prohibited values are defined (#308).
- o URI matching rules may be scheme specific (#309).
- o Don't dally beyond MAX_TRANSMIT_SPAN during retransmission (#310).
- o More about selecting a token length for anti-spoofing (#311).
- o Discuss spoofing ACKs (#312).
- o Qualify partial discard strategy implementation note as UDP only (#313).
- o Explicitly point out that UDP and DTLS don't mix (#314).
- o Point out security consideration re URIs and access control (#315).
- o Point to RFC5280 section 6 (#316).

- o Add a paragraph about cert status checking (#317).
- o RSA is out, ECDHE is in for cert-with-PSK, too (#318).
- o Point out that requests and responses don't always come in pairs (#319).
- o Clarify when there is a need for Unicode normalization (#320).
- o Point out that Uri-Host doesn't handle user-part (#321).
- o Clarify the use of non-FQDN Authority Names in certificates.
- o Numerous editorial improvements and clarifications.

Changes from ietf-14 to ietf-15: Address comments from IETF last-call, mostly implementation notes and editorial improvements. These should not impact interoperability.

- o Clarify bytes/characters and UTF-8/ASCII in "Decomposing URIs into Options" (#282).
- o Make reference to ECC/CCM DTLS ciphersuite normative (#286).
- o Add a quick warning that bitwise scanning for a payload marker is not a good idea (#287).
- o Make reference to PROBING_RATE explicit for saturation discussion (#288).
- o Mention PROCESSING_DELAY when discussion piggy-backing (#290).
- o Various editorial nits: Clarify use of noun "service" (#283), Reference terminology from lwig-terminology (#284), make reference to HTTP terms more explicit (#285), add a forward reference to 5.9.2.9 (#289), 8 kbit/s is not "conservative" (#291).
- o Add description of resource depletion attack (#292).
- o Add description of DoS attack on congestion control (#293).
- o Add discussion of using non-trivial token for protecting against hijacking (#294).
- o Clarify implementation note about per-destination Message ID generation.

Changed from ietf-13 to ietf-14:

- o Made Accept option non-repeatable.
- o Clarified that Safe options in a 2.03 Valid response update the cache.
- o Clarified that payload sniffing is acceptable only if no Content-Format was supplied.
- o Clarified URI examples (Appendix B).
- o Numerous editorial improvements and clarifications.

Changed from ietf-12 to ietf-13:

- o Simplified message format.
 - * Removed the OC (Option Count) field in the CoAP Header.
 - * Changed the End-of-Options Marker into the Payload Marker.
 - * Changed the format of Options: use 4 bits for option length and delta; insert one or two additional bytes after the option header if necessary.
 - * Promoted the Token Option to a field following the CoAP Header.
- o Clarified when a payload is a diagnostic payload (#264).
- o Moved IPsec discussion to separate draft (#262).
- o Added a reference to a separate draft on reverse-proxy URI embedding (#259).
- o Clarified the use of ETags and of 2.03 responses (#265, #254, #256).
- o Added reserved Location-* numbers and clarified Location-*.
- o Added Proxy-Scheme proposal.
- o Clarified terms such as content negotiation, selected representation, representation-format, message format error.
- o Numerous clarifications and a few bugfixes.

Changed from ietf-11 to ietf-12:

- o Extended options to support lengths of up to 1034 bytes (#202).

- o Added new Jump mechanism for options and removed Fenceposting (#214).
- o Added new IANA option number registration policy (#214).
- o Added Proxy Unsafe/Safe and Cache-Key masking to option numbers (#241).
- o Re-numbered option numbers to use Unsafe/Safe and Cache-Key compliant numbers (#241).
- o Defined NSTART and restricted the value to 1 with a MUST (#215).
- o Defined PROBING_RATE and set it to 1 Byte/second (#215).
- o Defined DEFAULT_LEISURE (#246).
- o Renamed Content-Type into Content-Format, and Media Type registry into Content-Format registry.
- o A large number of small editorial changes, clarifications and improvements have been made.

Changed from ietf-10 to ietf-11:

- o Expanded section 4.8 on Transmission Parameters, and used the derived values defined there (#201). Changed parameter names to be shorter and more to the point.
- o Several more small editorial changes, clarifications and improvements have been made.

Changed from ietf-09 to ietf-10:

- o Option deltas are restricted to 0 to 14; the option delta 15 is used exclusively for the end-of-options marker (#239).
- o Option numbers that are a multiple of 14 are not reserved, but are required to have an empty default value (#212).
- o Fixed misleading language that was introduced in 5.10.2 in coap-07 re Uri-Host and Uri-Port (#208).
- o Segments and arguments can have a length of zero characters (#213).
- o The Location-* options describe together describe one location. The location is a relative URI, not an "absolute path URI" (#218).

- o The value of the Location-Path Option must not be '.' or '..' (#218).
- o Added a sentence on constructing URIs from Location-* options (#231).
- o Reserved option numbers for future Location-* options (#230).
- o Fixed response codes with payload inconsistency (#233).
- o Added advice on default values for critical options (#207).
- o Clarified use of identifiers in RawPublicKey Mode Provisioning (#222).
- o Moved "Securing CoAP" out of the "Security Considerations" (#229).
- o Added "All CoAP Nodes" multicast addresses to "IANA Considerations" (#216).
- o Over 100 small editorial changes, clarifications and improvements have been made.

Changed from ietf-08 to ietf-09:

- o Improved consistency of statements about RST on NON: RST is a valid response to a NON message (#183).
- o Clarified that the protocol constants can be configured for specific application environments.
- o Added implementation note recommending piggy-backing whenever possible (#182).
- o Added a content-encoding column to the media type registry (#181).
- o Minor improvements to Appendix D.
- o Added text about multicast response suppression (#177).
- o Included the new End-of-options Marker (#176).
- o Added a reference to draft-ietf-tls-oob-pubkey and updated the RPK text accordingly.

Changed from ietf-07 to ietf-08:

- o Clarified matching rules for messages (#175)

- o Fixed a bug in Section 8.2.2 on Etags (#168)
- o Added an IP address spoofing threat analysis contribution (#167)
- o Re-focused the security section on raw public keys (#166)
- o Added an 4.06 error to Accept (#165)

Changed from ietf-06 to ietf-07:

- o application/link-format added to Media types registration (#160)
- o Moved content-type attribute to the document from link-format.
- o Added coaps scheme and DTLS-secured CoAP default port (#154)
- o Allowed 0-length Content-type options (#150)
- o Added congestion control recommendations (#153)
- o Improved text on PUT/POST response payloads (#149)
- o Added an Accept option for content-negotiation (#163)
- o Added If-Match and If-None-Match options (#155)
- o Improved Token Option explanation (#147)
- o Clarified mandatory to implement security (#156)
- o Added first come first server policy for 2-byte Media type codes (#161)
- o Clarify matching rules for messages and tokens (#151)
- o Changed OPTIONS and TRACE to always return 501 in HTTP-CoAP mapping (#164)

Changed from ietf-05 to ietf-06:

- o HTTP mapping section improved with the minimal protocol standard text for CoAP-HTTP and HTTP-CoAP forward proxying (#137).
- o Eradicated percent-encoding by including one Uri-Query Option per &-delimited argument in a query.
- o Allowed RST message in reply to a NON message with unexpected token (#135).

- o Cache Invalidation only happens upon successful responses (#134).
- o 50% jitter added to the initial retransmit timer (#142).
- o DTLS cipher suites aligned with ZigBee IP, DTLS clarified as default CoAP security mechanism (#138, #139)
- o Added a minimal reference to draft-kivinen-ipsecme-ikev2-minimal (#140).
- o Clarified the comparison of UTF-8s (#136).
- o Minimized the initial media type registry (#101).

Changed from ietf-04 to ietf-05:

- o Renamed Immediate into Piggy-backed and Deferred into Separate -- should finally end the confusion on what this is about.
- o GET requests now return a 2.05 (Content) response instead of 2.00 (OK) response (#104).
- o Added text to allow 2.02 (Deleted) responses in reply to POST requests (#105).
- o Improved message deduplication rules (#106).
- o Section added on message size implementation considerations (#103).
- o Clarification made on human readable error payloads (#109).
- o Definition of CoAP methods improved (#108).
- o Max-Age removed from requests (#107).
- o Clarified uniqueness of tokens (#112).
- o Location-Query Option added (#113).
- o ETag length set to 1-8 bytes (#123).
- o Clarified relation between elective/critical and option numbers (#110).
- o Defined when to update Version header field (#111).
- o URI scheme registration improved (#102).

- o Added review guidelines for new CoAP codes and numbers.

Changes from ietf-03 to ietf-04:

- o Major document reorganization (#51, #63, #71, #81).
- o Max-age length set to 0-4 bytes (#30).
- o Added variable unsigned integer definition (#31).
- o Clarification made on human readable error payloads (#50).
- o Definition of POST improved (#52).
- o Token length changed to 0-8 bytes (#53).
- o Section added on multiplexing CoAP, DTLS and STUN (#56).
- o Added cross-protocol attack considerations (#61).
- o Used new Immediate/Deferred response definitions (#73).
- o Improved request/response matching rules (#74).
- o Removed unnecessary media types and added recommendations for their use in M2M (#76).
- o Response codes changed to base 32 coding, new Y.XX naming (#77).
- o References updated as per AD review (#79).
- o IANA section completed (#80).
- o Proxy-Uri Option added to disambiguate between proxy and non-proxy requests (#82).
- o Added text on critical options in cached states (#83).
- o HTTP mapping sections improved (#88).
- o Added text on reverse proxies (#72).
- o Some security text on multicast added (#54).
- o Trust model text added to introduction (#58, #60).
- o AES-CCM vs. AES-CCB text added (#55).

- o Text added about device capabilities (#59).
- o DTLS section improvements (#87).
- o Caching semantics aligned with RFC2616 (#78).
- o Uri-Path Option split into multiple path segments.
- o MAX_RETRANSMIT changed to 4 to adjust for RESPONSE_TIME = 2.

Changes from ietf-02 to ietf-03:

- o Token Option and related use in asynchronous requests added (#25).
- o CoAP specific error codes added (#26).
- o Erroring out on unknown critical options changed to a MUST (#27).
- o Uri-Query Option added.
- o Terminology and definitions of URIs improved.
- o Security section completed (#22).

Changes from ietf-01 to ietf-02:

- o Sending an error on a critical option clarified (#18).
- o Clarification on behavior of PUT and idempotent operations (#19).
- o Use of Uri-Authority clarified along with server processing rules; Uri-Scheme Option removed (#20, #23).
- o Resource discovery section removed to a separate CoRE Link Format draft (#21).
- o Initial security section outline added.

Changes from ietf-00 to ietf-01:

- o New cleaner transaction message model and header (#5).
- o Removed subscription while being designed (#1).
- o Section 2 re-written (#3).
- o Text added about use of short URIs (#4).

- o Improved header option scheme (#5, #14).
- o Date option removed while being designed (#6).
- o New text for CoAP default port (#7).
- o Completed proxying section (#8).
- o Completed resource discovery section (#9).
- o Completed HTTP mapping section (#10).
- o Several new examples added (#11).
- o URI split into 3 options (#12).
- o MIME type defined for link-format (#13, #16).
- o New text on maximum message size (#15).
- o Location Option added.

Changes from shelby-01 to ietf-00:

- o Removed the TCP binding section, left open for the future.
- o Fixed a bug in the example.
- o Marked current Sub/Notify as (Experimental) while under WG discussion.
- o Fixed maximum datagram size to 1280 for both IPv4 and IPv6 (for CoAP-CoAP proxying to work).
- o Temporarily removed the Magic Byte header as TCP is no longer included as a binding.
- o Removed the Uri-code Option as different URI encoding schemes are being discussed.
- o Changed the rel= field to desc= for resource discovery.
- o Changed the maximum message size to 1024 bytes to allow for IP/UDP headers.
- o Made the URI slash optimization and method idempotence MUSTs
- o Minor editing and bug fixing.

Changes from shelby-00 to shelby-01:

- o Unified the message header and added a notify message type.
- o Renamed methods with HTTP names and removed the NOTIFY method.
- o Added a number of options field to the header.
- o Combines the Option Type and Length into an 8-bit field.
- o Added the magic byte header.
- o Added new ETag Option.
- o Added new Date Option.
- o Added new Subscription Option.
- o Completed the HTTP Code - CoAP Code mapping table appendix.
- o Completed the Content-type Identifier appendix and tables.
- o Added more simplifications for URI support.
- o Initial subscription and discovery sections.
- o A Flag requirements simplified.

Authors' Addresses

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

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

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

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

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

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 04, 2014

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
KoanLogic
E. Dijk
Philips Research
July 03, 2013

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

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 January 04, 2014.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	3
2. Terminology	3
3. Cross-Protocol Usage of URIs	4
4. HTTP to CoAP URI Mapping	5
4.1. Requirements	5
4.2. Proposals	6
4.2.1. Solution #1: Carsten's Mapping Proposal	6
4.2.2. Solution #2: Adding IPv6 Literals to Carsten's Mapping Proposal	6
4.2.3. Solution #3: Split CoAP URI in Query Arguments	7
4.2.4. Solution #4: CoAP URI as Single Query Parameter	7
4.2.5. Solution #5: Split CoAP URI in Path Components	7
4.3. Comparison Matrix	8
5. HTTP-CoAP Reverse Proxy	9
5.1. Proxy Placement	10
5.2. Response Code Translations	11
5.3. Media Type Translations	13
5.4. Caching and Congestion Control	13
5.5. Cache Refresh via Observe	14
5.6. Use of CoAP Blockwise Transfer	15
5.7. Security Translation	15
5.8. Other guidelines	16
6. IANA Considerations	16
7. Security Considerations	16
7.1. Traffic overflow	17
7.2. Handling Secured Exchanges	17
8. Acknowledgements	18
9. References	18
9.1. Normative References	18
9.2. Informative References	19
Appendix A. Change Log	20
Authors' Addresses	20

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

Following sections define requirements for the URI mapping function, identify potential solutions, and provide a matrix to evaluate the identified solutions against requirements. The aim is to define a clear framework to inform further WG discussion.

4.1. Requirements

- o REQ1: Syntactic correctness of the HTTP URI (i.e. handle percent-encoding when needed);
- o REQ2: HTTP URI must be able include all elements of a CoAP URI (e.g. coap(s) scheme, hostname, host literals IPv4/IPv6, port, path, query components, characters allowed in CoAP URI);
- o REQ3: The mapping operation must produce a string that can be directly used by a proxy as input to the process of Section 6.4. of [I-D.ietf-core-coap];
- o REQ4: HTTP URI should be easily readable/writable by humans, if possible (i.e. easy to read the CoAP URI embedded in it, e.g.: avoid multiple levels of percent encoding, etc.);
- o REQ5: HTTP cache friendliness of the mapping solution, i.e. maximize the caching of CoAP resources by HTTP intermediaries (a solution where entire CoAP URI is provided as a single query parameter is bad in this respect);
- o REQ6: Normalised form: preferably there should be only one normal/default way to encode the URI, so that we do not end up with multiple different cache entries for the same CoAP resource in intermediaries;
- o REQ7: HTTP URI should be as short as possible.

4.2. Proposals

4.2.1. Solution #1: Carsten's Mapping Proposal

This is the mapping proposal originally defined in [I-D.bormann-core-cross-reverse-convention].

URI template: TBD.

Example:

- o `http://proxy.example.com/.well-known/core-translate/1.2.3.4_4567/foo/bar?a=3`
- o `coap://1.2.3.4:4567/foo/bar?a=3`

Notes: How to include IPv6 literals was not defined in [I-D.bormann-core-cross-reverse-convention]. The CoAP scheme is derived from HTTP scheme (http or https). The "{Port}" part is optional.

4.2.2. Solution #2: Adding IPv6 Literals to Carsten's Mapping Proposal

Adding IPv6 literals support to [I-D.bormann-core-cross-reverse-convention].

URI template: `"/.well-known/core-translate/{authority-encoded}/{path}?{query}"`

Example 1:

- o `http://proxy.example.com/.well-known/core-translate/%5B2001:db8::1%5D:4567/foo/bar?a=3`
- o `coap://[2001:db8::1]:4567/foo/bar?a=3`

Example 2:

- o `http://proxy.example.com/.well-known/core-translate/server.coap.example.com:4567/foo/bar?a=3`
- o `coap://server.coap.example.com:4567/foo/bar?a=3`

Example 3:

- o `http://proxy.example.com/.well-known/core-translate/server.coap.example.com/foo/bar?a=3`

- o `coap://server.coap.example.com/foo/bar?a=3`

Example 4:

- o `http://proxy.example.com/.well-known/core-translate/1.2.3.4:4567/foo/bar?a=3`
- o `coap://1.2.3.4:4567/foo/bar?a=3`

4.2.3. Solution #3: Split CoAP URI in Query Arguments

This proposal splits the CoAP URI in parts and puts parts in to separate query arguments of the HTTP URI.

URI template: `"/.well-known/core-translate/host={host}&port={port}&path={path}?{query}"`.

Note: The query parts "host", "port", "path" and "query" are all optional in the URI.

TBD: discuss order of query arguments; and what to do with duplicates.

Example:

- o `http://proxy.example.com/.well-known/core-translate?host=%5B2001:db8::1%5D&port=4567&path=/foo/bar?a=3&b=5`
- o `coap://[2001:db8::1]:4567/foo/bar?a=3&b=5`

4.2.4. Solution #4: CoAP URI as Single Query Parameter

Inspired by certain web services that put HTTP callback URIs in URI-query parameters.

URI template: TBD.

Example:

- o `"http://proxy.example.com/.well-known/core-translate?uri=coap%3A%2F%2F%5B2001%3Adb8%3A%3A1%5D%3A4567%2Ffoo%2Fbar%3Fb%3Dbefore_colon%253Aafter_colon"`
- o Maps to `"coap://[2001:db8::1]:4567/foo/bar?b=before_colon%3Aafter_colon"`

4.2.5. Solution #5: Split CoAP URI in Path Components

URI template: /.well-known/core-translate/{scheme}/{+host}/{port}/{+path_abempty}/{+query}

Where:

- o scheme is "coap" or "coaps" or the empty string;
- o host matches the production defined in [RFC3986] Sec. 3.2.2. (need to percent-encode '[' and ']' in IP-literal);
- o port matches the production defined in [RFC3986] Sec. 3.2.3.;
- o path_abempty matches the production defined in [RFC3986] Sec. 3.3. (need to percent-encode any '/' occurrence);
- o query matches the production defined in [RFC3986] Sec. 3.4. (need to percent-encode any '/' and '?' occurrence);

CoAP URI is reconstructed as per [RFC3986] Sec. 5.3. carrying out the following substitutions before going through the algorithm:

- o if scheme is the empty string, make it "coap";
- o if port is the empty string, make it "5683";
- o TBD (possibly not needed): if path-abempty is the empty string, make it "/";

Example:

- o http://proxy.example.com/.well-known/core-translate/coap/server.coap.example.com/4567/foo%2Fbar/a=3
- o coap://server.coap.example.com:4567/foo/bar?a=3

4.3. Comparison Matrix

The following table compares solutions defined in Section 4.2 to requirements stated in Section 4.1.

	#1	#2	#3	#4	#5	Notes
REQ1	+	+	+	+	+	
REQ2	-	+	+	+	+	
REQ3	+	+	+	+	+	(a)
REQ4	+	+	o	-	o	
REQ5	+	+	-	-	+	

REQ6	?	?	?	?	?	
REQ7	+	+	+	+	+/o	
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+

Table 1: Evaluation of HTTP-CoAP URI Mapping Proposals Against Requirements

Legend:

+ = Meets the requirement

- = Does not meet the requirement

o = Partly meets the requirement

? = TBD

(a) = Details need to be defined for each solution.

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:

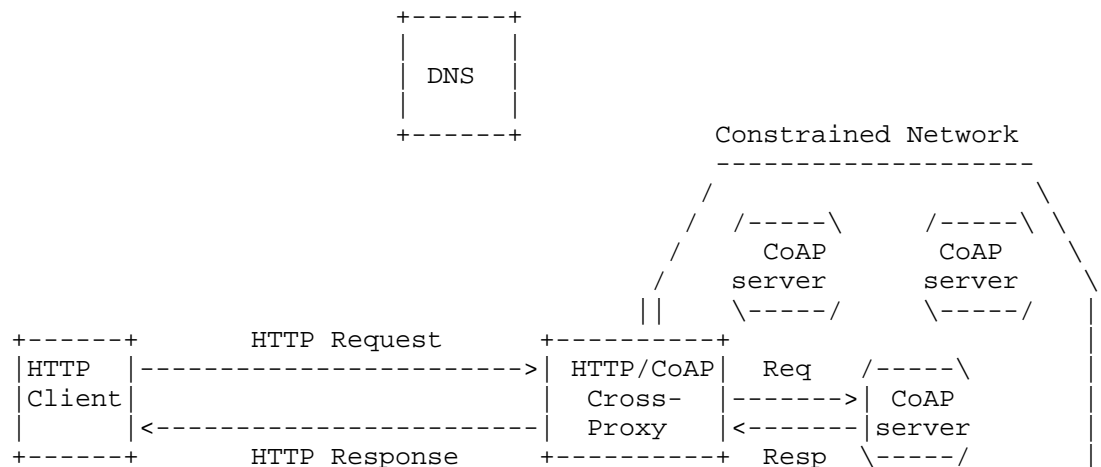
Caching: Efficient caching requires that all request traffic to a CoAP server is handled by the same proxy which receives HTTP requests from multiple source locations. This maximally reduces the load on (constrained) CoAP servers.

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.)



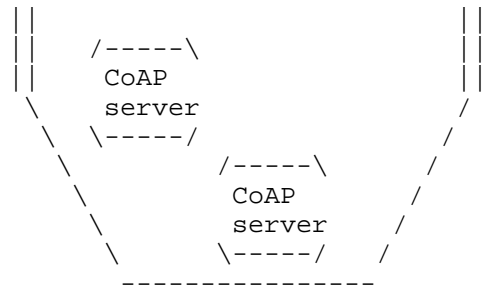


Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

5.2. Response Code Translations

Table 2 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 2: 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 SS placed.

Resources experiencing a high access rate coupled with high volatility MAY be observed [I-D.ietf-core-observe] by the Cross-Protocol Proxy to keep their cached representation fresh while minimizing the number CoAP messages. See Section 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 end-point before returning 413 (Request Entity Too Large) to the HTTP client.

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

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-12 (work in progress), June 2013.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-09 (work in progress), May 2013.
- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.

- [I-D.ietf-httpbis-p1-messaging]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", draft-ietf-httpbis-p1-messaging-22 (work in progress), February 2013.
- [I-D.ietf-httpbis-p2-semantics]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", draft-ietf-httpbis-p2-semantics-22 (work in progress), February 2013.
- [I-D.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-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-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-05 (work in progress), February 2013.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, January 2001.
- [RFC4732] Handley, M., Rescorla, E., IAB, "Internet Denial-of-Service Considerations", RFC 4732, December 2006.

Appendix A. Change Log

Changes from ietf-00 to ietf-01:

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

Authors' Addresses

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

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

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

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

Thomas Fossati
KoanLogic

Email: tho@koanlogic.com

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

Email: esko.dijk@philips.com

CoRE
Internet-Draft
Intended status: Informational
Expires: December 05, 2013

Z. Shelby
Sensinode
M.V. Vial
Schneider-Electric
June 03, 2013

CoRE Interfaces
draft-ietf-core-interfaces-00

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 December 05, 2013.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
2. Terminology	3
3. Function Set	4
3.1. Defining a Function Set	4
3.1.1. Path template	4
3.1.2. Resource Type	5
3.1.3. Interface Description	5
3.1.4. Data type	5
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	10
5.2. Batch	11
5.3. Linked Batch	11
5.4. Sensor	13
5.5. Parameter	13
5.6. Read-only Parameter	14
5.7. Actuator	14
5.8. Binding	15
5.9. Resource Observation Attributes	15
5.10. Future Interfaces	18
5.11. WADL Description	18
6. Security Considerations	22
7. IANA Considerations	22
8. Acknowledgments	22
9. Changelog	22
10. References	23
10.1. Normative References	23
10.2. Informative References	23
Appendix A. Profile example	23
Authors' Addresses	25

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>;rt="simple.sen.lt";if="core.s",
</s>;rt="simple.sen.tmp";if="core.s";obs,
</s>;rt="simple.sen.hum";if="core.s",
</a>;rt="simple.act";if="core.b",
</a>;rt="simple.act.led";if="core.a",
</a>;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 /l and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.

```
Req: POST /l (Content-type: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed
```

```
Req: GET /l
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 /l (Content-type: application/link-format)
</s/humidity>
Res: 2.04 Changed
```

```
Req: GET /l (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>
```

```
Req: GET /l
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 /l
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/link-format)
<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 Attributes

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 control how often a client is interested in receiving notifications and how much a resource value 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.

These query parameters MUST be treated as resources that are read using GET and set using PUT, and MUST NOT be included in the Observe request. Multiple parameters MAY be set at the same time by including the values in the query string of a PUT. Before being set, these parameters have no default value.

Resource	Parameter	Data Format
Minimum Period (s)	/ {resource} ?pmin	xsd:integer (>0)
Maximum Period (s)	/ {resource} ?pmax	xsd:integer (>0)
Change Step	/ {resource} ?st	xsd:decimal (>0)
Less Than	/ {resource} ?lt	xsd:decimal
Greater Than	/ {resource} ?gt	xsd:decimal

Minimum Period: When set, 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 set, 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 set, 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.

Less Than: When set, the value of the resource MUST be less than this parameter in order to send a new notification. This parameter has lower priority than the period parameters.

Greater Than: When set, the value of the resource MUST be greater than this parameter in order to send a new notification. This parameter has lower priority than the period parameters.

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 PUT /s/temp?pmin=10&pmax=60&st=1

Res: 2.04 Changed

Req: GET Observe /s/temp

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

The following example shows a request to check the current value of the pmin attribute of the Observable resource from the last example.

Req: GET /s/temp?pmin

Res: 2.05 Content (text/plain)
10

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"/>
    <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 -05 to WG Document -00

- o Updated the date and version.

Changes from -04 to -05

- o Made the Observation control parameters to be treated as resources rather than Observe query parameters. Added Less Than and Greater Than parameters.

Changes from -03 to -04

- o Draft refresh

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., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-14 (work in progress), March 2013.
- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.
- [I-D.shelby-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-shelby-core-resource-directory-05 (work in progress), February 2013.

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 Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 05, 2013

C. Bormann
Universitaet Bremen TZI
June 03, 2013

Representing CoRE Link Collections in JSON
draft-ietf-core-links-json-00

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 December 05, 2013.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
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., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-17 (work in progress), May 2013.
- [I-D.pbryan-zyp-json-ref] Bryan, P. and K. Zyp, "JSON Reference", draft-pbryan-zyp-json-ref-03 (work in progress), September 2012.
- [MNOT11] Nottingham, M., "Linking in JSON", November 2011, <http://www.mnot.net/blog/2011/11/25/linking_in_json>.
- [RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

Appendix A. Implementation

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

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

Author's Address

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

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 16, 2014

K. Hartke
Universitaet Bremen TZI
July 15, 2013

Observing Resources in CoAP
draft-ietf-core-observe-09

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on January 16, 2014.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	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. Aggregation	9
3.5. Reordering	9
3.6. Transmission	10
3.7. Cancellation	10
4. Server-side Requirements	11
4.1. Request	11
4.2. Notifications	11
4.3. Caching	12
4.4. Reordering	12
4.5. Transmission	13
5. Intermediaries	15
6. Web Linking	15
7. Security Considerations	16
8. IANA Considerations	16
9. Acknowledgements	17
10. References	17
10.1. Normative References	17
10.2. Informative References	17
Appendix A. Examples	18
A.1. Proxying	22
Appendix B. Modeling Resources to Tailor Notifications	24
Appendix C. Changelog	24
Author's Address	29

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 for it, though, to solve the full set of problems that the existing HTTP solutions solve, or to replace publish/subscribe networks that solve a much more general problem [RFC5989].

1.2. Protocol Overview

The protocol is based on the well-known observer design pattern [GOF]. In this design pattern, components called "observers" register at a specific, known provider called the "subject" that they are interested in being notified whenever the subject undergoes a change in state. The subject is responsible for administering its list of registered observers. If multiple subjects are of interest to an observer, it must register separately for all of them.

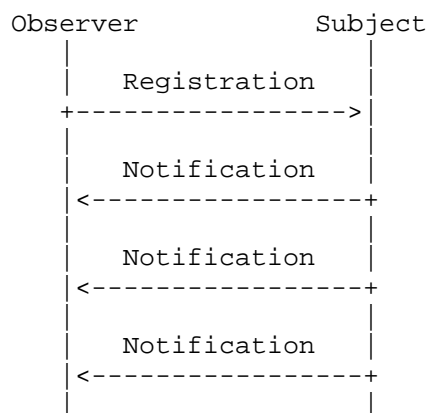


Figure 1: The Observer Design Pattern

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 endpoint and token specified in the request to the list of observers of that resource.

Notification: Whenever the state of a resource changes, the server notifies each client in the list of observers of the resource. Each notification is an additional CoAP response sent by the server in reply to the GET request and includes a complete, updated representation of the new resource state.

Figure 2 below shows an example of a CoAP client registering its interest in a resource and receiving three notifications: the first upon registration with the current state, and then two upon changes to the resource state. Both the registration request and the notifications are identified as such by the presence of the Observe Option defined in this document. In notifications, the Observe Option 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.

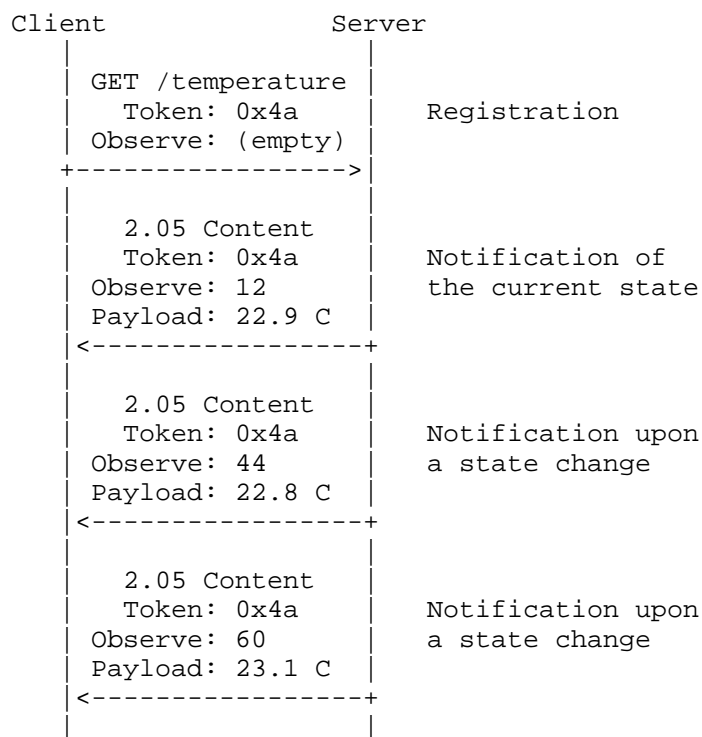


Figure 2: Observing a Resource in CoAP

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's entry 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 times out after several transmission attempts, then the client is assumed to be no longer interested and its entry is removed from the list of observers.

While a client is in the list of observers of a resource, the goal of the protocol is to keep the resource state observed by the client as closely in sync with the actual state at the server as possible.

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 a new notification. And third, the server may erroneously come to the conclusion that the client is no longer interested in the resource, which will cause the server to stop sending notifications and the client to assume an old state until it registers its interest again.

The protocol addresses this issue as follows:

- o It follows a best-effort approach for sending the current representation to the client after a state change: Clients should see the new state after a state change as soon as possible, and they should see as many states as possible. However, a client cannot rely on observing every single state that a resource might go through.
- o It labels notifications with a maximum duration up to which it is acceptable for the observed state and the actual state to be out of sync. When the age of the notification received reaches this limit, the client cannot use the enclosed representation until it receives a new notification.
- o It is designed on the principle of eventual consistency: The protocol guarantees that, if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the latest resource state.

1.3. 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 a new entry to the list of observers of the resource. The list entry consists of the client endpoint and the token specified by the client in the request. The value of the option in a request **MUST** be empty on transmission and **MUST** be ignored on reception.

The Observe Option is not critical for processing the request. If the server is unwilling or unable to add the client to the list of observers of the target resource, then the request falls back to a normal GET request.

In a response, the Observe Option identifies the message as a notification. This implies that the server has added the client to the list of observers and that it will notify the client of changes to the resource state. The value of the option is a 24-bit sequence number for reordering detection (see Section 3.5 and Section 4.4). The sequence number is encoded in network byte order using a variable number of bytes ('uint' format; see Section 3.2 of RFC XXXX [I-D.ietf-core-coap]).

The Observe Option is not part of the cache-key: a cacheable response obtained with an Observe Option in the request can be used to satisfy a request without an Observe Option, and vice versa. When a stored response 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 3.5) 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 included one or more ETag Options in the request (see Section 3.3). In the event that the resource changes in a way that would cause a normal GET request at that time to return a non-2.xx response (for example, when the resource is deleted), the server sends a notification with an appropriate response code (such as 4.04 Not Found) and removes 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 server. Like a response, a notification is considered fresh while its age is not greater than the value indicated by the Max-Age Option and no newer notification/response has been received.

The server will do its best to keep the resource state observed by the client as closely in sync with the actual state as possible. However, a client cannot rely on observing every single state that a resource might go through. For example, if the network is congested or the state changes more frequently than the network can handle, the server can skip notifications for any number of intermediate states.

The server uses the Max-Age Option to indicate an age up to which it is acceptable that the observed state and the actual state are inconsistent. If the age of the latest notification becomes greater than its indicated Max-Age, then the client MUST NOT use the enclosed representation unless it is validated.

3.3.2. Validation

When a client has one or more notifications stored in its cache for a resource, it can use the ETag Option in the GET request to give the server an opportunity to select a stored notification to be used.

The client MAY include an ETag Option for each stored response that is applicable in the GET request. Whenever the observed resource changes to a representation identified by one of the ETag Options, the server can select a stored response by sending a 2.03 (Valid) notification with an appropriate ETag Option instead of a 2.05 (Content) notification.

A client implementation needs to keep all candidate responses in its cache until it is no longer interested in the target resource or it issues a GET request with a new set of entity-tags.

3.4. Aggregation

Every successful GET request with an Observe Option yields a new, independent stream of notifications. Like a fresh response can be used to satisfy a request without contacting the server, the stream of notifications resulting from one request can be used to satisfy another request if the target resource is the same.

A client **MUST** aggregate GET requests with an Observe Option for the same target resource. The target resource **SHALL** be identified for this purpose by the request URI and all options in the request that are part of the cache-key (such as the Accept Option).

To make sure it has a current representation, it **MAY** issue a GET request without an Observe Option at any time. It is **RECOMMENDED** that the client does not issue a GET request (with or without Observe Option) for a resource while it still has a fresh notification/response in its cache. Additionally, the client **SHOULD** wait for a random amount of time between 5 and 15 seconds before issuing the request to avoid synchronicity with other clients.

3.5. Reordering

Messages with notifications can arrive in a different order than they were sent. Since the goal is to keep the observed state as closely in sync with the actual state as possible, a client **MUST NOT** update the observed state with a notification that arrives later than a newer notification.

For reordering detection, the server sets the value of the Observe Option in each notification to the 24 least-significant bits of a strictly increasing sequence number. An incoming notification is newer than the newest notification received so far when one of the following conditions is met:

$$\begin{aligned} & (V1 < V2 \text{ and } V2 - V1 < 2^{23}) \text{ or} \\ & (V1 > V2 \text{ and } V1 - V2 > 2^{23}) \text{ or} \\ & (T2 > T1 + 128 \text{ seconds}) \end{aligned}$$

where V1 is the value of the Observe Option of the newest notification received so far, V2 the value of the Observe Option of the incoming notification, T1 a client-local timestamp of the newest notification received so far, and T2 a client-local timestamp of the incoming notification.

Design Note: The first two conditions verify that V1 is less than V2 in 24-bit serial number arithmetic [RFC1982]. The third condition ensures that the time elapsed between the two incoming messages is not so large that the difference between V1 and V2 has become larger than the largest integer that it is meaningful to add to a 24-bit 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.

The client **MUST** specify a token in its GET request that is currently not in use for the client/server pair, as the sequence numbers provide an order only among the notifications resulting from the same request.

3.6. 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 for any previous notification.

If a client does not recognize the token in a confirmable notification, it **MUST NOT** acknowledge the message and **SHOULD** reject it with a Reset message; otherwise, the client **MUST** acknowledge the message as usual. In the case of a non-confirmable notification, rejecting the message with a Reset message is **OPTIONAL**.

An acknowledgement message signals to the server that the client is alive and interested in receiving further notifications; if the server does not receive an acknowledgement in reply to a confirmable notification, it will assume that the client is no longer interested and will eventually remove the associated entry from the list of observers.

3.7. Cancellation

A client that is no longer interested in receiving notifications for a resource can simply reject the next notification with a Reset message. In the case of a confirmable notification, the server will then remove the associated entry from the list of observers of this resource. In the case of a non-confirmable notification, the server may (but is not required to) remove the list entry. So the client may have to wait for a confirmable notification if the servers seems to ignore the Reset messages that the client sends to reject non-confirmable notifications.

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 target resource, but also to add a new entry to the list of observers of that resource. The list entry consists of the client endpoint and the token specified by the client in the request. If a client sends multiple requests, each request creates a new entry in the list.

Upon success, the server **MUST** return a current representation of the resource and **MUST** notify the client of subsequent changes to the resource state for each entry of the client in 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.

4.2. Notifications

A client is notified of changes to the resource state by additional responses sent by the server in reply to the GET request. Each such notification response (including the initial response) **MUST** include an Observe Option and **MUST** echo the token specified by the client in the GET request. If there are multiple entries in the list of observers, the order in which the clients are notified is not defined; the server is free to use any method to determine the order.

A notification **SHOULD** have a 2.05 (Content) or 2.03 (Valid) response code. However, in the event that the state of a resource changes in a way that would cause a normal GET request at that time to return a non-2.xx response (for example, when the resource is deleted), the server **SHOULD** notify the client by sending a notification with an appropriate response code (such as 4.04 Not Found) and **MUST** remove the client from the list of observers of the resource.

The Content-Format used in a notification **MUST** be the same as the one used in the initial response to the GET request. If the server is unable to continue sending notifications in this Content-Format, it **SHOULD** send a notification with a 4.06 (Not Acceptable) response code and **MUST** remove the client from the list of observers of the resource.

A non-2.xx notification **MUST NOT** include an Observe Option.

4.3. Caching

As notifications are just additional responses sent by the server, they are subject to caching as defined in Section 5.6 of RFC XXXX [I-D.ietf-core-coap].

4.3.1. Freshness

After returning the initial response, the server MUST try to keep the returned representation current, i.e., keep the resource state observed by the client as closely in sync with the actual resource state as possible.

Since becoming out of sync at times cannot be avoided, the server MUST indicate for each representation an age up to which it is acceptable that the observed state and the actual state are inconsistent. This age is application-dependent and MUST be specified in notifications using the Max-Age Option.

When the resource does not change and the client has a current representation, the server does not need to send a notification. However, if the client does not receive a notification, it cannot tell if the observed state and the actual state are still in sync. Thus, when the age of the latest notification becomes greater than its indicated Max-Age, the client must assume that the states have become inconsistent. The server MAY wish to prevent that by sending a notification with the unchanged representation just 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.

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 option to the sequence number that is current at that time 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 every state change). This removes the need to update the value of the Observe Option on retransmission when the resource state did not change.

Design Note: The choice of a 24-bit option value and a time span of 256 seconds allows for a notification rate of up to 65536 notifications per second. 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.

A server MAY choose to skip sending a notification if it knows that it will send another notification soon, for example, when the state is changing frequently. Similarly, it MAY choose to send a notification more than once. However, above all, the server MUST ensure that a client in the list of observers of a resource eventually observes the latest state if the resource does not undergo a new change in state. For example, when state changes occur in bursts, the server can skip some notifications, send the notifications in non-confirmable messages, and make sure that the client observes the latest state change by repeating the last

notification in a confirmable message when the burst is over.

The client's acknowledgement of a confirmable notification signals to the server that the client is interested in receiving further notifications. If a client rejects a confirmable notification with a Reset message, the client is no longer interested and the server **MUST** remove the associated entry from the list of observers. If the client rejects a non-confirmable notification, the server **MAY** remove the entry from the list of observers as well. (It is expected that the server does remove the entry if it has the information available that is needed to match the Reset message to the non-confirmable notification, but the server is not required to keep this information.)

At a minimum, the server **MUST** send a notification in a confirmable message instead of a non-confirmable message at least every 24 hours, so a client that went away or is no longer interested does not remain forever in the list of observers.

The server **MUST** limit the number of confirmable notifications for which an acknowledgement has not been received yet to **NSTART** (1 by default; 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 associated entry from the list of observers of the observed resource.
3. If the entry is still in the list of observers, start to transmit a new notification with a representation of the current resource state. Should the resource have changed its state more than once in the meantime, the notifications for the intermediate states are silently skipped.
4. If the transmission attempt completed in step 1 timed out, increment the retransmission counter and double the timeout for the new transmission; otherwise, reinitialize both the retransmission counter and the timeout as described in Section 4.2 of RFC XXXX [I-D.ietf-core-coap].

5. Intermediaries

A client may be interested in a resource in the namespace of an origin server that is reached through a chain of one or more CoAP intermediaries. In this case, the client registers its interest with the first intermediary towards the origin server, acting as if it was communicating with the origin server itself as specified in Section 3. It is the task of this intermediary to provide the client with a current representation of the target resource and send notifications upon changes to the target resource state, much like an origin server as specified in Section 4.

To perform this task, the intermediary SHOULD make use of the protocol specified in this document, taking the role of the client and registering its own interest in the target resource with the next hop towards the origin server. If the next hop does not return a response with an Observe Option, the intermediary MAY resort to polling the next hop or MAY itself return a response without an Observe Option.

The communication between each pair of hops is independent; each hop in the server role MUST determine individually how many notifications to send, of which message type, and so on. Each hop MUST generate its own values for the Observe Option, and MUST set the value of the Max-Age Option according to the age of the local current representation.

If two or more clients have registered their interest in a resource with an intermediary, the intermediary MUST register itself only once with the next hop and fan out the notifications it receives to all registered clients. This relieves the next hop from sending the same notifications multiple times and thus enables scalability.

An intermediary is not required to act on behalf of a client to observe a resource; an intermediary MAY observe a resource, for example, just to keep its own cache up to date.

See Appendix A.1 for examples.

6. Web Linking

A web link [RFC5988] to a resource accessible over CoAP (for example, in a link-format document [RFC6690]) MAY include the target attribute "obs".

The "obs" attribute, when present, is a hint indicating that the destination of a link is useful for observation and thus, for

example, should have a suitable graphical representation in a user interface. Note that this is only a hint; it is not a promise that the Observe Option can actually be used to perform the observation. A client may need to resort to polling the resource if the Observe Option is not returned in the response to the GET request.

A value MUST NOT be given for the "obs" attribute; any present value MUST be ignored by parsers. The "obs" attribute MUST NOT appear more than once in a given link-value; occurrences after the first MUST be ignored by parsers.

7. Security Considerations

The security considerations of RFC XXXX [I-D.ietf-core-coap] apply.

The considerations about amplification attacks are somewhat amplified when observing resources. Without client authentication, a server MUST therefore strictly limit the number of notifications that it sends between receiving acknowledgements that confirm the actual interest of the client in the data; i.e., any notifications sent in non-confirmable messages MUST be interspersed with confirmable messages. (An attacker may still spoof the acknowledgements if the confirmable messages are sufficiently predictable.)

As with any protocol that creates state, attackers may attempt to exhaust the resources that the server has available for maintaining the list of observers for each resource. Servers may want to access-control this creation of state. As degraded behavior, the server can always fall back to processing the request as a normal GET request (without an Observe Option) if it is unwilling or unable to add a client to the list of observers of a resource, including if system resources are exhausted or nearing exhaustion.

Intermediaries must be careful to ensure that notifications cannot be employed to create a loop. A simple way to break any loops is to employ caches for forwarding notifications in intermediaries.

8. IANA Considerations

The following entry is added to the CoAP Option Numbers registry:

Number	Name	Reference
6	Observe	[RFCXXXX]

9. Acknowledgements

Carsten Bormann was an original author of this draft and is acknowledged for significant contribution to this document.

Thanks to Daniele Alessandrelli, Jari Arkko, Peter Bigot, Angelo P. Castellani, Gilbert Clark, Esko Dijk, Thomas Fossati, Brian Frank, Bert Greevenbosch, Jeroen Hoebeke, Cullen Jennings, Matthias Kovatsch, Salvatore Loreto, Charles Palmer, Zach Shelby, and Floris Van den Abeele for helpful comments and discussions that have shaped the document.

This work was supported in part by Klaus Tschira Foundation, Intel, Cisco, and Nokia.

10. References

10.1. Normative References

- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

10.2. Informative References

- [GOF] Gamma, E., Helm, R., Johnson, R., and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, USA, November 1994.
- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext

Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC5989] Roach, A., "A SIP Event Package for Subscribing to Changes to an HTTP Resource", RFC 5989, October 2010.

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

Appendix A. Examples

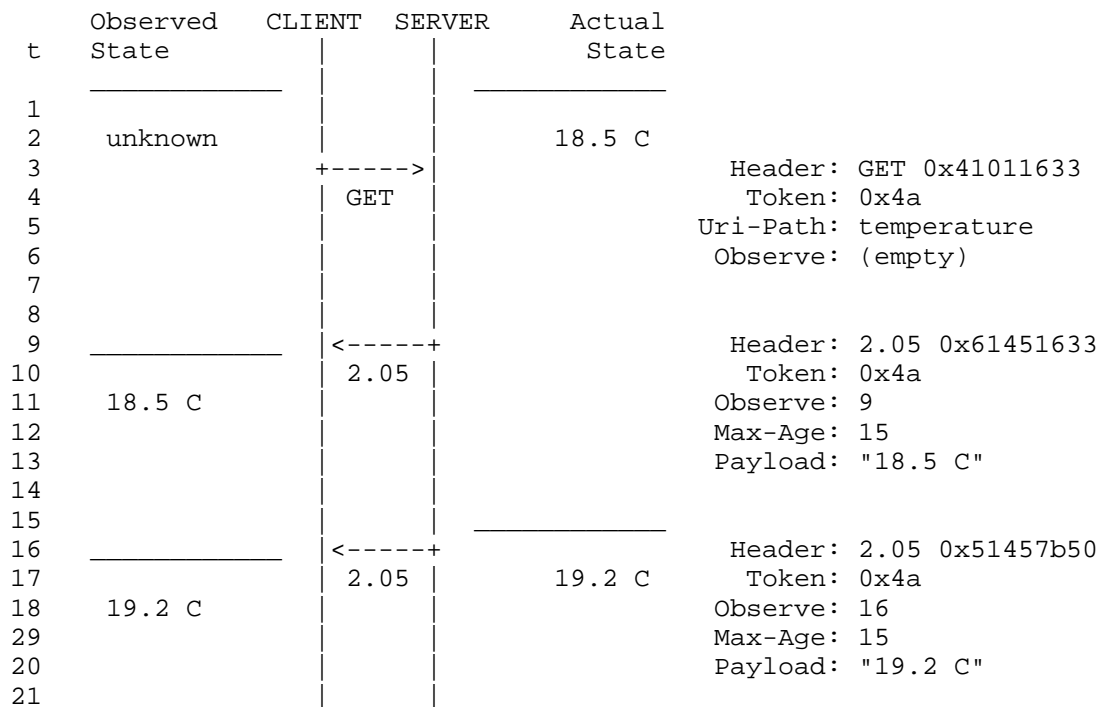


Figure 3: A client registers and receives one notification of the current state and one of a new state upon a state change

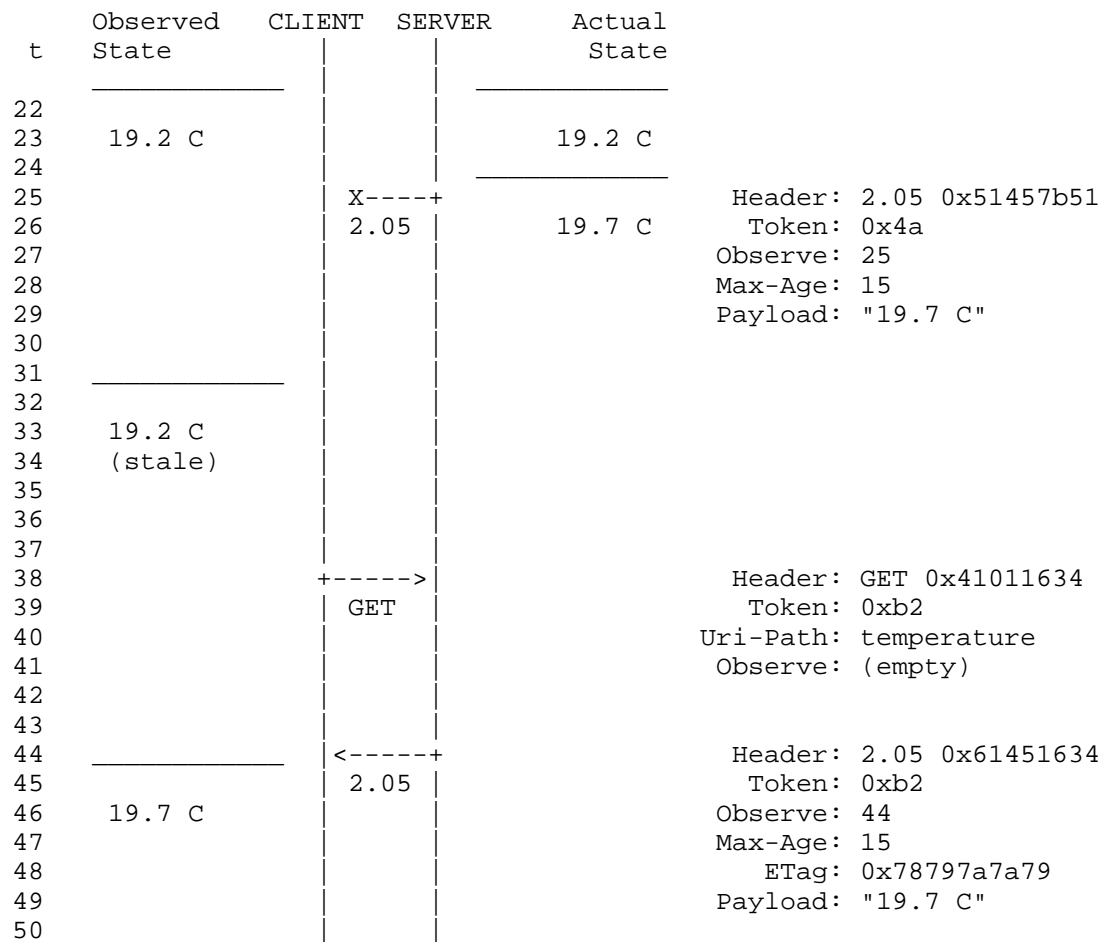


Figure 4: The client re-registers after Max-Age ends

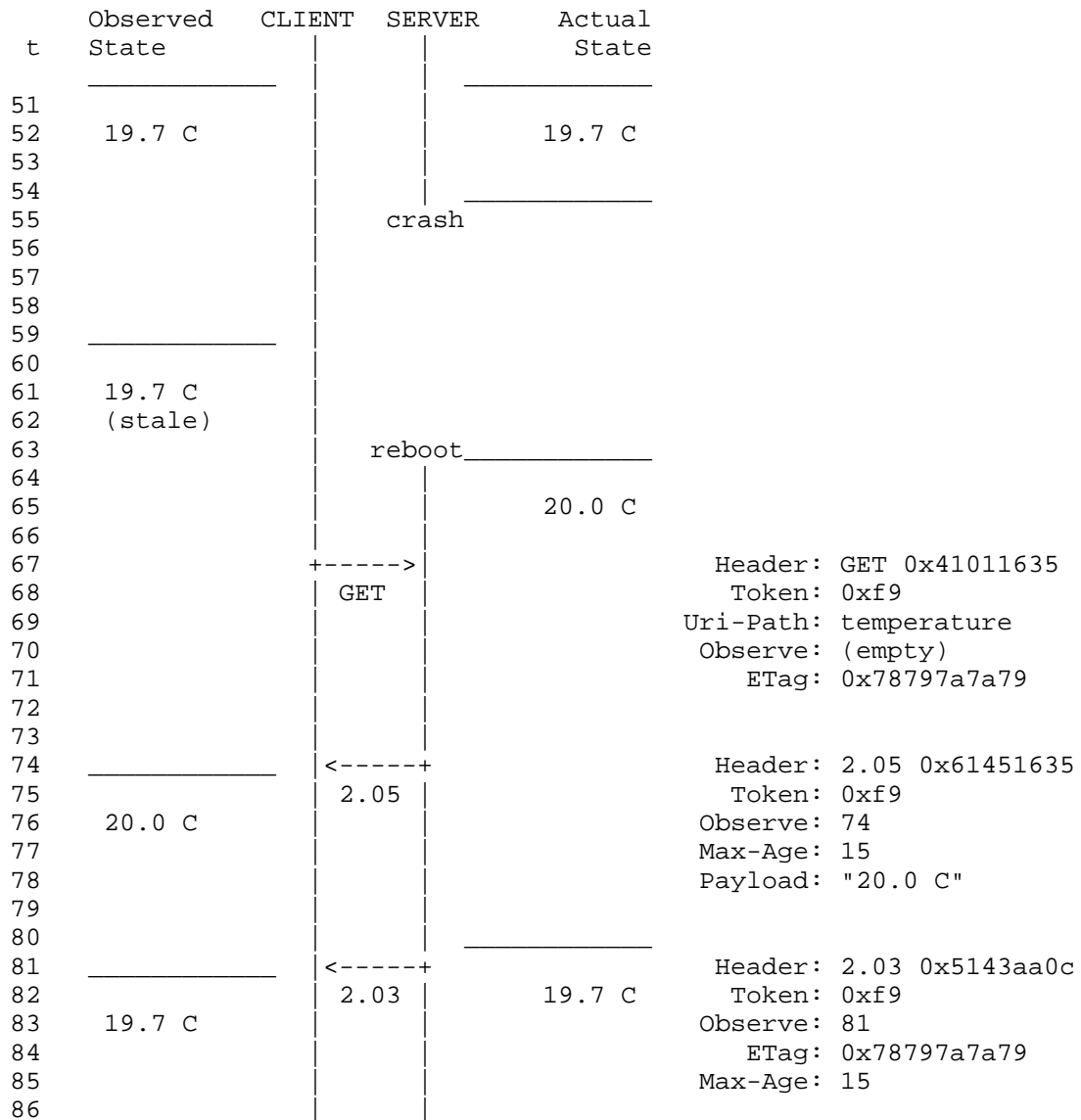


Figure 5: The client re-registers and gives the server the opportunity to select a stored response

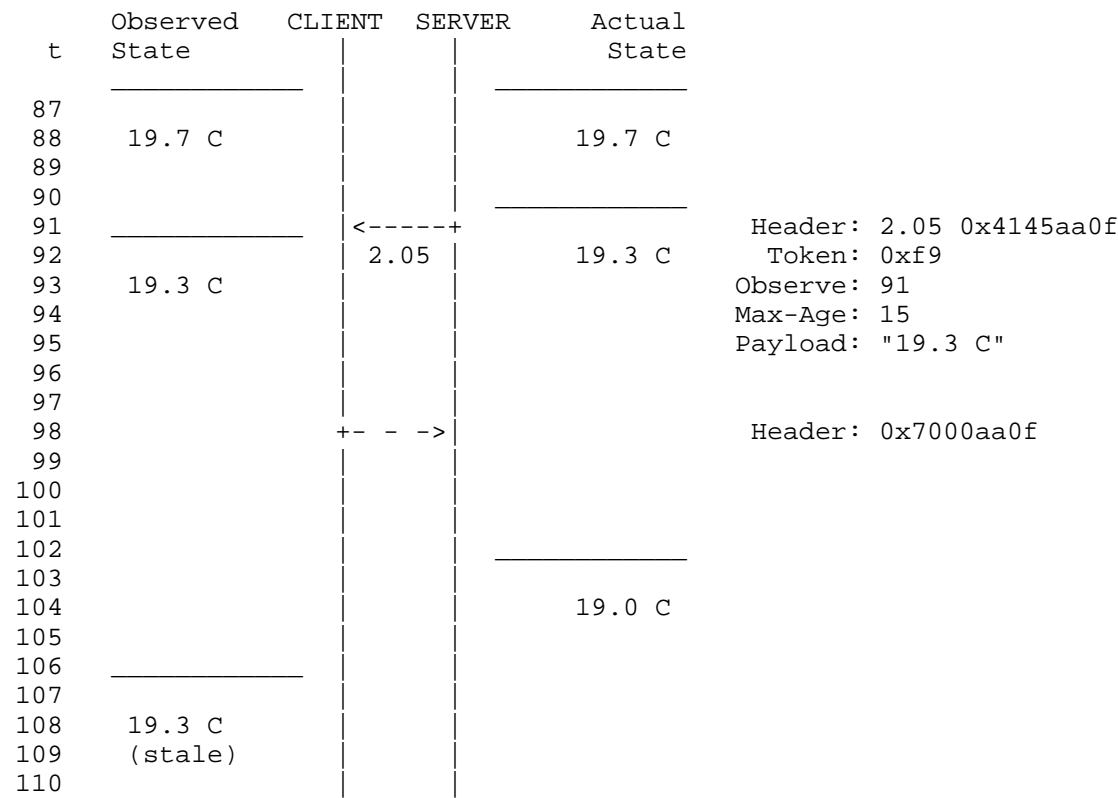


Figure 6: The client rejects a notification and thereby cancels the observation

A.1. Proxying

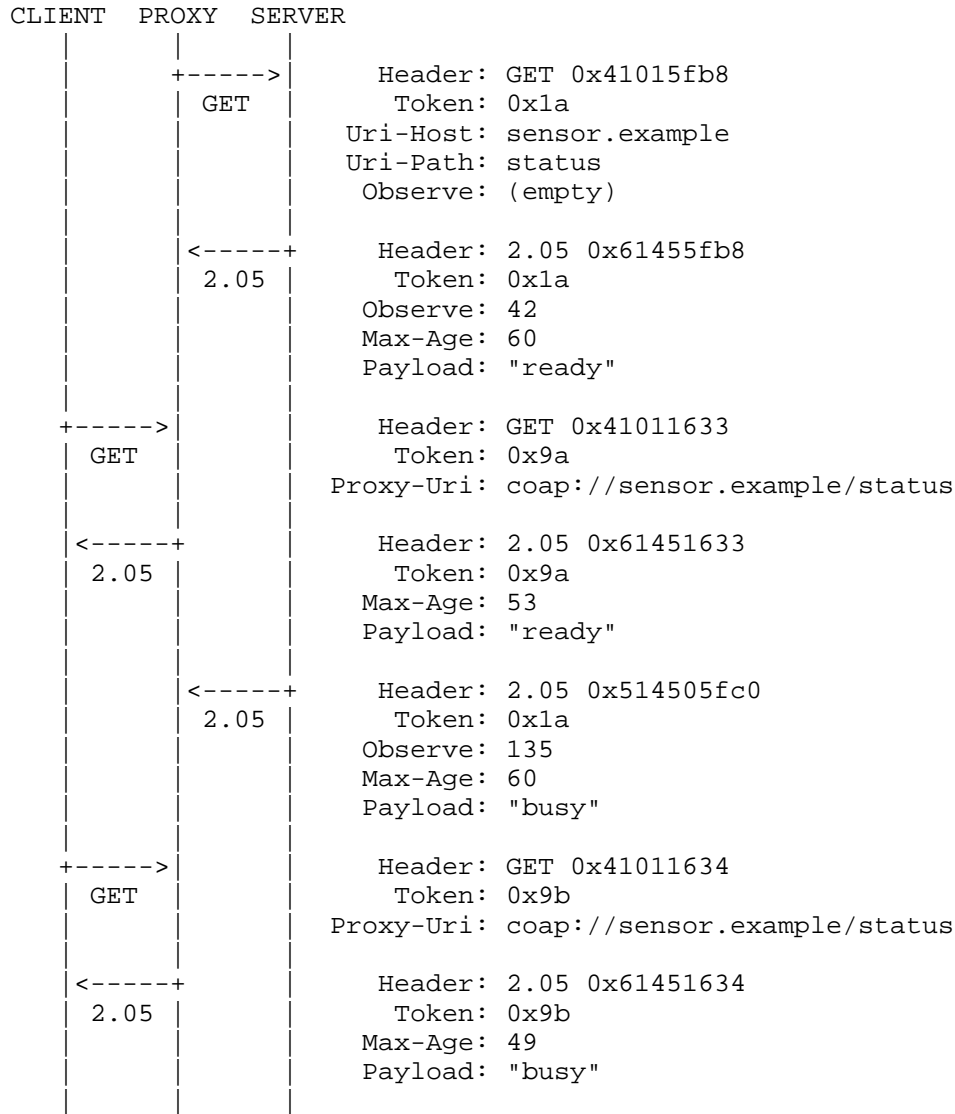


Figure 7: A proxy observes a resource to keep its cache up to date

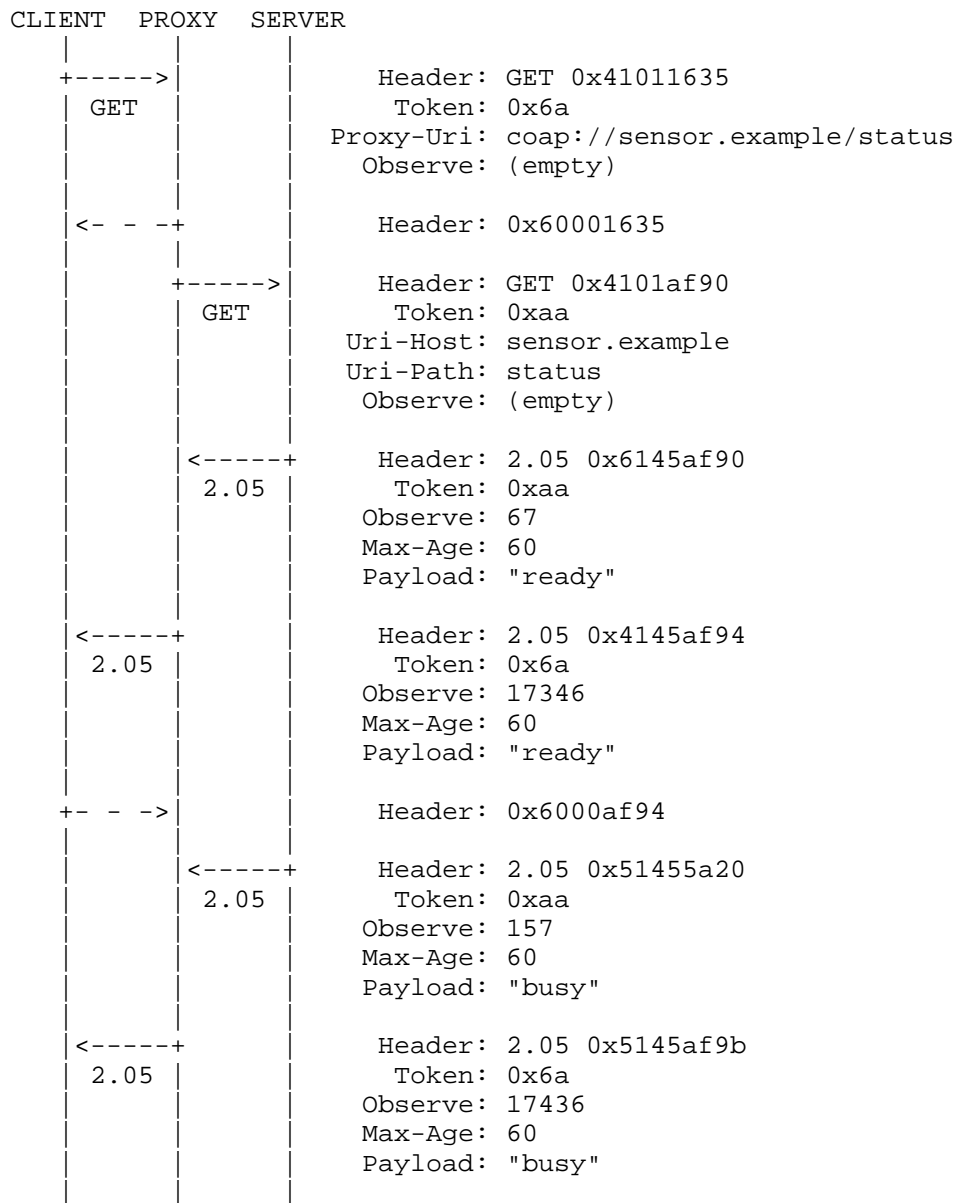


Figure 8: A client observes a resource through a proxy

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, slightly 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 parameterizing 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

(To be removed by RFC editor before publication.)

Changes from ietf-08 to ietf-09:

- o Removed the side effects of requests on existing observations. This includes removing that
 - * the client can use a GET request to cancel an observation;
 - * the server updates the entry in the list of observers instead of adding a new entry if the client is already present (#258, #281).
- o Clarified that a resource (and hence an observation relationship) is identified by the request options that are part of the Cache-Key (#258).
- o Clarified that a non-2.xx notification MUST NOT include an Observe Option.
- o Moved block-wise transfer of notifications to [I-D.ietf-core-block].

Changes from ietf-07 to ietf-08:

- o Expanded text on transmitting a notification while a previous transmission is pending (#242).
- o Changed reordering detection to use a fixed time span of 128 seconds instead of EXCHANGE_LIFETIME (#276).
- o Removed the use of the freshness model to determine if the client is still on the list of observers. This includes removing that
 - * the client assumes that it has been removed from the list of observers when Max-Age ends;
 - * the server sets the Max-Age Option of a notification to a value that indicates when the server will send the next notification;
 - * the server uses a number of retransmit attempts such that removing a client from the list of observers before Max-Age ends is avoided (#235);
 - * the server may remove the client from all lists of observers when the transmission of a confirmable notification ultimately times out.

- o Changed that an unrecognized critical option in a request must actually have no effect on the state of any observation relationship to any resource, as the option could lead to a different target resource.
- o Clarified that client implementations must be prepared to receive each notification equally as a confirmable or a non-confirmable message, regardless of the message type of the request and of any previous notification.
- o Added a requirement for sending a confirmable notification at least every 24 hours before continuing with non-confirmable notifications (#221).
- o Added congestion control considerations from [I-D.bormann-core-congestion-control-02].
- o Recommended that the client waits for a randomized time after the freshness of the latest notification expired before re-registering. This prevents that multiple clients observing a resource perform a GET request at the same time when the need to re-register arises.
- o Changed reordering detection from 'MAY' to 'SHOULD', as the goal of the protocol (to keep the observed state as closely in sync with the actual state as possible) is not optional.
- o Fixed the length of the Observe (3 bytes) in the table in Section 2.
- o Replaced the 'x' in the No-Cache-Key column in the table in Section 2 with a '-', as the Observe Option doesn't have the No-Cache-Key flag set, even though it is not part of the cache key.
- o Updated examples.

Changes from ietf-06 to ietf-07:

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

- o Clarified the semantics of the "obs" link target attribute (#236).

Changes from ietf-05 to ietf-06:

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

Changes from ietf-04 to ietf-05:

- o Recommended that a client does not re-register while a new notification from the server is still likely to arrive. This is to avoid that the request of the client and the last notification after max-age cross over each other (#174).
- o Relaxed requirements when sending a Reset message in reply to non-confirmable notifications.
- o Added an implementation note about careless GET requests (#184).
- o Updated examples.

Changes from ietf-03 to ietf-04:

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

Changes from ietf-02 to ietf-03:

- o Separated client-side and server-side requirements.
- o Fixed uncertainty if client is still on the list of observers by introducing a liveliness model based on Max-Age and a new option called "Max-OFE" (#174).
- o Simplified the text on message reordering (#129).
- o Clarified requirements for intermediaries.
- o Clarified the combination of blockwise transfers with notifications (#172).
- o Updated examples to show how the state observed by the client becomes eventually consistent with the actual state on the server.
- o Added examples for parameterization of observable resource.

Changes from ietf-01 to ietf-02:

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

Changes from ietf-00 to ietf-01:

- o Changed terminology from "subscriptions" to "observation relationships" (#33).
- o Changed the name of the option to "Lifetime".
- o Clarified establishment of observation relationships.
- o Clarified that an observation is only identified by the URI of the observed resource and the identity of the client (#66).
- o Clarified rules for establishing observation relationships (#68).
- o Clarified conditions under which an observation relationship is terminated.
- o Added explanation on how clients can terminate an observation relationship before the lifetime ends (#34).

- o Clarified that the overriding objective for notifications is eventual consistency of the actual and the observed state (#67).
- o Specified how a server needs to deal with clients not acknowledging confirmable messages carrying notifications (#69).
- o Added a mechanism to detect message reordering (#35).
- o Added an explanation of how notifications can be cached, supporting both the freshness and the validation model (#39, #64).
- o Clarified that non-GET requests do not affect observation relationships, and that GET requests without "Lifetime" Option affecting relationships is by design (#65).
- o Described interaction with blockwise transfers (#36).
- o Added Resource Discovery section (#99).
- o Added IANA Considerations.
- o Added Security Considerations (#40).
- o Added examples (#38).

Author's Address

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

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

CoRE Working Group
Internet Draft
Intended status: Standard Track
Expires: January 14, 2014

Ilkyun Park
Seok-Kap Ko
Seung-Hun Oh
Byung-Tak Lee
ETRI
July 14, 2013

Shim Header for CoAP Transfer over non-TCP/IP Networks
draft-ikpark-core-shim-00.txt

Abstract

This document defines shim header for the transfer of CoAP messages over non-TCP/IP constrained networks. In this environment, IP and UDP or TCP are not used, so that additional shim header as a container for addresses of sender/receiver and the length of CoAP header and its payload is required.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 14, 2014.

Copyright Notice

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

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

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
2. Terminology	5
3. Shim Header	5
3.1. Header Format	5
3.2. Identifying between CoAP Header and Shim Header	7
3.3. Example	7
4. Security Considerations	8
5. IANA Considerations	8
6. Acknowledgments	9
7. References	9
7.1. Normative References	9
7.2. Informative References	9

1. Introduction

CoAP [CoAP] is a data transfer protocol over constrained nodes and networks, e.g., 6LoWPAN. This protocol uses TCP/IP suit and some other additional protocols designed for low-power and lossy networks (LLN) like Routing Protocol for LLN (RPL) [RPL].

Nowadays many sensors are involved in variable area like industry, security, environmental and etc., and many of them don't use TCP/IP stack yet. They use various vender-specific transfer protocols. Also, they contain diverse media like IEEE 802.15.4, RS485, CAN, and RS232 (or UART). In the case IEEE 802.15.4, there is a standard protocol for the transmission of IPv6 packets [RFC4944], but many of nodes use vender-specific protocols instead of it.

Hence, a CoAP proxy box is required to interconnect this vender-specific non-TCP/IP network and LLN defined by IETF (e.g., 6LoWPAN). CoAP is not required for vender-specific constrained nodes, because the proxy can translate between vender-specific protocols and TCP/IP as CoAP cross-proxy. But as a CoAP-to-CoAP proxy, the operation of the proxy can be simplified, and the services are able to be abstracted by CoAP overall networks, applying CoAP for both vender-specific and LLN nodes are recommended.

But there are two restrictions on applying CoAP to vender-specific nodes. The first is the lack of address. Peer-to-peer media like UART and RS232 has no address to identify each node. RS485 has an integer value as an identifier, but there is no standardized way to present or carry this value over networks. IEEE 802.15.4 has Network Address in their standard, but this address is allocated by Coordinator dynamically. 64-bit extended address is too long and is sometimes not used.

The second is that CoAP has not PDU size information in its header. CoAP calculates the PDU size with the information from underlying protocol like the payload size in IP header.

As a result, to apply CoAP to vender-specific networks and nodes, new underlying protocol for CoAP must be defined to complement these two restrictions. In this document, new Shim Header is specifies as a underlying protocol for CoAP, and an example is added to explain this shim header.

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.

Shim Header

In front of a CoAP header, a slim header is inserted instead of IP header and this contains addresses of CoAP sender and receiver and the length of CoAP header and its payload.

Local ID

A local ID is an address of CoAP sender or receiver, and it is uniqueness in a vender-specific non-TCP/IP constrained network. This ID has a 16-bit length hexadecimal value, that is IEEE 802.15.4 Network Address compatible, and it can be extensible later.

CoAP Proxy

A CoAP proxy in this document is a CoAP-to-CoAP proxy from [CoAP]. In additional, this CoAP proxy can read and insert a shim header from/to the front of CoAP header.

Additional terminology for CoAP can be found in [CoAP].

3. Shim Header

3.1. Header Format

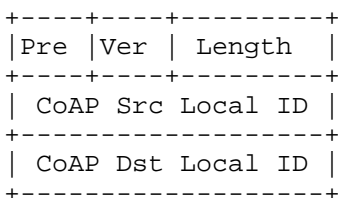


Figure 1 Shim Header Format

Fig. 1 shows a basic format of shim header. Next is the explanation of each field in the header.

- * Pre: Preamble is a 4-bit field, and has a meaning of the start of shim header. '0xa' is used currently.
- * Ver: Version is also a 4-bit field, currently the version number is '1'.
- * Length: is the length of CoAP header and its payload followed by the shim header. This has a value between 1 and 255.
- * CoAP Source Local ID, CoAP Destination Local ID: are the local IDs of the sender and receiver of its CoAP message. Each field has 16-bit field.

Fig. 2 shows the extended format of shim header. This format is used when the length of CoAP header and its payload is larger than 255. If the value of Length field is '0', this means its followed field is the 16-bit extended length field, not CoAP source ID. This field can have a range of value from 0 to 65535.

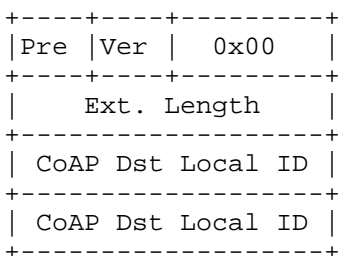


Figure 2 Shim Header Format with Extended Length

3.2. Identifying between CoAP Header and Shim Header

According to the type of constrained nodes, Shim header can be hardcoded, or not be involved in the CoAP node implementation. But at the following cases, CoAP and shim header must be identified when a packet receiver of a node meets a start point of some header.

- * The CoAP receiver has multiple heterogeneous media, and are not able to identify the received interface among them. And some media use shim header on their interfaces and the others are not.
- * The CoAP receiver receives CoAP messages from two or more other CoAP nodes.

As a result, the start point of shim header must be differentiated from it of CoAP header. According to the CoAP draft, CoAP header starts with version and message type, which has a range of values from 0x04 to 0x07. But in the case of shim header, it starts with preamble field, and its value is '0xa' to identify shim header against COAP header.

3.3. Example

As shown in [CoAP], first example in Appendix A illustrates a basic Confirmable CoAP request and response in piggy-backed manner. In this section, we add shim header to this example. In this case, the client and the server are connected via non-TCP/IP. The client can be a data collector node or a proxy node. The server may be a temperature sensor. In this example, we assume that the client has an ID of 0x0000, and the server has an ID of 0x0001.


```

Client      Server
(0x0000) (0x0001)

|          |          |
|          |          | Shim: Length=16, Src=0x0000, Dst=0x0001
|----->|          | Header: GET (T=CON, Code=0.01, MID=0x7d34)
| GET    |          | Uri-Path: "temperature"
|          |          |
|          |          | Shim: Length=11, Src=0x0001, Dst=0x0000
|<-----+|          | Header: 2.05 Content (T=ACK, Code=2.05,
|          |          | MID=0x7d34)
| 2.05   |          | Payload: "22.3 C"
|          |          |

```

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
| 0xa | 1 | Length=16 | Source ID=0x0000 |
+-----+-----+-----+-----+
| Destination ID=0x0001 |
+-----+-----+-----+-----+
| 1 | 0 | 0 | GET=1 | MID=0x7d34 |
+-----+-----+-----+-----+
| 11 | 11 | "temperature" (11 B) ...
+-----+-----+-----+-----+

```

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
| 0xa | 1 | Length=11 | Source ID=0x0001 |
+-----+-----+-----+-----+
| Destination ID=0x0000 |
+-----+-----+-----+-----+
| 1 | 2 | 0 | 2.05=69 | MID=0x7d34 |
+-----+-----+-----+-----+
| 1 1 1 1 1 1 1 1 | "22.3 C" (6 B) ...
+-----+-----+-----+-----+

```

4. Security Considerations

(TBD)

5. IANA Considerations

(TBD)

6. Acknowledgments

(TBD)

7. References

7.1. Normative References

- [CoAP] Z. Shelby, K. Hartke, C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18, June 28, 2013.
- [RFC6550] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, March 2012.
- [RFC4944] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

7.2. Informative References

- [RFC2119] S. Brander, "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [IEEE 802.15.4] IEEE Computer Society, "IEEE Std. 802.15.4-2003", October 2003.

Author's Addresses

Ilkyun Park
ETRI
1000-6 Oryong-dong, Buk-gu, Gwangju, 500-480,
Korea
Phone: +82-62-970-6651
Email: ikpark@etri.re.kr

Seok-Kap Ko
ETRI
1000-6 Oryong-dong, Buk-gu, Gwangju, 500-480,
Korea
Phone: +82-62-970-6677
Email: softgear@etri.re.kr

Seung-Hun Oh
ETRI
1000-6 Oryong-dong, Buk-gu, Gwangju, 500-480,
Korea
Phone: +82-62-970-6655
Email: osh93@etri.re.kr

Byung-Tak Lee
ETRI
1000-6 Oryong-dong, Buk-gu, Gwangju, 500-480,
Korea
Phone: +82-62-970-6624
Email: bytelee@etri.re.kr

core
Internet-Draft
Intended status: Standards Track
Expires: December 19, 2013

I. Ishaq
J. Hoebeke
F. Van den Abeele
iMinds-IBCN/UGent
June 17, 2013

CoAP Entities
draft-ishaq-core-entities-00

Abstract

This document describes a format to create entities that can be used for group communication using CoAP unicast messages.

Note

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

Status of This Memo

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

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

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

This Internet-Draft will expire on December 19, 2013.

Copyright Notice

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

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

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Requirements notation	2
2. Introduction	3
3. System Overview	4
4. Entity Creation	5
5. Validation Process	6
6. Entity Profile	7
6.1. The resources "r" entity field	7
7. Entity Usage	8
8. Examples	8
8.1. Entity Creation	8
8.2. Entity Profile	8
8.3. Entity Usage	10
9. Open topics	11
9.1. Open since v00	11
10. Security Considerations	11
11. IANA Considerations	11
12. References	11
12.1. Normative References	12
12.2. Informative reference	12
Authors' Addresses	13

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].

The above key words are used to establish a set of guidelines for CoAP entities. An implementation of CoAP entities MAY implement these guidelines; an implementation claiming compliance to this document MUST implement the set of guidelines.

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

Entity

A group of resources on CoAP servers that can be created, used or manipulated through a single CoAP request.

Entity Manager (EM)

The component that manages the entities. This component, which can reside e.g. on the Border Gateway of the LLN, is responsible for maintaining entities. Clients on the Internet can interact with an EM to create new entities and/or customize how these entities should behave.

2. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a RESTful protocol for constrained nodes. The networks that connect these nodes together are often referred to as low power and lossy networks (LLNs).

Typically, each of the constrained servers has at least one CoAP resource that may be queried by clients to obtain information about the nodes themselves (e.g. battery level), about the environment that they monitor (e.g. temperature of the room), or to trigger the nodes to perform real-world actions (switch the light on).

Depending on the application, information from individual nodes might not be sufficient, reliable, or useful. An application may need to aggregate and/or compare data from several nodes in order to obtain accurate results. In the same way, a single user request might need to trigger a series of actions on multiple actuators to perform a single user request.

Although multicast may be used to transmit the same request to several nodes [I-D.ietf-core-groupcomm], multicast communication in LLNs has some disadvantages. For instance, it is difficult to avoid duplication of messages, and duplication is undesirable in an LLN where bandwidth is limited for these constrained nodes. Furthermore, basic multicast is not reliable in an LLN, which is problematic for requests that require guaranteed delivery. Security of multicast is another issue. Currently CoAP relies on Datagram Transport Layer Security (DTLS) [RFC6347] for secure unicast communication. At the moment, DTLS requires non-standard extensions like [I-D.keoh-tls-multicast-security] to secure multicast. As demonstrated by the formation of the upcoming DTLS-IoT WG (pending a BoF at IETF 87) that aims to introduce multicast record layer support for DTLS, work is very much ongoing in this field but no standard solution is available as of today. Also, the creation of multicast groups, defining which nodes should be addressed when using a particular multicast address, is hard to realize inside LLNs. For instance, the approach in [I-D.ietf-core-groupcomm] suggests that every CoAP endpoint should implement the "core.gp" interface. Additionally, the use of multicast increases the footprint of the code that needs to fit on the constrained nodes, and it is to be expected that this functionality will not be available in many LLNs.

Consequently, in some cases the use of multicast might be not feasible or provide a suboptimal solution.

As an alternative, unicast-based solutions should be considered. Simple unicast solutions are defined in the CoRE Interfaces draft [I-D.ietf-core-interfaces]. Among other interface types, this draft defines the Batch interface type and its extension, the Linked Batch interface type. Batch interfaces are used to manipulate a collection of sub-resources at the same time. 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 [RFC6690]. The draft does not foresee any way to manipulate resources that are located on multiple smart objects with a single client request.

The current CoRE drafts do not foresee any unicast-based way to manipulate resources that are located on multiple nodes with a single client request. To overcome this shortcoming and be able to perform such composite requests, intelligence is typically added to the client application to make it communicate with the nodes individually. This leads to more complex user applications, and the added intelligence and programming cannot be shared with other applications easily. Furthermore, complex use applications may be unmanageable. Any modifications to those complex user applications may require significant testing time, thus limiting the flexibility of the user applications. Additionally a large overhead of communication between the client machine and the nodes is generated, especially when many nodes are involved in these actions. When the communication between the client and the nodes is done across the Internet, delays are unpredictable and a sequence of actuator commands might arrive out of order and possibly have unwanted results. Furthermore, if the communication occurs over costly links, communication between the client and the nodes might get unnecessarily expensive.

The discussed approaches are able to realize communication with a group of resources, but each exhibit some limitations. Therefore, in this Internet Draft we propose an alternative unicast-based approach for communication with a group of resources across multiple nodes.

3. System Overview

We call the component that manages the entities, the Entity Manager (EM). This component, which can reside e.g. on the Border Gateway of the LLN, is responsible for maintaining entities that are created from groups of CoAP servers (i.e. sensors and actuators) and/or

resources inside the LLN. Clients on the Internet can interact with an EM to create new entities and/or customize how these entities should behave. Optionally the client can elect to contact a resource directory [I-D.ietf-core-resource-directory] in order to discover which resources are available in the network.

The EM functionality does not have to be put on a dedicated device. Theoretically any CoAP server can be extended to become an EM. The choice of the most appropriate location to put the EM functionality depends on the size and topology of the network. For example, it can reside on a smart object in the constrained network with enough resources, in the Cloud, on the client device itself, or on a gateway at the edge of the LLN. The latter case has the added benefit that security can be centrally managed besides offloading the processing from constrained devices.

Regardless of the location of the EM, it will serve as a proxy between the client and the constrained devices. Client requests will be sent to the EM, which will analyze and verify the requests and then issue the appropriate requests to the constrained devices using CoAP. Once the EM receives responses from the constrained devices, it will combine them according to the client needs and will send back an aggregated response to the client.

When a client tries to create a new entity consisting of a group of resources inside LLNs, the EM performs a sanity check on the request in order to make sure that the resulting entity would make sense. For example it verifies that the resources inside the entity are valid, if they support a certain content format or if their data can be aggregated. Customization of the entity behavior is accomplished by creating profiles for the entities. A profile of an entity can specify for example whether to return the values of all resources in the entity, only the computed average of all values or a subset of all values.

4. Entity Creation

To facilitate the creation and manipulation of entities, an Entity Manager MUST implement the RESTful interface defined in this draft. A CoAP resource implementing this interface can be identified by using the resource type (rt) "core.em". We call this interface the Entity Management Interface and the corresponding resource the Entity Management Resource (available at e.g. "/e"). This interface supports only the CoAP POST request method. As payload of the request, it expects a collection of resources in CoRE link format [RFC6690], which together should form the entity. In the response, the Location-Path CoAP option MUST be used to specify the name of the newly created resource. The payload of the response is in plain text

and describes the results of the validation tests performed by the EM on the collection of resources.

When a client wants to create an entity consisting of several sub-resources, it MUST compose a CoAP POST request and send it to the Entity Management resource on the EM. The EM creates the entity, assigns it a unique URI, and stores the entity in the entity database for future usage. Then the EM starts the entity validation process (explained in the next subsection). The EM MUST inform the client about the URI to use in order to access or further customize the newly created entity and about the results of the validation of the entity. If the entity did not pass the validation process the client SHOULD fix any errors and resubmit the entity for validation again before the client can use the entity.

5. Validation Process

Whenever a client requests to create a new entity or to modify an existing entity, the EM SHOULD perform a validation process. The purpose of this validation process is twofold: 1) Make sure that the sub-resources in the entity exist and can be used. 2) Derive the properties of the entity based on the properties of the sub-resources it contains. If the entity passes validation the EM marks the entity as a valid entity and stores the entity along with its calculated properties in the entity database for future usage. If the entity fails validation it is still created, but marked as invalid. The entity validation is based on EM knowledge of the individual sub-resources through .well-known/core and their profiles and possibly based on additional functionality implemented by the EM (e.g. vendor-specific functionality).

If the Entity Manager does not know any of the subresources in an entity (e.g. based on knowledge in a resource directory) or does not know the sub-resource capabilities, it tries to obtain this information according to a fallback mechanism as follows.

- o The EM tries to contact the object containing the resource in order to obtain the resource profile, since this would provide the most complete information about the resource.
- o If the resource profile does not exist, the EM tries to obtain any information about this resource from .well-known/core of the respective object.
- o If this fails as well, the EM tries to query the resource directly to discover, at a minimum, if the resource exists or not.

The validation process that the entity manger performs on entities MUST ensure the following:

- o The individual sub-resources contained in the entity are valid (e.g. the resources exist on the respective nodes).
- o The requested operations can be performed on the individual sub-resources (e.g. which CoAP options are supported, which RESTful methods are allowed?).
- o The individual sub-resources do not conflict. A sample conflict can occur when an entity creation request contains two sub-resources on the same actuator asking it to do contradictory actions, e.g. open and close at the same time.
- o The responses sent by the individual sub-resources can be combined together by using a common denominator or by executing custom algorithms that reside at the EM.

6. Entity Profile

Once the EM knows all information about the subresources that should become part of the entity and once all necessary checks have passed, the EM SHOULD create a profile for the entity based on this information and its custom algorithms. This profile contains information related to the resource itself, as described in [I-D.greevenbosch-core-profile-description]. In addition, the profile is extended with an entity specific part, providing more information about the entity itself. The entity specific part is a JSON object with the name "entity". The value of this object is an array of entity specific fields.

6.1. The resources "r" entity field

The resources "r" entity field contains a list of the resources in the entity. It has the format depicted in Figure 1, where r1, r2, ... are strings containing the absolute URIs of the individual resources.

```
"r":[r1,r2,...]
```

Figure 1: resources "r" entity field syntax

When including the "r" entity field in the entity profile description, all individual resources of the CoAP entity MUST be included.

If the "r" entity profile field is available, the receiving party SHALL assume a non-listed URI is not a resource of the entity.

7. Entity Usage

Once an entity is created the response contains the URI of the dynamically created resource name. The client CAN now interact with the entity by issuing a single CoAP request to the resource representing the entity. When a request for an entity arrives, the following process flow SHOULD be executed.

- o The EM breaks down the request into its components and sends the individual requests to the respective objects using unicast CoAP messages. It can either do that in parallel or sequentially.
- o Once all needed answers are received, the EM creates a response for the client based on the individual responses and sends it to the client. Note that the amount of sub-resources that should respond, the way in which a response is formed and how it should look like can be configured by customizing the entity profile as will be explained later on.

8. Examples

8.1. Entity Creation

In the following simple example the client requests the creation of an entity consisting of two sub-resources: `coap://sen5.example.com/tmp` and `coap://sen8.example.com/tmp`. The EM creates the new entity, assigns it the URI `/1` and informs the client about the newly created entity. From now on, any client can access the newly created entity by accessing the `/1` resource on the EM.

```
Req: POST coap://em.example.com/e (application/link-format)
     Body: <coap://sen5.example.com/tmp>,
           <coap://sen8.example.com/tmp>
```

```
Res: 2.05 Content (text/plain)
     Body: /1 created
```

8.2. Entity Profile

Assume that the temperature sensor at `"coap://sen5.example.com/tmp"` from the previous example supports the "Uri-Host" (3), "ETag" (4), "Observe" (6), "Uri-Port" (7), "Uri-Path" (11) and "Content-Format" (12) CoAP options (op). This sensor further supports the

"application/senml+json" (55) content format (cf) and the allowed method is GET (1). This will result in Sen5 having the following profile [I-D.greevenbosch-core-profile-description]:

Req: GET coap://sen5.example.com/.well-known/profile?path=/tmp

Res: 2.05 Content (application/json)

Body:

```
{
  "profile":[
    {
      "path":"tmp",
      "op":[3,4,6,7,11,12],
      "cf":[55],
      "m":[1]
    }
  ]
}
```

Let us further assume that the second temperature sensor "coap://sen8.example.com/tmp" supports the same options as sen5 except for the observe option. Only the GET method is allowed and the supported content formats on this sensor are "text/plain" (0) and "application/senml+json" (55). Thus Sen8 will have the following profile:

Req: GET coap://sen8.example.com/.well-known/profile?path=/tmp

Res: 2.05 Content (application/json)

Body:

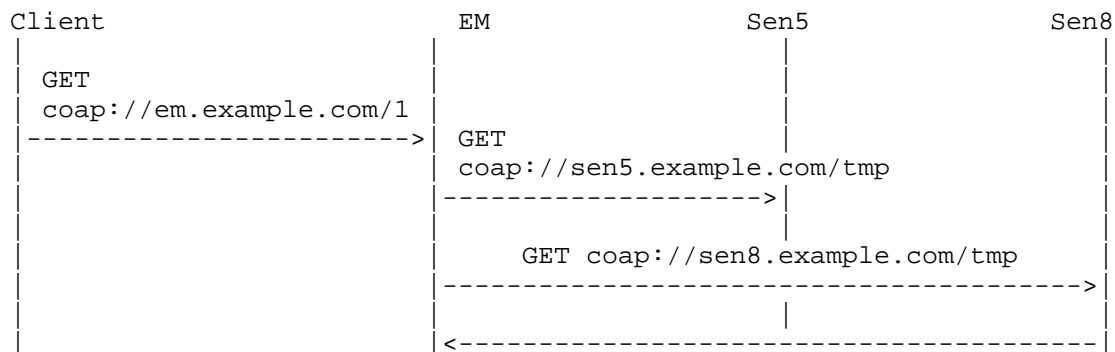
```
{
  "profile":[
    {
      "path":"tmp",
      "op":[3,4,7,11,12],
      "cf":[0,55],
      "m":[1]
    }
  ]
}
```

Based on these two profiles the EM constructs a profile for the newly created entity. This profile contains information related to the resource itself. In this example, this includes the options that are supported, the supported methods (only GET) and the content format "application/senml+json" (55). In addition, the profile is extended with an entity specific part, providing more information about the entity itself. The resulting profile of the entity looks as follows:

```
Res: 2.05 Content (application/json)
{
  "profile":[
    {
      "path":"/1",
      "op":[3,4,7,11,12],
      "cf":[55],
      "m":[1]
    }
  ],
  "entity":[
    {
      "r":["coap://sen5.example.com/tmp",
          "coap://sen8.example.com/tmp"]
    }
  ]
}
```

8.3. Entity Usage

The following Figure shows an example of using the entity that was created previous example. The client issues a GET request on the entity's resource "/1". This results in the EM issuing two GET requests to the individual sub-resources, waiting for replies from both of them and then sending back both results in one combined response back to the client.



	2.05 Content (text/plain) Body: 23.5
	<-----
	2.05 Content (text/plain) Body: 26.6
<-----	
2.05 Content (application/senml+json)	
Payload: {"e":[
{ "n": "Sen5/tmp", "v": "26.6", u="degC"},	
{ "n": "Sen8/tmp", "v": "23.5", u="degC"}]}	

9. Open topics

9.1. Open since v00

- o Use key words consistently.

10. Security Considerations

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

A client might request the creation of a large number of entities or entities that contain a large number of resources. This might lead to buffer overflow on the EM.

In an unprotected environment, an attacker can change the profile description of Entities. 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. As the Entity Manager amplifies a single requests into multiple requests per user, special care should be taken to avoid congestion and to avoid abuse of this mechanism by a malicious user that might want to flood the LLN. However, such threats are normal in environments that lack authentication.

11. IANA Considerations

- o A registry for entity profile fields as well as possible values needs to be set up.

12. References

12.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-17 (work in progress), May 2013.
- [I-D.greevenbosch-core-profile-description]
Greevenbosch, B., Hoebeke, J., and I. Ishaq, "CoAP Profile Description Format", draft-greevenbosch-core-profile-description-01 (work in progress), October 2012.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

12.2. Informative reference

- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-09 (work in progress), May 2013.
- [I-D.ietf-core-interfaces]
Shelby, Z. and M. Vial, "CoRE Interfaces", draft-ietf-core-interfaces-00 (work in progress), June 2013.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-ietf-core-resource-directory-00 (work in progress), June 2013.
- [I-D.keoh-tls-multicast-security]
Keoh, S., Kumar, S., and E. Dijk, "DTLS-based Multicast Security for Low-Power and Lossy Networks (LLNs)", draft-keoh-tls-multicast-security-00 (work in progress), October 2012.

Authors' Addresses

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

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

Floris Van den Abeele
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: floris.vandenabeele@intec.ugent.be

core
Internet-Draft
Intended status: Standards Track
Expires: December 26, 2013

Shi. Li
Huawei Technologies
J. Hoebeker
F. Van den Abeele
iMinds-IBCN/UGent
A. Jara
University of Murcia
June 24, 2013

Conditional observe in CoAP
draft-li-core-conditional-observe-04

Abstract

CoAP is a RESTful application protocol for constrained nodes and networks. Through the Observe option, clients can observe changes in the state of resources and obtain a current representation of the last resource state. This document defines a new mechanism in CoAP Observe so that a CoAP client can conditionally observe a resource on a CoAP server, only being informed about state changes meeting a specific condition or set of conditions. This offers possibilities to extend network intelligence, enhance scalability, and optimize the lifetime and performance in order to address the requirements from the Constrained Nodes and Networks.

Note

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

Status of This Memo

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

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

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

This Internet-Draft will expire on December 26, 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. Justification	3
1.2. Terminology	4
2. Motivation	4
2.1. Comparison to RESTful method	5
3. The Condition Option	5
4. Condition Types	9
5. Using the Condition Option	11
6. Cancellation, updating and existence of conditional relationships	12
6.1. Cancellation and updating	12
6.2. Existence	13
7. Discovery	15
8. Examples	16
9. Security Considerations	22
10. IANA Considerations	22
10.1. Condition option registry	22
10.2. Condition type registry	23
11. Further considerations	23
12. Acknowledgements	24
13. Normative References	24
Appendix A. OMA Information Reporting with CoAP Conditional Observe	24
Appendix B. Alternative approaches	27
B.1. Annex: Cancellation flag	27
B.2. Annex: Logic flag	28
Appendix C. Change log	29
Authors' Addresses	30

1. Introduction

CoAP [I-D.ietf-core-coap] is an Application Protocol for Constrained Nodes/Networks. The observe [I-D.ietf-core-observe] specification describes a protocol design so that a CoAP client and server can use the subject/observer design pattern to establish an observation relationship. When observe is used, the CoAP client will get a notification response whenever the state of the observed resource changed. However, in some scenarios, the CoAP client may only be interested in a subset of state changes of the resource, other state changes might be meaningless. This inability to suppress additional notifications results in superfluous traffic. This memo defines a new CoAP option "Condition" that can be used to allow the CoAP client to condition the observe relationship, and only when such condition is met, the CoAP server shall send the notification response with the latest state change. When such a condition fails, the CoAP server does not need to send the notification response.

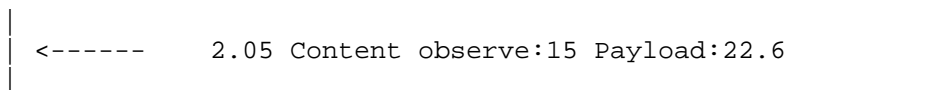
1.1. Justification

A GET request that includes an Observe Option creates an observation relationship. When a server receives such a request, it first services the request like a GET request without this option and, if the resulting response indicates success, establishes an observation relationship between the client and the target resource. The client is notified of resource state changes by additional responses sent in reply to the GET request to the client.

CoAP is used for Constrained Networks, especially used for transporting sensor data. Different sensor equipments have different properties, e.g. different change rates, data unit, different response time, etc. resulting in varying clients' interests that differ from mere state changes. As such, when a client wants to collect information from a sensor, it does not want to receive useless information. In addition, this would cause the transmission of irrelevant information in an already constrained network.

Consider the following example.

CLIENT		SERVER
GET/temperature observe:0	----->	
<----- 2.05 Content observe:5 Payload:22		
<----- 2.05 Content observe:10 Payload:22.3		



The diagram shows a network packet structure. It consists of a header field containing the text "<-----" followed by a payload field containing the text "2.05 Content observe:15 Payload:22.6". Vertical lines on either side represent the packet boundaries.

Figure 1: GET request with observe

In this example, the sensor acts as a server, and it collects the resource data every 5 seconds. When the client observes a resource on the server, it will receive a response whenever the server updates the resource, that is to say, mostly every 5 seconds the client will receive a notification response. However, the client might be a quite simple equipment not too sensitive to the resource state change, so it may not want to receive notifications that often. One possible solution could be to alter the sensor's configuration, e.g. to shorten the collecting frequency. However, the sensor should be able to provide services to many other clients, making it hard to find the best configuration that fits all clients' requirements.

1.2. Terminology

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

2. Motivation

The CoAP Observe Option gives clients the ability to observe changes in the state of resources. A notification relationship is established and whenever the state of the resource changes, the new representation is pushed to the observer. In many cases, an observer will typically be interested in state changes that satisfy a specific condition. In addition, similar conditional observations will prove useful for many different resources. For example, being informed when the state of a resource exceeds a specific value.

Defining an agreed set of commonly used conditional observations has a number of advantages. In a well-defined way, clients can observe different resources conditionally. At the same time, these resources can clearly announce how they can be observed, facilitating machine processing of this information. Also, intermediaries can process multiple conditional observations, having as goal to alleviate the load on the constrained network and devices. In the absence of such a set of commonly used conditional observations, where every application developer can specify its own mechanisms, these advantages are lost.

2.1. Comparison to RESTful method

In [I-D.shelby-core-interfaces] a simple RESTful mechanism is described to provide additional conditions to the Observe Option through the use of URI query parameters. It proves that conditional observations are useful but begs the question how the two approaches stack up to each other. The authors think that both approaches come with their advantages and disadvantages. Here we try to give an overview for both approaches. This short list of arguments for and against both approaches isn't meant to be exhaustive. We as authors of this draft are aware that we are - by definition - biased and we welcome any feedback via our contact details and/or via the CoRE mailing list.

When using the RESTful approach care should be given to avoid overlap with URI query parameters that a resource might want to use itself. Therefore a clear RESTful interface should define how all the possible condition type are mapped to said interface. Specifying conditions via URI query parameters also requires a separate request for the conditions because - by default - all URI query parameters in a request are part of the Cache-Key. Furthermore, it makes identifying conditions harder when compared to a dedicated CoAP condition option. The current approach also doesn't allow multiple clients observing the same resource with different conditions. Including this into the RESTful approach, while keeping caching in mind, has yet to be considered. Another unspecified matter is signalling which condition types are supported by a resource.

The main advantage of the approach proposed in the interfaces draft is that due to its RESTful design it doesn't rely on Application protocol specific mechanisms (like CoAP options). This means that the interface can easily be supported by any other (application) protocol that supports the RESTful paradigm. It also means that servers and clients can limit their implementation of CoAP to the basic building blocks of the protocol and use those to add additional logic (in the form of RESTful interfaces) as opposed to supporting yet another option. Another - albeit a purist - argument is that CoAP options should limit themselves to controlling the CoAP protocol and should not influence the representations offered by a resource, as these should be defined within the REST interactions themselves.

3. The Condition Option

Type	C/E	Name	Data type	Length	Default
18	E	Condition	uint	1-5 B	(none)

Table 1: Condition Option number

The Condition Option has the option number 18. The last bit indicates it is an elective option and the second to last bit indicates that this is a Proxy Unsafe option (similar to the Observe Option). The delta between the Condition Option and the Observe Option is 12.

The Condition Option can be present in both request and response messages. In both cases, it must be used together with the Observe Option, since it extends the meaning of the Observe Option.

In a GET request message, the Condition Option represents the condition the client wants to apply to the observation relationship. It is used to describe the resource states the client is interested in.

In the response to the initial GET request message, the Condition Option, together with the Observe Option, indicates that the client has been added to the list of observers and that notifications will be sent only when the resource state meets the condition specified in the Condition Option. In all further notifications, the Condition Option identifies the condition to which the notification applies.

Basically, a similar procedure as described in the observe draft [I-D.ietf-core-observe] is followed, but extended with additional behaviour by taking the Condition Option into account. The exact semantics are defined in the sections below.

The size of the Condition Option value is not fixed and can range from 1 to 5 bytes. The value carried is in a specific format that consist of two parts: a mandatory condition header and an optional condition value. The condition header has a fixed length of 1 byte and the condition value, when present, can range from 1 to 4 bytes. The condition header consists of 3 fields: the condition type (TYPE), reliability flag (R) and value type (V).

The Condition Option may occur more than once. If multiple Condition Options are present in a request, their relationship is "AND", meaning that the server will only send a notification when both conditions are fulfilled. In the notifications to such a request, the same Condition Options are present. The Figure 10 presents an example of a multiple condition with "AND" conjunction.

Note that in order to establish an "OR" relationship between conditions, a client simply needs to send separate requests using

different source endpoints. The Figure 11 presents an example of OR condition with multiple requests, which are sent in two messages via different source transport ports.

Since this solution could be considered as non-optimal, an alternative solution is proposed for discussion in the Annex "Logic flag", where multiple OR relationships with multiple conditional options can be defined, similar as has been presented for the "AND" conjunction.

In case of multiple Condition options, the main reason for choosing "AND" semantics over "OR" semantics is motivated by the fact that in case of the "OR" semantics the notification must be send anyway (as opposed to the "AND" case). This choice minimizes the amount of useless notifications that are sent over the network. The authors foresee that for some constrained devices supporting more than 1 condition per relationship might not be possible, in that case the client should try to establish the required behaviour using a minimal number of 1-condition relationships and filtering out unwanted notifications at the client-side.

```

      0
    0 1 2 3 4 5 6 7
+-----+
| TYPE      |R| V |
+-----+

      0          1
    0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+-----+-----+
| TYPE      |R| V |          VAL      |
+-----+-----+

      0          1          2
    0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 . . 7
+-----+-----+-----+
| TYPE      |R| V |          VAL      |
+-----+-----+-----+

      0          1          2          3
    0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 . . 7 0 . . 7
+-----+-----+-----+-----+
| TYPE      |R| V |          VAL      |
+-----+-----+-----+-----+

      0          1          2          3          4
    0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 . . 7 0 . . 7 0 . . 7
+-----+-----+-----+-----+-----+

```



```

+-----+-----+-----+-----+-----+-----+-----+-----+
| TYPE   | R | V |                                     VAL                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 2: Condition Option value

TYPE: The condition type is a 5 bit integer indicating the type of the condition in the observe request or in a notification. Every value of TYPE represents the type ID of a specific condition type. Every condition type can be complemented by a condition value in the VAL field, further specifying the condition in more detail. Below is the definition of identified condition types.

R: In an observe request, the reliability flag indicates whether notifications for that condition need to be send non-confirmable (0) or confirmable (1). In the initial response, this flag indicates the server's willingness or ability to send the notifications confirmable or non-confirmable, as requested by the client. In all further notifications, this flag can be changed depending on the server's decision. In case of a request containing multiple Condition Options, the client must use the same value of the R flag in all Condition Options. If the server receives a request with multiple Condition Options, which do not all share the same value of the R flag, the server MUST respond with a 4.00 "Bad Request" response code. Note that the observe draft states that the message type of a notification is independent from the type used for the request or any previous notification. The R flag doesn't violate this behaviour and a client should expect that a CON notification might arrive without this being explicitly signalled by the server (as a NON-notification signalling this, might not arrive).

V: The value type is a 2 bit field in a request or response that gives the data type of the condition value, when present in the Condition Option. If no condition value is present, this field has no meaning and must be 0. Table 2 gives an overview of all available value types. The Duration data type is defined in Appendix C.2 of [I-D.bormann-coap-misc]. The representation of floating point numbers in a common format that is understandable by constrained devices is outside the scope of this document.

Value type (2 bit)	Id.
Integer	0
Duration (s)	1

Float	2	
+-----+	+-----+	+-----+

Table 2: Value types

VAL: The condition value is 1 to 4 byte value of the type indicated by the V flag. The condition value is used to indicate the value that further specifies the condition type (e.g. a threshold). Condition types can require the presence of a condition value. When a condition value is present in an observe request, the same value must be used in the initial response. In all further notifications, the condition value can be left out to reduce the size of the option.

4. Condition Types

Table 3 gives an overview of all currently identified condition types. If supported by the server, different condition types can be combined in a request to express a logical AND relationship. By default, logical OR of condition types is always supported through sending separate requests using different source endpoints.

Condition type (5 bit)	Id.	Condition Value	
Cancellation	0	no	
Time series	1	no	
Minimum response time	2	yes	
Maximum response time	3	yes	
Step	4	yes	
AllValues<	5	yes	
AllValues>	6	yes	
Value=	7	yes	
Value<>	8	yes	
Periodic	9	yes	

Table 3: Condition types

Time series: This condition indicates that a client wants to receive all state changes of a resource, but that it does not want to receive a notification in case the time since the last notification was sent became greater than the max-age of the resource and the resource did not change during this period. --> This is a variant of the observe draft [I-D.ietf-core-observe] that deals with eventual consistency, which may result in notifications even if the resource did not change in order to ensure the observer has a fresh representation of the resource. Note that the observe draft states that 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. In case of the Time series condition a client might opt to divert from this behaviour and use these older notifications anyway. The client signals this to the server by choosing a Time series condition value larger than the Max-Age of the resource and including this option in its observe request. In case the server thinks this behaviour isn't feasible, it signals this to the client by removing the time series Condition option from its response. In case the server's response does echo the Time series condition, then the client is allowed to divert from the behaviour specified in the observe draft and use these 'older' notifications. Similar considerations hold true for the other timing condition types that are defined in this draft.

Minimum response time: For this condition, the value specified in the condition value field gives the minimum time in seconds the server should leave between subsequent notifications.

Maximum response time: For this condition, the value specified in the condition value field gives the maximum time in seconds the server is allowed to leave between subsequent notifications.

Step: For this condition, the value specified in the condition value field gives the minimum state change of a resource (since the last notification) before the server should send a new notification.

AllValues<: This condition indicates that a client is only interested in receiving notifications whenever the state of the resource changes and the value is less than the value specified in the condition value field.

AllValues>: This condition indicates that a client is only interested in receiving notifications whenever the state of the resource changes and the value is greater than the value specified in the condition value field.

Value=: This condition indicates that a client is only interested in receiving notifications whenever the state of the resource changes

and the value is equal to the value specified in the condition value field.

Value<>: This condition indicates that a client is only interested in receiving a single notification whenever the state becomes higher or lower than the value specified in the condition value field. Once the notification has been sent, no new notifications are sent for subsequent state changes where the value remains higher or lower. As such, a single notifications is sent whenever a threshold is passed in either direction.

Periodic: This condition indicates the periodic interval in seconds with which new notifications should be sent.

5. Using the Condition Option

Whenever a client wants to initiate a Conditional Observation relationship, it sends a GET request with both an Observe and at least one Condition Option. The Condition Option extends the meaning of the Observe Option by including a condition that describes the resource states the client is interested in. It represents the condition the client wants to apply to the observation relationship.

When a server receives such a request, it first services the request the same way as described in [I-D.ietf-core-observe]. Next, if the server supports the Condition Option, it analyses the Condition Option to find the condition requested by the client. If the condition is supported, the relationship is stored and the client is informed about the successful establishment of the conditional relationship. This is done by sending a response containing both the Observe and Condition Option, which implies that the client has now been added to the list of observers and will only be informed about state changes or resource states satisfying the condition described in the Condition Option.

Since the Condition Option is elective, an observe request that includes the Condition Option will automatically fall back to a basic observe request if the server does not support the Condition Option. There is no default value for the Condition Option. Also, if the Condition Option is supported, but the requested condition is not supported by the resource, the request will also fall back to a basic observe request, resulting in a response only containing the Observe Option. This implies that the client will now receive notifications as described in [I-D.ietf-core-observe] and that the client itself is responsible for processing the resource state in the notifications in order to identify whether the condition of interest is fulfilled.

Whenever the state of a resource that supports conditional observations changes on the server, the server needs to check the established conditional relationships. Whenever the relationship condition(s) is(are) met, the server sends the notification response to the client that has established the relationship. In case the server is still waiting for a confirmable notification to be acknowledged or the 3 seconds on average for a non-confirmable notification to elapse, it MUST adhere to the behaviour specified in section 4.5 of [I-D.ietf-core-observe]. The response contains both the Observe Option and the Condition Option, where the latter option describes the condition that has been fulfilled. If not met, the server does not send any response to the client. Furthermore, the server also doesn't send a response when the last notification is older than Max-Age.

A client is allowed to use multiple Condition Options in an observe request in order to express a logical AND relationship between different condition types. When a server receives such a request and it does not support composed conditions, the request will also fall back to a basic observe request, resulting in a response only containing the Observe Option. If the server supports this, it will store the relationship and send back a response echoing the same multiple Condition Options.

In case a client wants to establish multiple different conditional relationships with the same server, it needs to use a different source endpoint for every conditional relationship.

6. Cancellation, updating and existence of conditional relationships

6.1. Cancellation and updating

In case a client wants to cancel an existing conditional relationship, it has to perform a GET request to the target resource without Observe and Condition Option using the same source endpoint used to establish the conditional relationship (i.e. source endpoint of the original request). Upon reception of such a GET request, the server will remove the client from the list of conditional observers for that resource.

Alternatively, the client can also send a confirmable request containing a Condition Option with condition type `_Cancellation_`. The source endpoint used by the client together with the request URI uniquely identifies the conditional relationship the client has with the server. Upon reception of such a message, the server knows that the client wants to terminate the relationship it has established.

This cancellation mechanism implies that whenever a client wants to establish multiple different conditional relationships with the same resource on the same server, it needs to use a different source endpoint for every conditional relationship.

When a client has established a conditional relationship and it sends a new conditional observe request using the same source endpoint to the same resource, the existing relationship is removed and the new relationship established. This way, a client is able to update existing relationships.

Apart from the cancellation through sending a GET request without Observe and Condition Option, a conditional relationship can also be cancelled by sending a RST message in response of a confirmable notification. When a client rejects a non-confirmable notification with a RST, the server can remove the client from the list of observers interested in the specific condition of the resource in case the server maintains state information about non-confirmable notifications.

Additionally, if the server is for whatever reason not able to further fulfill the conditional relationship of a client, the server can also send a confirmable notification containing a Condition Option with condition type 'Cancellation' (echoing the Token in the notification). The client's endpoint and the request URI uniquely identify a conditional relationship with a server. As such, upon reception of such a message, the client can infer the request URI from the request associated with the Token contained in the response and knows the relationship that has been terminated by the server.

Finally, when a server sends a confirmable notification that is not acknowledged by the observer, the server may terminate the relationship after the last unsuccessful delivery attempt of said notification.

Note: The possibilities to establish a Cancellation flag have also been evaluated, see Annex "Cancellation flag". This has been discarded since, as it is too complex for processing, when multiple conditions are defined.

6.2. Existence

The observe draft [I-D.ietf-core-observe] specifies that at a minimum a server must send a notification in a confirmable message at least every 24 hours. In case of monitoring critical events this 24-hour interval is most likely too short. Therefore additional, more flexible mechanisms for checking the existence of a (conditional) relationship are needed. This paragraph presents two ideas in the

form of CoAP options that augment the default periodic confirmable message.

A client has the possibility to establish and remove conditional relationships and a server can also inform a client about the removal of a conditional relationship. Next to this, there is the issue of how long the relationship is guaranteed to exist in the absence of any explicit removal from either the client or server side (e.g. a client that wants to maintain the relationship for a very long time) or in the absence of frequent notifications. To this end, a mechanism is needed for a client to know whether it is still on the list of observers and for a server to know whether a client is still an observer.

In order for a client to know whether it is still on the list of observers after a long period without notifications or without confirmable notifications, the server can use the Pledge Option, as defined in [I-D.bormann-coap-misc]. By adding this option to its notifications, the server indicates how long it minimally promises to maintain that specific conditional relationship. In case no new notifications or non-confirmable notifications are sent and the duration indicated in the Pledge Option is to expire, the client must renew the relationship by resending the same request, preferable as a confirmable message.

In case the duration indicated in the Pledge Option expires at the server side and the client did not renew the relationship, the server must remove the relationship by sending an explicit cancellation message (a confirmable notification to the client's source endpoint containing a Condition Option with condition type 'Cancellation'). As such, the use of the Pledge Option extends the establishment and removal mechanism with a server-initiated mechanism to realize intermediate refreshments of the relationship.

The second mechanism, for a server to determine whether a client is still an observer, is realized by adding a Keep-Alive Option to the observe request. The size of the Keep-Alive Option value is 1 byte and represents a duration in seconds, using the Duration data type as defined in Appendix C.2 of [I-D.bormann-coap-misc].

Type	C/E	Name	Format	Length	Default
30	E	Keep-alive	Duration in s	1 B	(none)

Table 4: Keep-alive Option number

The Keep-Alive Option has the option number 30, meaning that it is an elective, Proxy Safe option, but not a cache key.

When the client adds the Keep-Alive option to its conditional observe request, it requests the server to confirm that the relationship is still alive every time the duration expires and no notifications or only non-confirmable notifications have been sent during that period. If the option is supported by the server, the option is echoed in its first response. Every time the duration expires and no notifications or only non-confirmable notifications have been sent, the server sends a confirmable notification to the client with an empty payload (since the condition is not fulfilled). As such, the use of the Keep-alive Option extends the establishment and removal mechanism with a client-initiated mechanism to realize intermediate refreshments of the relationship. Furthermore it allows a client to explicitly specify the 24-hour interval mentioned in the observe draft.

The Pledge Option allows a server to request a client to confirm its interest and the Keep-Alive option allows a client to request a server to confirm whether it is still an observer. In case neither of the two options are supported, the only way for the client to ensure the relationship is still existing in the absence of incoming notifications is to periodically re-establish the relationship and the only way for the server to ensure the client is still interested is to send a confirmable notification from time to time.

7. Discovery

The Condition Option enables the establishment of well-defined set of conditional observations. It is equally important for a resource to be able to announce in a well-defined way which conditional observations it supports. Clients can then discover these capabilities and process them automatically.

In [I-D.ietf-core-observe], the "obs" attribute is introduced as a target attribute. It is used as a flag without any value, indicating that the resource is observable. In order to describe the conditional observe capabilities of a resource, a value is added to the obs attribute. To describe which of the 2^5 possible condition types a resource supports, a 32-bit value is used where a bit-value of 1 at position X (starting from 0 and from right to left) indicates that the condition type X is supported. As such, by a client can discover the supported observe capabilities by parsing the value of the "obs" attribute. In case no value is present, the "obs" attribute preserves its original meaning.

8. Examples

This section gives a number of short examples with message flows to illustrate the use of Condition Option in a observe GET request. Note that, in order to keep the figures readable, the Condition Option that is included in every notification is not shown.

The first example (Figure 3) shows that the client sets the Condition Option to Type: 1 - No Value (1/-). The condition is Time Series, meaning that the client wants to receive all state changes. In case the state of the resource does not change, no notifications are sent, also not in case max-age expires. We assume a max-age value of 60 seconds.

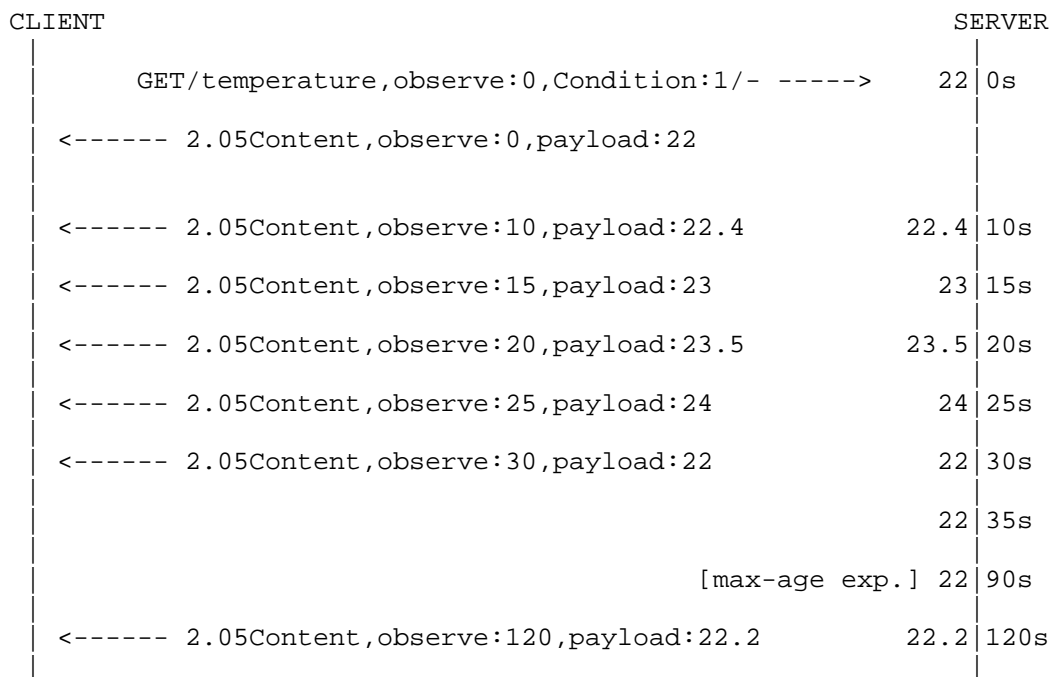


Figure 3: Condition Option with value 1/- (Time Series)

The following example (Figure 4) shows that the client sets the Condition Option to Type: 2 - Value: 10 (2/10), This means that the server shall wait at least 10s between sending notification responses, indicating changes in the state of the resource, to the client.

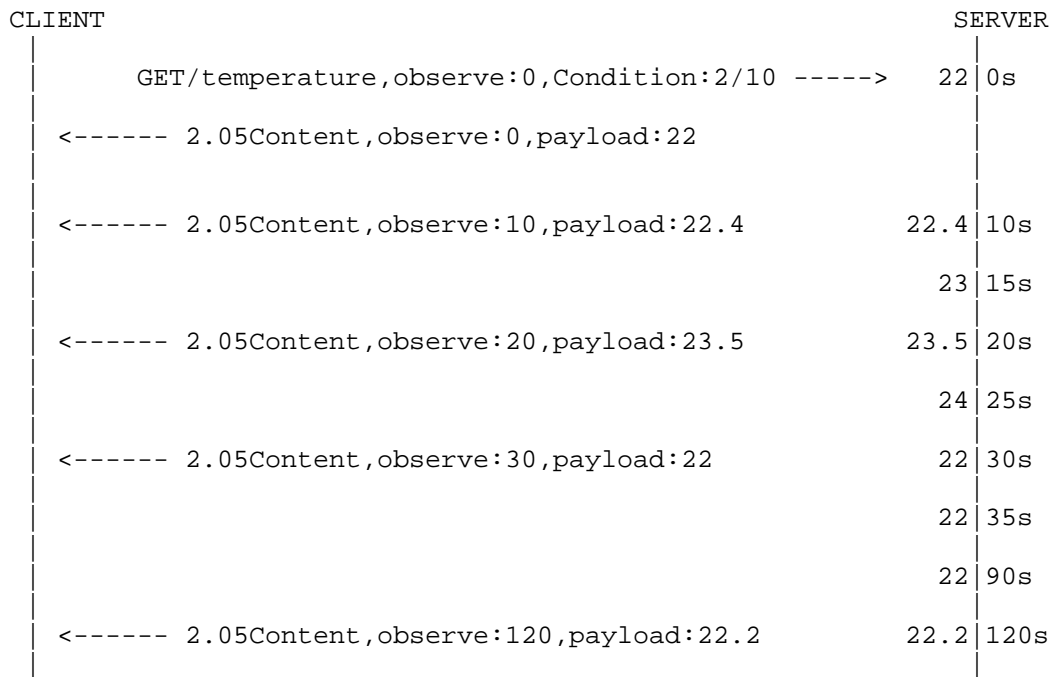
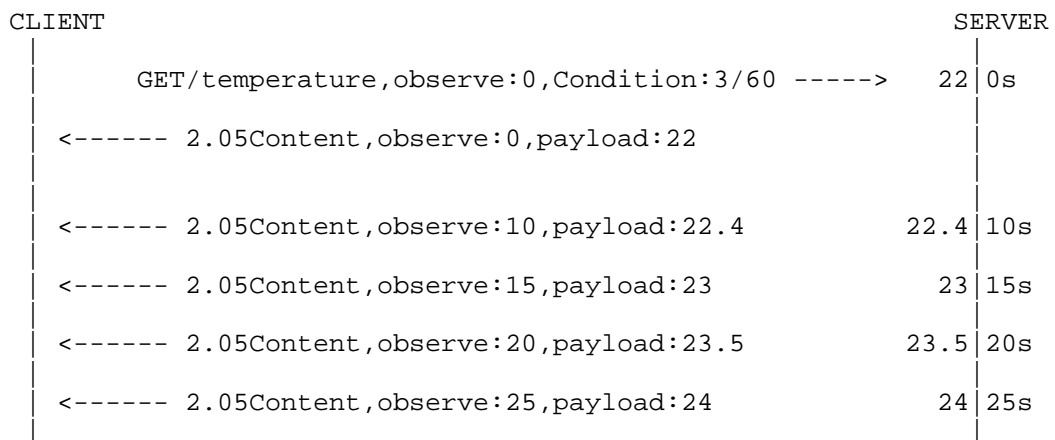


Figure 4: Condition Option with value 2/10 (Minimum Response Time)

The next example (Figure 5) shows that the client sets the Condition Option to Type: 3 - Value: 60 (3/60). The server will send notifications upon every state change, but will leave maximally 60s between subsequent notifications, even if they do not incur a state change.



<----- 2.05Content,observe:30,payload:22	22		30s
	22		35s
<----- 2.05Content,observe:90,payload:22	22		90s
<----- 2.05Content,observe:120,payload:22.2	22.2		120s

Figure 5: Condition Option with value 3/60 (Maximum Response Time)

Figure 6 shows a client setting the Condition Option to Type: 4 - Value: 1 (4/1). The server will now send notifications every time the change in state of the resource is at least 1.

CLIENT		SERVER	
GET/temperature,observe:0,Condition:4/1 ----->	22		0s
<----- 2.05Content,observe:0,payload:22			
		22.4	10s
<----- 2.05Content,observe:15,payload:23		23	15s
		23.5	20s
<----- 2.05Content,observe:25,payload:24		24	25s
<----- 2.05Content,observe:30,payload:22		22	30s
		22	35s
		22	90s
		22.2	120s

Figure 6: Condition Option with value 4/1 (Step)

The example in Figure 7 shows that the client sets the Condition Option to Type: 6 - Value: 23 (6/23). The server will send notifications to the client only if the resource value is bigger than 23.

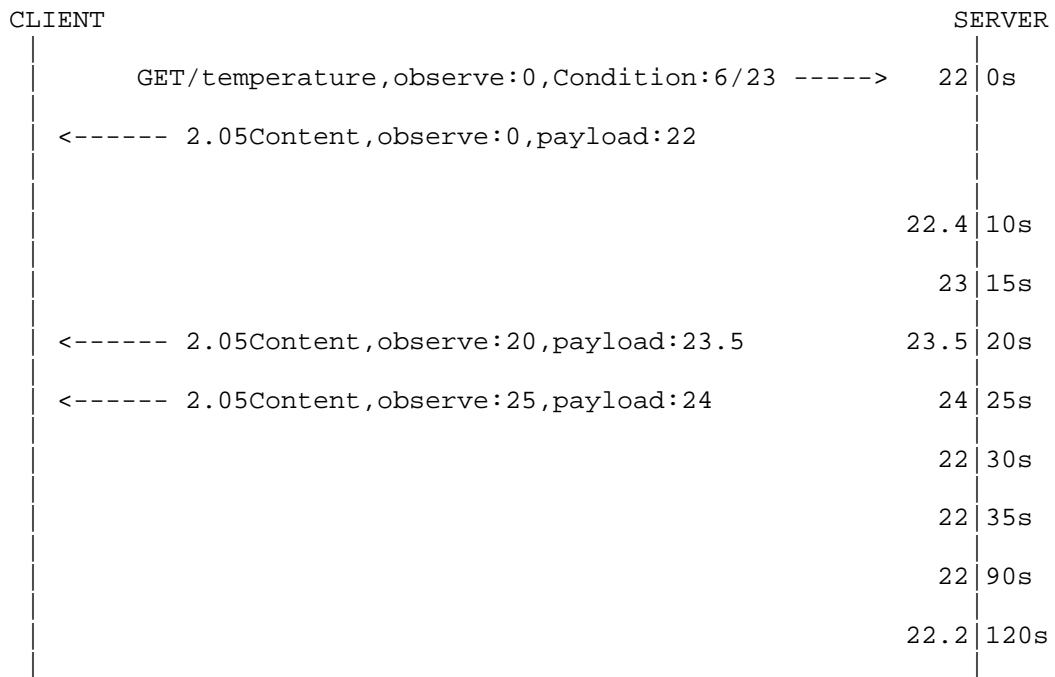


Figure 7: Condition Option with value 6/23 (AllValues>)

Figure 8 is an example of a client setting the Condition Option to Type: 8 - Value: 23 (8/23). The server will send a single notification whenever the state becomes higher or lower than 23.

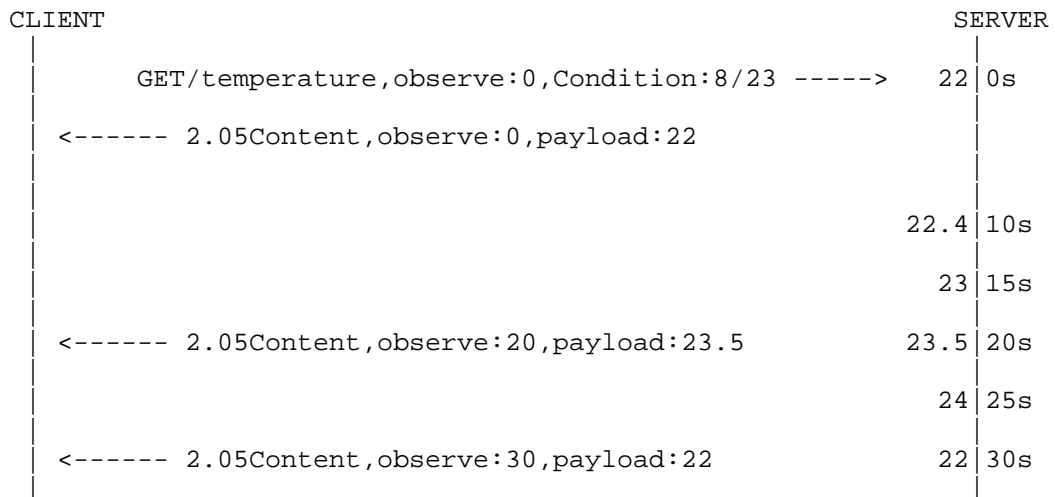




Figure 8: Condition Option with value 8/23 (Values<>)

Figure 9 is an example of a client setting the Condition Option to Type: 9 - Value: 30 (9/30). The server will send notifications every 30 seconds, independent whether the resource has changed or not.

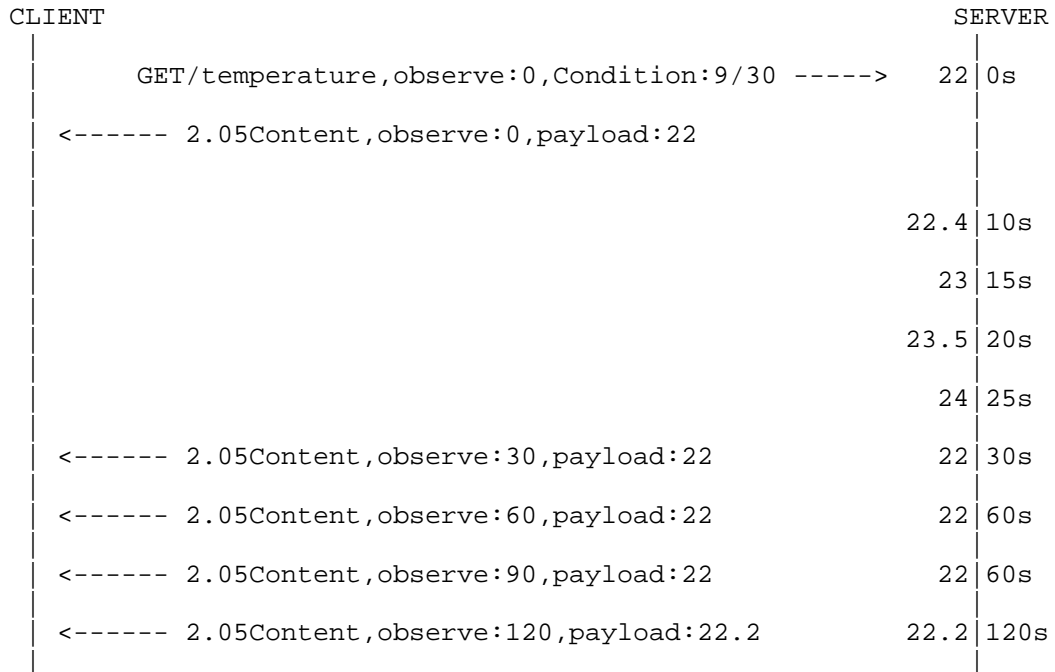


Figure 9: Condition Option with value 9/30 (Periodic)

In the following examples, we illustrate the combination of different conditions. The example in Figure 10 shows the client adds two range Condition Options in the request, one set to 6/5, another one set to 5/15. It means that the range is within 5 and 15, i.e. value > 5 AND value < 15. Since it is an AND condition, the two conditions can be specified in the same observe with multiple options.

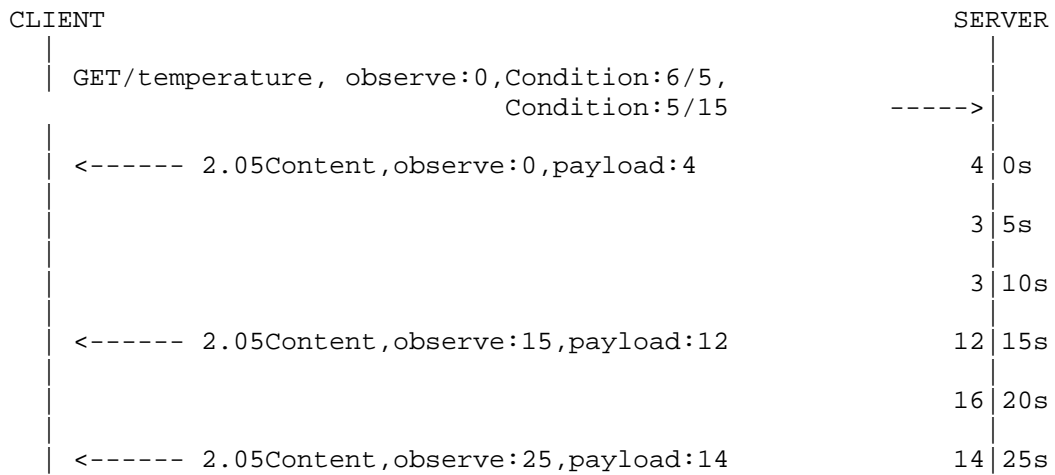


Figure 10: Two Condition Options to define in-side a range option
6/5 AND 5/15 (AllValues> AND AllValues<)

The last example (Figure 11) shows the client adds two Observe request messages to build a range, one sets to 6/22, another one sets to 5/16. It means that the range is out of range between 16 and 22, i.e. value > 22 OR value < 16. This requires two messages, since it is the OR option, which is defined with multiple observe messages. An example of the application for this option can be found in the implementation of the conditional observer [SENSORS].

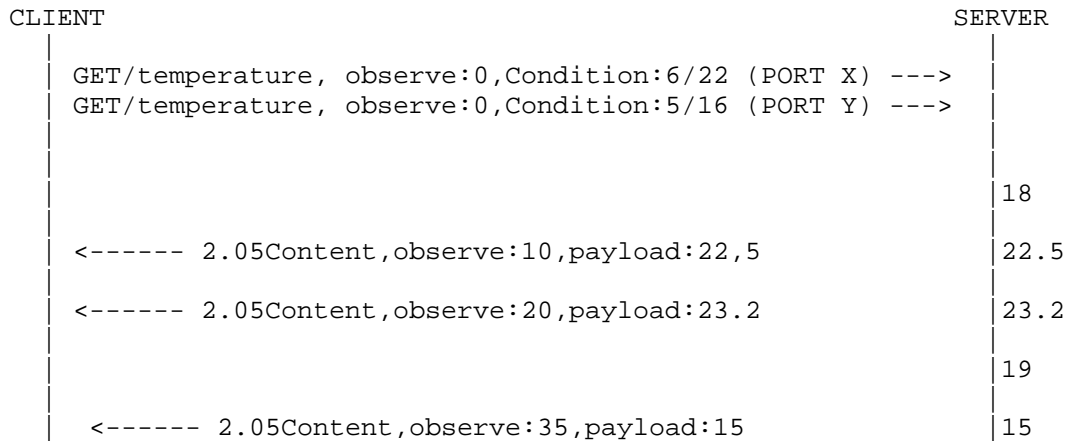


Figure 11: Two Observe requests to define out-side a range
6/22 OR 5/15 (Allvalues> OR Allvalues<)

Further, in [CPSCOM], an evaluation can be found regarding the feasibility of implementing conditional observations on real constrained devices, together with a basic performance comparison between conditional observe (server-filtering) and normal observe in combination with client-side filtering.

9. Security Considerations

As the Condition Option is used together with the Observe option, when it is used it must follow the security considerations as described in Observe draft [I-D.ietf-core-observe].

10. IANA Considerations

10.1. Condition option registry

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

Number	Name	Reference
26	Condition	[RFCXXXX]
28	Keep-alive	[RFCXXXX]

Table 3: Condition and Keep-alive Option number

10.2. Condition type registry

The Condition types defined in this draft are identified by a string, such as "Step". In order to minimize the overhead of using these condition types, this document defines a registry for the condition types to be used in CoAP and assigns each a numeric identifier.

Each entry in the registry must include the condition type registered with IANA, the numeric identifier in the range 0-31 to be used for that condition type in CoAP, and a reference to a document defining the usage of that condition type.

Initial entries in this registry are as follows:

Condition type	Id.	Reference
Cancellation	0	[RFCXXXX]
Time series	1	[RFCXXXX]
Minimum response time	2	[RFCXXXX]
Maximum response time	3	[RFCXXXX]
Step	4	[RFCXXXX]
AllValues<	5	[RFCXXXX]
AllValues>	6	[RFCXXXX]
Value=	7	[RFCXXXX]
Threshold	8	[RFCXXXX]
Periodic	9	[RFCXXXX]

Table 5: Condition Option type

11. Further considerations

Intermediaries, caching, retransmissions

12. Acknowledgements

Thanks to the IoT6 European Project (STREP) of the 7th Framework Program (Grant 288445).

13. Normative References

- [CPSCOM] Ketema, G., Hoebeke, J., Moerman, I., Demeester, P., Li, Shi., and A. Jara, "Efficiently observing Internet of Things Resources", The 2012 IEEE International Conference on Cyber, Physical and Social Computing November 20-23, 2012, Besancon, France, Novemer 2012.
- [I-D.bormann-coap-misc] Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-13 (work in progress), March 2012.
- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-17 (work in progress), May 2013.
- [I-D.ietf-core-link-format] Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-14 (work in progress), June 2012.
- [I-D.ietf-core-observe] Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.
- [I-D.shelby-core-interfaces] Shelby, Z. and M. Vial, "CoRE Interfaces", draft-shelby-core-interfaces-05 (work in progress), March 2013.
- [OMADM] Alliance, OMA., "Lightweight Machine to Machine Technical Specification", OMA-TS-LightweightM2M-V1_0-20130301-D Open Mobile Alliance (OMA), March 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [SENSORS] Castro, M., Jara, A., and A. Skarmeta, "Architecture for Improving Terrestrial Logistics Based on the Web of Things", Sensors 12, no. 5, 6538-6575, 2012, May 2012.

Appendix A. OMA Information Reporting with CoAP Conditional Observe

The Open Mobile Alliance (OMA) has defined the Lightweight Machine to Machine (LWM2M) Technical Specification [OMADM]. This specification uses CoAP as the transport for the device management in terms of registration, service discovery, bootstrapping, service enablement, and information reporting.

Information reporting presents a similar philosophy that CoAP conditional observe. In details, it is defined for the periodic reporting of data, or an event-triggered reporting based on changes in some of the resource values.

Information reporting allows to send an Observe GET request for an Object Instance (Objects is the organization of resources in OMA LWM2M), which results in asynchronous notifications whenever that Object Instance changes (periodically or as a result of an event). The minimum and maximum period of notifications can be controlled by including the minimum (pmin) and/or maximum (pmax) period for notifications to be sent in seconds.

An example of the logical operation defined by OMA LWM2M is a GET with Observe option:

```
//{Object ID}/{Object Instance ID}/{Resource ID}  
?pmin={minimum period}&pmax={maximum period}
```

The answer is a 2.05 Content with Observe option, 4.00 Bad Request, 4.04 Not Found, 4.05 Method Not Allowed, or in case of Asynchronous Response 2.04 Changed.

Note, that the resources are logically organized into Objects. Thereby, multiple resources are defined per Object, and each resource is given a unique identifier within that Object. Each Resource is defined to have one or more Operations that it supports. A Resource MAY contain multiple instances as defined in Object specification. An Object defines a grouping of Resources. Object MUST be instantiated, which is called Object Instance before using the functionality of an Object. After Object Instance is created, that Object Instance and Resources which belong to that Object Instance. As a result, it is required to reference the Object ID, Object Instance ID, and finally Resource ID.

The operations defined for the information reporting interface are: observe, notify, and cancel observation.

First, the observe interface, following the OMA LWM2M requirements, needs to define the Minimum and Maximum period parameters.

Minimum period indicates the minimum time in seconds the device SHOULD wait between sending a new notification.

Maximum period indicated the maximum time in seconds the device SHOULD wait between sending the next notification (regardless if the value has changed).

These options correspond with the Minimum response time (option 2), and Maximum response time (option 3) of the presented Conditional Observe.

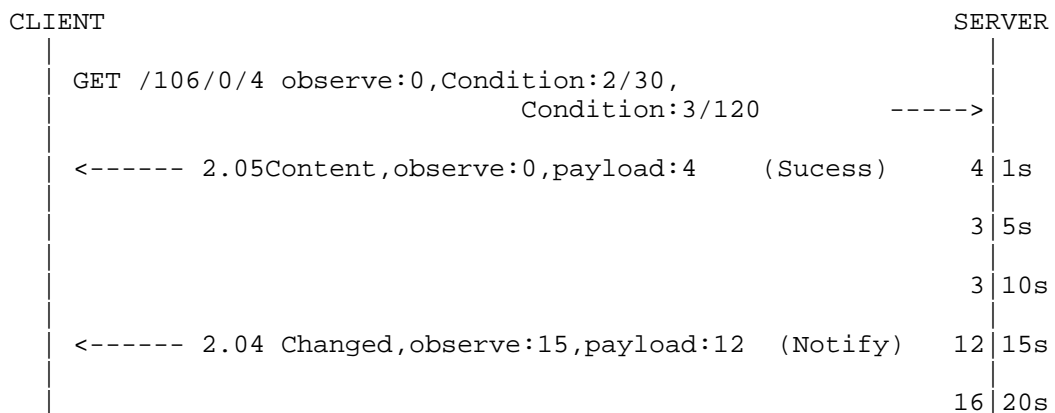
Second, the cancel observation interface of the OMA LWM2M for the information reporting corresponds with the Cancellation (option 0).

Finally, the notification interface is similar.

Therefore, OMA LWM2M Information Reporting can be implemented with the Conditional Observe. In addition, Conditional Observe offers a set of extra features in order to set up more complex logic for the observation (subscription).

The main difference and advantage of using Conditional Observe is that the minimum and maximum period options will be defined as part of the condition, since they are two options offered by conditional observe. Thereby, it is not required the definition of minimum and maximum period by parameters (i.e., using the ?pmin={minimum period} and pmax={maximum period}).

The Figure 12 presents an example to conditional observe the OMA Object 106 (which is an IPSO Sensor), Object instance 0, and the resource ID 4 (which is, for this example, a temperature value). This carries out a registration with a minium period of 30 seconds and a maximum period of 120 seconds.



```
|
| <----- 2.04 Changed,,observe:25,payload:14 (Notify) 14|25s
```

Figure 12: Example of conditional observe following the OMA LWM2M Information Reporting requirement. This example is observing the OMA Object 106, Object Instance ID 0, and Resource ID 4. This is defining a minimum period of 30 seconds and maximum period of 120 seconds.

Appendix B. Alternative approaches

In this appendix, we include some alternative solutions the authors have discussed regarding cancellation of relationships and the logical combination of different relationships. The mechanisms described here allow more flexibility, but introduce additional (undesired?) complexity. We put their description in this appendix to trigger further discussion and provide more background.

B.1. Annex: Cancellation flag

An alternative way to allow the explicit establishment and removal of conditional relationships and to allow the establishment of multiple conditional relationships to the same server using the same source transport address, is through the introduction of a 1 bit cancellation flag (C) as part of the 1 byte condition header. The following paragraphs describe how this flag would change this behaviour.

C: The cancellation flag, when used in an observe request, indicates whether the client wants to establish a conditional observation relationship (0) or cancel an existing conditional relationship (1). In the initial response, the server uses the same value as in the request. In all further notifications, this flag has no meaning and must be 0. In case of a request containing multiple Condition Options, the client must use the same value of the C flag in all Condition Options. The cancellation flag allows a client to establish multiple conditional observation relationships and remove individual relationships from the same address and port.

When using this cancellation flag, a client is able to establish multiple conditional relationships using the same source transport address. Different from [I-D.ietf-core-observe] a GET without observe issued by a client, will not result in the removal of established conditional relationships. Instead the client has the possibility to explicitly terminate any established conditional relationship by sending the same observe request, but with the

Condition Option having the C flag set in order to trigger the cancellation of the request. This way, the same end point can manage multiple conditional observation relationships without the risk of accidentally removing them.

When a server receives a cancellation request, it removes the relationship indicated by the Condition Option and sends back a response containing both the Observe and the Condition Option with the cancellation flag set to 1.

In case a client wants to terminate all existing conditional observation relationships with a server, it should send a request with Condition Option, where the Condition Type is set to the reserved value 0 and the cancellation flag to 1. Upon reception of this message, the server removes all existing relationships and sends back a response containing the same Condition Type.

In case a client has established a conditional relationship that is the result of a request with multiple Condition Options, the client can cancel this relationship by sending the same request, but now with all cancellation flags set to 1.

If the server is for whatever reason not able to further fulfil the conditional relationship of a client, the server can also send a confirmable notification containing the Condition Option with the C flag set to 1 in order to terminate the observation relationship.

B.2. Annex: Logic flag

Different condition types can be combined in a request to express a logical AND relationship. By default, logical OR of condition types is always supported through sending separate requests. In order to express an out-of-range condition (notification when value is lower than X OR higher than Y), two separate conditional observe requests have to be sent. Through the introduction of a 1 bit logic flag (L) as part of the 1 byte condition header, this can be avoided.

L: The logic flag in a Condition Option indicates how the condition should be combined logically with the condition in the next Condition Option. A value of 0 means AND and a value of 1 means OR. The flag has no meaning if the Condition Option is the only or last Condition Option in the request. Through the use of the L flag it is possible to logically combine different conditions in a single request (e.g. C1 AND C2 OR C3 AND C4). As a drawback, when used, it complicates processing at the server side.

This example (Figure 12) shows the same example from the (Figure 11) but with this alternative. This example presents as the client adds

two Observe options to build a range, one set to 6/22, another one set to 5/16. It means that the range is out of range between 16 and 22, i.e. value > 22 OR value < 16. The first option is defined by default with the Logic flag equal to 0. The second option is defined with the Logic flag equal to 1, since it is the OR option. Note as it has been simplified from two messages to only one message with two options, and the most important, it has avoided the management of multiple ports.

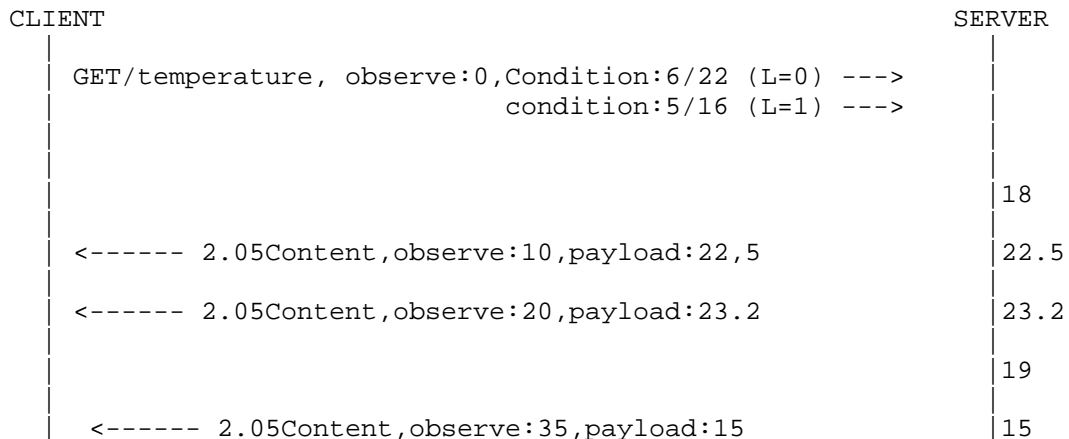


Figure 13: Two Observe options with Logic flag to define out-side a range 6/22 OR 5/15.

Appendix C. Change log

Changes in v04

- o Updated draft to be consistent with updated observe draft:
 - * Took request URI into consideration for Cancellation.
 - * Explicitly stated that the timing conditions allow a client to use a representation that is older than Max-Age without verifying the representation first. This collides with a MUST NOT from the observe draft.
 - * Stated that a server should follow the text from the observe draft for an unacknowledged notification in regards to the transmission of new notifications and the cancellation of existing relationships.

- o Added section 2.1 comparing the RESTful approach from draft-shelby-core-interfaces to our approach for conditional observations.
- o Clarified why "AND" semantics are preferred over "OR" semantics in case of multiple Condition options.
- o Clarified that the R flag doesn't violate the behaviour defined in the Observe draft.
- o Clarified that a client might opt to use representations older than Max-Age without validating these first in case of most of the timing conditions. A server has the ability to deny the client this sort of behaviour however.
- o Updated source endpoint terminology.

Changes in v03

- o Examples for most condition types
- o Update the option number according to the new numbering scheme
- o Added reference to paper validating implementation on constrained device

Changes in v02

- o Restructured entire document
- o Detailed description of the Condition Option and updated format of the Condition Option value
- o Added more Condition Types
- o New section on cancellation, updating and existence of conditional relationships
- o New section on discovery

Authors' Addresses

Shitao Li
Huawei Technologies
Huawei Base
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Phone: +86-25-56624157
Email: lishitao@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

Floris Van den Abeele
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: floris.vandenabeele@intec.ugent.be

Antonio J. Jara
University of Murcia
Department of Information Technology and Communications
Computer Science Faculty
Campus de Espinardo
Murcia ES-30100
Spain

Phone: +34-868-88-8771
Email: jara@um.es

CORE WG
Internet-Draft
Intended status: Standards Track
Expires: January 12, 2014

A. Rahman
InterDigital Communications, LLC
July 11, 2013

Enhanced Sleepy Node Support for CoAP
draft-rahman-core-sleepy-03

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 January 12, 2014.

Copyright Notice

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

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

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

Table of Contents

1. Terminology and Conventions	2
2. Introduction	3
3. Proposal	3
3.1. RD Based Sleep Tracking	3
3.2. Example of Synchronous RD Based Sleep Tracking	4
3.3. Example of Asynchronous RD Based Sleep Tracking	7
3.4. RD Caching Proxy	11
4. Performance Results	11
4.1. Prototype Setup	11
4.2. Initial Results	12
4.3. Comparison with Observe	12
5. Acknowledgements	13
6. IANA Considerations	13
7. Security Considerations	13
8. References	13
8.1. Normative References	13
8.2. Informative References	13
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 will otherwise 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 is 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 forward 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.ietf-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 for other nodes, and not the actual resource representation [RFC6690]. 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.ietf-core-resource-directory].

We also define a new lookup-type ("ss") for the RD lookup interface specified in [I-D.ietf-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.ietf-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.

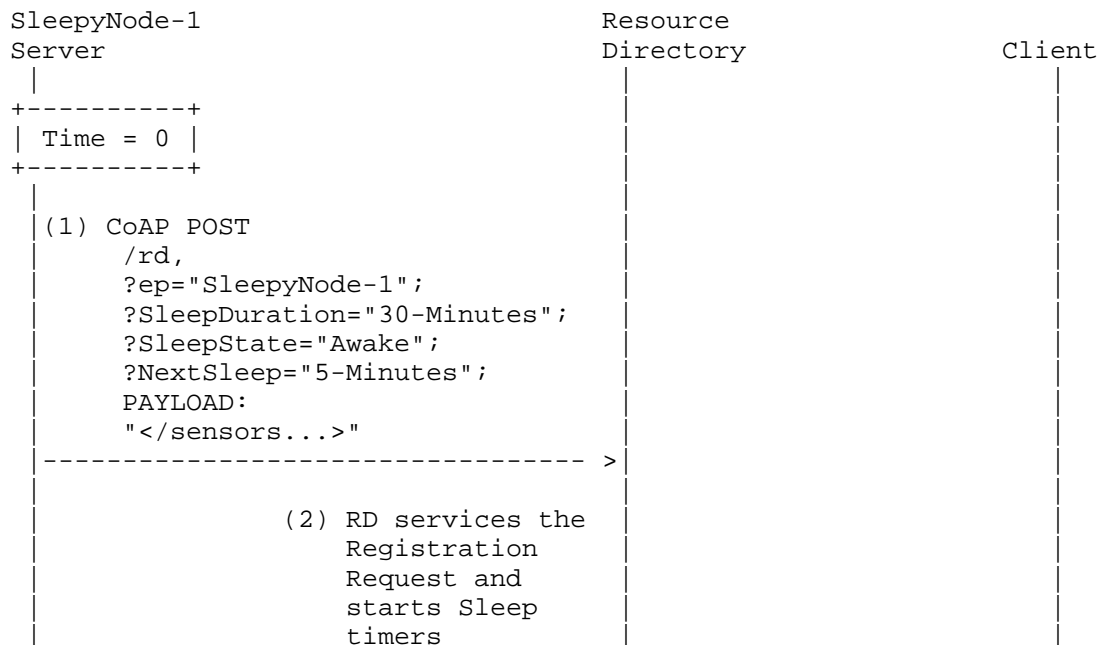
(4) SleepyNode-1 falls asleep.

(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.





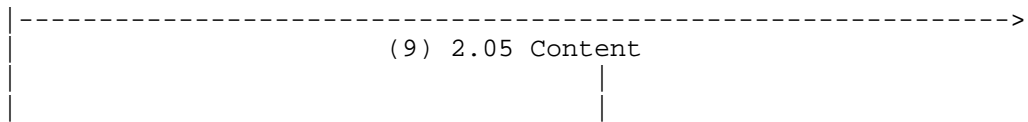


Figure 1: Synchronous Resource Directory Based Sleep Tracking

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.

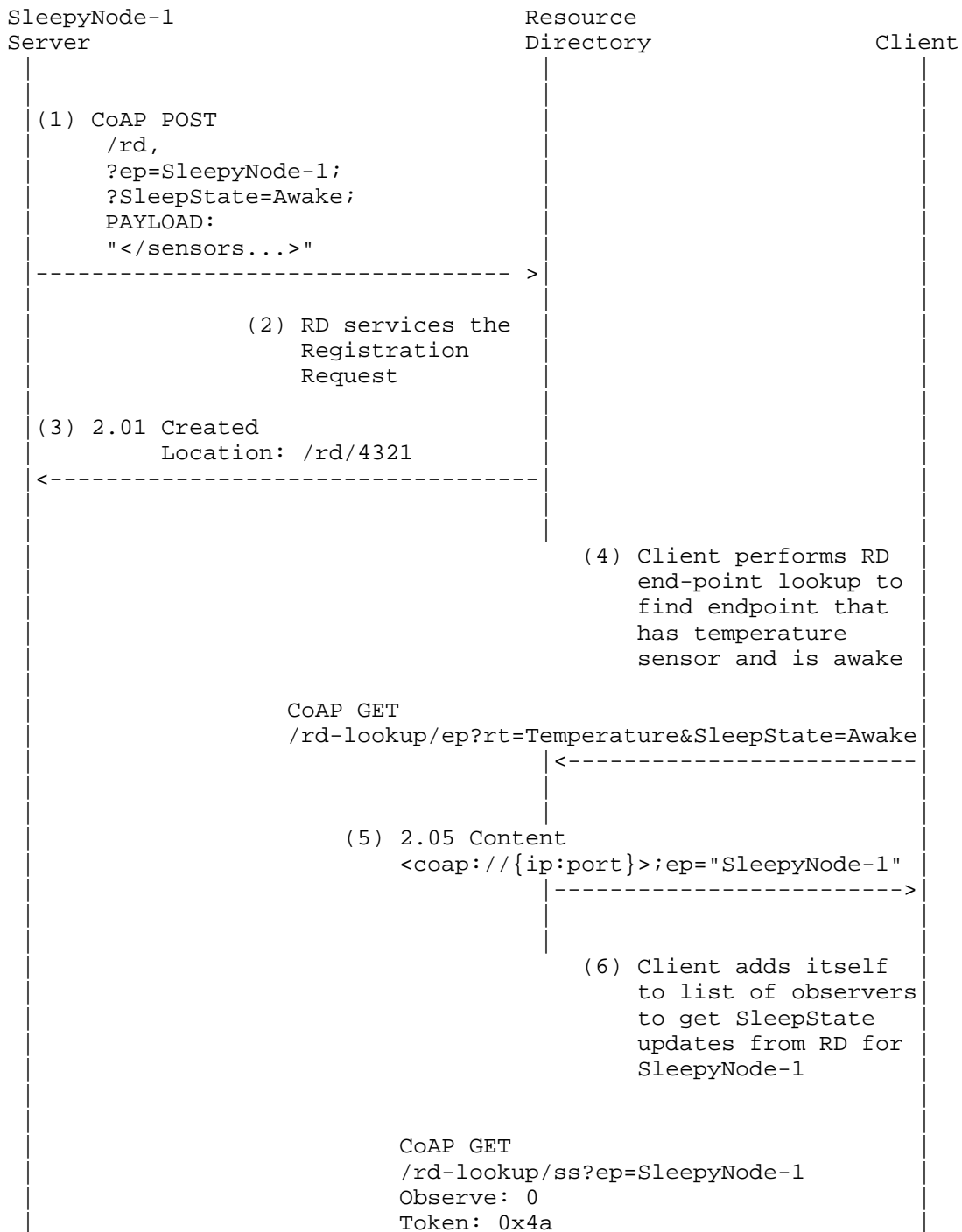
(14) RD notifies the client through previously established observe relationship.

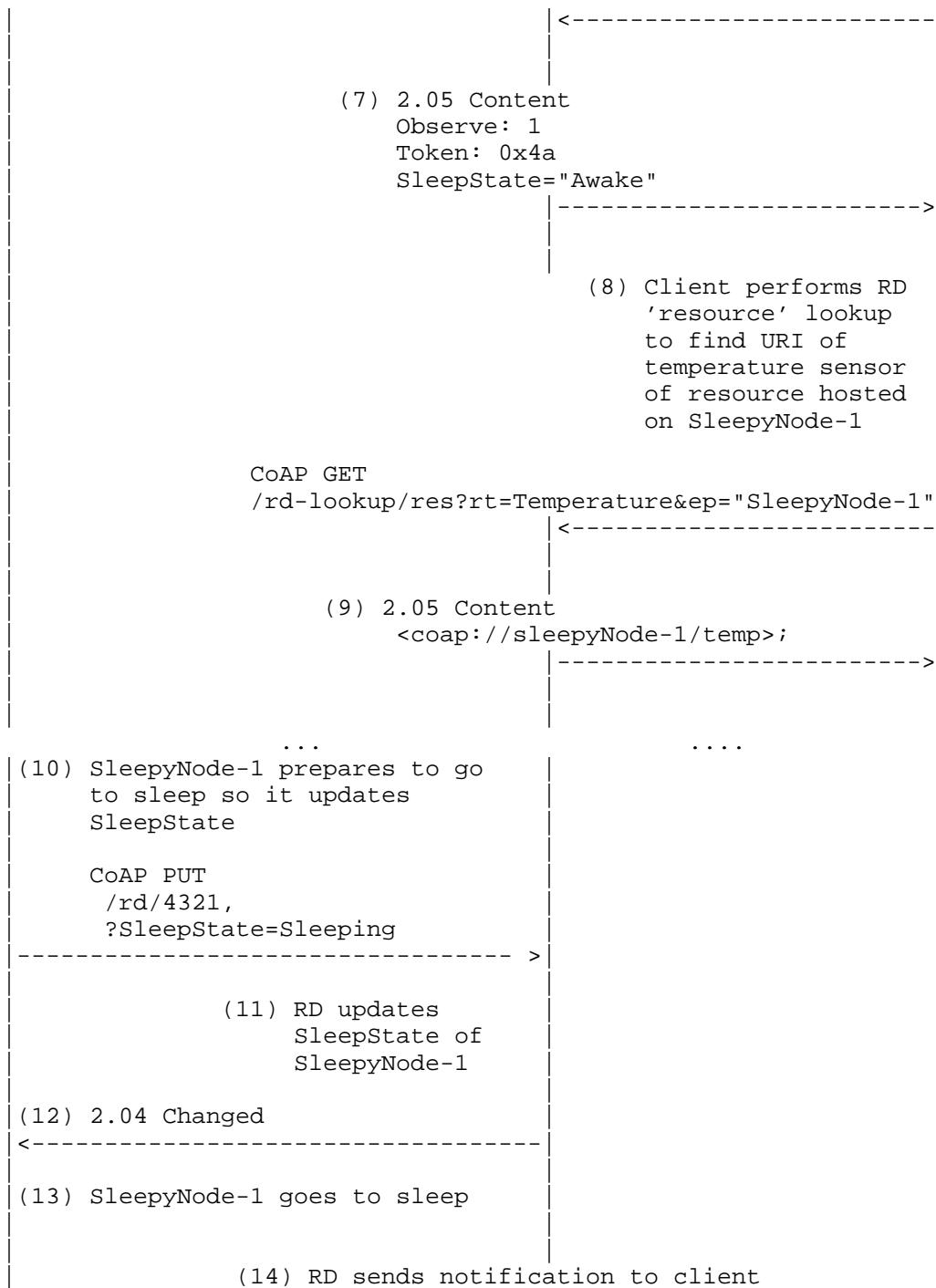
(15) Client application wants to get the temperature now but does not send the request as it knows SleepyNode-1 is currently sleeping.

(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.





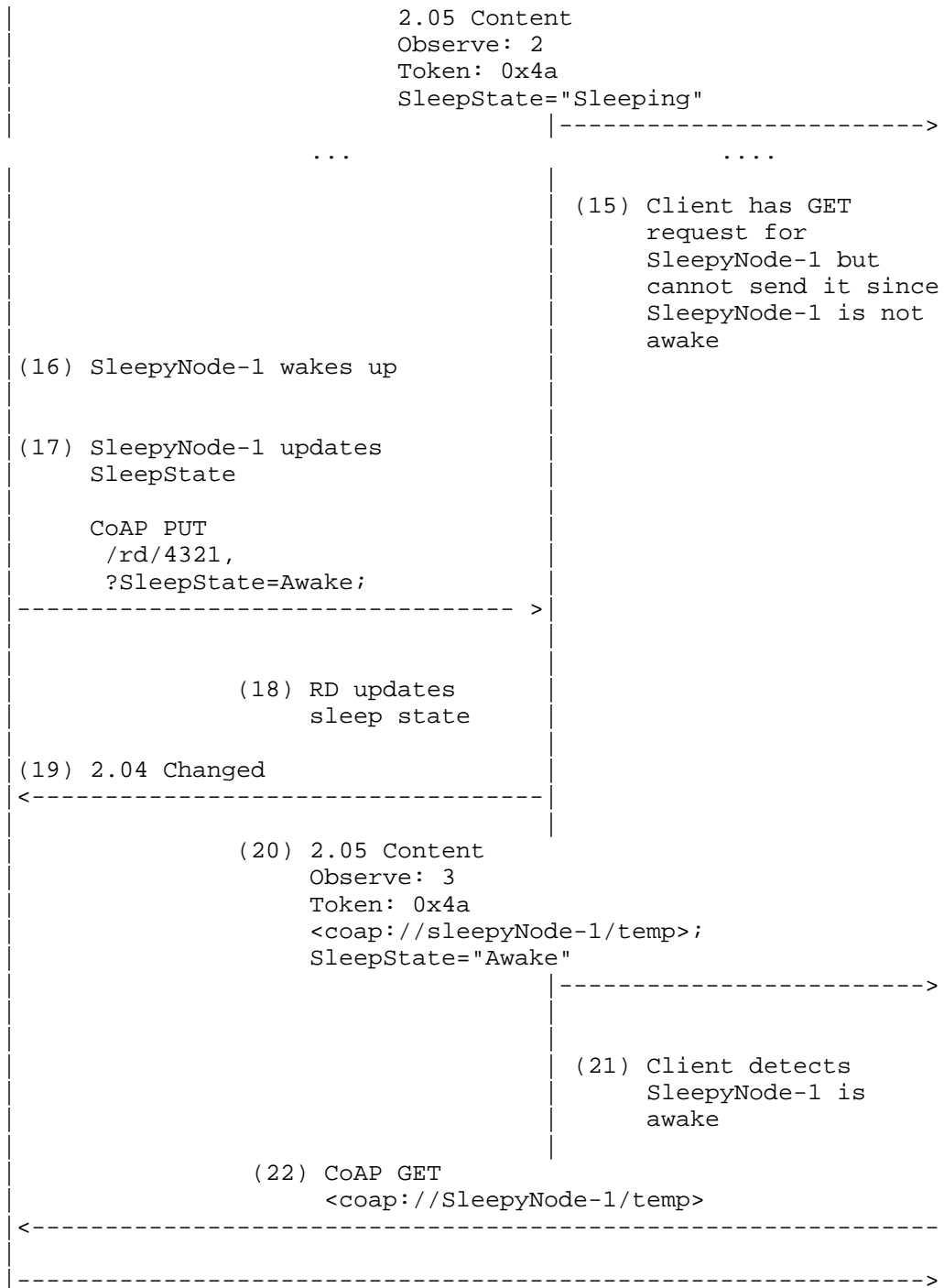


Figure 2: Asynchronous Resource Directory Based Sleep Tracking

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.ietf-core-resource-directory].

4. Performance Results

4.1. Prototype Setup

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.

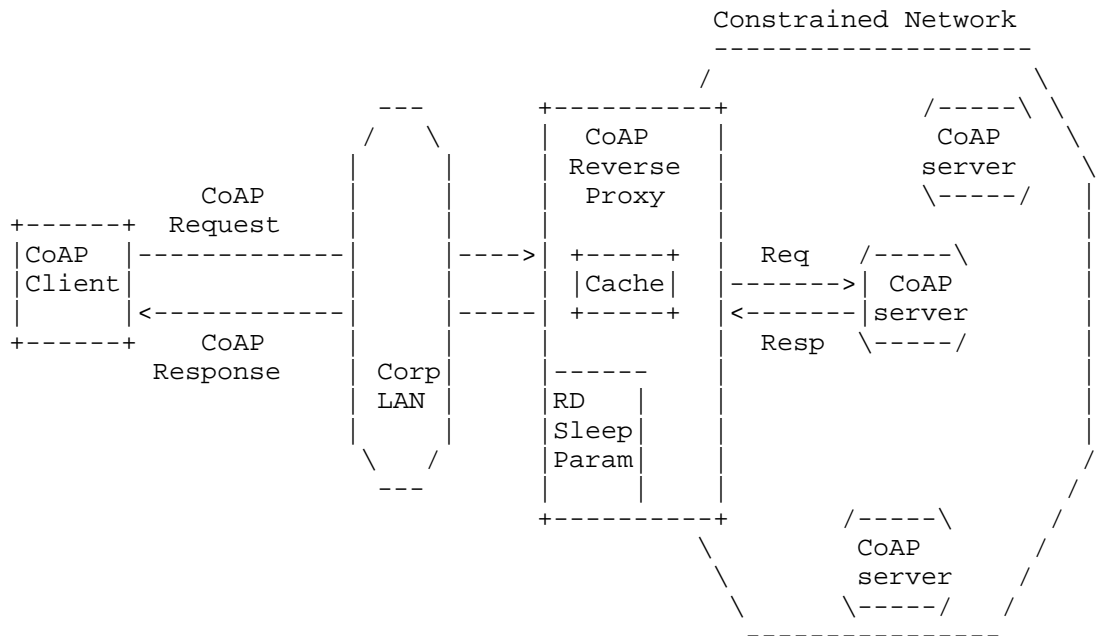


Figure 3: Prototype Configuration

4.2. Initial Results

Baseline performance results of A CoAP system with RD Based Sleepy Node support described in Section 4.1 are given in [IETF86Results].

4.3. Comparison with Observe

A comparison of the performance of a CoAP system with RD based Sleepy Node support vs a CoAP system supporting only Observe will be presented in IETF-87 (Berlin).

5. Acknowledgements

Thanks to Esko Dijk, Thomas Fossati, Salvatore Loreto, Dale Seed, and Zach Shelby for valuable discussions and feedback on this document.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

TBD. (All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.)

8. References

8.1. Normative References

[I-D.ietf-core-coap]

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

[I-D.ietf-core-observe]

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

[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.

8.2. Informative References

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

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

[IETF86Results]

Rahman, A., "IETF-86 Sleepy Node Performance Results", March 2013, <<http://www.ietf.org/proceedings/86/slides/slides-86-core-1.pdf>>.

[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 Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 13, 2014

T. Savolainen
K. Hartke
Nokia
B. Silverajan
Tampere University of Technology
July 12, 2013

CoAP over WebSockets
draft-savolainen-core-coap-websockets-00

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on January 13, 2014.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
1.1. Overview	4
1.2. Terminology	6
2. CoAP over WebSockets	6
2.1. Opening Handshake	6
2.2. Message Format	7
2.3. Message Transmission	8
2.4. Connection Health	8
2.5. Closing the Connection	8
3. CoAP over WebSockets URIs	8
4. Security Considerations	9
5. IANA Considerations	9
5.1. URI Scheme Registrations	9
5.2. WebSocket Subprotocol Registration	11
6. Acknowledgements	12
7. References	12
7.1. Normative References	12
7.2. Informative References	12
Appendix A. Examples	13
Authors' Addresses	16

1. Introduction

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

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

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

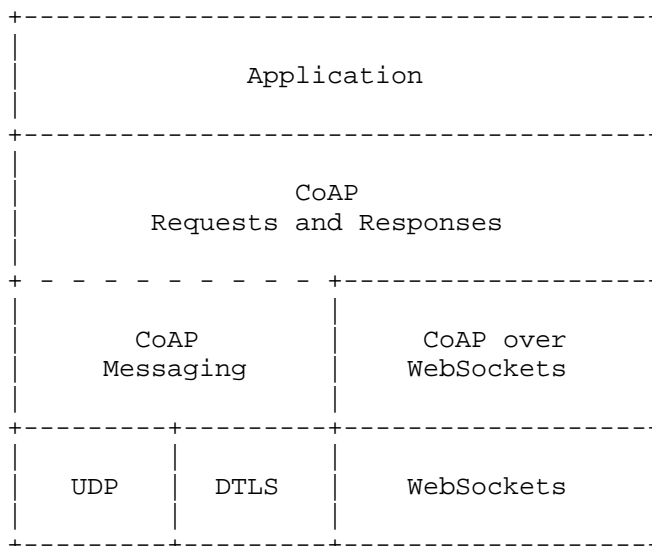


Figure 1: Abstract layering of CoAP extended by WebSockets

1.1. Overview

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

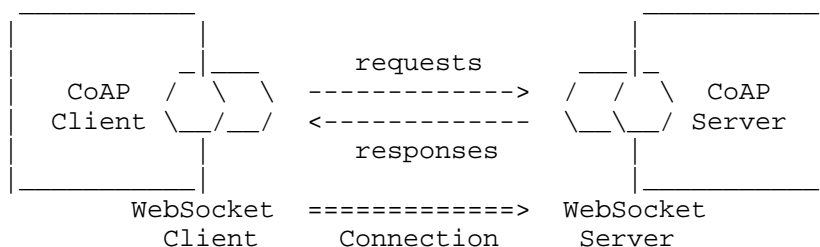


Figure 2: CoAP client (WebSocket client) accesses CoAP server (WebSocket server)

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

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

```

coap+ws://example.org/path/to/endpoint?/sensors/temperature?u=degC
└────────────────────────────────────────────────────────────────────────┘
ws://example.org/path/to/endpoint
└────────────────────────────────────────────────────────────────────────┘
Uri-Path: "sensors"
Uri-Path: "temperature"
Uri-Query: "u=degC"

```

Figure 3: The "coap+ws" URI Scheme

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

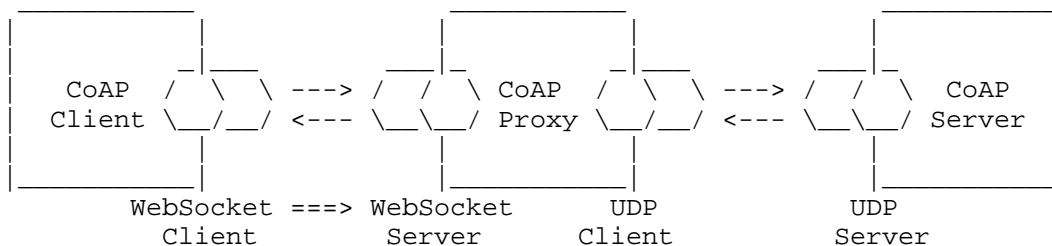


Figure 4: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

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

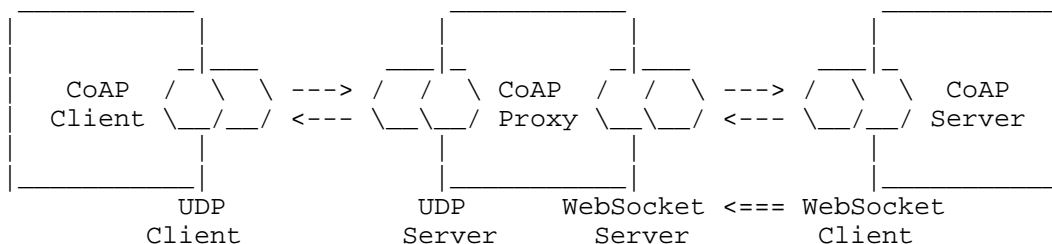


Figure 5: CoAP Client (UDP client) accesses sleepy CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

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

problem can be avoided if the intermediary is turned into a reverse proxy or a mirror server [I-D.vial-core-mirror-server].

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

1.2. Terminology

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455] and [I-D.ietf-core-coap].

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

2. CoAP over WebSockets

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

2.1. Opening Handshake

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

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

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

Figure 6: Example of an Opening Handshake

2.2. Message Format

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

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

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

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

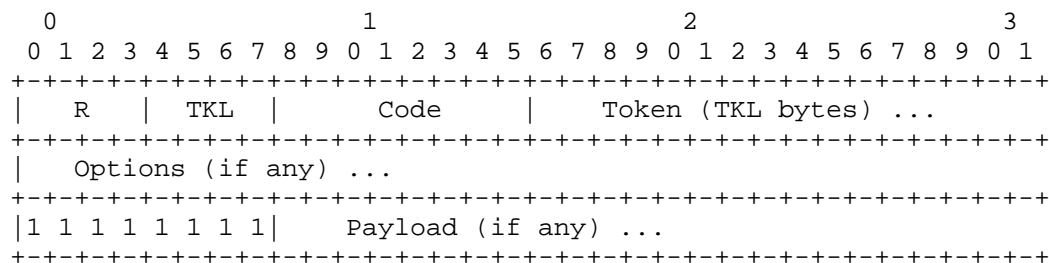


Figure 7: CoAP Message Format over WebSockets

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

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

2.3. Message Transmission

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

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

Retransmission and deduplication of messages is provided by the WebSocket Protocol. CoAP over WebSockets therefore does not make a distinction between Confirmable or Non-Confirmable messages, and does not provide Acknowledgement or Reset messages. Where [I-D.ietf-core-coap] makes a distinction between Confirmable and Non-Confirmable requests or responses, the normative text on Confirmable messages **SHALL** apply. Where [I-D.ietf-core-coap] makes a distinction between piggy-backed and separate responses, the normative text on separate responses **SHALL** apply.

2.4. Connection Health

If a client does not receive any response for some time after sending a CoAP request, the connection between the WebSocket client and the WebSocket server may be lost or temporarily disrupted without the client being aware of it. In this case, the client can send an unsolicited Pong frame to check the health of the WebSocket Connection, as specified in Section 5.5.3 of [RFC6455].

2.5. Closing the Connection

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

If there are requests for which the CoAP client has not received a response yet, the request is cancelled when the connection is closed.

If the CoAP client observes a resource [I-D.ietf-core-observe] over a WebSocket Connection, the CoAP server (or intermediary in the role of the CoAP server) **MUST** remove the client from the list of observers when the connection is closed.

3. CoAP over WebSockets URIs

For the first configuration discussed in Section 1.1, this document

defines two new URI schemes that can be used for identifying CoAP resources and providing a means of locating these resources: "coap+ws" and "coap+wss".

Similar to the "coap" and "coaps" schemes, the "coap+ws" and "coap+wss" schemes organize resources hierarchically under a CoAP origin server. The key difference is that the server is potentially reachable on a WebSocket endpoint instead of a UDP endpoint. The endpoint is identified by an embedded "ws" or "wss" URI respectively. The remainder of the URI identifies a resource which can be operated on by the methods defined by the CoAP protocol.

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

```
coap-ws-URI = "coap+" ws-URI-nq [ "?" path-abempty [ "?" query ] ]
coap-wss-URI = "coap+" wss-URI-nq [ "?" path-abempty [ "?" query ] ]

ws-URI-nq = "ws:" "://" host [ ":" port ] path-abempty
wss-URI-nq = "wss:" "://" host [ ":" port ] path-abempty
```

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

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

4. Security Considerations

CoAP over WebSockets and CoAP over TLS-secured WebSockets do not introduce additional security issues beyond CoAP and DTLS-secured CoAP respectively [I-D.ietf-core-coap]. The security considerations of [RFC6455] apply.

5. IANA Considerations

5.1. URI Scheme Registrations

5.1.1. "coap+ws"

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

URI scheme name.
coap+ws

Status.
Permanent.

URI scheme syntax.
Defined in Section 3.

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

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol.

Interoperability considerations.
None.

Security considerations.
See Section 4.

Contact.
IETF Chair <chair@ietf.org>

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

References.
This document.

5.1.2. "coap+wss"

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

URI scheme name.
coap+wss

Status.
Permanent.

URI scheme syntax.
Defined in Section 3.

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

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol secured with TLS.

Interoperability considerations.
None.

Security considerations.
See Section 4.

Contact.
IETF Chair <chair@ietf.org>

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

References.
This document.

5.2. WebSocket Subprotocol Registration

This document requests the registration of the subprotocol name "coap.v1" in the WebSocket Subprotocol Name Registry.

Subprotocol Identifier.
coap.v1

Subprotocol Common Name.
Constrained Application Protocol (CoAP).

Subprotocol Definition.
This document.

6. Acknowledgements

Thanks to Nadir Javed for helpful comments and discussions that have shaped the document.

7. References

7.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

7.2. Informative References

- [I-D.becker-core-coap-sms-gprs]
Becker, M., Li, K., 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.dijk-core-sleepy-reqs]
Dijk, E., "Sleepy Devices using CoAP - Requirements", draft-dijk-core-sleepy-reqs-00 (work in progress), June 2013.
- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-12 (work in progress), June 2013.

- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP",
draft-ietf-core-observe-08 (work in progress),
February 2013.
- [I-D.ietf-hybi-websocket-multiplexing]
Tamplin, J. and T. Yoshino, "A Multiplexing Extension for
WebSockets", draft-ietf-hybi-websocket-multiplexing-11
(work in progress), July 2013.
- [I-D.vial-core-mirror-server]
Vial, M., "CoRE Mirror Server",
draft-vial-core-mirror-server-01 (work in progress),
April 2013.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax
Specifications: ABNF", STD 68, RFC 5234, January 2008.

Appendix A. Examples

This section gives examples for the first two configurations discussed in Section 1.1.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap+ws" URI might be as follows. Figure 8 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI <coap+ws://example.org/path/to/endpoint?/sensors/temperature?u=degC>, for example, from a resource representation that it retrieved previously.
2. It establishes a WebSocket Connection to the endpoint identified by the embedded "ws" URI, <ws://example.org/path/to/endpoint>.
3. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=degC") options.
4. It waits for server to return a response.
5. The CoAP client uses the connection for further requests, or the connection is closed.

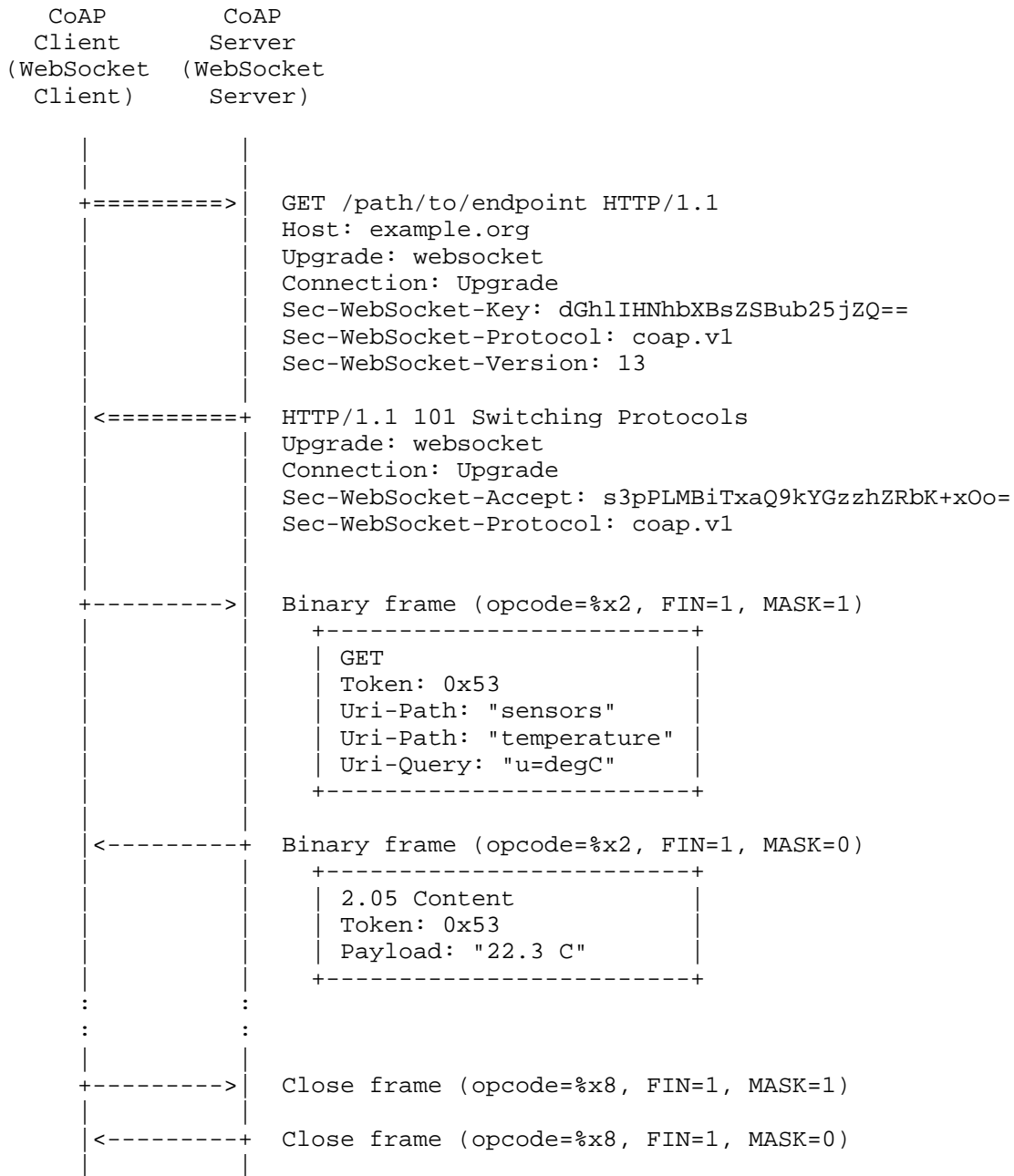


Figure 8: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 9 shows how a CoAP client uses a CoAP forward proxy with a WebSocket endpoint to retrieve the representation of the resource `<coap://[2001:DB8::1]/>`. The use of the forward proxy and the address of the WebSocket endpoint are determined by the client from local configuration rules. The request URI is specified in the Proxy-Uri Option. Since the request URI uses the "coap" URI scheme, the proxy fulfills the request by issuing a Confirmable GET request over UDP to the CoAP server and returning the response to the client.

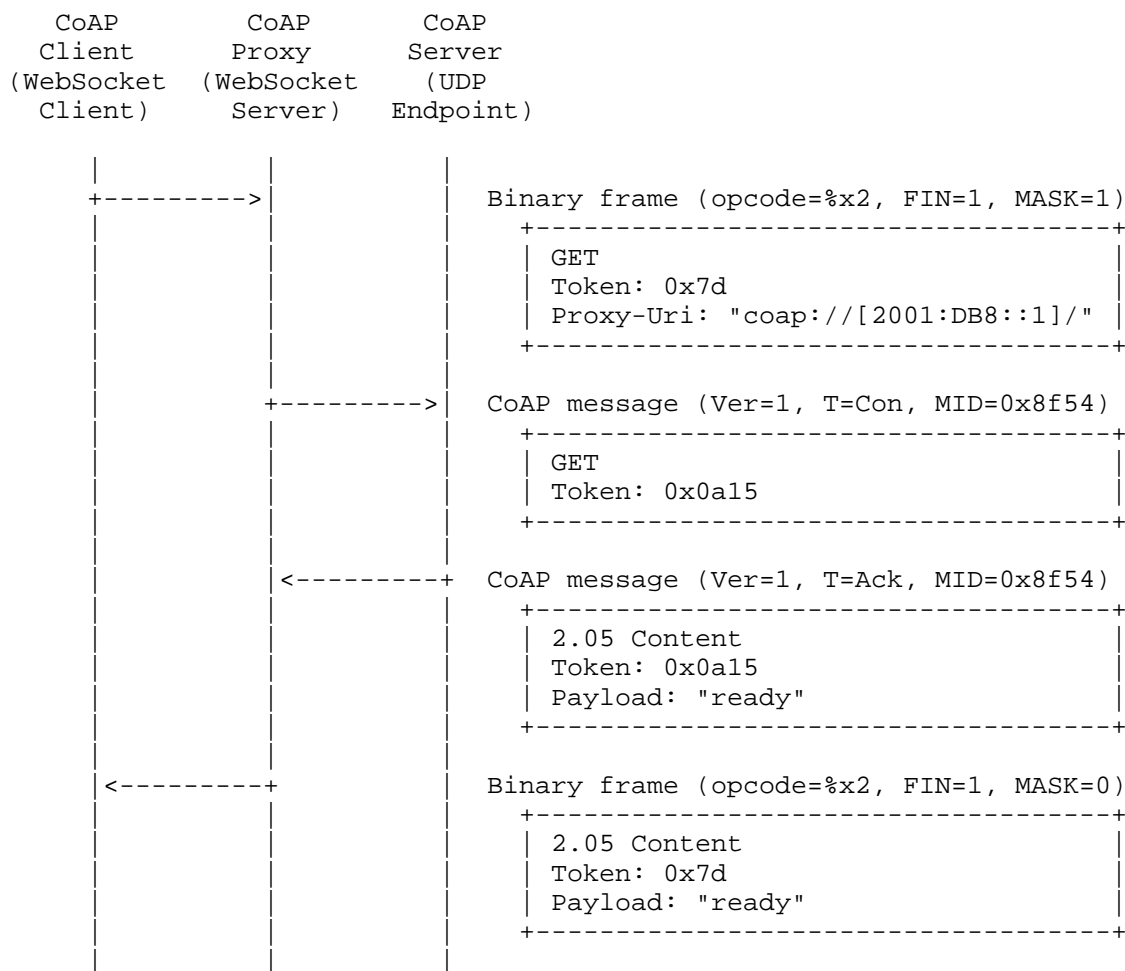


Figure 9: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSockets-enabled CoAP proxy

Authors' Addresses

Teemu Savolainen
Nokia
Hermiankatu 12 D
Tampere FI-33720
Finland

Email: teemu.savolainen@nokia.com

Klaus Hartke
Nokia
Hermiankatu 12 D
Tampere FI-33720
Finland

Email: klaus.hartke@nokia.com

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2014

C. Schmitt
B. Stiller
University of Zurich
T. Kothmayr
TU Muenchen
W. Hu
CSIRO ICT Centre
July 4, 2013

DTLS-based Security with two-way Authentication for IoT
<draft-schmitt-two-way-authentication-for-iot-00>

Abstract

In this draft the first key idea for a full two-way authentication security scheme for the Internet of Things (IoT) based on existing Internet standards, specifically the Datagram Transport Layer Security (DTLS) protocol, is introduced. By relying on an established standard, existing implementations, engineering techniques, and security infrastructure can be reused, which enables an easy security uptake. The proposed security scheme is, therefore, based on RSA, the most widely used public key cryptography algorithm. It is designed to work over standard communication stacks that offer UDP/IPv6 networking for Low power Wireless Personal Area Networks (6LoWPANs).

Status of this Memo

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

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

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

This Internet-Draft will expire on January 5, 2014.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
1.1. Document structure	3
2. Terminology	4
3. High Level Design Requirements	4
3.1. Implementation of A Standards Based Design	4
3.2. Focus on Application Layer and End-to-End Security	4
3.3. Support for unreliable transport protocols	5
4. DTLS Protocol for Wireless Sensor Networks	5
4.1. DTLS Standard - RFC 6347	5
4.2. A Standard Based End-to-End Security Architecture	6
5. Hardware Requirements	7
6. Security Considerations	8
7. Conclusion	8
8. Formal Syntax	9
9. References	10
9.1. Norminative References	10
9.2. Informative References	10
Authors' Addresses	11

1. Introduction

Today, there is a multitude of envisioned and implemented use cases for the Internet of Things (IoT) and wireless sensor networks (WSNs). In many of these scenarios it is intended to make the collected data globally accessible to authorized users and data processing units through the Internet. Most of these data collected in such scenarios is of sensitive nature due to the relation to location and personal information or IDs. Even seemingly inconspicuous data, such as the energy consumption measured by a smart meter, can lead to potential infringements in the users' privacy, e.g., by allowing an eavesdropper to conclude whether or not a user is currently at home. From an industry perspective, there is also a pressing need for security solutions based on standards as pointed out by the market research firm Gartner Inc. [1]. Regarding the infrastructure, security risks are aggravated by the trend toward a separation of sensor network infrastructure and applications. Therefore, a true end-to-end security solution is required to achieve an adequate level of security for IoT. Protecting the data once it leaves the scope of the local network is not sufficient.

A similar scenario in the traditional computing world would be a user browsing the Internet over an unsecured WLAN. Assuming attackers in physical proximity of the user it can happen that the attacker can capture the traffic between the user and a Web server. Countermeasures against such attacks include the establishment of a secured connection to the Web server via HTTPS, the use of a VPN tunnel to securely connect to a trusted VPN endpoint, and using wireless network security such as WPA.

These solutions are comparable to security approaches in the IoT area. Using WPA is similar to the traditional use of link layer encryption. The VPN solution is equivalent to creating a secure connection between a sensor node and a security end-point, which may or may not be the final destination of the sensor data. Establishing a HTTPS connection with the server is comparable to the approach described in this draft: The use of the DTLS protocol in an end-to-end security architecture for IoT is investigated, where DTLS is an adaption of the widespread TLS protocol, used to secure HTTPS, for unreliable datagram transport.

1.1. Document structure

Section 2 mentions conventions used in this draft. Afterwards the assumed high level design requirements are briefly mentioned in Section 3. Section 4 describes idea of bringing DTLS to Wireless sensor networks. Section 5 defines the hardware requirements, followed by security considerations. The draft is concluded in

Section 7.

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].

3. High Level Design Requirements

Due to the usage of DTLS for establishing an end-to-end security architecture for IoT three high-level design decisions MUST be made.

3.1. Implementation of A Standards Based Design

Standardization has helped the widespread uptake of technologies. Radio chips can rely on IEEE 802.15.4 for the physical and the MAC layer. Routing functionality is provided by the so-called 'IPv6 Routing Protocol for Low power and Lossy Networks' (RPL) [RFC6550] or 6LoWPAN [RFC4944]. COAP [2] defines the application layer. So far, no such efforts have addressed security in a wider context of IoT.

3.2. Focus on Application Layer and End-to-End Security

An end-to-end protocol provides security even if the underlying network infrastructure is only partially under the user's control. As the infrastructure for Machine-to-Machine (M2M) communication is getting increasingly commoditized, this scenario becomes more likely: The European Telecommunications Standard Institute (ETSI) plans to standardize the transport of local device data to a remote data center. For stationary installations security functionality could be provided by the gateway to the higher-level network. However, such gateways MAY present a high-value target for an attacker. If the devices are mobile, as it is possible within a logistic application, there may be no gateway to a provider's network that is under the user's control, similar to how users of smart phones connect directly to their carrier's network. Another example that favours end-to-end security is a multi-tenancy office building being equipped with a common infrastructure for metering and climate-control purposes. Tenants share the infrastructure but are still able to keep their devices' data private from other members of the network.

DTLS is located between the transport and the application layer. Thus, it is not necessary that providers of the infrastructure support security mechanisms. It is purely in the hands of the two communicating applications to establish security. If the security is

provided by a network layer protocol (e.g., IPsec) the same is true to a lower degree, because network stacks of both devices MUST support the same security protocols.

3.3. Support for unreliable transport protocols

Reliable transport protocols like TCP incur an overhead over simpler, unreliable protocols such as UDP. Especially for energy starved, battery powered devices this overhead is often too costly and TCP has been shown to perform poorly in low-bandwidth scenarios [3]. This is reflected in the design of the emerging standard COAP, which uses UDP transport and defines a binding to DTLS for security [2]. By using DTLS in conjunction with UDP this draft does not force the application developer to use reliable transport - as it would be the case if TLS would be used. It is still possible to use DTLS over transport protocols like TCP, since DTLS only assumes unreliable transport.

This is a weaker property than the reliability provided by TCP. However, adaptations of DTLS for unreliable transport introduce additional overhead when compared to TLS. There MAY be a benefit in using TCP during the handshake phase but the DTLS reliability mechanism SHOULD be adapted to the special requirements of constraint networks.

4. DTLS Protocol for Wireless Sensor Networks

4.1. DTLS Standard - RFC 6347

The Datagram Transport Layer Security (DTLS) protocol in version 1.2 was standardized under the RFC 6347 [RFC6347]. All messages sent via DTLS are prepended with a 13 Byte long DTLS record header. This header specifies the content of the message (e.g. application data or handshake data), the version of the protocol employed, as well as the 64 bit sequence number and the record length. The top two bytes of the sequence number are used to specify the epoch of the message, which changes once new encryption parameters have been negotiated between client and server.

If no security has been negotiated yet, the DTLS record header is followed by the plaintext, otherwise by the DTLS block cipher. If a block cipher is used, the plaintext is prepended by a random Initialization Vector (IV), which has the size of the cipher block length. This approach protects against attacks where attackers can adaptively choose plaintext. The plaintext is followed by a Hash-based Message Authentication Code (HMAC), which allows the receiver

to detect if the DTLS record has been altered. Finally, the message is padded to a multiple of the cipher block length. Unlike TLS, DTLS does not allow for stream ciphers, because they are sensitive to message loss and recording. Instead DTLS uses block ciphers in the Cipher-Block Chaining (CBC) mode of operation.

The key material and cipher suite, consisting of a block cipher and a hash algorithm, are negotiated between the client and the server during the handshake phase, which commences before any application data can be transferred. Three types of handshake exist: unauthenticated, server authenticated, and fully authenticated handshakes. During an unauthenticated handshake neither party authenticated with the other. In contrast, in a server-authenticated handshake only the server proves its identity to the client. In a fully authenticated handshake the client has to authenticate itself to the server as well.

4.2. A Standard Based End-to-End Security Architecture

The proposed system architecture in this draft is following the IoT model. It is assumed that IPv6 connects the Internet and parts of it run 6LoWPAN. The transport layer in 6LoWPAN is UDP, which can be considered unreliable; the routing layer is RPL or Hydro [[3]]. Both routing protocols are similar enough and, therefore, a change SHOULD have negligible impact on the results. IEEE 802.15.4 is used for the physical and MAC layer. Based on this protocol stack DTLS was selected as the security protocol and placed in the application layer on top of the UDP transportation layer. Figure 1 shows the network stack used in this draft [6], while BLIP is a special 6LoWPAN implementation including several IP protocols [7].

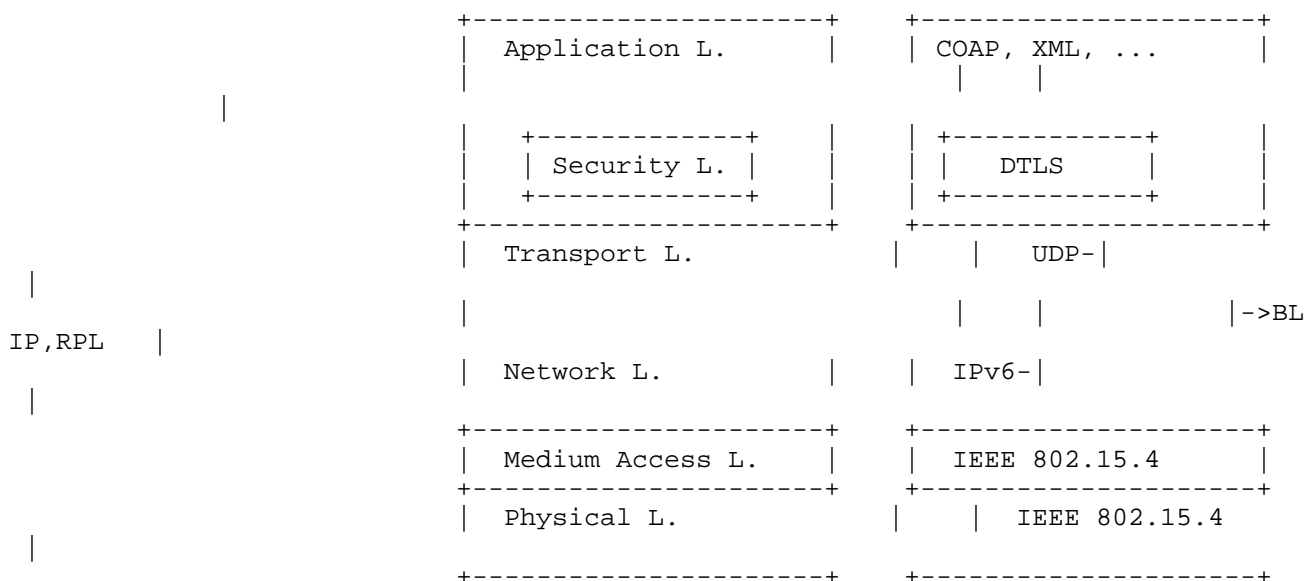


Figure 1: Assumed Network Stack

In order to support end-to-end communication security the need for proper authentication of data publishing devices and access control throughout the network is required. Thus, an Access Control (AC) server is integrated in the assumed system architecture. The AC is a trusted entity and a resource-rich server, on which access rights for the publisher (= sensor nodes) of the secured network are stored. The identity of a default subscriber is usually preconfigured on a publisher before it is deployed. If any additional subscribers want to initialize a connection with the publisher, they first have to obtain an access ticket from the AC. The AC verifies that the subscriber has the right to access the information available from the publisher. In the next step the publisher only has to evaluate the identity of the subscriber and has to verify the ticket it has received from the AC. This requires a unique identity for a publisher in the network. In the Internet, identities are usually established via public key cryptography (PKC) and identifiers are provided through X.509 certificates. An X.509 certificate contains, among other information, the public key of an entity and its common name. A trusted third party, called the Certificate Authority (CA), signs the certificate. The CA serves two purposes: Firstly, the signature allows the receiver to detect modifications to the certificate. Secondly, it also states that the CA has verified the identity of the entity that requested the certificate.

5. Hardware Requirements

Hu et al. showed that RSA, the most commonly used public key algorithm in the Internet, can be used in sensor networks with the assistance of a Trusted Platform Module (TPM), which costs less than 5% of a common sensor node [4]. A TPM is an embedded chip that provides tamper proof generation and storage of RSA keys as well as hardware support for the RSA algorithm. The certificate of a TPM equipped publisher and the certificate of a trusted CA MUST be stored on the publisher prior to deployment.

For publishers that are not equipped with TPM chips the authentication can be proposed via the DTLS pre-shared key cipher-suite, which requires a small number of random bytes, from which the actual key is derived, to be preloaded to the publisher before deployment. This secret MUST also be made available to the AC server, which will disclose the key to devices with sufficient authorization.

6. Security Considerations

The following security goals are addressed by the key idea presented in this draft:

Authenticity

Recipients of a message can identify their communication partners and can detect if the sender information has been forged.

Integrity

Communication partners can detect changes to a message during transmission.

Confidentiality

Attackers cannot gain knowledge about the content of a secured message.

By choosing DTLS as the security protocol those goals can be achieved. DTLS is a modification of TLS for the unreliable UDP and inherits its security properties [5].

7. Conclusion

In this draft the key idea of a standard-based security architecture with two-way authentication for the IoT was introduced. During a fully authenticated DTLS handshake authentication can be performed, while the handshake is based on an exchange of X.509 certificates containing RSA keys. The proposed architecture provides message integrity, confidentiality, and authenticity with affordable energy, end-to-end latency, and memory overhead [6]. Thus, it can be assumed that DTLS is a feasible security solution for the emerging IoT. A fully authenticated handshake with strong security through 2048 bit RSA keys is considered as feasible for sensor nodes equipped with a TPM chip, since a fully authenticated, RSA-based handshake consumes as little as 488 mJ [6]. These additional memory requirements of fewer than 20 kB RAM are well below the 48 kB of memory offered by the sensor node used [6].

Sensor nodes without a TPM chip forego protection against physical tampering, but can still perform a DTLS handshake based on Elliptic Curve Cryptography (ECC), which could be performed on the same platform with little more than 100 mJ of energy usage [6].

For the future it MAY be possible to apply these techniques to DTLS together with an Authenticated Encryption with Associated Data (AEAD) mode of operation. Another focus MAY be the inclusion of more constrained nodes without a TPM in the proposed architecture, for which a variant of the DTLS pre-shared key cipher suites SHALL be used.

8. Formal Syntax

6LoWPAN - IPv6 over Low power Wireless Personal Area Network (RFC 4944)

AC - Access Control Server

BLIP - Berkeley Low-power IP stack

CA - Certificate Authority

CBC - Cipher-Block Chaining

COAP - Constrained Application Protocol

DTLS - Datagram Transport Layer Security protocol (RFC 6347)

ECC - Elliptic Curve Cryptography

ETSI - European Telecommunications Standard Institute

HMAC - Hash-based Message Authentication Code

IoT - Internet of Things

IV - Initialization Vector

PKC - Public Key Cryptography

RPL - Routing Protocol for Low power and Lossy Networks (RFC 6550)

TCP - Transmission Control Protocol (RFC 793)

TLS - The Transport Layer Security (TLS) Protocol Version 1.2 (RFC 5246)

TPM - Trusted Platform Module

UDP - User Datagram Protocol (RFC 768)

WSN - Wireless Sensor Network

9. References

9.1. Norminative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [2] Shelby et al., Z., "Constrained Application Protocol (CoAP)", <http://tools.ietf.org/html/draft-ietf-core-coap-14>", March 2013.
- [3] Dawson-Haggerty et al., S., "Hydro: a hybrid routing protocol for low-power and lossy networks", In Proceedings of the 1st IEEE International Conference on Smart Grid Communications, SmartGridComm, Gaithersburg, Maryland, U.S.A.. , 2010.
- [5] Modadugu et al., N., "The Design and Implementation of Datagram TLS", In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, California, U.S.A.. , 2004.

9.2. Informative References

- [1] LeHong, H., "Hype Cycle for the Internet of Things", Tech. rep., Gartner Inc. , 2012.
- [4] Hu, W., "Toward trusted wireless sensor networks", ACM Transactions on Sensor Networks, Vol. 7, No.5. , 2010.
- [6] Kothmayr et al., T., "DTLS based security and two-way authentication for the Internet of Things", Elsevier,

Journal Ad Hoc Networks , 2013.

- [7] Dawson-Haggerty, S. and D. Culler, "Berkeley IP Information, Berkeley WEBS Wireless Embedded Systems, <http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip>", 2010.

Authors' Addresses

Corinna Schmitt
Univerity of Zurich
Department for Informatics
Communication Systems Group
Binzmuehlestrasse 14
Zurich 8050
Switzerland

Email: schmitt@ifi.uzh.ch

Burkhard Stiller
Univerity of Zurich
Department for Informatics
Communication Systems Group
Binzmuehlestrasse 14
Zurich 8050
Switzerland

Email: stiller@ifi.uzh.ch

Thomas Kothmayr
Technische Universitaet Muenchen
Department of Informatics
Datenbanksysteme (Informatik 3)
Boltzmannstr. 3
Garching 85748
Germany

Email: kothmayr@in.tum.de

Wen Hu
CSIRO ICT Centre
1 Technology Court
Pullenvale QLD 4069
Australia

Email: Wen.Hu@csiro.au

CoRE Working Group
Internet-Draft
Intended Status: Standards Track
Expires: January 6, 2014

G. Selander
M. Sethi
Ericsson
L. Seitz
SICS Swedish ICT
July 5, 2013

Access Control Framework for Constrained Environments
draft-selander-core-access-control-00

Abstract

The Constrained Application Protocol (CoAP) is a light-weight web transfer protocol designed to be used in constrained nodes and constrained networks. Communication security support for CoAP, including authentication, encryption, integrity protection, is well understood and a DTLS binding for CoAP is specified, but authorization and access control are not described in detail.

This document describes a generic and dynamic access control framework suitable for constrained environments using CoAP. The framework builds on standards and well known paradigms for access control, externalizing authorization decision making to unconstrained nodes while performing authorization decision enforcement and verification of local conditions in constrained devices.

In addition, this document provides alternative or complementary key management to the CoAP security modes.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at

<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

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

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

Table of Contents

1. Introduction	4
1.1 Terminology	5
2. Requirements, rationale and outline	5
2.1 Requirements	5
2.2 Rationale	6
2.3 Outline of Access Control Framework	8
3. Roles	10
3.1 Resource Owner	10
3.2 Authorization Server	10
3.3 Client	11
3.4 Resource Server	11
4. Assertion profiles	12
4.1 Assertion requirements	12
4.2 Assertion wrapping	13
4.3 Assertion transport	13
5. Message protection profiles	14
5.1 Communication security profile	14
5.1.1 Assertion caching in resource server (informational)	15
5.2 Object security profile	16
5.2.1 GET	17
5.2.2 PUT/POST	17
5.2.3 DELETE	18
5.2.4 Assertion replay protection (informational)	18

6. Intermediary processing and notifications	19
6.1 Intermediary nodes	19
6.2 Mirror Server	20
6.3 Observe	20
7. Trust management	20
7.1 Access Control Lists	21
7.2 Key provisioning schemes	23
7.2.1 ACF PK Scheme	23
7.2.2 ACF SSK Scheme	24
7.3 Key provisioning examples	25
7.3.1 The authorization server as certification authority . .	26
7.3.2 The authorization server for configuration of raw public keys	26
7.3.3 The assertion as an authentication token	26
7.3.4 Using the assertion to transport the nonce	27
7.3.5 TLS extension for transport of nonce	27
8. ACF Profiles	27
8.1 Assertion Profile: Compact SAML-XACML	27
8.2 Communication Security Profile: DTLS PSK with Bind Key . .	31
8.3 Object Security Profile: JWE-protected CoAP payload	32
9. Security Considerations	33
10. IANA Considerations	34
11. References	35
11.1 Normative References	35
11.2 Informative References	35
Authors' Addresses	37

1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a light-weight web transfer protocol suitable for applications in embedded devices used in services such as smart energy, smart home, building automation, remote patient monitoring etc. Due to the nature of these use cases including critical, unattended infrastructure and the personal sphere, security and privacy are critical components.

CoAP message exchanges can be protected with different security protocols. The CoAP specification defines a DTLS binding for CoAP, which provides communication security services including authentication, encryption, integrity protection and replay protection.

Authorization and access control - i.e. controlling who has access to what - is addressed with access control lists, which are assumed to have been provisioned to the devices and which contain e.g. lists of identifiers that may start DTLS sessions with the devices.

There are some limitations with this approach:

1. By restricting the scope of access control to the granularity of identifiers with "root" access, it is not possible to give different privileges to different entities that are allowed to access the same device. For example, it may be desirable to give some clients the right to GET resources but others the right to POST or PUT resources to the same device; or to give the same client different access rights for different resources on the same device.

2. Furthermore, in certain use cases the granularity of GET/PUT/POST/DELETE is not sufficient to specify the relevant access restrictions. For example, the access policy may depend on local conditions of the device such as: date and time, proximity, geo-location, detected effort (press 3 times) or other aspects of the current state of the device.

3. Another limitation is that it is neither defined how to change the access privileges except by means of re-provisioning, nor how such changes would be authorized.

The use of more flexible access control also enables more advanced business settings, for example: outsourcing of operations of sensor networks, data harvesting in leased networks, etc.

All these aspects could in principle be addressed with customized

access control lists, but if access control should be standardized for CoAP, it seems more appropriate to use of existing access control standards instead of re-inventing the wheel for a particular protocol or use case.

This document proposes a framework that allows fine-grained and flexible standard-compatible access control applicable to a generic setting including use cases with constrained devices such e.g. class 1 devices [I-D.ietf-lwig-terminology].

1.1 Terminology

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

Certain security-related terms are to be understood in the sense defined in [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "access control", "confidentiality", "credential", "encryption", "sign", "signature", "data integrity", and "verify".

Terminology for constrained environments is defined in [I-D.ietf-lwig-terminology]. These terms include, but are not limited to, "constrained device", "constrained network", and "device class".

2. Requirements, rationale and outline

This section gives an overview of the requirements that form the basis for the proposed Access Control Framework, as well as the rationale behind it, and furthermore it gives a rough outline of how the framework is applied.

2.1 Requirements

The Access Control Framework SHALL support

- access control in a constrained environment with constrained devices or networks, in particular,

- no additional messages SHOULD be sent or received by a constrained device exclusively for performing access control.

- access control applicable to a variety of use cases and access purposes, in particular
 - differentiated access rights for different requesting entities,
 - access control at least at the granularity of RESTful resources,
 - access policies based on local conditions (e.g. state of device, time, position),
- changes to access policies without re-provisioning, and
- state-of-the-art security (in particular access control) standards and best practices (in particular end-to-end security between resource and authorized client).

The Access Control Framework SHALL be compatible with a large variety of client-server authentication methods, message protection mechanisms (communication/object security), device key management procedures and trust anchors (secret keys, raw public keys, certificates).

2.2 Rationale

Consider the main setting in focus where a CoAP client wants to access a resource hosted on a CoAP server, which is potentially a constrained device, and where the access rights should be determined by the owner of the resource.

To comply with requirements on generic applicability, granular, and standardized access control, we need to be able to support some fairly general and rich access control policy language, e.g. XACML [XACML].

Managing and evaluating XACML or similar policies is too heavyweight for constrained devices. As a consequence we propose to externalize the main authorization decision to a less constrained node called the Authorization Server, acting on behalf of the resource owner.

On the other hand, access control enforcement SHOULD be performed in a trusted environment associated to the resource and as close to the resource as possible, in order to provide end-to-end security between resource and authorized client.

Moreover, verifications of local conditions SHOULD be performed in conjunction with accessing the resource for the following reasons:

- Transporting information about local conditions in the resource server to an authorization server for each policy decision (or on a regular basis) introduces delays and/or adds additional messages exclusively for the purpose of performing access control.
- Local conditions may have changed at the time of enforcement.

We therefore target enforcement and local decisions to take place in the constrained device hosting the resource, or in a proxy-type device offloading a severely constrained device hosting the resource.

We express local conditions as constraints under which an externally granted authorization decision is valid, and which are verified at the time and location of enforcement. For example if using XACML, such constraints can be expressed by Obligations.

In order to convey the authorization decisions (including local decision constraints) from the authorization server to the device where access control is enforced we use authorization assertions, which are digitally signed data objects containing authorization information. Examples of authorization assertions are SAML XACML authorization decision assertions [SAML-XACML] and OAuth 2.0 MAC tokens [I-D.ietf-oauth-v2-http-mac].

The assertions need to be lightweight so that constrained devices can parse and enforce authorization decisions. However, as this framework is targeting a variety of different constrained devices, protection mechanisms and use cases, we allow different assertion formats just as we must support different authentication methods and message protection mechanisms.

In order to encompass these different setups, this framework requires the specification of accompanying "profile", the contents of which are defined later in this document.

NOTES

1. The authorization server must be trusted by all involved parties, in particular by the resource server (and the resource owner). We assume that this trust relationship is manifested through trusted keys established a priori in the resource server and the authorization server (Section 7).
2. The existence of such a trust relationship, once introduced to support authorization and access control, can be utilized to optimize key establishment, authentication and message protection between the

client and the server. We define key provisioning schemes (section 7.2) that profiles of this framework may take advantage of.

2.3 Outline of Access Control Framework

We briefly outline the roles, the targeted access control procedure, the default components of the Access Control Framework (ACF) and what parts are specified by profiles.

The roles are:

- A Resource Owner specifying the policies for access to the resources.
- An Authorization Server (AS) performing the authorization decision based on the access control policies and provisioned with one or more trusted keys from the Resource Server.
- A potentially constrained Resource Server (RS) hosting resources and provisioned with one or more trusted keys from the AS.
- An Origin Client (OC) requesting authorization and access to a resource. As there may be client intermediaries, e.g. forward proxies, the actual CoAP client requesting the resource server may be different from the origin client. When there are no other clients to confuse with, we refer to the origin client simply as "the client".

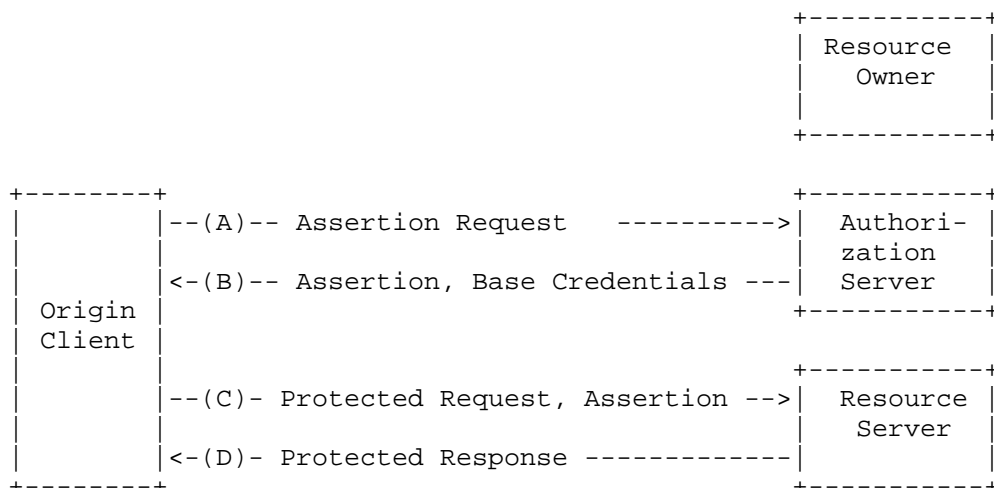


Figure 1: Roles and access control procedure

The default procedure for resource access works as follows (see Figure 1):

The client requests (A) from the authorization server an assertion for access to the resource.

The authorization server makes the authorization decision on behalf of the resource owner and, if granted, responds (B) to the client with an Assertion and optionally one or more Base Credentials to be used for protecting the message exchange between client and resource server. A base credential may e.g. be the public key of the resource server, a public key certificate of the origin client, or a so-called Bind Key (Section 7.2.2) bootstrapping the trust relation between the authorization server and resource server. Additional information may be conveyed between authorization server and the origin client, e.g. which Message Protection Profile (see below) is being used in communication with the resource server.

The client attaches the assertion to the request (C) submitted to the resource server, where the request/response message exchange is protected either using communication security, e.g. DTLS [RFC6347], or using object security, e.g. using JWE and JWS [I-D.ietf-jose-json-web-encryption][I-D.ietf-jose-json-web-signature]. There are also hybrid cases where both communication and object security is used, e.g. when the assertion signed by the authorization server is additionally signed and/or encrypted by the client and sent over DTLS.

In either case, the message protection may be based on a key provisioning scheme (Section 7.2) using base credentials provided by the authorization server. Alternatively, DTLS can be based on pre-provisioned credentials as defined by the security modes in [I-D.ietf-core-coap], or other secure provisioning schemes.

The resource server verifies the assertion using a trusted key, evaluates potential local conditions and, if all conditions are met, allows the requested action on the resource. Finally the resource server responds (D) to the request. The response uses the same protected message exchange mechanism as the request.

The Access Control Framework (ACF) as specified in this document defines the roles (Section 3), the overall architecture and message flow (presented in this section), the embedding of the assertion in CoAP (Section 4.3), the extended access control lists (Section 7.1), and key provisioning schemes between origin client and resource server enabled by the authorization server (Section 7.2).

The reason for including key management in the access control framework is because authorization and access control are intimately linked to authentication and secure messaging: Authorization is bound to certain entity, and secure messaging enables exclusive access. Therefore it is in line with the trust model and a clear gain in provisioning effort, and computational and memory performance to use a common key management for the different security functions.

The different applications of this access control framework in the different use cases are expressed in terms of profiles. A profile of the access control framework consists of an Assertion Profile and a client-server Message Protection Profile:

- An Assertion Profile specifies the assertion format including the procedure for signing and verifying the assertion, and how trusted AS keys are used (Section 4).
- A Message Protection Profile describes the protection of the resource request/response, what keys are used (including base credentials) and how (Section 5).

3. Roles

This section describes the roles of the components in the authorization framework.

3.1 Resource Owner

The resource owner specifies the policies for access to the resources. The access policy language or format for expressing policy related conditions may be specified in an assertion profile if it affects the format of the assertion or procedures related to composing, signing, verifying or parsing the assertion.

The role of the resource owner is obviously an important one, but is not directly involved in the access control framework specified here.

3.2 Authorization Server

The authorization server is acting on behalf of the resource owner and has several functions:

1. Storing and using trusted keys of the resource server (trusted RS keys). The establishment and regular re-establishment of the trusted

keys is out of scope.

2. Performing the authorization decision. How this is done is out of scope.

3. Generating, signing, and issuing assertions for requesting clients to be verified by a resource server and potentially intermediate nodes. The details are specified in the assertion profile.

4. Communicating with clients (receiving access request, responding with assertion, base credentials or other data). The details are specified in the message protection profile.

NOTE

The authorization server must have knowledge about its resource servers' supported assertion and communication security profiles.

3.3 Client

A client wishing to access a resource, requests an assertion from the authorization server. If the request is granted it receives an assertion, and optionally base credentials and other information, to use when communicating with a resource server. The client requests access to a resource from the resource server and attaches the assertion. The origin client may need to encrypt and/or sign the assertion before sending it to the resource server. The base credentials may be used to protect message exchanges between client and resource server. The procedures are described in the message protection profile.

3.4 Resource Server

The resource server, is a potentially constrained device hosting the resource owner's resources. It receives, processes and responds to client requests. The resource server has an established trust relation with an authorization server (identified with a trusted AS identifier), manifested through one or more trusted AS keys. The resource server verifies that an assertion received in an access request is valid and from a trusted source involving a trusted AS key. If the assertion depends on the client's identity, the resource server verifies that the requesting client is the same as the entity authorized by the assertion. The assertion verification and message protection procedures are specified in the respective profiles.

4. Assertion profiles

The assertion is a secure object containing authorization information passed from the authorization server via a client to the resource server. The assertion is digitally signed by the authorization server to enable data origin verification and integrity protection. The assertion format is specified in assertion profile, an example is given in Section 8.1.

4.1 Assertion requirements

In order to enable the resource server to enforce the authorization decision, the assertion SHALL provide the following information:

- o Which resource does the decision apply to.
- o Which action (GET, PUT, POST, DELETE) does the decision apply to.
- o Which client does the decision apply to, and how can this client be authenticated (if necessary).
- o Which authorization server has issued this assertion (AS identifier). This information MAY be implicit from the signature of the assertion.
- o Under what other conditions is the assertion valid (expiration date, conditions evaluated by the resource server at access time).

The assertion SHALL include an Assertion Identifier which is unique for a given resource server. If the ACF SSK key provisioning scheme (Section 7.2.2) is used, the assertion SHALL include a Nonce N which is different for different assertions of a given resource server. The nonce may be the assertion identifier.

The assertion MUST be signed by the authorization server such that it can be verified by the resource server using a trusted key. An assertion MAY be signed with a secret symmetric key, which is then called the Assertion Key (AK). An assertion MAY be signed with a private key in an asymmetric signature scheme, for example the authorization server's private key (PrK_AS). The resource server, or other nodes verifying the assertion, MUST have access to the relevant key (AK or PK_AS) at the time of performing the verification.

The case of an asymmetric signature scheme is RECOMMENDED if, in addition to the resource server, an intermediary node should also be able to verify the assertion, since the intermediary would only require to be provisioned with a public key rather than a secret key.

The assertion MAY be (partially) encrypted and signed by the origin client, if so specified in the message protection profile. The assertion identifier SHOULD NOT be encrypted. If the ACF SSK key provisioning scheme is used, the Nonce, or Assertion Identifier if that is used as nonce, SHALL NOT be encrypted.

When using a communication security profile, assertions MAY be cached (Section 5.1.1) and there is no need for replay protection, assuming the communication security protocol supports replay protection (e.g. DTLS, IPsec). When using an object security profile, assertions SHALL NOT be cached and assertion replay protection SHALL be applied (Section 5.2.4).

4.2 Assertion wrapping

Since assertions are to be consumed by constrained devices, the protection of the assertion must be lightweight and compact. For example XML-Signature [XMLDSig] is not considered to fulfill this requirement. This specification RECOMMENDS the use of JSON Web Signatures (JWS) [I-D.ietf-jose-json-web-signature] as a means of signing assertions. It is furthermore RECOMMENDED to use the JWS Compact Serialization in order to further reduce the size of the signed assertion object.

4.3 Assertion transport

The assertion needs to be delivered to the Resource Server in some way. There are several possible methods to achieve this:

1. One alternative is to transmit the assertion in CoAP as a URI query value. When using this option it is RECOMMENDED to use the key "authz". A limitation with this approach is that the URI query option in CoAP is defined as Unsafe-to-Forward, meaning that a potential forward proxy that does not understand the authorization query would not forward such a request.
2. Another alternative is to transport the assertion using a dedicated CoAP option. Such an option SHOULD use an option number in the "Specification Required" range of 256..2047. The option SHOULD be defined as Critical, and Safe-to-Forward. It SHOULD NOT be used as cache-key and it SHOULD NOT be repeated. The content format SHOULD be string and the Length between 1-1023 bytes. A Default value SHOULD NOT be specified.
3. A third alternative, if DTLS is used for communication security, is a DTLS adaptation of the TLS Authorization Extensions [RFC5878].

This provides an alternative to the TLS extension defined in Section 8.2.

5. Message protection profiles

The message protection between origin client and resource server is specified in a profile. The request/response messages may be protected with communication security (e.g. DTLS, IPsec) or with object security (e.g. JWE, JWS). The message protection profile SHALL specify:

- o Which security protocol is used for authentication and secure messaging between origin client and resource server.
- o Which cipher suites are used, and how cipher suite negotiations of the security protocol are used.
- o Whether the assertion is encrypted and/or signed by the client.
- o Which keys are used and how they are obtained a) in the origin client (including the use of base credentials) and b) in the resource server.
- o Which CoAP security mode is assumed (if any, [I-D.ietf-core-coap]) and what ACF key provisioning scheme (if any, section 7.2).
- o How to handle intermediary nodes (Section 6).
- o Processing in origin client, resource server and intermediary nodes at sending and receiving messages.

A resource server SHALL support one and only one message protection profile, either a communication security profile or an object security profile. How the authorization server gets information about the message protection profile supported by the resource server is not in scope.

5.1 Communication security profile

A communication security profile defines the detailed security protocol used between client and resource server in the case when there are two security protocol phases: One security establishment phase performing authentication and key establishment (such as DTLS handshake protocol or IKE) and one secure messaging phase (such as DTLS record protocol or IPsec) wherein CoAP messages are sent.

Communication security protocols like DTLS and IPsec provides protection to the entire application layer protocol message including headers, assertion and payload.

The "detailed security protocol" in the communication security profile may also include object security components, as is illustrated in the examples below.

Example 1: The assertion is a secure object in itself, being signed by the authorization server, but it may be necessary to also prove the intent of the authorized client to perform the request it is authorized with. This may be achieved by an authentication protocol authenticating a client as described in the assertion. Alternatively it may be addressed with a client signature on the assertion.

Example 2: The content of the assertion may require confidentiality protection between client and server e.g. for privacy reasons. This may be achieved by an end-to-end communication security protocol (see below) which encrypts the messages. Alternatively it may be addressed with encryption by the client of the assertion.

We distinguish between the end-to-end communication security setting where the communication security protocol runs between origin client and resource server, and the hop-by-hop security setting where there are intermediary nodes (Section 6.1).

In the end-to-end security setting, the resource server authenticates the origin client. The resource server SHALL verify that the subject authorized with the assertion coincides with the authenticated client.

In the end-to-end security setting the client signature on the assertion MAY be omitted (compare DTLS hop-by-hop, Section 6.1). If communication security is established using public keys, the public key of the client MAY be replaced with a public key identifier in the assertion.

5.1.1 Assertion caching in resource server (informational)

There may be cases of repeated access requests from one client where a single assertion could provide authorization of all requests, e.g. requests within the life time of the assertion. To avoid repeatedly sending the same assertion in each request, assertions could be cached.

Assertions may support caching/multiple use i.e. that the same assertion is valid for multiple requests.

The same assertion may be sent multiple times within a secure session, but if the resource server supports assertion caching this is not recommended.

Assertion caching shall only be used with session based communication security profiles and if supported by the resource server. The assertion shall be deleted when the session expires.

Assertions shall be verified before caching. Not all parts of the assertion need to be cached. If caching is done, the assertion identifier, client identifier, local conditions, resource identifier, and the action shall be cached. Local conditions in the assertion shall be verified at each request or, considering Observe (Section 6.3), before each response.

After communication security has been established with a resource server supporting caching of assertions, on reception of a request, the following access control priority sequence shall be used:

- 1) If the client identifier is category A (Section 7.1), the request is granted. In this case, further access control processing is omitted.
- 2) Else if the client identifier is category B check for a cache hit. If a matching assertion is found, verify the local conditions and grant access if they are fulfilled.
- 3) Else if the request contains an assertion verify it, and if successful, cache it. Then verify the local conditions, and if valid then grant access.

TBD Assertion caching in proxies. TBD Response code if assertion is cached

5.2 Object security profile

The authentication protocol for establishing the context of a secure session, e.g. the DTLS handshake, may be a significant cost for a constrained device. An alternative is to not have a security establishment phase and instead send secure objects in CoAP request and response.

The requirements on message protection profiles as listed in the

beginning of section 5 apply. In this case the required security protocol is manifested through the processing and passing of the secure objects.

The assertion is a secure object in itself which is passed in the CoAP request. In addition to the assertion, the CoAP request and the CoAP response may contain a secure payload object. The object security profile defines what object format is being used, e.g. JWE [I-D.ietf-jose-json-web-encryption].

Assertions MUST NOT be cached in case of an object security profile. Assertion are one-time only, and the resource server SHALL maintain a replay cache (Section 5.2.4).

Different request methods (GET/PUT/POST/DELETE) may have different secure object formats for request and response.

5.2.1 GET

Assertions for GET requests SHOULD be signed by the client. The client signature SHALL use the key indicated by the client identifier in the assertion. In some cases, e.g. extremely constrained devices and other settings where implicit authentication is feasible, the client signature MAY be omitted. The assertion MAY be (partially) encrypted by the client.

If the resource request is authorized, then the requested action is performed and what would have been the unprotected CoAP response payload SHALL be wrapped into a secure object and sent as response payload. The GET response SHALL be integrity protected and MAY be encrypted.

NOTE that in the case of implicit authentication (no client signature on the assertion) the resource server does not establish the identity of the requesting client. However, by protecting the response for the authorized client, only the authorized client can get access to the response. This is however an entry for DoS so should be used only when the potential security consequences are considered acceptable.

5.2.2 PUT/POST

In the PUT/POST request, the assertion or the payload SHALL be signed by the client. The assertion and the payload MAY be (partially) encrypted, e.g. for privacy reasons.

The object secure PUT/POST response payload MAY be signed or encrypted by the resource server.

5.2.3 DELETE

Assertions for a DELETE request SHALL be signed by the client. The object secure DELETE response payload MAY be signed or encrypted by the resource server.

5.2.4 Assertion replay protection (informational)

In case an assertion is eavesdropped it may be replayed at a later stage by an unauthorized party and cause unnecessary processing and power consumption in the resource server. In the case of a communication security protocol with replay protection this is not an issue. In the case of an object security protocol, even if the assertion may be in parts encrypted, it may nevertheless be eavesdropped and replayed. For this purpose we recommend the use of a Replay Cache where the latest legitimate requests are being stored, as outlined in this section.

For each entry in the replay cache there are two values: the Assertion Identifier and the associated assertion expiration date. The replay cache is being managed in the following way during reception of object secured requests. When a request is received:

1. The expiration date of the assertion is checked.
2. The identifier of the assertion is matched against the replay cache.

If any of these verifications fail, the request is immediately silently dropped to avoid unnecessary processing. The resource server may also increase a counter that an invalid request was received, such a counter may be used as input for a decision to e.g. stop receiving or go to sleep a randomized time interval.

If the verifications are successful then the signature of the assertion is verified. If that is also successful, then the a new entry is added to the replay cache. Before a new entry is being added, the cache is cleaned from entries associated to expired assertions. If the cleaned replay cache is full, the new entry pushes out the entry with shortest remaining lifetime, and if several of equal remaining lifetimes, the oldest of these is pushed out.

The size of the replay cache and assertion lifetimes should be tuned so that the replay cache can contain all entries corresponding to

valid assertions. The importance of this recommendation depends on the impact of processing invalid requests.

6. Intermediary processing and notifications

This section describes the security implications of intermediary processing and notifications for access control.

6.1 Intermediary nodes

There may be intermediary nodes between origin client and resource server, including forward proxies, reverse proxies, cross-proxies, gateways, etc. From an access control point of view the resource server SHOULD be able to verify that a received CoAP request is originating from the origin client referenced in the received assertion. This has implications on the assertion and message protection profiles.

We distinguish between the end-to-end security setting where no intermediary nodes need be trusted and the hop-by-hop security setting where at least one intermediary node must be trusted. Note that an object security profile may provide end-to-end security independently of proxies, but would not allow any inspection or modification of the payload.

DTLS generally needs to be hop-by-hop in case of proxies, this requires some degree of trust in a proxy which may not be acceptable for some applications. A resource server sending back the response via the forward proxy trusts the forward proxy with the plain text response (e.g. a GET response) and that the proxy has established secure communication with the origin client.

In the hop-by-hop setting, neither DTLS nor CoAP offers any means for resource server to authenticate the origin client.

If the resource server has established DTLS with a forward proxy which proxies requests from an origin client, then the assertion MUST be signed by the origin client in addition to the authorization server signature. The resource server can not authenticate the origin client directly, but it can infer from a correctly signed valid and fresh assertion that the origin client is authorized and has an intent to perform the request.

Using the authorization server as a trusted party it is possible to define a message protection profile which also provides base

credentials for proxies or for clients use of proxies, which support DTLS establishment between origin client and a proxy (Section 7.3.1).

6.2 Mirror Server

The access control framework can also be applied to the scenario where a mirror server as defined in [I-D.vial-core-mirror-proxy] is present. In such a scenario, the resource servers behave as the clients of the mirror server. The access control enforcement in this case, would be made at the mirror server instead of in a constrained resource server, and the trusted AS keys would have to be provisioned to the mirror server. However, to a client wishing to access a resource, the mirror server behaves as any other resource server and is indistinguishable (transparent), thereby requiring no change for the communication between client and the mirror server. The communication between the mirror server and the constrained resource server may or may not be secured, and is oblivious to the ACF profiles used between the client and the mirror server.

6.3 Observe

The access control framework can also be applied, as it is, in the case where the CoAP observe option [I-D.ietf-core-observe] is used. With the observe option, the clients can register an interest in a particular resource by sending a CoAP request containing the observe option to a resource server. The resource server would in this case maintain the state information for this expressed interest and send responses on state changes only as long as the assertion and local conditions presented in the original interest request are valid. The local conditions may need to be verified at each state change. Once the assertion expires, the resource server will remove any state information for the interest expressed. The client would then have to send a new CoAP request with an observe option expressing interest and a new assertion for demonstrating that it is allowed access.

7. Trust management

The access control framework specified here involves at least three entities, and thus a number of different trust relations are possible. The framework does not make any assumptions on the a priori trust relationship between origin client and resource server (if any). The access control framework requires however an a priori trust establishment between resource server and authorization server manifested, in particular, in one or more trusted AS keys stored in

the resource server. How these keys are established is out of scope, but we define different categories of keys in Section 7.1.

The primary purpose for the AS keys is to enable the resource server to verify that an assertion received in a request from a client was issued by the authorization server and has not been modified. Data origin authentication and integrity verification is performed by verifying a digital signature of the assertion. This digital signature may either be a private key signature generated with the private key of the authorization server (PrK_AS), or a Message Authentication Code (MAC) calculated with a secret symmetric Assertion Key (AK) unique for this resource server and assertion. In the case of a private key signature, the resource server must have access to the public key of the authorization server (PK_AS) at the time of verifying the assertion. In the case of a MAC, the resource server must have access to AK at the time of verifying the assertion. Two key provisioning schemes are defined in this section.

The trusted AS keys may also be utilized for other purposes than to support assertion verification. By means of authentication and authorization of a client, the authorization server may invoke a trust relation between the resource server and the client. This is the main purpose of the key provisioning schemes detailed later in this section. For example, the AS may create a certificate of a public key of the client which can be verified with an AS key. As another example, the AS may create a secret Bind Key for the origin client which the resource server can derive from a nonce and an AS key (Section 7.2.2). The public key or the bind key of the client can subsequently be used by the resource server e.g. for authentication of and secure messaging with a hitherto unknown client. The latter is detailed in the message protection profiles.

7.1 Access Control Lists

Considering that different deployments may have very different access control requirements we want to keep the simple access control scheme indicated in [I-D.ietf-core-coap], and allow a gradual extension to comply with more advanced access control settings.

The use of an access control lists in the resource server containing identities of exchanged public keys, or pre-shared keys, of clients [I-D.ietf-core-coap] provides an easily described criterion for determining if a requesting client is allowed to establish secure communications: Clients that can be authenticated as one in the list are allowed. In the absence of any other access control, authenticated clients are also implicitly authorized to access any resource ("all-or-nothing" access control).

Similarly with a list of root trust anchors in security mode Certificate [I-D.ietf-core-coap]: Clients that can be authenticated using a public key certificate signed by a trust root from the list, are allowed to establish secure communication (and potentially to become implicitly authorized). In this case the list contains identities of trust roots which are trusted with issuing certificates to authorized clients.

We propose to maintain backward compatibility with these access control lists but at the same time enable a distinction between authentication/secure communication and authorization. There SHALL be two categories of client keys/identities in the access control lists of resource servers:

- o Category A: identities or keys of clients which are allowed to set up secure communication to a resource server, and to access all resources.
- o Category B: identities or keys of clients which are allowed to set up secure communication, but access to a resource requires an appropriate valid assertion.

Requests from clients in category A requires no assertion and can bypass the access control.

Clients that have been authorized access to a resource by an authorization server should obviously have the right to set up secure communication to this resource server, and thus are in category B. Hence the authorization server constitutes a trust root for clients of category B. The root trust anchor is in this case a trusted AS key.

Summarizing, there are four kinds of access control lists: Category A, its trust anchors, Category B, and its trust anchors. Not all lists need to be populated. Implementations complying with the security mode RawPublicKey [I-D.ietf-core-coap] define access control lists of category A. For implementations complying with the present specification category B trust anchors are necessary. In Section 7.3 we show examples that the latter access control list may also be sufficient.

One example of a member of the category B access control list is a pre-provisioned, semi-trusted forward proxy which is allowed to set up secure communication with a resource server, but without gaining access to any resource.

NOTES

1. While the trusted AS key is a category B trust anchor, the property of trusted AS keys relate to verification of assertions, and not all category B trust anchors are necessarily used for verifying assertions.

2. For object security the trusted AS keys are sufficient for determining access rights as there is no need to set up secure communication first. There is no access without assertion in the object security setting - corresponding to category A of the communication security setting. The reason for excluding this option is that in the object security setting there is always at least one object to be verified by the resource server and there is no generality lost in assuming that this object is the assertion.

TBD - change of access control lists: Can the ACL in itself be a resource that can be accessed like any other and whose changes can be authorized with the methods proposed here?

7.2 Key provisioning schemes

There are two basic key provisioning schemes defined in this access control framework: the ACF PK scheme built on public keys exchanged between RS and AS, and the ACF SSK scheme built on a shared secret key between RS and AS. The key provisioning schemes define which keys are assumed to be provisioned to what nodes, and how these keys can be used for deriving and/or provisioning of other keys.

The key provisioning schemes apply both to communication security and object security. The assertion is by definition a secure object. The key management is more complex for communication security than object security and assertions, since the former involves key management both for communication security and for secure objects.

Note that these key provisioning schemes are complementing the Security Modes defined in the DTLS binding of CoAP [I-D.ietf-core-coap] and may replace the need for provisioning keys between resource server and a client. See examples of Section 7.3.

7.2.1 ACF PK Scheme

The ACF PK scheme assumes that the public key of the resource server, PK_RS, is provisioned to the authorization server and vice-versa; the public key of the assertion server PK_AS is provisioned to the resource server. The trusted AS key is in this case PK_AS. With this scheme the assertion is signed by the authorization server with its private key PrK_AS and verified by the resource server using PK_AS.

The content of the assertion provides a secure information channel from the authorization server to the resource server, and may be used as a means for provisioning. For example, the public key of the origin client, PK_OC transported within the signed assertion to the resource server is integrity protected.

When an origin client requests an assertion for access to a resource, the public key of this resource server can be attached in the response as one of the base credentials, as a means to provision PK_RS to this origin client.

As an alternative to provisioning PK_OC to the resource server in an assertion, the authorization server may issue a public key certificate for PK_OC as a base credential together with the response carrying the assertion. This certificate may then be used by the origin client in a certificate based authentication protocol with the resource server. PK_AS is in this case a category A or category B trust anchor. A similar provisioning scheme is also applicable to trust establishment with a forward proxy, see Section 7.3.1.

7.2.2 ACF SSK Scheme

The ACF SSK scheme assumes that there is a 128 bit shared secret key SSK between resource server and authorization server, and specifies how to derive the assertion key AK and the Bind Key BK from SSK. The trusted AS key is in this case SSK. The assertion is signed by the authorization server with AK and verified by the resource server with AK.

We assume that the shared secret key SSK is pseudo-random, i.e. not biased. It may for example be the result of a randomness extraction step from an initial key material IKM as defined in [RFC5869].

AK and BK is derived from SSK by the authorization server and resource server through a data expansion step as defined in [RFC5246]:

$$\begin{aligned} P_hash(secret, seed) = & HMAC_hash(secret, A(1) + seed) + \\ & HMAC_hash(secret, A(2) + seed) + \\ & HMAC_hash(secret, A(3) + seed) + \dots \end{aligned}$$

where '+' indicates concatenation and A() is defined as:

A(0) = seed

A(i) = HMAC_hash(secret, A(i-1))

In the present case:

- o hash is SHA-256,
- o 'secret' is SSK - the shared secret, and
- o 'seed' is the nonce, N.

The nonce N associated to the resource access request is defined by the authorization server. A Nonce SHALL NOT be reused with the same SSK. Since the nonce is required for key derivation, it must be available when the key is needed by the resource server.

With one iteration of the P_SHA256, the output data of 256 bits is defined to be the concatenation of the two 128 bit keys AK and BK, where AK is the Assertion Key and BK is the Bind Key:

$$AK + BK = P_SHA256(SSK, N)$$

How the assertion key and the bind key is used in protecting the assertion and client-server communication, respectively, is defined in an ACF profile.

7.3 Key provisioning examples

The CoAP security modes makes assumptions on what keys are provisioned in the resource server.

- o In security mode PreSharedKey there is a list of keys corresponding to trusted nodes. With the terminology in Section 7.1, these clients may be of category A or B.

- o In security mode RawPublicKey there is a list of identities of trusted nodes, these clients may be of category A or B.

- o In security mode Certificate there is a list of trust anchors, these trust anchors may be of category A or B.

This access control framework requires the provisioning of a trusted AS key. In either of the cases above, the additional effort of provisioning a trusted AS key, which in particular is a category B trust anchor, is negligible.

This section takes the opposite standpoint: assuming the use of this access control framework with the authorization server functionality and trusted AS key provisioned to the resource server, what other keys (if any) need to be pre-provisioned to the resource server and what keys can be communicated during the access request procedure?

Phrased differently: To what extent could this access control framework remove the need for, or simplify, the provisioning of keys?

7.3.1 The authorization server as certification authority

Given that the trusted AS key may be used as a category A/B trust anchor, the authorization server may issue public key certificates to origin clients that are verified in the resource server. This removes the need to pre-provision client public keys to the resource server.

The origin client may also use certificates signed by the authorization server to set up DTLs with an intermediary node such as a forward proxy. This would require the proxy to be provisioned with PK_AS, which however is a natural assumption: A forward proxy may also perform firewall functionality on behalf of the resource server; to verify some properties of the assertions and drop requests with e.g. expired assertions, assertions with invalid signatures, etc.

Alternatively, the TLS Authorization Extensions [RFC5878] may be used with the assertion replacing the need for a certificate.

7.3.2 The authorization server for configuration of raw public keys

Since the origin client has a secure channel to the authorization server which is a trusted party, the origin client can obtain trusted information directly from the authorization server. The public key of the resource server can e.g. be provided as a base credential, removing the need to pre-provision resource server public key to the origin client.

7.3.3 The assertion as an authentication token

There are two aspects of authentication tokens considered here.

1. The assertion may contain the public key of the client thereby carrying authentication information about the client.
2. The assertion may be signed by the client thereby constitute a verifiable token for authentication of the client.

Given that the assertion contains the public key of the client, the resource server may use the assertion to establish DTLs in security mode RawPublicKey with the client, removing the need to pre-provision client public keys to the resource server or to pre-provision or use client public key certificates.

The use of an origin client signature of the assertion provides an additional means for authenticating the client, removing the need for a separate secure payload object, or providing authentication information over multiple hops.

7.3.4 Using the assertion to transport the nonce

The assertion provides transport of the nonce to the origin client and resource server.

In the ACF SSK scheme, with object security, the transportation of nonce in the assertion enables the resource server to derive of the bind key, which removes the need to pre-provision shared secret keys between origin client and resource server.

7.3.5 TLS extension for transport of nonce

The authentication protocol may provide transport of the nonce between origin client and resource server.

In the ACF SSK scheme, with DTLS, the transportation of a nonce in a dedicated TLS extension (Section 8.2) enables the resource server to derive the bind key, which removes the need to pre-provision shared secret keys between origin client and resource server.

8. ACF Profiles

8.1 Assertion Profile: Compact SAML-XACML

In this section we give an example of an assertion profile, using SAML [SAML] and XACML [XACML]. This profile is based on the SAML 2.0 Profile of XACML Version 2.0 [SAML-XACML] with a number of significant changes to this profile, in order to account for the constrained nature of the resource host.

Since the full syntax of XACML Requests, Responses, and SAML assertions includes a large number of features, we have defined a subset of both standards, in order to simplify the processing on the resource server. Furthermore the XML representation of this subset is too verbose for efficient transmission over limited channels, therefore we have defined a compact JSON-based notation of this SAML and XACML subset.

SAML assertions have the following required elements:

- An assertion identifier
- The issue instant
- The SAML version number
- The Issuer of the assertion

Furthermore the following optional elements are relevant for the purposes of this profile:

- The Subject of the assertion
- A Statement (in our case an authorization statement)

The SAML 2.0 Profile of XACML defines an XACML authorization decision statement as a sequence of an XACML Response and an optional XACML Request. It is obvious in our case that the Request **MUST** be part of the statement to give the necessary context, otherwise it would be impossible to determine what the Response applies to.

The following restrictions have been defined to create a useful subset of the SAML assertion format:

- The assertion identifier **SHALL** be a 128 bit random number encoded as UUID [RFC4122].
- The issue instant is encoded in UTC form without time zone component.
- The SAML version number is implicitly assumed to have the value "2.0".
- The encoding of the Issuer is up to the implementers, but it **SHOULD** enable consumers of the assertion to determine the identity of the trusted entity that issued this assertion, and thus to derive the key used to verify the signature of the assertion.
- The Subject **SHALL** either contain a subject identifier that can be used to bind the assertion to a subject in some way that is implicit to the consumer, or a subject's public key that can be used in an authentication scheme.
- The Statement **SHALL NOT** contain an XACML Response. The Response is implicitly assumed to contain a PERMIT decision. An entity generating assertion **MUST NOT** generate assertions for decisions other than PERMIT. If the XACML Response contained XACML Obligations these **MUST** be included in the Statement. It is **RECOMMENDED** not to write

Policies that would result in more than one Obligation in a Response.

The resource server MUST know how to process the Obligation, otherwise it MUST reject the assertion and deny the request with error message.

- The XACML Request MUST contain the two single attributes identifying the resource and the action. The subject identifier MUST NOT be part of the Request since it is inferred from the SAML subject identifier. The resource identifier SHALL be the resource URI, without the scheme name. The query part MAY also be omitted if that level of granularity is not desired in the access control policies. The action identifier SHALL be one of the values: GET, POST, PUT, DELETE. In the case of a POST or PUT action, the payload value of the action SHALL be encoded as an action attribute with the attribute identifier "VAL". The entity generating the assertion MUST discard all other attributes from the Request.

The following compact encoding defines the assertion format in this profile, listing the JSON elements in the order in which they are expected to appear in the final assertion object:

- The assertion identifier SHALL use the member name "ID".
- The issue instant SHALL use the member name "II".
- The Issuer SHALL use the member name "IS".
- The Subject element SHALL use the member name "SK" if it represents a subject-key, where it is assumed that the consuming device and the issuer of the assertion share a common understanding of the type of key that is encoded in that value. If the Subject element is an identifier, it SHALL use the member name "SU".
- The Statement SHALL be a JSON object with the name "ST" containing the following members:
 - o If the XACML Response contained any Obligations, they SHALL be represented in JSON objects or members with the name "OB", the content or value format of these objects MUST be understood by the consuming device.
 - o The action identifier SHALL use the member name "ACT".
 - o If the action is a PUT or POST, the payload value SHALL use the member name "VAL".
 - o The resource identifier SHALL use the member name "RES".

In order to protect the integrity of the assertion JWS [I-D.ietf-jose-json-web-signature] SHALL be used. If only the AS is supposed to sign the assertion it is RECOMMENDED to use JWS Compact Serialization, if an additional signature by the the origin client is required (see Sections 5.2.1-5.2.3) it is RECOMMENDED to use JWS JSON Serialization. If the origin client is required to encrypt the assertion, this MUST be performed in the same way as specified for the payload in Section 8.3, where the encrypted plaintext is the JWS Compact Serialization of the assertion signed by the AS.

Implementations using the ACF PK Scheme (see Section 7.2.1) MUST support the ES256 algorithm. PrK_AS MUST be used for generating the signature and PK_AS for verifying the signature.

Implementations using the ACF SSK Scheme (see Section 7.2.2) MUST support the HS256 algorithm. The Assertion Key AK MUST be used as key for generating the message authentication code.

The algorithms corresponding to the identifiers above are specified in [I-D.ietf-jose-json-web-algorithms].

In order to verify the assertion, the RS needs to perform the following steps:

1. Verify if the assertion has expired. This profile assumes that each RS uses a fixed validity time for any assertion. Therefore expiration can be easily checked by verifying the time elapsed since the issue instant. In cases where the RS lacks the means to measure time reliably, this step is skipped.
2. Verify if the assertion was issued by a trusted AS, by checking the Issuer field of the assertion.
3. Verify that the assertion applies to the CoAP request, i.e. that it applies to the same resource and action.
4. Verify that the origin client is the Subject of the assertion. This might not be possible if there are intermediary nodes, in this case the assertion can have an additional signature by the OC to confirm its intent to perform the authorized action. Note that certain actions allow implicit authentication (see Sections 5.2.1-5.2.3), in which case this step can be skipped.
5. Verify the AS signature of the assertion.
6. If the assertion contains any Obligations, check that these are satisfied.

The following example illustrates an assertion generated according to this profile, without the JWS signature:

```
01 {
02   "ID": "ID_ffda55f9...097bdd21e6",
03   "II": "2013-02-15T10:02:52Z",
04   "IS": "AAA-Server",
05   "SK": "BvDgLAXSHe...0RLhfwS1fue",
06   "ST": {
07     "OB": {
08       "NB": "09:00:00Z",
09       "NA": "17:00:00Z"
10     },
11     "ACT": "GET"
12     "RES": "node346/tempSensor"
13   }
14 }
```

Note that the assertion shown would have a size of 208 bytes without pretty printing. The corresponding assertion following the XML representation of SAML and XACML would be 2281 bytes large. Also note that this assertion carries an obligation on the time interval during which access is allowed (NB = not before, NA = not after). This obligation must be pre-defined on the consuming device so that it can parse and enforce it.

8.2 Communication Security Profile: DTLS PSK with Bind Key

This section gives an example of a communication security profile based on DTLS PSK [RFC4279] using the bind key (BK) of the ACF SSK key provisioning scheme as if it was a pre-shared key. PSK-cipher suites and cipher suite negotiations are identical.

The setup is identical to the CoAP security mode PreSharedKey [I-D.ietf-core-coap] with the exception that the key BK is not pre-provisioned to the resource server but needs to be derived from a nonce and the shared secret key SSK (Section 7.2.2).

Using the coaps:// schema, DTLS security context is established before the CoAP request in which the assertion is sent. Therefore, in order to derive a bind key to be used in the DTLS handshake the nonce needs also to be sent in the DTLS handshake.

When using this profile in an end-to-end communication security setting, the nonce SHALL be included as a DTLS extension in ClientHello.

The extension type is defined as follows:

```
enum {  
    key_derivation(19), (65535)  
} ExtensionType;
```

The "extension_data" field of this extension contains a "key_derivation_data" value, defined as follows:

```
struct {  
    opaque nonce[16];  
    opaque as_id<1..32>;  
} key_derivation_data
```

The following modifications to the DTLS handshake SHALL apply:

Before composing the ClientHello message, the origin client reads out the nonce and the AS identifier from the assertion and composes the key_derivation extension as defined above. This extension is sent in the ClientHello message.

After receiving the ClientHello message, the resource server looks up the SSK based on the AS identifier, and computes BK as described in section 7.2.2, using the provided nonce. BK is used as the PSK from here on, and the protocol proceeds as specified in [RFC4279].

8.3 Object Security Profile: JWE-protected CoAP payload

This section gives an example of an object security profile, i.e. protection of CoAP payload by means of secure objects. When using this profile the Javascript Object Signing and Encryption (JOSE) standards SHALL be used. Section 5.2 and its subsections specify when a payload or assertion is to be signed and encrypted. If a payload is to be signed but not encrypted the JSON Web Signature (JWS) standard [I-D.ietf-jose-json-web-signature] SHALL be used. If a payload is to be signed and encrypted the JSON Web Encryption (JWE) standard [I-D.ietf-jose-json-web-encryption] SHALL be used. If only a single signature is needed on the assertion, it is RECOMMENDED to use the Compact Serialization instead of the JSON Serialization in order to keep the payload size small.

Implementations using the ACF PK Scheme (see Section 7.2.1), MUST

support the ES256 JWS algorithm. For JWE the ECDH-ES key sharing algorithms MUST be supported, and for authenticated encryption the A128GCM algorithm MUST be supported.

Implementations using the ACF SSK Scheme (see Section 7.2.2) MUST support the HS256 algorithms for JWS. For JWE the dir key sharing algorithm MUST be supported together with the A128GCM authenticated encryption algorithm.

Implementations using the A128GCM algorithm MUST ensure that the specific key freshness requirements of the GCM mode are observed (see [NIST-SP-800-38D] Section 8.3).

The algorithms corresponding to these identifiers are specified in [I-D.ietf-jose-json-web-algorithms].

9. Security Considerations

The present framework aims to protect the resources on resource servers, the servers themselves, and the services offered. The means proposed to protect these assets is to enforce more granular access restrictions on accessing the devices than all-or-nothing. Due to the setup of the framework, there is also a need to protect the authorization decisions and the keys used to protect the entire resource access procedure.

The authorization server is a Trusted Third Party from the point of view of the resource owner, which if compromised could e.g. issue assertions to unauthorized parties or use a bind key to decrypt an eavesdropped GET response secure object payload.

In order to enforce a policy decision, the resource server must authenticate the requesting client, and match the identifier of the authenticated entity with the subject identifier of the assertion. As a consequence, in the case of DTLS, the handshake protocol must potentially be executed without knowing that the request will be authorized which opens up for a potential DoS attack.

Comparing with authorization using category A identifiers of the access control lists (equivalent to [I-D.ietf-core-coap]), an unauthorized CoAP request masquerading as a legitimate will also not be detected until in the third pass of DTLS handshake.

DTLS introduces stateless cookies in ClientHello to prevent DoS attacks. The server having received a ClientHello without cookie from an unverified client sends a HelloVerifyRequest message expecting the client to return ClientHello with a cookie

corresponding to the parameter values. Only if the cookie can be successfully verified by the server will the DTLS handshake protocol continue. These cookies are unfortunately not very useful in the setting where the client is much more powerful than the server, and the very sending of a HelloVerifyRequest may cost much more than e.g. a verification of a digital signature (e.g. a MAC).

If DTLS Handshake is considered a serious threat for DoS, an alternative approach is to carry some lightweight (short and easily computed) keyed integrity information in a TLS extension (TBD) of the ClientHello. Using the established key material, the server can infer with reasonable assurance if a request is legitimate and drop other requests.

While DTLS offers bundled encryption and integrity protection of both payload and headers, the object security approach allows for a trade-off between protection against performance. Depending on the trust model, assertion and payload may need to be encrypted because eavesdropping will reveal information about the client's request, which may be privacy sensitive. Wrapping of the payloads as secure objects allows differentiated protection of the content based on its sensitiveness.

10. IANA Considerations

<IANA considerations text>

TBD: Text about CoAP option numbers (see 4.3). Text about DTLS/TLS extension (see 8.2).

11. References

11.1 Normative References

[I-D.ietf-core-coap]

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

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

11.2 Informative References

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and Keranen, A., "Terminology for
Constrained Node Networks", draft-ietf-lwig-terminology-04
(work in progress), April 2013.

[SAML]

Cantor, S., Kemp, J., Philpott, R., Maler, E., "Assertions
and Protocols for the OASIS Security Assertion Markup
Language(SAML) V2.0", OASIS, March 2005.

[XACML]

Rissanen, E., "eXtensible Access Control Markup Language
(XACML) Version 3.0", OASIS, September 2012.

[SAML-XACML]

Rissanen E., and Lockhart H., "SAML 2.0 Profile of XACML ,
Version 2.0", OASIS, August 2010.

[I-D.ietf-oauth-v2-http-mac]

Mills, W. and Tschofenig H., Ed., "OAuth 2.0 Message
Authentication Code (MAC) Tokens", draft-ietf-oauth-v2-
http-mac-03 (work in progress), February 2013.

[I-D.ietf-jose-json-web-encryption]

Jones, M., Rescorla, E., and Hildebrand J., "JSON Web
Encryption (JWE)", draft-ietf-jose-json-web-encryption-11
(work in progress), May 2013.

[I-D.ietf-jose-json-web-signature]

Jones, M., Bradley, J., and Sakimura N., "JSON Web
Signature (JWS)", draft-ietf-jose-json-web-signature-11

(work in progress), May 2013.

[XMLDSig]

Eastlake, D., Reagle, J., Solo, D., Hirsch, F., and
Roessler T., "XML Signature Syntax and Processing (Second
Edition)", W3C Recommendation, June 2008.

[I-D.vial-core-mirror-proxy]

Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-
proxy-01 (work in progress), July 2012.

[I-D.ietf-core-observe]

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

[I-D.ietf-jose-json-web-algorithms]

Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-
json-web-algorithms-11 (work in progress), May 2013.

[NIST-SP-800-38D]

Dworkin, M., "Recommendation for Block Cipher Modes of
Operation: Galois/Counter Mode (GCM) and GMAC", NIST
Special Publication 800-38D, November 2007.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI
36, RFC 4949, August 2007.

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

[RFC5878] Brown, M. and R. Housley, "Transport Layer Security (TLS)
Authorization Extensions", RFC 5878, May 2010.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand
Key Derivation Function (HKDF)", RFC 5869, May 2010.

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

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally
Unique Identifier (UUID) URN Namespace", RFC 4122, July
2005.

[RFC4279] Eronen, P., Ed., and H. Tschofenig, Ed., "Pre-Shared Key
Ciphersuites for Transport Layer Security (TLS)",

RFC 4279, December 2005.

Authors' Addresses

Goeran Selander
Ericsson
Farogatan 6
16480 Kista
SWEDEN

EMail: goran.selander@ericsson.com

Mohit Sethi
Ericsson
Hirsalantie 11
02420 Jorvas
FINLAND

EMail: mohit.m.sethi@ericsson.com

Ludwig Seitz
SICS Swedish ICT AB
Scheelevagen 17
22370 Lund
SWEDEN

EMail: ludwig@sics.se

CoRE
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.

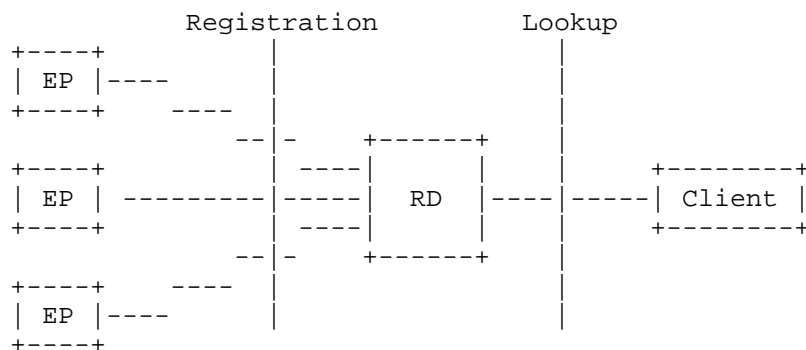


Figure 1: The resource directory architecture.

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

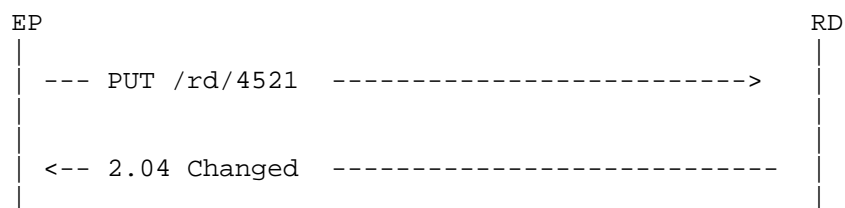
The following response codes are defined for this interface:

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.

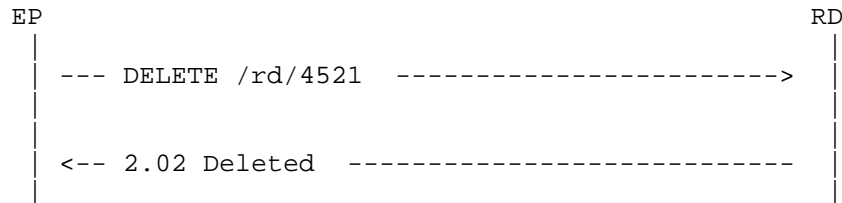
The following responses codes are defined for this interface:

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 | RD
| |
| |----- GET /rd-lookup/res?rt=temperature ----->
| |
| |<-- 2.05 Content "<coap://node1/temp>;rt="temperature" ----
```

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

```
Res: 2.05 Content
<coap://{ip:port}/temp>
```

The following example shows a client performing an endpoint lookup:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/ep?et=power-node ----->      |
|                                                         |
|<-- 2.05 Content "<coap://{ip:port}>;ep="node5" -----|
|                                                         |

```

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

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

The following example shows a client performing a domain lookup:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/d ----->                      |
|                                                         |
|<-- 2.05 Content "</rd>;d=domain1,</rd>;d=domain2 -----|
|                                                         |

```

Req: GET /rd-lookup/d

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

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

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/gp ----->                    |
|                                                         |

```

```
<-- 2.05 Content </rd-group/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: January 16, 2014

B. Silverajan
Tampere University of Technology
T. Savolainen
Nokia
July 15, 2013

CoAP Communication with Alternative Transports
draft-silverajan-core-coap-alternative-transport-02

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. UDP remains the optimal solution for CoAP use in IP-based constrained environments and nodes. However the need for M2M communication using non-IP networks, improved transport level end-to-end reliability and security, NAT and firewall traversal issues, and mechanisms possibly incurring a lower overhead to CoAP/HTTP translation gateways 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 January 16, 2014.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
1.1. Using Alternative Transports	3
1.2. Use Cases	3
2. CoAP Transport URI	5
3. Alternative Transport Analysis and Properties	6
3.1. Other Considerations	9
4. IANA Considerations	9
5. Security Considerations	9
6. Acknowledgements	9
7. Informative References	10
Appendix A. Expressing transport in the URI in other ways . . .	12
A.1. Transport in URI path or query component	12
A.2. Transport in the URI authority component	13
A.3. Transport as part of a 'service:' URL scheme	13
Authors' Addresses	13

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 providing a request/response model that constrained nodes can use to communicate with other nodes, be those servers, proxies, gateways, less constrained nodes, or other constrained nodes.

As the Internet continues taking shape by integrating new kinds of networks, services and devices, the need for a consistent, lightweight method for resource representation, retrieval and manipulation becomes evident. Owing to its simplicity and low overhead, CoAP is a highly suitable protocol for this purpose. However, the CoAP endpoint can reside in a non-IP network, be separated from its peer by NATs and firewalls or simply has no possibility to communicate over UDP. Consequently in addition to UDP, alternative transport channels for conveying CoAP packets could be considered.

This document looks at how CoAP can be used by nodes for resource retrieval, in an end-to-end manner regardless of the transport channel available. It looks at current usage of CoAP in this regard

today and provides other possible scenarios. Then the document looks at how resources using CoAP can contain resource information that provide endpoint as well as transport identifiers, without imposing incompatibilities with [I-D.ietf-core-coap] and maintaining conformance to [RFC3986].

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.

1.1. Using Alternative Transports

Extending CoAP's REST-based usage over alternative transports allows CoAP 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 channels. As opposed to implementing new resource retrieval schemes, an application in an end-node can continue relying on using CoAP for this purpose, but lets CoAP take into account the change in end point identification and transport protocol. This simplifies development and memory requirements. Resource representations are also visible in an end-to-end manner for any CoAP client.

Inevitably, if two CoAP endpoints reside in distinctly separate networks with orthogonal transports, a CoAP proxy node is needed between the 2 networks so that CoAP Requests and Responses can be fulfilled properly. The processing and computational overhead for conveying CoAP packets from one underlying transport to another, would be less than that of an application-level gateway performing individual packet-based, protocol translation between CoAP to another resource retrieval scheme.

1.2. Use Cases

CoAP has been designed to work on top of UDP, that is, on top of a transport that can lose, reorder, and duplicate packets. UDP has been chosen as the transport protocol over IP due its lightweight nature and connectionless characteristics. In addition to point-to-point communication, this allows multicast and group communications [I-D.ietf-core-groupcomm]. Moreover, DTLS can be employed to secure CoAP communication.

At this time of writing, the use of CoAP is also being specified for other environments as follows:

1. CoAP Request and Response messages can be sent via SMS or USSD between CoAP end-points in a cellular network [I-D.becker-core-coap-sms-gprs]. A CoAP Request message can also be sent via SMS from a CoAP client to a sleeping CoAP Server as a wake-up mechanism for subsequent communication via GPRS. The Open Mobile Alliance (OMA) specifies both UDP and SMS as transports for M2M communication in cellular networks. The OMA Lightweight M2M protocol being drafted uses CoAP, and as transports, specifies both UDP binding as well as Short Message Service (SMS) bindings [OMALWM2M] for the same reason.
2. The WebSocket protocol is being used as a transport channel between WebSocket enabled CoAP end-points on the Internet [I-D.savolainen-core-coap-websockets]. This is particularly useful as a means for web browsers, particularly in smart devices, to allow embedded client side scripts to upgrade an existing HTTP connection to a WebSocket connection through which CoAP Request and Response messages can be exchanged with a WebSocket-enabled server. This also allows a browser containing an embedded CoAP server to behave as a WebSocket client by opening a connection to a WebSocket enabled CoAP Mirror Server to register and update its resources.
3. [I-D.jimenez-p2psip-coap-reload] specifies how CoAP nodes can use a peer-to-peer overlay network called RELOAD, as a resource caching facility for storing wireless sensor data. When a CoAP node registers its resources with a RELOAD Proxy Node (PN), the node computes a hash value from the CoAP URI and stores it as a structure together with the PN's Node ID as well as the resources. Resource retrieval by CoAP nodes is accomplished by computing the hash key over the Request URI, opening a connection to the overlay and using its message routing system to contact the CoAP server via its PN.

We also envisage CoAP being extended atop other transport channels, such as:

1. Using TCP to facilitate the traversal of CoAP Request and Response messages. This allows easier communication between CoAP clients and servers separated by firewalls and NATS. This also allows CoAP messages to be transported over push notification services from a notification server to a client app on a smartphone, that may previously have subscribed to receive change notifications of CoAP resource representations, possibly by using CoAP Observe-functionality [I-D.ietf-core-observe].

2. The transportation of CoAP messages in Delay-Tolerant Networks [RFC4838], using the Bundle Protocol [RFC5050] for reaching sensors in extremely challenging environments such as acoustic, underwater and deep space networks.
3. Any type of non-IP networks supporting constrained nodes and low-energy sensors, such as Bluetooth and Bluetooth Low Energy (either through L2CAP or with GATT), ZigBee, Z-Wave, 1-Wire, DASH7 and so on.
4. Instant Messaging and Social Networking channels, such as Jabber and Twitter.

2. 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. For a URI that is expressed generically as

URI = scheme ":" "/" authority path-abempty ["?"query]

A simple example COAP URI, "coap://server.example.com/sensors/temperature" can be interpreted as follows:

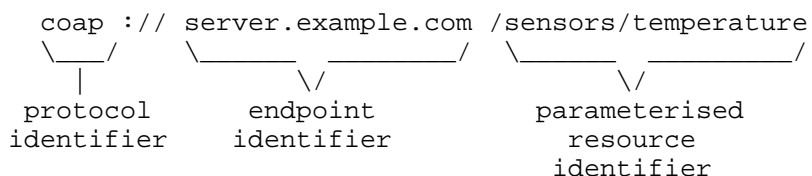


Figure 1: The CoAP URI format

The resource path is explicitly expressed, and the endpoint identifier, which contains the host address at the network-level is also directly bound to the scheme name containing the application-level protocol identifier. The choice of a specific transport for a scheme, however, cannot be embedded with a URI, but is defined by convention or standardisation of the protocol using the scheme. As examples, [RFC5092] defines the 'imap' scheme for the IMAP protocol over TCP, while [RFC2818] requires that the 'https' protocol

identifier be used to differentiate using HTTP over TLS instead of TCP.

To express an alternative transport binding to CoAP, a scheme name can follow a convention of the form "coap+<transport-name>", where the name of the transport is clearly and unambiguously described. Each scheme name formed in this manner can be used to differentiate the use of CoAP over an alternative transport instead of the use of CoAP over UDP or DTLS. The endpoint identifier, path and query components together with each scheme name would be used to uniquely identify each resource.

Examples of such URIs are:

- o coap://server.example.com/sensors/temperature for using CoAP over UDP
- o coaps://server.example.com/sensors/temperature for using CoAP over DTLS
- o coap+tel://+15105550101/sensors/temperature for using CoAP over SMS or USSD with the endpoint identifier being a telephone subscriber number
- o coap+ws://www.example.com/WebSocket?/.well-known/core?rt=core.ms for using CoAP over WebSockets with the endpoint at ws://www.example.com/WebSocket
- o coap+ble.l2cap://[12:34:56:78:90:AB]:4/pulse where the scheme name can possibly be used to describe the future use of CoAP over L2CAP using Bluetooth Low Energy, but not L2CAP using classic Bluetooth.

When such a URI is provided from an end-application to its CoAP implementation, the scheme name can be checked to allow the CoAP to use the appropriate transport for the specified endpoint identifier. The CoAP Transport URI can also be supplied as a Proxy-Uri option by a CoAP end-point to a CoAP forward proxy in order to communicate with a CoAP end-point residing in a network using a different transport. Section 6.4 of [I-D.ietf-core-coap] provides an algorithm for parsing the received URI to obtain the request's options.

3. Alternative Transport Analysis and Properties

In this section we consider the various characteristics of alternative transports for successfully supporting various kinds of functionality for CoAP. CoAP factors lossiness, unreliability, small packet sizes and connection statelessness into its protocol logic. We discuss general transport differences and their impact on carrying CoAP packets here. Note that Properties 1, 2, and 3 are related.

Property 1: Uniqueness of an end-point identifier.

Transport protocols providing non-unique end-point IDs for nodes may only convey a subset of the CoAP functionality. Such nodes may only serve as CoAP servers that announce data at specific intervals to a pre-specified end point, or to a shared medium.

Property 2: Unidirectional or bidirectional CoAP communication support.

This refers to the ability of the CoAP end-point to use a single transport channel for both request and response messages. Depending on the scenario, having a unidirectional transport layer would mean the CoAP end-point might utilise it only for outgoing data or incoming data. Should both functionalities be needed, 2 unidirectional transport channels would be necessary.

Property 3: 1:N communication support.

This refers to the ability of the transport protocol to support broadcast and multicast communication. CoAP's request/response behaviour depends on unicast messaging. Group communication in CoAP is bound to using multicasting. Therefore a protocol such as TCP would be ill-suited for group communications using multicast. Anycast support, where a message is sent to a well defined destination address to which several nodes belong, on the other hand, is supported by TCP.

Property 4: Transport-level reliability.

This refers to the ability of the transport protocol to provide a guarantee of reliability against packet loss, ensuring ordered packet delivery and having error control. When CoAP Request and Response messages are delivered over such transports, the CoAP implementations elide certain fields in the packet header. As an example, if the usage of a connection-oriented transport renders it unnecessary to specify the various CoAP message types, the Type field can be elided. For some connection-oriented transports, such as WebSockets, the version of CoAP being used can be negotiated during the opening transfer. Consequently, the Version field in CoAP packets can also be elided.

Property 5: Message encoding.

While parts of the CoAP payload are human readable or are transmitted in XML, JSON or SenML format, CoAP is essentially a low overhead binary protocol. Efficient transmission of such packets would therefore be met with a transport offering binary encoding support, although techniques exist in allowing binary payloads to be transferred over text-based transport protocols such as base-64 encoding. A fuller discussion about performing CoAP message encoding for SMS can be found in Appendix A.5 of [I-D.bormann-coap-misc]

Property 6: Network byte order.

CoAP, as well as transports based on the IP stack use a Big Endian byte order for transmitting packets over the air or wire, while transports based on Bluetooth and Zigbee prefer Little Endian byte ordering for packet fields and transmission. Any CoAP implementation that potentially uses multiple transports has to ensure correct byte ordering for the transport used.

Property 7: MTU correlation with CoAP PDU size.

Section 4.6 of [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.

Property 8: Transport latency.

A confirmable CoAP request would be retransmitted by a CoAP end-point if a response is not obtained within a certain time. A CoAP end-point registering to a Resource Directory uses a POST message that could include a lifetime value. A sleepy end-point similarly uses a lifetime value to indicate the freshness of the data to a CoAP Mirror Server. Care needs to be exercised to ensure the latency of the transport being used to carry CoAP packets is small enough not to interfere with these values for the proper operation of these functionalities.

3.1. Other Considerations

This section outlines miscellaneous considerations concerning transport bindings with the CoAP URI.

1. A CoAP endpoint using a connection-oriented transport should be responsible for proper connection establishment prior to sending a CoAP Request message. Both communicating endpoints may monitor the connection health during the Data Transfer phase. Finally, once data transfer is complete, at least one end point should perform connection teardown gracefully.
2. A CoAP server, such as a sensor, may make its data available over multiple types of transports concurrently. For example, this can be done to allow the value to be retrieved via UDP as well as TCP. However, this could be carried out only when necessary to avoid a resource being identified by more than one URI. On the other hand, when using only a single underlying transport, URI aliasing should not be practised [WWWArchv1]. For some scenarios where there is an availability of DNS for lookups as well as updates, SRV records can be used. In these cases, the "_service" field can be "coap", and the "_proto" field carries the name of the transport to be used.
3. As the usage of each alternative transport results in an entirely new scheme, IANA intervention is required for the registration of each scheme name. The registration process follows the guidelines stipulated in [RFC4395], particularly where permanent URI scheme registration is concerned.

4. IANA Considerations

This memo includes no request to IANA.

5. 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.

6. Acknowledgements

Discussions with Klaus Hartke, Graham Klyne, Markus Becker and Golnaz Karbaschi provided useful insights and ideas for this draft.

7. 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.bormann-coap-misc]
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-25 (work in progress), May 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., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-10 (work in progress), July 2013.
- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.
- [I-D.jimenez-p2psip-coap-reload]
Jimenez, J., Lopez-Vega, J., Maenpaa, J., and G. Camarillo, "A Constrained Application Protocol (CoAP) Usage for REsource LOcation And Discovery (RELOAD)", draft-jimenez-p2psip-coap-reload-03 (work in progress), February 2013.
- [I-D.savolainen-core-coap-websockets]
Savolainen, T., Hartke, K., and B. Silverajan, "CoAP over WebSockets", draft-savolainen-core-coap-websockets-00 (work in progress), July 2013.

- [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.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2609] Guttman, E., Perkins, C., and J. Kempf, "Service Templates and Service: Schemes", RFC 2609, June 1999.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [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.
- [RFC5092] Melnikov, A. and C. Newman, "IMAP URL Scheme", RFC 5092, November 2007.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.
- [RFC6568] Kim, E., Kaspar, D., and JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6568, April 2012.
- [RFC6733] Fajardo, V., Arkko, J., Loughney, J., and G. Zorn, "Diameter Base Protocol", RFC 6733, October 2012.
- [WWWArchv1]
<http://www.w3.org/TR/webarch/#uri-aliases>, "Architecture of the World Wide Web, Volume One ", December 2004.

Appendix A. Expressing transport in the URI in other ways

Other means of indicating the transport as a distinguishable component within the CoAP URI are possible, but have been deemed unsuitable by working group consensus or because they violate [RFC3986], and are incompatible with existing practices outlined in [I-D.ietf-core-coap]. They are however, retained in this section for historical documentation and completeness.

A.1. 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 [RFC6733] 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;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.

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 or a "tt" link target attribute may then need to be registered.

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.

A.2. 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.

A.3. Transport as part of a 'service:' URL scheme

The "service:" URL scheme name was introduced in [RFC2609] and forms the basis of service description used primarily by the Service Location Protocol. An abstract service type URI would have the form

```
"service:<abstract-type>:<concrete-type>"
```

where <abstract-type> refers to a service type name that can be associated with a variety of protocols, while the <concrete-type> then providing the specific details of the protocol used, authority and other URI components.

Adopting the "service:" URL scheme to describe CoAP usage over alternative transports would be rather trivial. To use a previous example, a CoAP service to discover a Resource Directory and its base RD resource using TCP would take the form

```
service:coap:tcp://host.example.com/.well-known/core?rt=core-rd
```

The syntax of the "service:" URL scheme differs from the generic URI syntax and therefore such a representation should be treated as an opaque URI as Section 2.1 of [RFC2609] recommends.

Authors' Addresses

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi

Teemu Savolainen
Nokia
Hermiankatu 12 D
FI-33720 Tampere
Finland

Email: teemu.savolainen@nokia.com

CoRE Working Group
Internet-Draft
Intended status: Standard Track
Expires: January 14, 2014

Seok-Kap Ko
Ilkyun Park
Seung-Hun Oh
Byung-Tak Lee
ETRI
July 14, 2013

ASCII Encoding for Constrained Application Protocol (CoAP/A)
draft-softgear-core-coapa-00.txt

Abstract

This document defines ASCII encoding method for Constrained Application Protocol(CoAP). CoAP has defined a binary encoding method to exchange requests and responses between constrained nodes or gateways. However, some communication modules in constrained nodes support ASCII data only. These modules are cheaper and widely deployed. To support these modules, this document describes the method to convert binary CoAP messages to base64 styled ASCII messages.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 14, 2014.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	4
2. Terminology	4
3. Base64 encoding for CoAP	5
3.1. Overview	5
3.2. Examples	6
4. Security Considerations	7
5. IANA Considerations	7
6. Acknowledgments	8
7. References	8
7.1. Normative References	8

1. Introduction

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks. Current CoAP draft document [COAP] describes a binary encoding method for exchanging requests and response between nodes. Binary encoding is compact and fast. However, in real deployment environment, many constrained nodes use ASCII encoding only. Many ZigBee communication modules and WiFi communication modules for embedded devices allow ASCII characters only. Some modules can send a limited size (about 50 bytes) message at a time. Because these communication modules are cheaper and widely deployed, ASCII version of CoAP is required.

To support ASCII encoding, an approach to design new encoding method may be inefficient. Therefore, this document reuses Base64 encoding/decoding. Binary CoAP messages are converted to ASCII messages before transmission. ASCII encoded messages are converted to binary CoAP messages again after receiving via communication channel.

This document replaces some characters in the standard Base64 character set because of escape characters, for example, "+" character. To support fragmentation and reassembly if a communication module allows small size of message for transmitting or receiving, this document adds "#" character at end of the each messages.

Because Base64 encoding/decoding logic is very simple, it requires just small code size for it. The code for it could be written with about 120 lines source code.

2. Terminology

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

3. Base64 encoding for CoAP

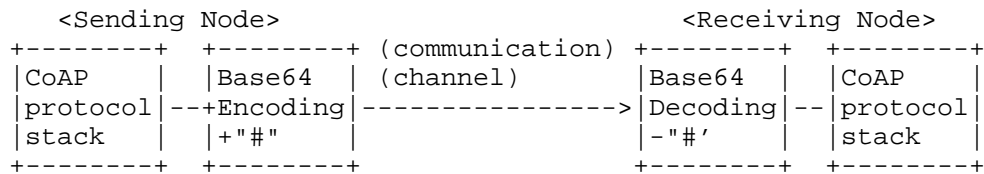
3.1. Overview

A CoAP node MAY consist of the CPU module and the communication module logically or physically. Some communication modules do not support binary encoded messages. To use the communication modules which do not support binary messages, the CPU module in CoAP node SHOULD encode a binary coded CoAP message to an ASCII based text message before passing the message to the communication module. After the communication module passes the message to the CPU module, the CPU module SHOULD decode the ASCII based text message to the binary CoAP message. A CoAP node can use a normal CoAP protocol software stack with this translation.

In this document, we use Base64 encoding[BASE64]. Base64 is a binary-to-text encoding scheme that represent binary data in an ASCII string by a radix-64 representation. There are some variations in Base64 characters. For example, MIME's Base64 uses A-Z, a-z, 0-9, '+', and '/'. And, '=' is used as a padding character. The number of encoded bytes per original bytes is approximately 4/3 (33% overhead). In this document, we use Base64url encoding that '-'(minus) character is used instead of '+'(plus) and '_'(underline) character is used instead of '/' (slash). This is a URL and file name safe version of Base64 variations. And, some communication module may translate '+' as an escape character. Therefore, the '+'(plus) character SHOULD NOT be used for the communication.

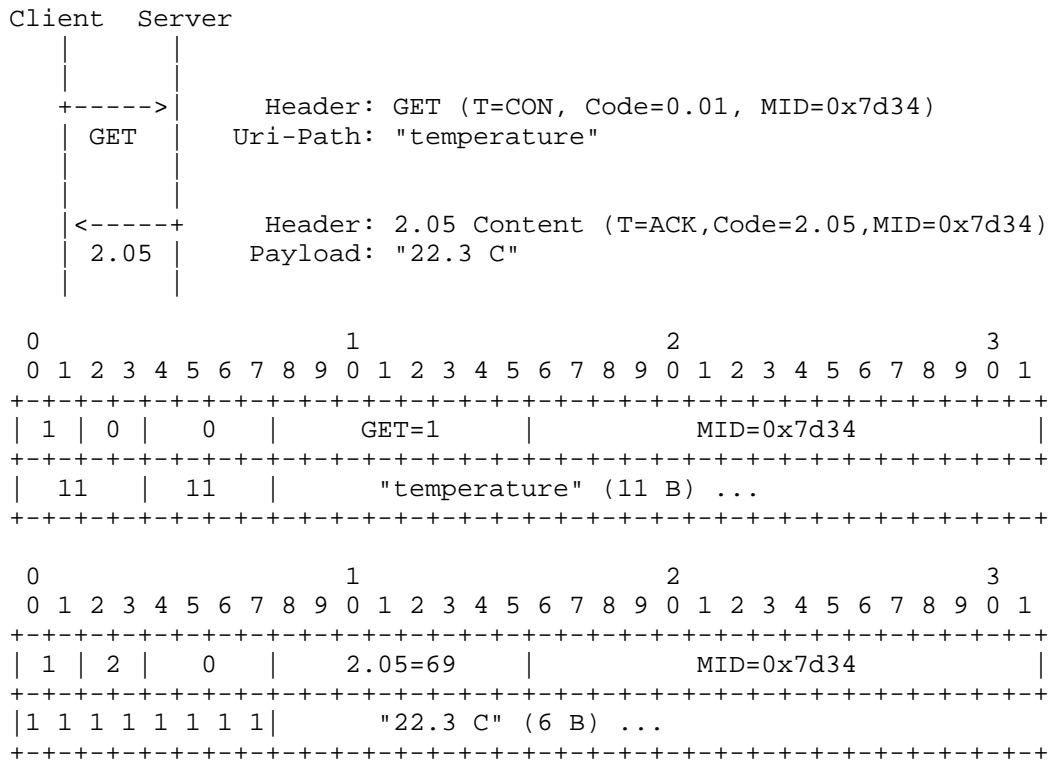
In this document, '#' is used as the end of message. Therefore, the CPU module in CoAP node SHOULD add '#' character at the end of Base64 encoded message. It helps the message fragmentation and reassembly. The sender side MAY send the partial encoded messages. The receiver side SHOULD reassemble the messages until '#' is received. The sender MAY send some white space characters, i.e., carriage return, line feed, space, or tab, between fragmented messages. The receiver SHOULD ignore all characters besides Base64url character set and '=' character.

The following figure shows how ASCII encoding method for Constrained Application Protocol works. The Base64 encoding/decoding software module runs independently from CoAP protocol stack.



3.2. Examples

In the appendix A in CoAP draft-18, a confirmable request and piggy-backed response message example is described.



Original binary encoded CoAP Request:

40 01 7d 34 bb 74 65 6d 70 65 72 61 74 75 72 65

Original binary encoded CoAP Response:

60 45 7d 34 ff 32 32 2e 33 43

These messages are converted to the Base64 encoded messages as the follows.

Text encoded CoAP Request:
QAF9NLt0ZWlwZXJhdHVyZQ==#

This message is written in hex string as the follows.

51 41 46 39 4e 4c 74 30 5a 57 31 77 5a 58 4a 68 64
48 56 79 5a 51 3d 3d 23

Text encoded CoAP Response:
YEV9NP8yMi4zIEM=#

This message is written in hex string as the follows.

59 45 56 39 4e 50 38 79 4d 69 34 7a 49 45 4d 3d 23

4. Security Considerations

TODO - fill in

5. IANA Considerations

TODO - fill in

6. Acknowledgments

TODO - fill in

7. References

7.1. Normative References

- [COAP] Z. Shelby, K. Hartke, C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18, June 28, 2013.
- [BASE64] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings", IETF RFC-4648, October 2006.

Author's Addresses

Seok-Kap Ko
ETRI
176-11 Cheomdan Gwagi-ro, Buk-gu, Gwangju, 500-480,
Korea
Phone: +82-62-970-6677
Email: softgear@etri.re.kr

Ilkyun Park
ETRI
176-11 Cheomdan Gwagi-ro, Buk-gu, Gwangju, 500-480,
Korea
Phone: +82-62-970-6651
Email: ikpark@etri.re.kr

Seung-Hun Oh
ETRI
176-11 Cheomdan Gwagi-ro, Buk-gu, Gwangju, 500-480,
Korea
Phone: +82-62-970-6655
Email: osh93@etri.re.kr

Byung-Tak Lee
ETRI
176-11 Cheomdan Gwagi-ro, Buk-gu, Gwangju, 500-480,
Korea
Phone: +82-62-970-6624
Email: bytelee@etri.re.kr

Core Working Group
Internet Draft
Intended status: Informational
Expires: March 29, 2014

J. Zhu
M. Qi
China Mobile
Sep 29, 2013

Group Authentication
draft-zhu-core-groupauth-01

Abstract

The group communication is designed for the communication of Internet of Things. A threat is identified in [I-D.ietf-core-groupcomm] that current DTLS based approach is unicast oriented and there is no supporting on group authentication feature. Unicast oriented authentication will causing serious burden when a large number of terminal nodes will be involved inevitably. In another aspect, some terminals will own the same characteristics, such as owning same features, in the same place, working in the same time, etc. With this mechanism, all terminals can be authenticated together with little signaling and calculation at the same time. It will reduce the network burden and save time. This draft describes the security of group authentication and an group authentication implementation method for the Internet of things.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on March 29, 2014.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Conventions used in this document	3
2.1. Definitions.....	3
3. Problem Statement	4
3.1. Use cases	4
3.2. Problem statement	5
4. Requirement	6
5. Group Authentication Solution	7
5.1. Introduction	7
5.2. Detailed group scenario description	7
5.3. Group scenario procedure	9
6. Security Considerations	10
7. IANA Considerations	11
8. Conclusions	11
9. Acknowledgement.....	11
10. References	11
10.1. Normative References	11
10.2. Informative References	11

1. Introduction

With the development of Internet of Things, a large number of

terminal nodes will be involved inevitably. The unicast authentication communication from big amount terminals will merge together in the network, and causing serious burden to the server. Although IP multicast technical is introduced for group communication in [I-D.ietf-core-groupcomm], IP multicast relies on the unicast authentication at initial stage. In another aspect, some terminals will own the same characteristics, such as owning same features, in the same place, working in the same time, etc. With this mechanism, all terminals can be authenticated with little signaling and calculation at the same time. It will reduce the network burden and save time.

This draft describes the security of group authentication and an group authentication implementation method for the Internet of things.

2. Conventions used in this document

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

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

2.1. Definitions

Terminals: It is such a constrained device also recognized as group node in this document which communicates with server, through direct or indirect connections, which can be seen by the server. If some constrained devices like Zigbee node only communicates with sink node and it can't be seen by the server, such constrained device is not recognized as terminal.

Group agent: It is a device on behalf of group nodes (terminals) to make mutual authentication with network server.

Network server: It is the core network server which is responsible for authenticating the legality of terminal and provides specific services for them.

3. Problem Statement

3.1. Use cases

Nowadays the normal authentication mechanism in network is a traditional unicast authentication method between a single terminal and a single network entity. The authentication mechanism will be finished based one 1-2 round of challenge-response conversation separately. If there are many terminals, the cost for authentication will be increased.

Now for some M2M service, it may be a large amount of terminal used for an M2M service. These terminals are placed in the same location, will be used for the same purpose, and own same behavior. These terminals can be worked together as a group. In these scenarios, the existing authentication mechanism is no longer appropriate. When a large number of terminals want to access network server, huge number of authentication signaling will be generated by the unicast authentication method. What is more, it will cause network congestion and lead to DoS attack. For some terminal devices which is restricted with limited computing capability and power, the traditional unicast authentication will increase the computational burden of these terminals and drain their poor battery.

The following use cases are identified at this point:

Smart Metering: A large amount of smart power meter terminals are deployed in a block. The smart meter uploads meter report frequently through the network to smart meter server. What is more, smart meter server queries all terminals periodically to check whether the terminal is workable or not. So smart meters report at the same time, or the smart meter server need to re-configure all smart meters at the same time. In fact, there are other type of smart metering which has no agent node. This will lead to overload since each apartment needs to equip with a meter and they access network parallel at the same time and it will not be considered in this draft.

Remote Vehicle Management: IOT terminals contains GPS location reporter, remote air condition control, etc. would be installed in some special Vehicles like Taxi. It will send information such as position information, navigation, remote diagnosis, on-board communication, news and

entertainment information etc. to the network server in order to make better vehicle scheduling, vehicle monitoring and vehicle controlling. So it needs to connect to the network and make authentication at first. However, such vehicles would gather in a small place like airport, train station, etc. The frequency of connection from these terminals to server will cause overloading.

Intelligent home: various sensors equipped with communication modules are deployed in a house to monitor house conditions and make a control when necessary. These sensors collect and report house related information like the status of door open/close, indoor temperature to its owner through a network, and take actions by following the regulating instructions send by the owner.

3.2. Problem statement

In the current smart metering service use cases, a large amount of smart power meter terminals are deployed in a block. The smart meter uploads meter report frequently through the network to smart meter server. What is more, smart meter server queries all terminals periodically to check whether the terminal is workable or not. Therefore, the meter requires frequent and network communication.

In such use cases, when all the meters access network parallel at the same time, or when the server sends message to all meters, the terminals will connect to the network in a short time period (1sec ~ 1min). Assume there are 19 buildings in the block, and each building has 25 floors on average with 10 apartments in each floor. If each apartment is equipped with 1 smart power meter, then 4760 meters will be deployed in total in the block. This will cause pressure to the network.

So an agent node has been introduced to aggregate the message from these meters and then send out these meters data to the server together. After the agent is introduced, the connection between meters and servers is split into two parts: one is the connection between meters, the other is and the one between agent and server. Usually the agent is responsible for the authentication of the meters.

The server is responsible for the authentication of the agent only and gets all information about meters such as ID, data, from agent.

The current security mechanism is:

1. Each meter is authenticated with the agent. Agent will authenticate the meter one by one. After that, agent should make mutual authentication with server. Then server can confirm agent identity.
2. Meter will set up security connection with agent, and agent will also set up security connection with server. When a meter wants to send data to server. It should send the data confidential protected to agent first. Agent will decrypt the data and transfer it to server by using the security protection mechanism between agent and server.

However, this procedure has the following security problems:

1. Since all meters are authenticated by the agent and no direct authentication from server to meter. The server can get meter's ID and data only through agent. So the agent Due to the key position in the authentication, the security protection about agent is very important. Server could not authenticate meters directly. It can only rely on the agent. However, the agent would be placed in un-secure place or owned by different user rather than the server owner. If the agent is compromised or lay to server, agent can act as a middle attacker that makes fake authentication to meters and report fake ID to servers.
2. Another security problem is related with agent and server. Under this scenario, all information from meters will be transferred through agent. So agent will know all information generated by meters. However, under some scenario, agent would be owned and used by different user other than the meters' and servers' owner. So under this assumption, the agent should not get the message from meter to server. So meters should set up an secure end-to-end tunnel with server. It should request another authentication and key generation procedure in addition to authenticate with agent. This will bring complexity and overhead to the system.

4. Requirement

In order to reduce the cost and simplify a lot of overhead with the same characteristics of these groups of meter or sensor node group-based operations, it is needed to provide group authentication. For example, when smart meters perform bulk configuration information updates, it is needed to ensure that the correct identity of the user node within the group, to prevent the configuration information is wrong node receives. In addition, when smart meters report meter readings to the electricity system platform, it is also needed to be able to prove the correctness of the identity of smart meters, to prevent malicious node reporting false readings.

5. Group Authentication Solution

5.1. Introduction

Group authentication is a kind of authentication technologies that a group of users or terminals can be authenticated together at the same time. Instead of authenticating a number of terminals of a group one by one, group authentication mechanism treats these terminals in the group as a whole, and authenticates them together. Each group has a unique identifier, and an agent, which can be called as group agent, group gateway, etc.

Group authentication comprises following two phases as following:

1. The first phase is that user/terminal should be authenticated whether it belongs to a given group. This can be implemented through the proprietary authentication technology in a group, such as Zigbee or any others.
2. The second phase is that mutual authentication should be made between a given network entity, and a group agent who is responsible to delegate all terminals in the group.

After the authentication, terminals and network entity can generate separated session keys individually if there is some demand to make individual communication between network entity and each terminal.

5.2. Detailed group scenario description

For group authentication, there is detailed network description as following. There are 5 nodes inside a given group. They are A1, A2, A3, A4, and A5 which is group agent. And the given group can be named as group A. All nodes in group A can communicate with each other. What is more, A5 is able to communicate with network entity directly. Network entity will store the group information, such as identifiers, root keys used for all nodes inside the group. Network entity is also responsible for generating group authentication vector. The scenario is shown as below.

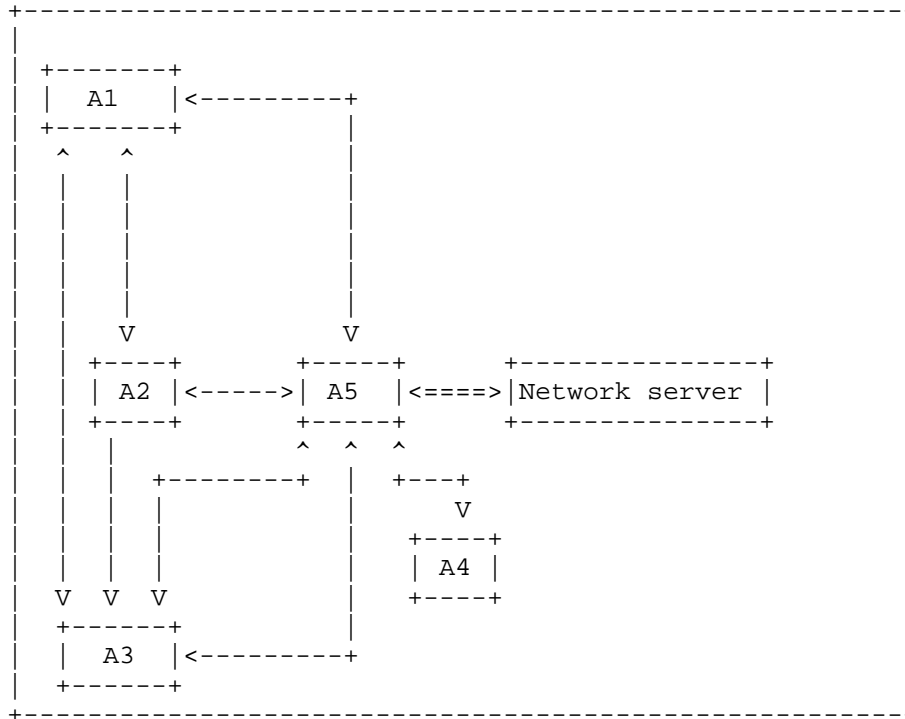


Figure 1 Group Authentication Architecture

- o A5 (group agent) communicates with other nodes, i.e. A1, A2, A3, A4 by inner group protocol. All nodes should contain such models as inner group communication model, group authentication mode. Inner group communication model can be used to sending/receiving

the group authentication message. Group authentication model can be used to generate authentication vectors/response and to authenticate peers.

- o Group agent will make mutual authentication with network entities. There are two kinds of network entities. Network server is responsible for mutual authentication action with group agent. And Network Server is responsible for group authentication vector generation and forwarding AV to network server. After the authentication, terminals and network entity can generate separated session keys individually if there is some demand to make individual communication between network entity and each terminals.

- o Group agent who represents the whole group, communicates with network entity, and generate group session key through authentication with the network server.

- o Pre-configure of the group

All the group nodes should be configured with sub key k_1 , k_2 , k_3 , k_4 , k_g , which will be used for mutual authentication in the group and separated communication.

5.3. Group scenario procedure

As mentioned above, group authentication can be divided into two phases.

In the first phase, group member, say A_i , sends authentication request to group agent at first as following.

1. Group member A_i sends message to trigger authentication at first.
2. Group agent sends authentication request to each group member.
3. Group member A_i verifies group agent at first. If success, A_i will generate session key for the communication with group agent, and sends response containing such session key back to group agent. If not success, the authentication is failed and group authentication procedure will be abort.
4. Group agent authenticates each group member A_i through the response message and record the authentication result in a mapping

table.

After the inner group authentication, all of group members are authenticated by group agent, and second phase can be performed.

5. Group agent sends message to network server to trigger the authentication outside the group.
6. Meanwhile, group agent sends authentication vector request to network server with group agent identity.
7. Network Server will generate authentication vector according to group agent identity.
8. What is more, network server should be able to recognize that is a group authentication is performed based on group agent identity. Network Server will generate session key for each group members by using pre-configured group member information and the same keying material in above step.
9. Network Server will send such authentication vector and session keys together back to network server.
10. Network server will perform mutual authentication with group agent.
11. Group agent authenticates group agent and send authentication response back to network server.
12. Network server authenticates group agent. If success, it can be considered that group agent and all group terminals is authenticated successfully.
13. Group agent will communicate with network server to choose the confidential and integrity protection algorithms.
14. After that, group agent will send keying material, selected algorithms to each group member.
15. Group member will generate session keys.

After these two phases, each terminal is authenticated with network server and generate independently session key with network server.

6. Security Considerations

TBD

7. IANA Considerations

There are no IANA considerations associated to this memo.

8. Conclusions

This memo describes the problem raised by using one-to-one authentication for huge number of Internet of Things terminals.

After that, group authentication requirement is raised and a group authentication mechanism is proposed. By using the proposed group authentication mechanism, the exploited group agent can behalf of all involved group nodes to make mutual authentication with network server, but the agent can not realize the content transmitted between both of them since it is just a intermediate node to forward messages which are encrypted by specific session key between the group node and network server. The trust relationship is between group nodes and network server. After authentication, the trust relationship between group nodes and group agent are not needed.

9. Acknowledgement

Thanks very much to Bert Greevenbosch, Stefanie Gerdes, Kepeng Li for their helpful comments and significant suggestions to revise this document.

10. References

10.1. Normative References

10.2. Informative References

Authors' Addresses

Judy Zhu
China Mobile
Unit 2, 32 Xuanwumenxi Ave,
Xicheng District,
Beijing 100053, China
Email: zhuhongru@chinamobile.com

Minpeng Qi
China Mobile
Unit 2, 32 Xuanwumenxi Ave,
Xicheng District,
Beijing 100053, China
Email: qiminpeng@chinamobile.com