

CoRE
Internet-Draft
Intended status: Standards Track
Expires: August 24, 2017

K. Kuladinithi, Ed.
ComNets, Hamburg University of Technology
M. Becker
Tridonic GmbH & Co KG
K. Li
Alibaba Group
T. Poetsch
New York University Abu Dhabi
February 20, 2017

Transport of CoAP over SMS
draft-becker-core-coap-sms-gprs-06

Abstract

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

Status of This Memo

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 24, 2017.

Copyright Notice

Copyright (c) 2017 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. Motivation	3
1.2. Terminology	3
1.3. Requirements Language	4
2. Scenarios	4
2.1. MO-MT Scenarios	4
2.2. MT Scenarios	4
2.3. MO Scenarios	5
3. Message Exchanges	6
3.1. Message Exchange for SMS in a Cellular-To-Cellular Mobile-Originated and Mobile-Terminated Scenario	6
4. Encoding Schemes of CoAP for SMS transport	7
5. Message Size Implementation Considerations	7
6. Addressing	8
7. Options	8
7.1. New Options for mixed IP operation.	8
8. URI Scheme	9
9. Transmission Parameters	9
10. Multicast	9
11. Security Considerations	9
12. IANA Considerations	10
12.1. CoAP Option Number	10
12.2. URI Scheme Registration	10
13. References	10
13.1. Normative References	10
13.2. Informative References	11
Appendix A. SMS encoding	12
A.1. ASCII-optimized SMS encoding	12
Appendix B. Changelog	16
Acknowledgements	17
Contributors	17
Authors' Addresses	17

1. Introduction

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

1.1. Motivation

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

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

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

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

1.2. Terminology

This document uses the following terminology:

CoAP Server and Client

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

Mobile Station (MS)

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

1.3. 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].

2. Scenarios

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

2.1. MO-MT Scenarios

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

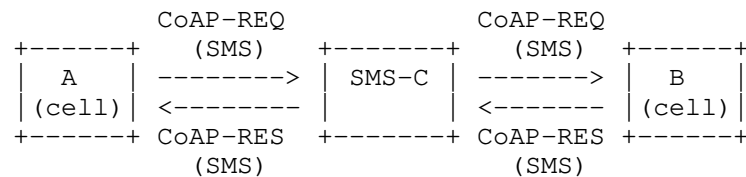


Figure 1: Cellular and Cellular Communication (only SMS-based)

2.2. MT Scenarios

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

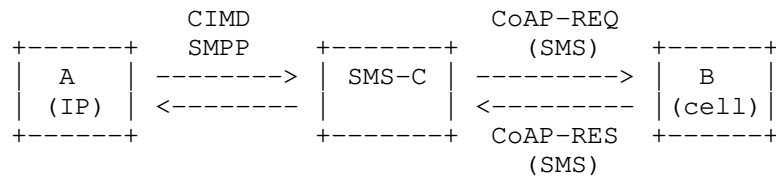


Figure 2: IP and Cellular Communication

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

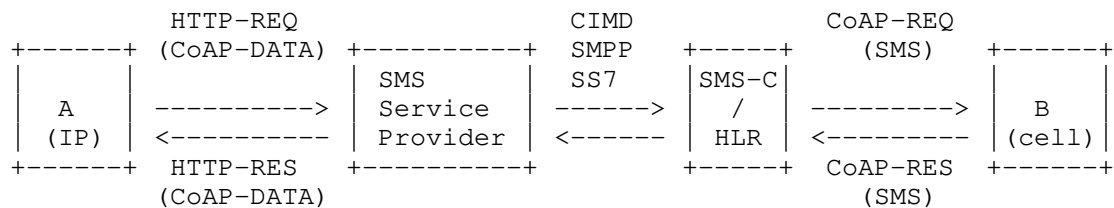


Figure 3: IP and Cellular Communication (using an SMS Service Provider)

2.3. MO Scenarios

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

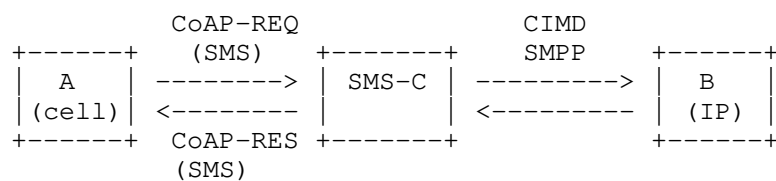


Figure 4: Cellular and IP Communication

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

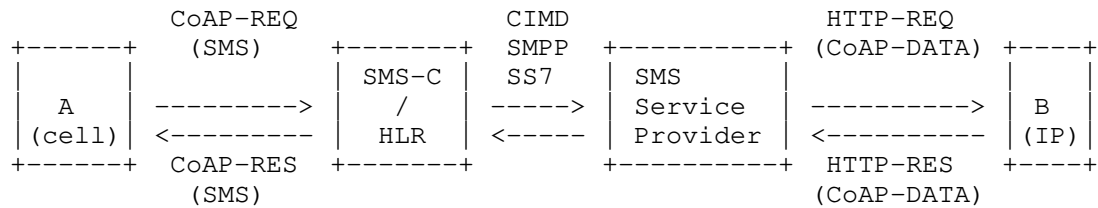


Figure 5: IP and Cellular Communication (using an SMS Service Provider)

3. Message Exchanges

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

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

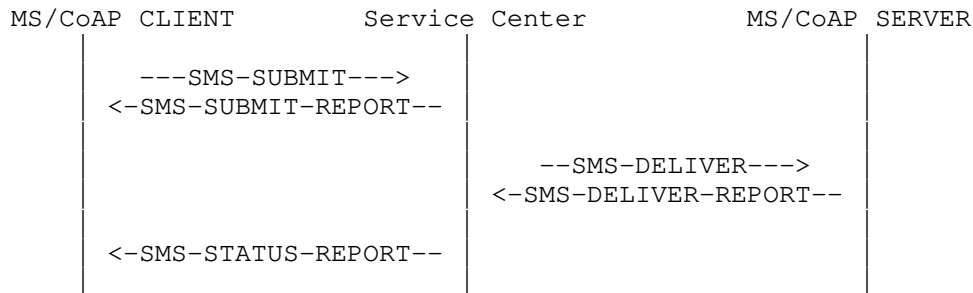


Figure 6: CoAP Messages over SMS

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

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

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

Step 3: The Service Center stores the received SMS message and forwards it to the CoAP Server, using an SMS-DELIVER message. The

CoAP Client address is specified as a TP Originating Address (see [ts23_040]).

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

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

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

4. Encoding Schemes of CoAP for SMS transport

Short messages can be encoded by using various alphabets: GSM 7 bit default alphabet ([ts23_038]), 8 bit data alphabet, and 16 bit UCS2 data alphabet ([iso_ucs2]). These encodings lead to message sizes of 160, 140, and 70 characters, respectively. Whereas the support of 7 bit encoding is mandatory on a MS, the two other encodings are dependent on the language that needs to be encoded, e.g. UCS2 for Arabic, Chinese, Japanese, etc. Furthermore, the supported encoding highly depends on the implementations of the MS itself.

According to [ts23_038], GSM 7 bit encoding shall be supported by all MSs offering SMS services. Since not all MSs support 8 bit short message encoding, the preferred encoding scheme for CoAP messages over SMS is therefore 7 bit, e.g. Base64 ([RFC4648]) or SMS encoding in Appendix A.1.

More considerations about SMS encoding can be found in Appendix A.

5. Message Size Implementation Considerations

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

Possible options for larger CoAP messages are:

Concatenated short messages

Most MSs are able to send concatenation short messages in order to transmit longer messages. The total length of a concatenated short message can consist of up to 255 single messages and result

in total length of 39015 7 bit characters or 34170 bytes.
 Resulting from this, the maximum length of each individual message reduces to 153 (160 - 7) characters (133 bytes).

CoAP block-wise transfer

According to [RFC7959], the Block Size (SZX) of block-wise transfer in CoAP is represented as a three-bit unsigned integer. Thus, the possible block sizes are to the power of two. (Block size = $2^{(SZX + 4)}$). Due to the limitations of 160 characters (140 bytes) for one short message, the maximum value of SZX is 3 (Block size = 128 byte).

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

6. 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.

7. Options

7.1. New Options for mixed IP operation.

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

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

Table 1: New CoAP Option Numbers

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

The options SHOULD NOT be used in the response.

The options MUST NOT occur more than once.

8. URI Scheme

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

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

Example of such URI :

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

In the URI, 0015105550101 is a telephone subscriber number.

9. Transmission Parameters

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

10. Multicast

Multicast is not possible with SMS transports.

11. 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.

12. IANA Considerations

12.1. CoAP Option Number

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

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

12.2. URI Scheme Registration

According to [I-D.silverajan-core-coap-alternative-transports] this document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+sms". The registration request complies with [RFC7595].

13. References

13.1. Normative References

- [etsi_ts101_748] ETSI, "Technical Report: Digital cellular telecommunications system; Abbreviations and acronyms (GSM 01.04 version 8.0.0 release 1999)", 2000.
- [iso_ucs2] ISO, "ISO/IEC10646: "Universal Multiple-Octet Coded Character Set (UCS)"; UCS2, 16 bit coding.", 2000.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<http://www.rfc-editor.org/info/rfc7595>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<http://www.rfc-editor.org/info/rfc7959>>.
- [ts23_038] ETSI 3GPP, "Technical Specification: Alphabets and language-specific information (3GPP TS 23.038 version 11.0.0 Release 11)", 2012.

13.2. Informative References

- [cimd] Nokia, "CIMD Interface Specification (SMSCDOC8000.00, Nokia SMS Center 8.0)", 2005.
- [I-D.silverajan-core-coap-alternative-transport] Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transport-09 (work in progress), December 2015.
- [oma_lightweightm2m_ts] OMA, "Lightweight Machine to Machine Technical Specification", 2013.
- [RFC1924] Elz, R., "A Compact Representation of IPv6 Addresses", RFC 1924, DOI 10.17487/RFC1924, April 1996, <<http://www.rfc-editor.org/info/rfc1924>>.
- [smpp] SMPP Developers Forum, "Short Message Peer to Peer Protocol Specification v3.4 Issue 1.2", 1999.
- [ts23_040] 3GPP, "Technical realization of the Short Message Service (SMS)", 3GPP-23.040 a00, March 2011.

- [ts23_682] ETSI 3GPP, "Technical Specification Group Services and System Aspects; Architecture Enhancements to facilitate communications with Packet Data Networks and Applications; (Release 11)", 2012.
- [ts23_888] ETSI 3GPP, "Technical Specification Group Services and System Aspects; System Improvements for Machine-Type Communications; (3GPP TR 23.888 version 1.6.0, Release 11)", 2011.
- [ucp] Vodafone, "Short Message Service Centre (SMSC) External Machine Interface (EMI) Description Version 4.3d", 2011.

Appendix A. SMS encoding

For use in SMS applications, CoAP messages can be transferred using SMS binary mode. However, there is operational experience showing that some environments cannot successfully send a binary mode SMS.

For transferring SMS in character mode (7-bit characters), base64-encoding [RFC4648] is an obvious choice. 3 bytes of message (24 bits) turn into 4 characters, which consume 28 bits. The overall overhead is approximately 17 %; the maximum message size is 120 bytes (160 SMS characters).

If a more compact encoding is desired, base85 encoding could be employed (however, probably not the version defined in [RFC1924] -- instead, the version used in tools such as btoa and PDF should be chosen). However, this requires division operations. Also, the base85 character set includes several characters that cannot be transferred in a single 7-bit unit in SMS and/or are known to cause operational problems. A modified base85 character set can be defined to solve the latter problem. 4 bytes of message (32 bits) turn into 5 characters, which consume 35 bits. The overall overhead is approximately 9.3 %; the resulting maximum message size is 128 bytes (160 SMS characters).

Base64 and base85 do not make use of the fact that much CoAP data will be ASCII-based. Therefore, we define the following ASCII-optimized SMS encoding.

A.1. ASCII-optimized SMS encoding

Not all 128 theoretically possible SMS characters are operationally free of problems. We therefore define:

Shunned code characters: @ sign, as it maps to 0x00

LF and CR signs (0x0A, 0x0D)

uppercase C cedilla (0x09), as it is often mistranslated in gateways

ESC (0x1B), as it is used in certain character combinations only

Some ASCII characters cannot be transferred in the base SMS character set, as their code positions are taken by non-ASCII characters. These are simply encoded with their ASCII code positions, e.g., an underscore becomes a section mark (even though underscore has a different code position in the SMS character set).

Equivalently translated input bytes: \$, @, [, \,], ^, _, ` , {, |, }, ~, DEL

In other words, bytes 0x20 to 0x7F are encoded into the same code positions in the 7-bit character set.

Out of the remaining code characters, the following SMS characters are available for encoding:

Non-equivalently translated (NET) code characters: 0x01 to 0x08, (8 characters)

0x0B, 0x0C, (2 characters)

0x0E to 0x1A, (13 characters)

0x1C to 0x1F, (4 characters)

Of the 27 NET code characters, 18 are taken as prefix characters (see below), and 8 are defined as directly translated characters:

Directly translated bytes: Equivalently translated input bytes are represented as themselves

0x00 to 0x07 are represented as 0x01 to 0x08

This leaves 0x08 to 0x1F and 0x80 to 0xFF. Of these, the bytes 0x80 to 0x87 and 0xA0 to 0xFF are represented as the bytes 0x00 to 0x07 (represented by characters 0x01 to 0x08) and 0x20 to 0x7F, with a prefix of 1 (see below). The characters 0x08 to 0x1F are represented as the characters 0x28 to 0x3F with a prefix of 2 (see below). The characters 0x88 to 0x9F are represented as the characters 0x48 to 0x5F with a prefix of 2 (see below). (Characters 0x01 to 0x08, 0x20

to 0x27, 0x40 to 0x47, and 0x60 to 0x7f with a prefix of 2 are reserved for future extensions, which could be used for some backreferencing or run-length compression.)

Bytes that do not need a prefix (directly translated bytes) are sent as is. Any byte that does need a prefix (i.e., 1 or 2) is preceded by a prefix character, which provides a prefix for this and the following two bytes as follows:

char	pfx	.	char	pfx
0x0B	100	.	0x15	200
0x0C	101	.	0x16	201
0x0E	102	.	0x17	202
0x0F	110	.	0x18	210
0x10	111	.	0x19	211
0x11	112	.	0x1A	212
0x12	120	.	0x1C	220
0x13	121	.	0x1D	221
0x14	122	.	0x1E	222

Table 2: SMS prefix character assignment

(This leaves one non-shunned character, 0x1F, for future extension.)

The coding overhead of this encoding for random bytes is similar to Base85, without the need for a division/multiplication. For bytes that are mostly ASCII characters, the overhead can easily become negative. (Conversely, for bytes that for some reason are more likely to be non-ASCII than in a random sequence of bytes, the overhead becomes greater.)

So, for instance, for the CoAP message in Figure 7:

ver	tt	code	mid	
1	ack	2.05	17033	
content_type		40		
token		sometok		

3c 2f 3e 3b 74 69 74 6c 65 3d 22 47 65 6e 65 72	</>;title="Gener
61 6c 20 49 6e 66 6f 22 3b 63 74 3d 30 2c 3c 2f	al Info";ct=0,</
74 69 6d 65 3e 3b 69 66 3d 22 63 6c 6f 63 6b 22	time>;if="clock"
3b 72 74 3d 22 54 69 63 6b 73 22 3b 74 69 74 6c	;rt="Ticks";titl
65 3d 22 49 6e 74 65 72 6e 61 6c 20 43 6c 6f 63	e="Internal Cloc
6b 22 3b 63 74 3d 30 2c 3c 2f 61 73 79 6e 63 3e	k";ct=0,</async>
3b 63 74 3d 30	;ct=0

Figure 7: CoAP response message as captured and decoded

The 116 byte unencoded message is shown as ASCII characters in Figure 8 (\xDD stands for the byte with the hex digits DD):

```
bEB\x89\x11(\xA7sometok</>;title="General Info";ct=0,</time>
;if="clock";rt="Ticks";title="Internal Clock";ct=0,</async>;ct=0
```

Figure 8: CoAP response message shown as unencoded characters

The only non-ASCII characters in this example are in the beginning of the message. According to the translation instructions above, the four bytes:

```
89 11 ( A7
```

need the prefixes:

```
2 2 0 1
```

As each prefix character always covers three unencoded bytes, we need the prefix characters for 220 and 100, which are \x1C and \x0B, respectively (Table 2).

The equivalent SMS encoding is shown as equivalent-coded SMS characters in Figure 9 (7 bits per character, \x1C is the 220 prefix and \x0B is the 100 prefix, the rest is shown in equivalent encoding), adding two characters of prefix overhead, for a total length of 118 7-bit characters or 104 (103.25 plus padding) bytes:

```
bEB\x1CI1(\x0B'sometok</>;title="General Info";ct=0,</time>
;if="clock";rt="Ticks";title="Internal Clock";ct=0,</async>;ct=0
```

Figure 9: CoAP response message shown as SMS-encoded characters

Appendix B. Changelog

RFC editor: please remove this appendix.

Changed from draft-05 to draft-06:

- o Update references and addresses
- o Integrate relevant text from coap-misc as an appendix.
- o Section 2 & 3 are merged to section 1

Changed from draft-04 to draft-05:

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

Changed from draft-03 to draft-04:

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

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.

Changed from draft-01 to draft-02:

- o Added security considerations: Transport and Object Security.
Section 11

- o Reply-To-* changed to Response-To-*. Section 12
- o Added URI scheme.
- o Added possible CON/NON/ACK interactions.
- o Added possible M2M proxy scenarios.
- o Added reference to bormann-coap-misc for other SMS encoding. Section 4
- o Updated requirements on Uri-Host and Uri-Port for coap+tel://.
- o Chose CoAP option numbers and updated the option number table to meet draft-ietf-core-coap-10. />
- o Added an IANA registration for the URI scheme. Section 12.2

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.

Contributors

Appendix A has been contributed by Carsten Bormann.

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

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

Authors' Addresses

Koojana Kuladinithi (editor)
ComNets, Hamburg University of Technology
Am Schwarzenberg-Campus 3
Hamburg 21073
Germany

Phone: +49 40 428 783533
Email: koojana.kuladinithi@tuhh.de

Markus Becker
Tridonic GmbH & Co KG
Faerbergasse 15
Dornbirn 6851
Austria

Phone: +43 5572 395 45637
Email: markus.becker@tridonic.com

Kepeng LI
Alibaba Group
Wenyixi Road, Yuhang District
Hangzhou, Zhejiang 311121
China

Email: kepeng.lkp@alibaba-inc.com

Thomas Poetsch
New York University Abu Dhabi
P.O. Box 129188
Abu Dhabi 129188
United Arab Emirates

Phone: +971 2 628 5069
Email: thomas.poetsch@nyu.edu

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2017

C. Bormann
Universitaet Bremen TZI
A. Betzler
Fundacio i2CAT
C. Gomez
I. Demirkol
Universitat Politecnica de Catalunya/Fundacio i2CAT
July 08, 2016

CoAP Simple Congestion Control/Advanced
draft-bormann-core-cocoa-04

Abstract

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

This specification defines some simple advanced CoRE Congestion Control mechanisms, Simple CoCoA. It is making use of input from simulations and experiments in real 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 9, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Context	3
3. Area of Applicability	4
4. Advanced CoAP Congestion Control: RTO Estimation	4
4.1. Blind RTO Estimate	5
4.2. Measured RTO Estimate	5
4.2.1. Modifications to the algorithm of RFC 6298	6
4.2.2. Discussion	6
4.3. Lifetime, Aging	7
5. Advanced CoAP Congestion Control: Non-Confirmables	7
5.1. Discussion	8
6. IANA Considerations	8
7. Security Considerations	8
8. References	8
8.1. Normative References	8
8.2. Informative References	9
Appendix A. Advanced CoAP Congestion Control: Aggregate Congestion Control	10
A.1. Proposed Algorithm	10
A.2. Example 1	10
A.3. Example 2	11
A.4. Discussion	12
Acknowledgements	12
Authors' Addresses	13

1. Introduction

(See Abstract.)

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

1.1. Terminology

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

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

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

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

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

2. Context

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

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

The present specification also assumes that, outside of exchanges, non-confirmable messages can only be used at a limited rate without an advanced congestion control mechanism (this is mainly relevant for [RFC7641]). It is also intended to address the [RFC5405] guideline about combining congestion control state for a destination; and to clarify its meaning for CoAP using the definition of an endpoint.

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

3. Area of Applicability

The present algorithm is intended to be generally applicable. The objective is to be "better" than default CoAP congestion control in a number of characteristics, including achievable goodput for a given offered load, latency, and recovery from bursts, while providing more predictable stress to the network and the same level of safety from catastrophic congestion. It does require three state variables per scope plus the state needed to do RTT measurements, so it may not be applicable to the most constrained devices (class 1 as per [RFC7228]).

The scope of each instance of the algorithm in the current set of evaluations has been the five-tuple, i.e., CoAP + endpoint (transport address) for Initiator and Responder. Potential applicability to larger scopes needs to be examined.

Aggregate Congestion Control (Appendix A) is not yet supported by research as well as the other algorithms in this specification. Its use is more interesting on the cloud side, where a single CoAP endpoint may need to talk to thousands of other endpoints and may need to control the burstiness of the resulting aggregate traffic.

4. Advanced CoAP Congestion Control: RTO Estimation

For an initiator that plans to make multiple requests to one destination endpoint, it may be worthwhile to make RTT measurements in order to obtain a better RTO estimation than that implied by the default initial timeout of 2 to 3 s. This is based on the usual algorithms for RTO estimation [RFC6298], with appropriately extended default/base values, as proposed in Section 4.2.1. Note that such a mechanism must, during idle periods, decay RTO estimates that are shorter or longer than the basic RTO estimate back to the basic RTO estimate, until fresh measurements become available again, as proposed in Section 4.3.

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

for a mechanism with variable backoff factors is presented in Section 4.2.1.

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

4.1. Blind RTO Estimate

The initial RTO estimate for an endpoint is set to 2 seconds (the initial RTO estimate is used as the initial value for both `E_weak_` and `E_strong_` below).

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

4.2. Measured RTO Estimate

The RTO estimator runs two copies of the algorithm defined in [RFC6298], as modified in Section 4.2.1: One copy for exchanges that complete on initial transmissions (the "strong estimator", `E_strong_`), and one copy for exchanges that have run into retransmissions, where only the first two retransmissions are considered (the "weak estimator", `E_weak_`). For the latter, there is some ambiguity whether a response is based on the initial transmission or the retransmissions. For the purposes of the weak estimator, the time from the initial transmission counts. Responses obtained after the third retransmission are not used to update an estimator.

The overall RTO estimate is an exponentially weighted moving average ($\alpha = 0.5$ and 0.25 , respectively) computed of the strong and the weak estimator, which is evolved after each contribution to the weak estimator (1) or to the strong estimator (2), from the estimator that made the most recent contribution:

$$\text{RTO} := 0.25 * \text{E_weak_} + 0.75 * \text{RTO} \quad (1)$$

$$\text{RTO} := 0.5 * \text{E_strong_} + 0.5 * \text{RTO} \quad (2)$$

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

4.2.1. Modifications to the algorithm of RFC 6298

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

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

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

If an RTO estimation is lower than 1 s or higher than 3 s, instead of applying a binary backoff factor in both cases, a variable backoff factor is used. For RTO estimations below 1 s, the RTO for a retransmission is multiplied by 3, while for estimations above 3 s, the RTO is multiplied only by 1.5 (this updated choice of numbers to be verified by more simulations). This helps to avoid that exchanges with small initial RTOs use up all retransmissions in a short interval of time and exchanges with large initial RTOs may not be able to carry out all retransmissions within MAX_TRANSMIT_WAIT (93 s).

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

4.2.2. Discussion

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

Some evaluation has been done on earlier versions of this specification [Betzler2013]. A more recent (and more comprehensive) reference is [Betzler2015]. Additional investigation is required.

4.3. Lifetime, Aging

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

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

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

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

5. Advanced CoAP Congestion Control: Non-Confirmables

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

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

Non-confirmables outside exchanges (e.g., [RFC7641] notifications sent as non-confirmables) are governed by the following rules:

1. Of any 16 consecutive messages towards this endpoint that aren't responses or acknowledgments, at least 2 of the messages must be confirmable.
2. The confirmable messages must be sent under an RTO estimator, as specified in Section 4.
3. The packet rate of non-confirmable messages cannot exceed $1/\text{RTO}$, where RTO is the overall RTO estimator value at the time the non-confirmable packet is sent.

5.1. Discussion

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

[RFC7641], Section 4.5.1, specifies that the rate of NONs SHOULD NOT exceed $1/\text{RTT}$ on average, if the server can maintain an RTT estimate for a client. CoCoA limits the packet rate of NONs in this situation to $1/\text{RTO}$. Assuming that the RTO estimation in CoCoA works as expected, $\text{RTO}[k]$ should be slightly greater than the $\text{RTT}[k]$, thus CoCoA would be more conservative. The expectation therefore is that complying with the NON rate set by CoCoA leads to complying with [RFC7641].

6. IANA Considerations

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

7. Security Considerations

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

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<http://www.rfc-editor.org/info/rfc2914>>.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, DOI 10.17487/RFC5405, November 2008, <<http://www.rfc-editor.org/info/rfc5405>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<http://www.rfc-editor.org/info/rfc6298>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

8.2. Informative References

- [Betzler2013] Betzler, A., Gomez, C., Demirkol, I., and J. Paradells, "Congestion control in reliable CoAP communication", ACM MSWIM'13 p. 365-372, DOI 10.1145/2507924.2507954, 2013.
- [Betzler2015] Betzler, A., Gomez, C., Demirkol, I., and J. Paradells, "CoCoA+: an Advanced Congestion Control Mechanism for CoAP", Ad Hoc Networks Vol. 33 pp. 126-139, DOI 10.1016/j.adhoc.2015.04.007, October 2015.
- [I-D.bormann-core-congestion-control] Bormann, C. and K. Hartke, "Congestion Control Principles for CoAP", draft-bormann-core-congestion-control-02 (work in progress), July 2012.
- [I-D.eggert-core-congestion-control] Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<http://www.rfc-editor.org/info/rfc5348>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<http://www.rfc-editor.org/info/rfc5681>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

Appendix A. Advanced CoAP Congestion Control: Aggregate Congestion Control

(The mechanism defined in this appendix has received less research than the ones in the main body of this specification.)

A.1. Proposed Algorithm

To avoid possible congestion when sending many packets to different destination endpoints in parallel, the overall number of outstanding interactions towards different destination endpoints should be limited. An upper limit PLIMIT determines the maximum number of outstanding interactions towards different destination endpoints that are allowed in parallel. When a request is to be sent to a destination endpoint, PLIMIT is determined according to Equation (3) in the case that no RTO information is already available for the destination endpoint, or using Equation (4) in case that valid RTO information is available for the destination endpoint. Both formulas use LAMBDA, as defined in Equation (5).

$$\text{PLIMIT} = \text{LAMBDA} \quad (3)$$

$$\text{PLIMIT} = \max(\text{LAMBDA}, \text{LAMBDA} * \text{ACK_TIMEOUT} / \text{mean}(\text{RTO})) \quad (4)$$

$$\text{LAMBDA} = \max(4, \text{KNOWN_DEST_ENDPOINTS} / 4) \quad (5)$$

mean(RTO) is the average value of all valid RTO estimations maintained by the device. LAMBDA is the maximum of a constant value (4 by default) and the rounded up value of KNOWN_DEST_ENDPOINTS/4, where KNOWN_DEST_ENDPOINTS is the overall number of "known" destination endpoints (i.e. destination endpoints for which an RTO estimate is maintained).

A new interaction may only be processed if the current overall number of outstanding interactions is lower than the PLIMIT calculated when the request is initiated.

A.2. Example 1

In the following we give an example, with LAMBDA = 4 (our proposed default LAMBDA):

Assume that a sender has so far obtained RTO estimations for two destination endpoints A (RTO = 0.5 s) and B (RTO = 1.5 s), and currently pcount (a variable which accounts for the number of outstanding interactions towards endpoints) is equal to 0. Now three transactions are initiated consecutively in the following order: one for A, one for B and one for a new destination C.

When an interaction with node A is initiated, LAMBDA is calculated

$$\text{LAMBDA} = \max(4, 3/4) = 4.$$

Then PLIMIT is calculated:

$$\text{PLIMIT} = \max(4, (4*2 \text{ s})/\text{mean}(0.5 \text{ s}, 1.5 \text{ s})) = \max(4, 8 \text{ s}/1 \text{ s}) = \max(4, 8) = 8$$

This means that with the current RTO information the sender has obtained about the destination endpoints, up to 8 outstanding interactions to different destination endpoints would be allowed. By initiating an interaction with A, pcount is increased to 1, which is still below PLIMIT. Thus, the interaction may be processed. The same applies to B: pcount increases to 2 after obtaining the same PLIMIT value of 8.

Destination C is unknown to CoCoA, therefore the updated PLIMIT before processing the interaction with node C is 4. The CoAP request may be processed (pcount = 3). If two more interactions with different unknown destination endpoints would have been initiated, only the first one would have met the requirements to process it (PLIMIT = 4, pcount = 4). The second interaction would have increased pcount to 5, which is not permitted, since PLIMIT is 4. It may occur that pcount exceeds PLIMIT in particular cases, in this case, the interaction is not permitted as well. If the number of destinations exchanges are initiated with would increase further, eventually LAMBDA could grow beyond 4, allowing for more interactions to be sent in parallel.

A.3. Example 2

Let us now assume that a sender has so far obtained RTO estimations for 101 destination endpoints, their average RTO is 1 s, and currently pcount is equal to 0. When a new transaction is initiated with a destination endpoint for which an RTO estimate is available, LAMBDA is calculated

$$\text{LAMBDA} = \max(4, 101/4) = 26$$

Based on this, PLIMIT is calculated as follows:

$$\text{PLIMIT} = \max(26, (26*2 \text{ s})/1 \text{ s}) = \max(26, 52) = 52$$

This means that with the current RTO information that the sender has obtained about the destination endpoints, up to 52 outstanding interactions to known destination endpoints would be allowed.

However, if the new exchange is to be initiated with an "unknown" destination endpoint (i.e. an endpoint for which an RTO estimate is not available), then PLIMIT would be 26.

A.4. Discussion

The idea of the proposal is to allow more parallel transactions to different destination endpoints if we have low RTO estimations for them (which can be interpreted as good connections and low degree of congestion). If the RTO estimations are large or interactions with unknown destinations are initiated, the mechanism behaves more conservatively by reducing the maximum number of parallel interactions towards different destinations, but allowing at least LAMBDA outstanding interactions. The second term of the `max()` statement used to calculate LAMBDA avoids behaving too restrictively when exchanges with many different destination endpoints are initiated. If no RTO information is available for a destination endpoint, PLIMIT is simply set to be LAMBDA.

If at any moment `pcount` would exceed PLIMIT, CoAP does not immediately perform the transaction. Further, it is important that in parallel, NSTART for each destination endpoint applies (which, for now, we assume to be 1). The default value used for LAMBDA (equal to 4 as per this document) determines how aggressive/conservative CoCoA behaves by default for a limited set of destination endpoints and it should be chosen carefully. The term `KNOWN_DEST_ENDPOINTS/4` loosens the hard limit of exchanges when large numbers of destination endpoints are addressed.

It will be necessary to see whether this approach is effective in the sense that it avoids congestion in use cases where transactions to a multitude of different destination endpoints are initiated. An important aspect of such evaluations would be whether LAMBDA is too conservative when dealing with few destination endpoints and whether it allows for a dynamic adjustment of parallel exchanges with large numbers of destination endpoints. On the other hand, a more safe approach would use `max(RTO)` instead of `mean(RTO)`. Other concerns include the fact that the congestion degree of the paths to "known" destination endpoints influence whether a new interaction is permitted to some new endpoint which may be in very different conditions in terms of congestion. However, it is desirable to avoid adding a lot of complexity to the current CoCoA mechanisms.

Acknowledgements

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

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

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

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CAS15/00336. His contribution to this work has been carried out in part during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge, in collaboration with Prof. Jon Crowcroft.

Authors' Addresses

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

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

August Betzler
Fundacio i2CAT
Mobile and Wireless Internet Group
C/ del Gran Capita, 2
Barcelona 08034
Spain

Email: august.betzler@entel.upc.edu

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

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

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

Email: ilker.demirkol@entel.upc.edu

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: February 1, 2013

C. Bormann
K. Hartke
Universitaet Bremen TZI
July 31, 2012

Congestion Control Principles for CoAP
draft-bormann-core-congestion-control-02

Abstract

The CoAP protocol needs to be implemented in such a way that it does not cause persistent congestion on the network it uses. Congestion control is a complex issue -- the proper rationale for the congestion control mechanisms chosen in CoAP is probably more material than the CoAP protocol specification itself. This informational document attempts to pull out the background material and more extensive considerations behind the CoAP congestion control mechanisms, while leaving the basic MUSTs and MUST NOTs in the main spec.

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 1, 2013.

Copyright Notice

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

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

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Objectives	4
1.2.1. TCP-Friendliness	4
1.2.2. Actually working well	4
1.2.3. Getting actual implementation	5
2. Input	6
2.1. RFC 2914	6
2.2. RFC 5405	6
2.3. draft-eggert-core-congestion-control	7
3. coap-11 Congestion Control Principles	8
4. How do other protocols do it	11
4.1. DNS	11
4.2. SIP	11
4.3. TCP	11
4.4. HTTP	12
5. Advanced CoAP Congestion Control	14
5.1. RTT Measurement	14
5.2. Block Slow-Start	14
6. Changes Planned for Base Specifications	16
7. IANA Considerations	17
8. Security Considerations	18
9. Acknowledgements	19
10. References	20
10.1. Normative References	20
10.2. Informative References	20
Authors' Addresses	22

1. Introduction

With few exceptions, it is simply incompetent to build an implementation of a packet-based protocol without considering congestion control. Unfortunately, detailed, evidence-based knowledge about congestion control is limited to a small group of people. It has become customary for these to try to encode their knowledge into the protocol definitions, in an attempt to replace competence by conformance.

This has worked relatively well for TCP, not the least because the art of TCP implementation is itself limited to a rather small group of experts, which over the years often have acquired some knowledge of congestion control principles, complementing the desire for conformance by substantial competence again. Conversely, application developers are a much larger, much more diverse group. Worse, protocol complexity for which the rationale is not apparent to the developers might simply not be implemented. Giving congestion-unaware developers UDP sockets that are not protected by TCP's congestion control may lead to disasters.

With this background, an application protocol that is threatening to be widely deployed and does not rely on the built-in congestion control properties of TCP presents a serious worry.

This document attempts to present a more extensive rationale for CoAP's minimal, but effective congestion control design, as well as some updates to it. This rationale is not included in [I-D.ietf-core-coap] or [I-D.ietf-core-observe] as the specification is threatening to become too long with all the rationale and implementation considerations discussion already included. While the present document discusses normative statements, it is not intended to supplement or replace the normative statements in [I-D.ietf-core-coap] and [I-D.ietf-core-observe], but just to provide additional explanation.

((Editorial note: the updates to the mandates discussed here partially still need to make it into the next version of [I-D.ietf-core-coap]. A summary of the updates needed is in Section 6.))

1.1. Terminology

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

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

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

1.2. Objectives

The objectives of adding congestion control to the CoAP protocol specification can be on two different levels, with one additional (third) consideration.

1.2.1. TCP-Friendliness

Much of the knowledge that the IETF has accumulated on congestion control focuses on not being worse than its flagship transport protocol, TCP, and being "fair" to instances of TCP competing for capacity. Since fairness is not really a well-defined term, we reduce it to "friendliness".

One objective of this document is to discuss how CoAP can be employed in a TCP-friendly way, and what are the minimum mandates the protocol needs to make in order to ensure this for reasonable applications.

(Note that TCP itself is not TCP-friendly when abused, e.g., when opening 10000 connections in close succession; so there will be no attempt to stay TCP-friendly when CoAP is abused, either.)

Conclusion: CoAP needs to be TCP-friendly, but probably not more so than TCP itself.

1.2.2. Actually working well

Making sure that the network continues to work well in the presence of a strong deployment of active CoAP endpoints is a much harder objective to achieve. There is only limited knowledge about the characteristics of the constrained node/networks CoAP will be used in. They might exhibit congestion in surprising ways.

It may turn out the collected wisdom that has been derived from TCP deployment experience in the mostly browser-oriented Internet does not transfer to the Internet of Things, and that we need to invent new mechanisms for the latter.

But this is research.

Imposing the need for a completed solution that meets requirements entirely unknown at this time would be an instance of the Fallacy of

Perfection [GF].

We will need to accumulate additional knowledge, on a research basis, and with experience coming in from larger CoAP deployments. One likely outcome is that constrained node/networks will simply continue to evolve to be able to cope with TCP and CoAP.

Conclusion: For now, we will focus on staying safe where TCP would have stayed safe.

1.2.3. Getting actual implementation

The protocol specification may specify whatever it wants; if there is significant complexity in implementing a mandate and the rationale is not apparent for implementers, compliance will be but a lucky coincidence - even more so in implementations for highly constrained systems. A design that achieves stable operation outside pathological situations and is implemented is preferable to a picture-perfect design that is a beautiful part of the specification and then ignored.

Binding the inevitable complexity of a congestion control scheme to mechanisms that already have to be implemented for other functional reasons seems the most fruitful approach for obtaining compliance. This consideration, together with the main design objective of CoAP - being implementable on constrained nodes and networks - has been the overriding design objective.

2. Input

The word "congestion" occurs more than a hundred times in lid-abstracts.txt, indicating that there is a lot of documents under construction that might become relevant to this document. We select a few existing documents here and pick up a few salient points.

2.1. RFC 2914

[RFC2914], "Congestion Control Principles", is the BCP that lays out the basic principles for congestion control in the Internet. While it does allude to non-TCP protocols, it mainly focuses on TCP and TCP-like behavior.

2.2. RFC 5405

[RFC5405], "Unicast UDP Usage Guidelines for Application Designers", makes additional points for the usage of UDP. It is also a BCP document. Its considerations have mostly been made without looking at specific application protocols, and with a view to guiding application protocol developers towards congestion-controlled transport protocols (which is unfortunately not an appropriate choice for CoAP). It does consider the case of low data-volume applications (section 3.1.2 is therefore the most relevant section for this document). It clearly needs to be interpreted intelligently in order to arrive at congestion control guidelines for a new application protocol. E.g., it recommends:

Applications that at any time exchange only a small number of UDP datagrams with a destination SHOULD still control their transmission behavior by not sending on average more than one UDP datagram per round-trip time (RTT) to a destination.

Instead, a CoAP client that does receive a response without the need for a retransmission should be able to send an ensuing request right away, without the need to do any such rate control -- this keeps the spirit, but not the letter of that requirement.

While [RFC5405] does provide a good set of "don't forget" points, some of its requirements appear to attempt to err on the side of caution, without regards to the specific characteristics of an application. Fortunately, these requirements are often phrased as a SHOULD, so it is possible to explain when and why they should not be heeded.

2.3. draft-eggert-core-congestion-control

[I-D.eggert-core-congestion-control], "Congestion Control for the Constrained Application Protocol (CoAP)", was the original document that led to CoAP's congestion control design. This document provides good historical context and should be read in conjunction with the present document. However, the "credit-based" mechanism proposed in its section 3.2 is probably too complicated to be implemented in constrained nodes; CoAP now uses a simpler algorithm that uses the information the implementation already has to keep (i.e., it is based on limiting the outstanding exchanges).

3. coap-ll Congestion Control Principles

CoAP is a protocol that attempts to minimize the complexity of its implementation. It is mainly intended for interactions that are not really flow-shaped, so traditional congestion control mechanisms simply do not have useful information to work on.

Basic CoAP [I-D.ietf-core-coap] uses a strict lock-step protocol for its requests and responses (both on the reliability layer with CON/ACK and one level higher with requests and responses), with exponential back-off in case of non-delivery. The initial timeout is dithered between 2 and 3 seconds and grows up to between 32 and 48 seconds.

This is inherently TCP-friendly, similar to the way protocols like DNS operate.

[I-D.ietf-core-coap] goes on to require:

In order not to cause congestion, Clients (including proxies) SHOULD strictly limit the number of simultaneous outstanding interactions that they maintain to a given server (including proxies). 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 a response has not yet been received but is still expected (which may both occur at the same time, counting as one outstanding interaction). A good value for this limit is the number 1. (Note that [RFC2616], in trying to achieve a similar objective, did specify a specific number of simultaneous connections as a ceiling. While revising [RFC2616], this was found to be impractical for many applications [I-D.ietf-httpbis-p1-messaging]. For the same considerations, this specification does not mandate a particular maximum number of outstanding interactions, but instead encourages clients to be conservative when initiating interactions.)

The rationale for this design is that it is very easy to implement for a constrained device: a constrained device will already have a hard limit on the number of slots available for initiating transactions. Similarly, even back-end systems already need to bind state to outstanding transactions; adding some form of congestion control state to these does not require maintaining new objects, just new fields. In any case, having some form of limit is not elective: in the text the SHOULD needs to be changed into a MUST, even though it may not be easy to pinpoint the exact criterion for compliance.

In the following, we refer to the initiator parameter that limits the number of outstanding interactions as NSTART.

Clients SHOULD also heed this [RFC5405] guideline:

an application SHOULD perform congestion control over all UDP traffic it sends to a destination, independently from how it generates this traffic. For example, an application that forks multiple worker processes or otherwise uses multiple sockets to generate UDP datagrams SHOULD perform congestion control over the aggregate traffic.

Note that [RFC5405] is not explicit here with respect to what it considers to be a "destination"; it also uses the term "destination host" when it appears to provide specific discussion about all protocol entities at an IP address. [RFC5405] duly notes the failure of the congestion manager approach [RFC3124], but appears to wish it back into existence. For the purposes of CoAP, probably "destination" here should be used as with the CoAP term destination endpoint (i.e., including the UDP port number). Still, an implementation that e.g. uses a new source port per request (i.e. a new source endpoint, which is a valid strategy) probably needs to heed this SHOULD for the entirety of the combination of its own endpoint abstractions.

For certain exchanges in CoAP, there is a chance that a request would never elicit a response (e.g., due to a crashed server) but there is also no (protocol) timeout governing this exchange. Therefore, the count of outstanding interactions needs to decay at some rate; a decay rate below that at which TCP sends to a very lossy channel (e.g., 7 B/s) should be safe.

There are also some special congestion control considerations with responses to multicast requests, see [I-D.ietf-core-coap] section 4.5; servers are expected to provide estimates for group size and a target rate as well as a response size. Where those estimates are hard to come up with, a default response dithering window of 10 seconds should be added to [I-D.ietf-core-coap], as well an admonition for a client not to use multicast requests when such a default window would be way off. Finally, a server that receives another multicast request within the dithering window for a request that it already is answering SHOULD move the dithering window for its next response to after the first dithering window.

Finally, the text in [I-D.ietf-core-coap] needs to be reviewed whether it always clearly separates the discussion for avoiding network congestion from any mechanisms for avoiding server overloading.

[I-D.ietf-core-observe] adds one additional behavior: servers may send NON messages as notifications for state changes, which is

outside of exchanges that would be governed by NSTART. This functionality needs to be supported with some discussion of congestion control. Generally, servers SHOULD NOT send more than one NON message every 3 seconds on average ([RFC5405] section 3.1.2), and they SHOULD NOT send NON messages while waiting for CON messages to be acknowledged (however, CON retransmissions should send the new resource state if it has changed since, see [I-D.ietf-core-observe] section 4.5). There already was a decision to add a requirement to require sending a CON message at least every 24 hours before continuing with NON messages; probably the parameter of no more than a NON per 3 seconds should be increased for servers that check the client that rarely (e.g., to the rate at which TCP sends into a very lossy channel, e.g., 7 B/s).

4. How do other protocols do it

While CoAP congestion control could be designed from first principles, it is maybe more realistic to have a look at how other protocols address its respective version of the problem.

4.1. DNS

The DNS protocol, which in many characteristics is quite close to CoAP, does not have any explicit mechanisms for congestion control at all. Many documents consider DNS to be "sporadic messages", not worth of congestion control.

[RFC4336] says:

(The short flows generated by request-response applications, such as DNS and SNMP, don't cause congestion in practice, and any congestion control mechanism would take effect between flows, not within a single end- to-end transfer of information.)

(This simple packet-for-packet request-response nature is now changing a bit with DNS being used for voluminous keying information and growing TXT records.)

4.2. SIP

SIP uses a 0.5 s initial timeout (T1 "RTT Estimate"), and uses binary exponential increase after that. That is similar to CoAP, but starts from a smaller initial estimate. CoAP is more conservative (initial RESPONSE_TIMEOUT is 2 s to 3 s) as we expect latencies in constrained networks to be higher than in the networks used for telephony.

4.3. TCP

A well-known problem with relying on TCP's built-in congestion control is that, even with all congestion-control mechanisms in place, simply multiplying the number of instances may lead to eventual congestion.

About a decade ago, TCP has increased its initial congestion window (IW) to about 3 full-size packets, or up to 4 packets with MSS <= 1095 bytes (which is comparable to CoAP's maximum packet sizes) [RFC3390]. As an Experimental specification, moving to an IW of 10 packets (IW10) is being examined [I-D.ietf-tcpm-initcwnd]. A related change is also planned in that document that will avoid resetting this initial window when the SYN or SYN/ACK is lost. This would mean that it is considered appropriate to send about 15 kB of data on a single connection without any congestion control feedback whatsoever,

except that some SYN+SYN/ACK exchange made it through. While [I-D.ietf-tcpm-initcwnd] is not yet approved, it is a WG document and there is widespread feeling of its inevitability even beyond the experimental status that is being planned now.

The numbers 3, 4, and 10 clearly provide some additional context for the selection of appropriate values of NSTART.

Conclusion: For now, it is probably appropriate to RECOMMEND keeping NSTART at or below a value chosen from the space between 3 and 10.

4.4. HTTP

HTTP is running on top of TCP, so it is TCP-friendly by definition. However, as HTTP 1.0 was using one TCP connection per request, and it became clear that browser usage would entail fetching many objects in parallel, congestion was still observed, and client implementations started to limit the number of simultaneously active connections to one server. Even when persistent connections were added (and later codified in HTTP 1.1) this remained a concern. Under 8.1.4 "Practical considerations", [RFC2616] defines a limit on the number of simultaneous connections from one client to one server.

Clients that use persistent connections SHOULD limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. A proxy SHOULD use up to 2*N connections to another server or proxy, where N is the number of simultaneously active users. These guidelines are intended to improve HTTP response times and avoid congestion.

Intended as a guideline, this has been implemented to the letter in browser clients for a decade. However, using this as a hard limit is simply not appropriate for all environments. This led server implementers to widely deploy workarounds, such as splitting up a website between multiple servers ("domain sharding") in order to increase the connection concurrency.

From this historical evidence we can learn that well-meaning limitations can cause a lot of pain when implemented slavishly. The httpbis effort has learned this lesson and removed the suggestion for a hard limit (see [HTTPBIS131], [HTTPBIS715]). Note that it now says:

Clients (including proxies) SHOULD limit the number of simultaneous connections that they maintain to a given server (including proxies).

Previous revisions of HTTP gave a specific number of connections as a ceiling, but this was found to be impractical for many applications. As a result, this specification does not mandate a particular maximum number of connections, but instead encourages clients to be conservative when opening multiple connections.

In particular, while using multiple connections avoids the "head-of- line blocking" problem (whereby a request that takes significant server-side processing and/or has a large payload can block subsequent requests on the same connection), each connection used consumes server resources (sometimes significantly), and furthermore using multiple connections can cause undesirable side effects in congested networks.

Note that servers might reject traffic that they deem abusive, including an excessive number of connections from a client.

Conclusion: There is no doubt that CoAP should follow this hard-learned expertise.

5. Advanced CoAP Congestion Control

5.1. RTT Measurement

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

One important consideration not relevant for TCP is the fact that a CoAP round-trip may include application processing time, which may be hard to predict, and may differ between different resources available at the same endpoint. Servers will only trigger early ACKs (with a non-piggybacked response to be sent later) based on the default timers, e.g. after 1 s. A client that has arrived at a RTT estimate much shorter than the 2 to 3 s used as a default **SHOULD** therefore not expend all of its retransmissions in the shorter estimated timescale.

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

5.2. Block Slow-Start

The CoAP protocol is not optimized for making good use of available network capacity; given a good offered load, a lightly-loaded network and some time, a TCP connection will always overtake a series of CoAP requests.

However, the [I-D.ietf-core-block] protocol can be used by inventive clients to emulate TCP slow start. E.g., a client can do a request for block 0, and, if a response comes back without a loss, it can fire off the requests for block 1 and block 2 at the same time, etc., using each response in a similar way that TCP would clock its data segments based on ACKs, waiving NSTART. Similar approaches may work to increase channel utilization for any other REST usage that requires multiple requests.

Clearly, the slow start period **MUST** terminate on the first loss/retransmission. How exactly the congestion window is to be

maintained after that (a "congestion avoidance period" for CoAP) is a subject for further study. See also [I-D.mathis-tcpm-tcp-laminar] for fresh approaches to maintaining the necessary variables in TCP. Another alternative would be an implementation that emulates [RFC5348].

6. Changes Planned for Base Specifications

[I-D.ietf-core-coap]:

1. Change SHOULD in second paragraph of [I-D.ietf-core-coap] 4.7 to MUST; define protocol parameter NSTART.
2. Add reference to (and/or cite) [RFC5405] guideline about combining congestion control state for a destination; clarify its meaning for CoAP using the definition of an endpoint.
3. Add a mechanism of decaying outstanding transactions at a rate of about 7 B/s.
4. Add default "Leisure" (response dithering windows) of 10 seconds to [I-D.ietf-core-coap] 8.2, as well as an admonition for a client not to use multicast requests when such a default window would be way off.

[I-D.ietf-core-observe]:

1. servers SHOULD NOT send more than one NON notification every 3 seconds to an endpoint on average ([RFC5405] section 3.1.2); define protocol parameter MAXNONRATE.
2. servers SHOULD NOT send NON messages while waiting for CON messages to be acknowledged (however, CON retransmissions should send the new resource state if it has changed since, see [I-D.ietf-core-observe] section 4.5).
3. require sending a CON message at least every 24 hours before continuing with NON messages.
4. consider increasing MAXNONRATE for servers that check the client that rarely (e.g., to the rate at which TCP sends into a very lossy channel, e.g., 7 B/s).

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

7. IANA Considerations

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

8. Security Considerations

(TBD. The security considerations of, e.g., [RFC2581], [RFC2914], and [RFC5405] apply. Some issues are already discussed in the security considerations of [I-D.ietf-core-coap].)

9. Acknowledgements

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

Michael Scharf did a review of CoAP congestion control issues that asked a lot of good questions that this draft attempts to answer.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

10.2. Informative References

- [GF] Bormann, C., "Garrulity and Fluff", IAB Smart Object Workshop, 2011, <<http://www.iab.org/wp-content/IAB-uploads/2011/04/Bormann.pdf>>.
- [HTTPBISc715] "Changeset 715", October 2009, <<http://trac.tools.ietf.org/wg/httpbis/trac/changeset/715>>.
- [HTTPBIS131] "increase connection limit", HTTPBIS ticket #131, closed 2009-12-02, September 2008, <<http://trac.tools.ietf.org/wg/httpbis/trac/ticket/131>>.
- [I-D.eggert-core-congestion-control] Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.
- [I-D.ietf-core-block] Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-08 (work in progress), February 2012.
- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-11 (work in progress), July 2012.
- [I-D.ietf-core-observe] Hartke, K., "Observing Resources in CoAP",

draft-ietf-core-observe-05 (work in progress), March 2012.

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

Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part 1: Message Routing and Syntax", draft-ietf-httpbis-pl-messaging-20 (work in progress), July 2012.

[I-D.ietf-tcpm-initcwnd]

Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", draft-ietf-tcpm-initcwnd-04 (work in progress), June 2012.

[I-D.mathis-tcpm-tcp-laminar]

Mathis, M., "Laminar TCP and the case for refactoring TCP congestion control", draft-mathis-tcpm-tcp-laminar-01 (work in progress), July 2012.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[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.

[RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, June 2001.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

[RFC4336] Floyd, S., Handley, M., and E. Kohler, "Problem Statement for the Datagram Congestion Control Protocol (DCCP)", RFC 4336, March 2006.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, September 2008.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

Authors' Addresses

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

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

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: April 25, 2014

C. Bormann
Universitaet Bremen TZI
October 22, 2013

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

Abstract

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

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

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

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

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

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 25, 2014.

Copyright Notice

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

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

Table of Contents

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

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

1. Introduction

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

1.1. Terminology

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

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

2. The Main Four

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

as the IEEE 802.15.4 based 6LoWPAN [RFC4919].

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

2.1. The CoAP protocol

The CoAP protocol is defined in three specifications:

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

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

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

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

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

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

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

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

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

2.2. Discovery

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

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

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

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

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

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

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

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

2.3. Further reading

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

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

3. Informational Drafts

3.1. Implementation

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

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

will include discussion of power-related terminology.

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

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

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

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

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

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

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

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

3.2. Multicast and Group Communication

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

This led to an additional milestone on the CoRE charter:

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

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

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

3.3. Security

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

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

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

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

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

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

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

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

3.4. Intermediaries

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

3.5. Congestion Control

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

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

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

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

4. CoAP over X

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

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

5. Optional components of CoRE

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

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

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

5.1. CoAP-misc

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

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

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

5.2. Generalizing Media Types

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

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

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

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

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

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

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

5.4. Extending Observe

5.5. Service discovery

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

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

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

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

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

5.6. Server discovery, Naming, etc.

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

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

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

5.7. More support for sleepy nodes

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

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

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

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

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

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

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

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

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

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

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

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

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

6. Replaced drafts

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

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

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

7. IANA Considerations

This document has no actions for IANA.

8. Security Considerations

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

9. Acknowledgements

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

Akbar Rahman contributed text to this roadmap.

10. References

10.1. Normative References

[I-D.ietf-core-block]

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

[I-D.ietf-core-coap]

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

[I-D.ietf-core-observe]

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

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

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

[I-D.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-07 (work in progress), August 2013.

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

10.2. Informative References

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

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

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

[I-D.garcia-core-security]

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

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

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

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

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

[I-D.greevenbosch-core-authreq]

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

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

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

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

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

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

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

[I-D.hartke-core-codtls]

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

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

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

[I-D.ietf-core-groupcomm]

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

[I-D.ietf-core-interfaces]

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

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

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

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

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

[I-D.ietf-lwig-cellular]

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

[I-D.ietf-lwig-guidance]

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

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

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

[I-D.ietf-lwig-terminology]

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

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

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

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

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

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

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

[I-D.kovatsch-lwig-coap]

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

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

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

[I-D.pporamba-dtls-certkey]

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

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

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

[I-D.rahman-core-sleepy]

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

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

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

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

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

Author's Address

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

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

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 5, 2013

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
KoanLogic
E. Dijk
Philips Research
July 4, 2012

Best Practices for HTTP-CoAP Mapping Implementation
draft-castellani-core-advanced-http-mapping-00

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 5, 2013.

Copyright Notice

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

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

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

Table of Contents

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

1. Terminology and Conventions

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

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

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

At least two different kinds of HC proxies exist:

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

2. Introduction

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

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

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

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

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

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

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

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

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

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

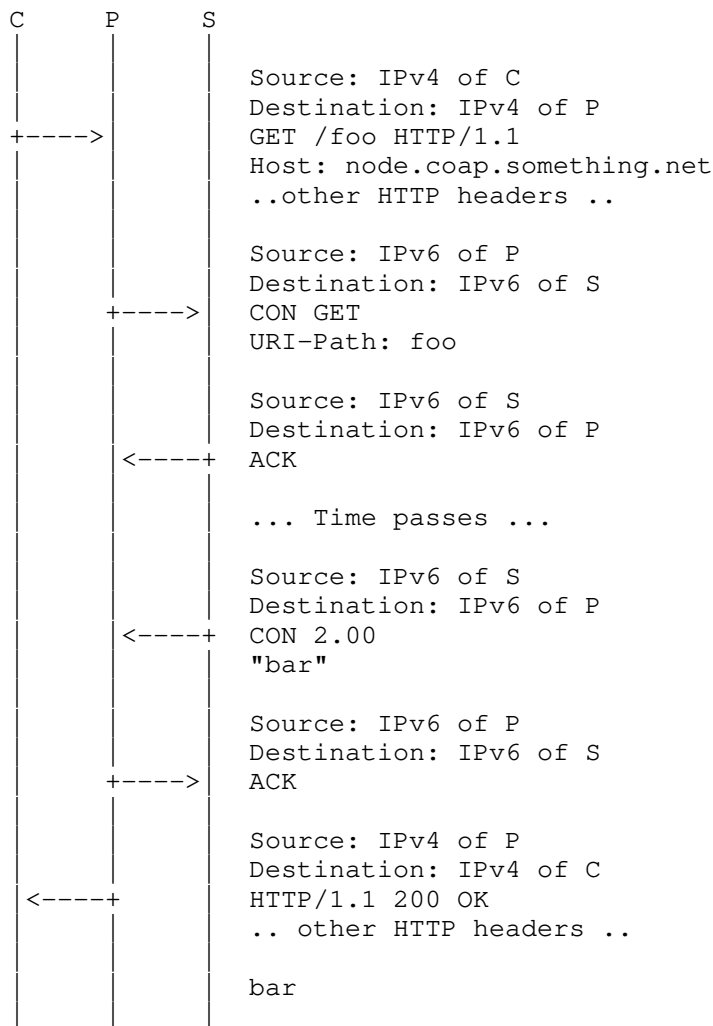


Figure 1: HTTP/IPv4 to CoAP/IPv6 Mapping

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

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

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

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

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

4. Multiple Message Exchanges Mapping

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

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

4.1. Relevant Features of Existing Standards

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

4.1.1. Multipart Messages

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

4.1.2. Immediate Message Delivery

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

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

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

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

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

4.1.3. Detailing Source Information

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

4.2. Multicast Mapping

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

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

4.2.1. URI Identification and Mapping

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

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

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

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

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

4.2.2. Request Handling

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

Multicast request handling consists of the following steps:

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

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

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

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

4.2.3. Examples

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

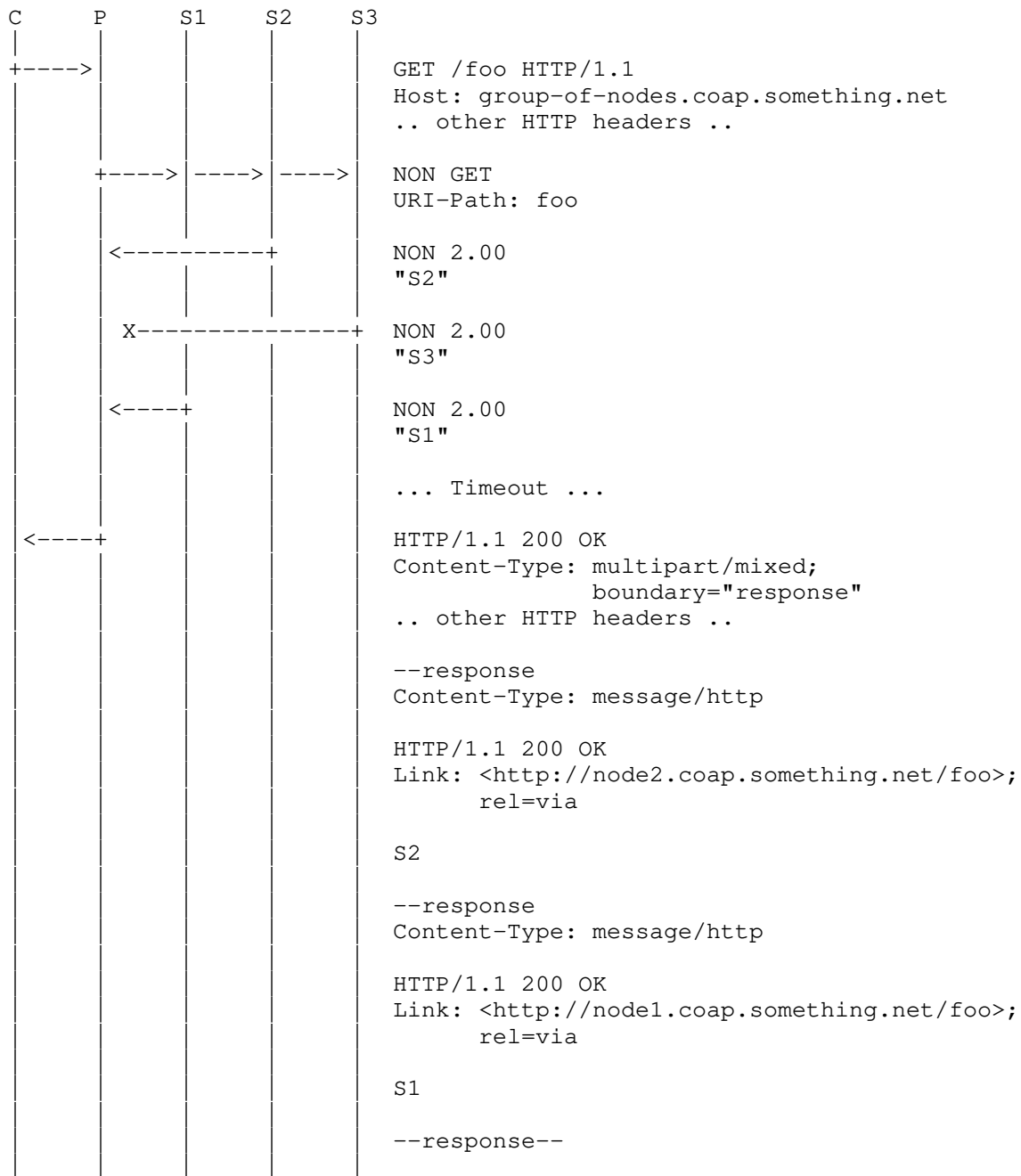


Figure 2: Unicast HTTP to Multicast CoAP Mapping

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

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

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

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

4.3. Multicast Response Caching

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

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

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

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

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

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

4.4. Observe Mapping

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

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

4.4.1. Identification

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

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

4.4.1.1. Observable URI Mapping

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

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

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

4.4.1.1.1. Discovery

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

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

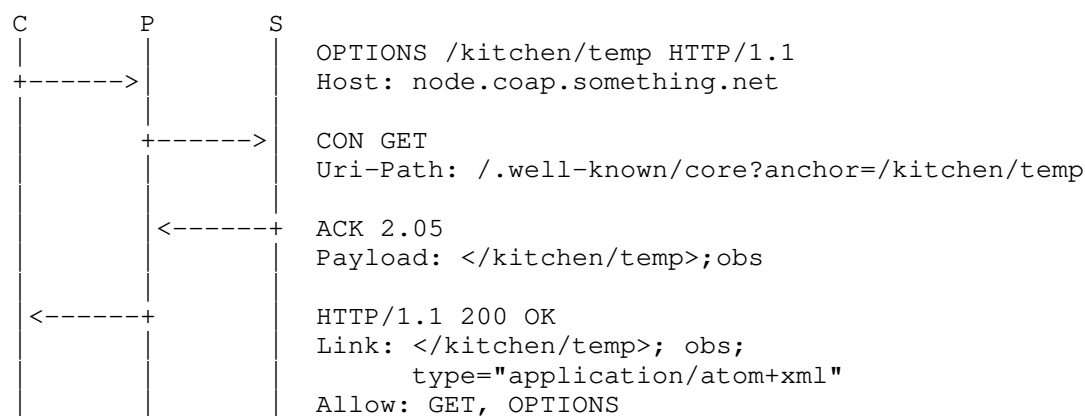


Figure 3: Discover Observability with HTTP OPTIONS

4.4.1.2. Differentiation Using HTTP Header

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

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

4.4.1.2.1. Expect Header

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

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

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

4.4.1.2.2. Prefer Header

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

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

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

4.4.2. Notification(s) Mapping

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

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

4.4.2.1. Multipart Messaging

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

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

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

4.4.2.2. Using ATOM Feeds

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

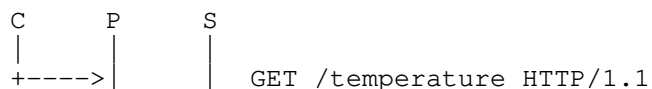
Figure 3 shows a resource supporting 'application/atom+xml' media-type. In such case clients can listen to update notification by regularly polling the resource via opportunely spaced GETs, i.e. driven by the advertised max-age value.

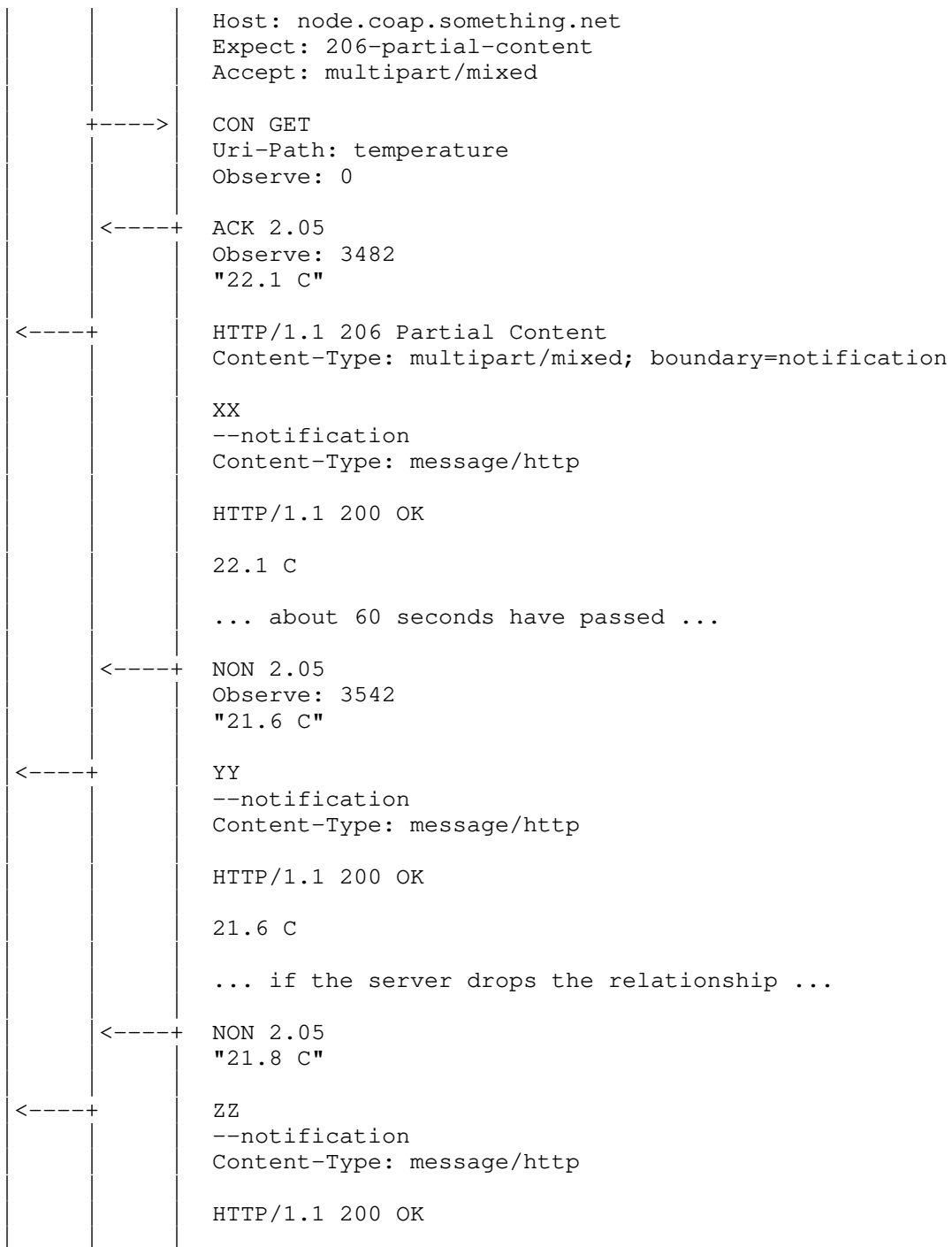
4.4.3. Examples

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

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

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





			21.8 C
			--notification--
			0

Figure 4: HTTP Streaming to CoAP Observe

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

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

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

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

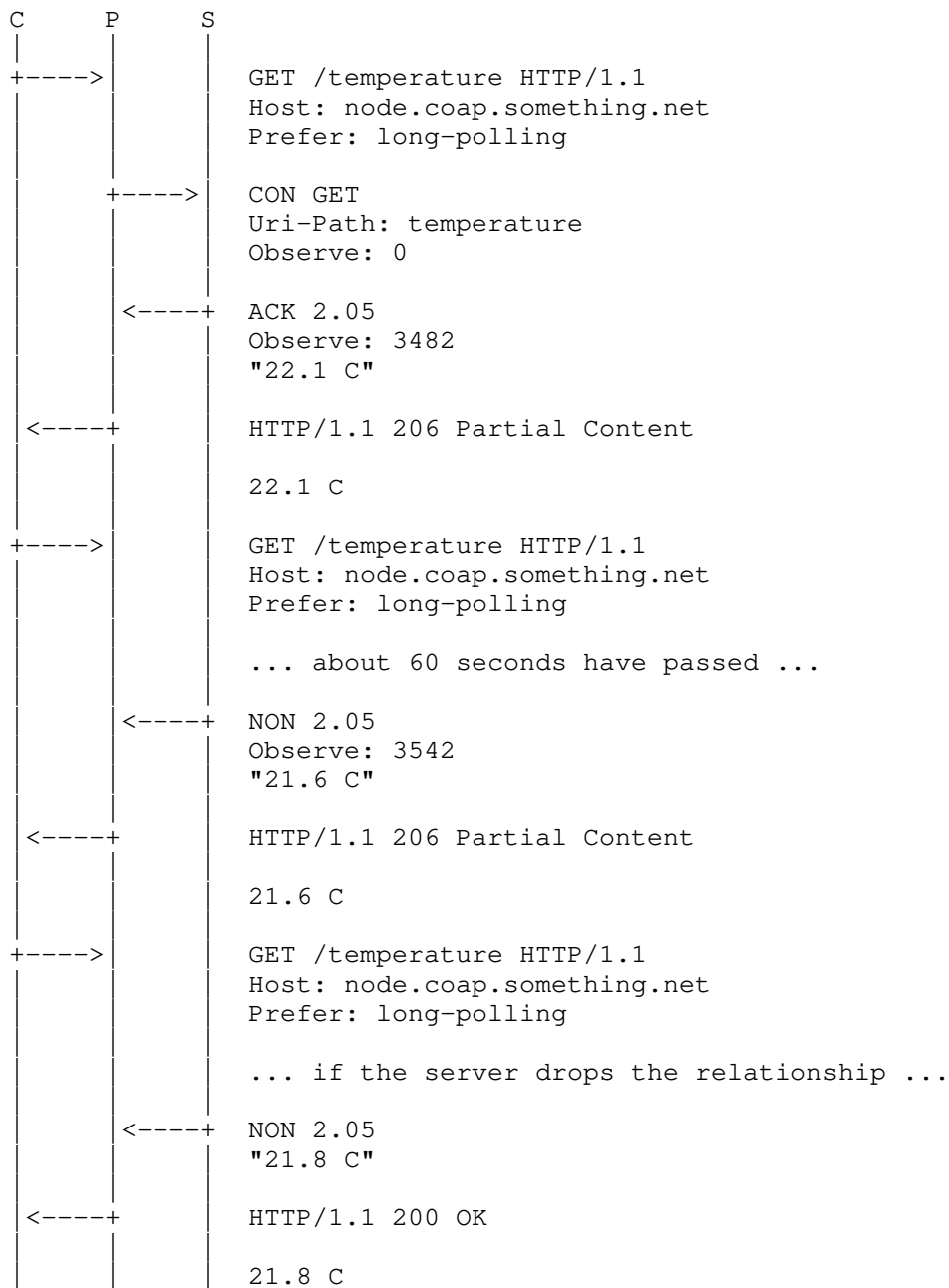


Figure 5: HTTP Long Polling to CoAP Observe

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

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

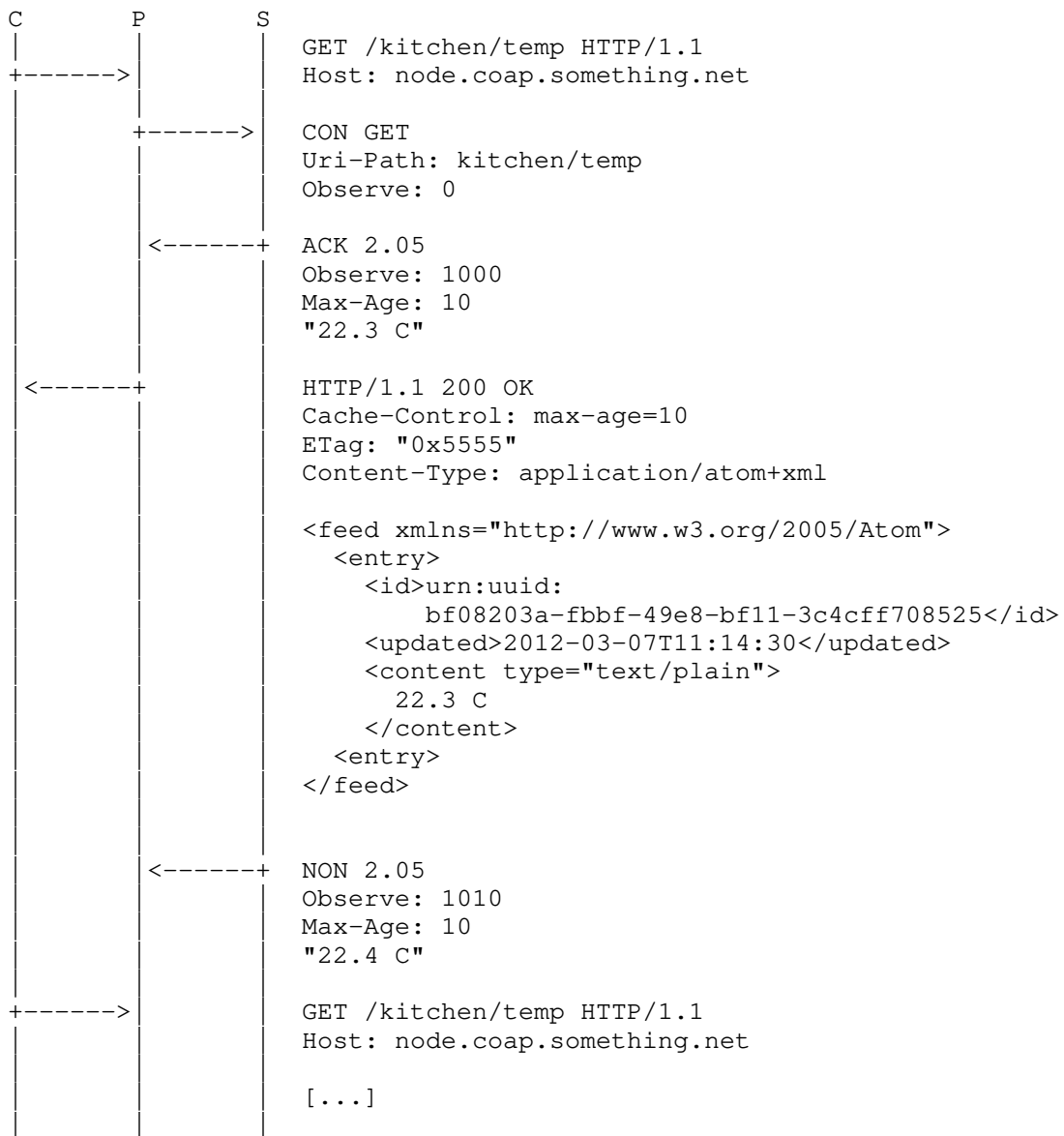


Figure 6: Observation via Atom feeds

5. HTML5 Scheme Handler Registration

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

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

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

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

6. Placement and Deployment

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

The arguments supporting CS placement are as follows:

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

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

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

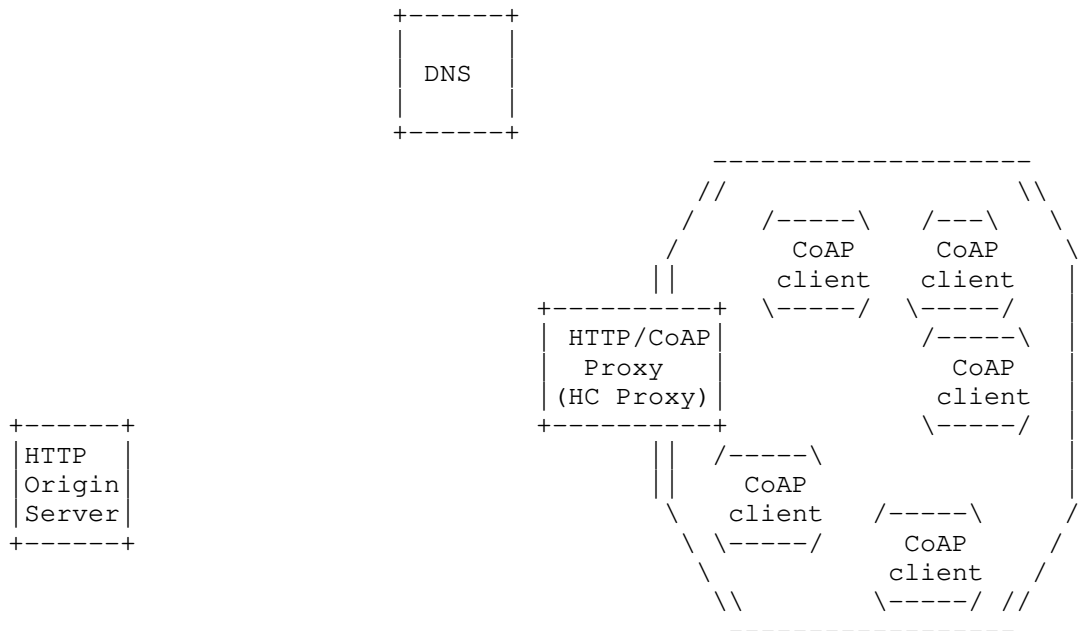


Figure 7: Client-side HC Proxy Deployment Scenario

7. Examples

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

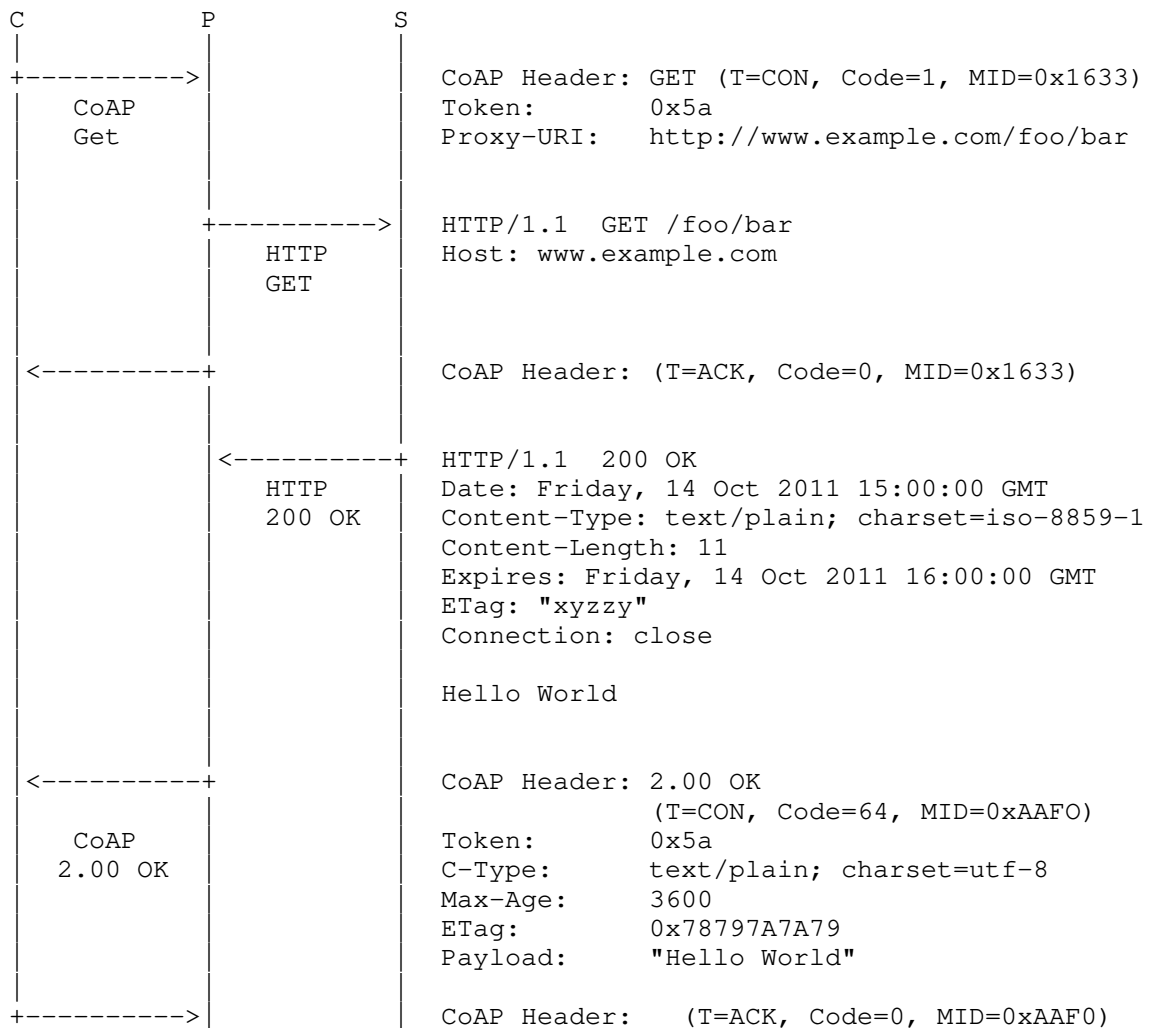


Figure 8: A Basic CoAP-HTTP GET Request

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

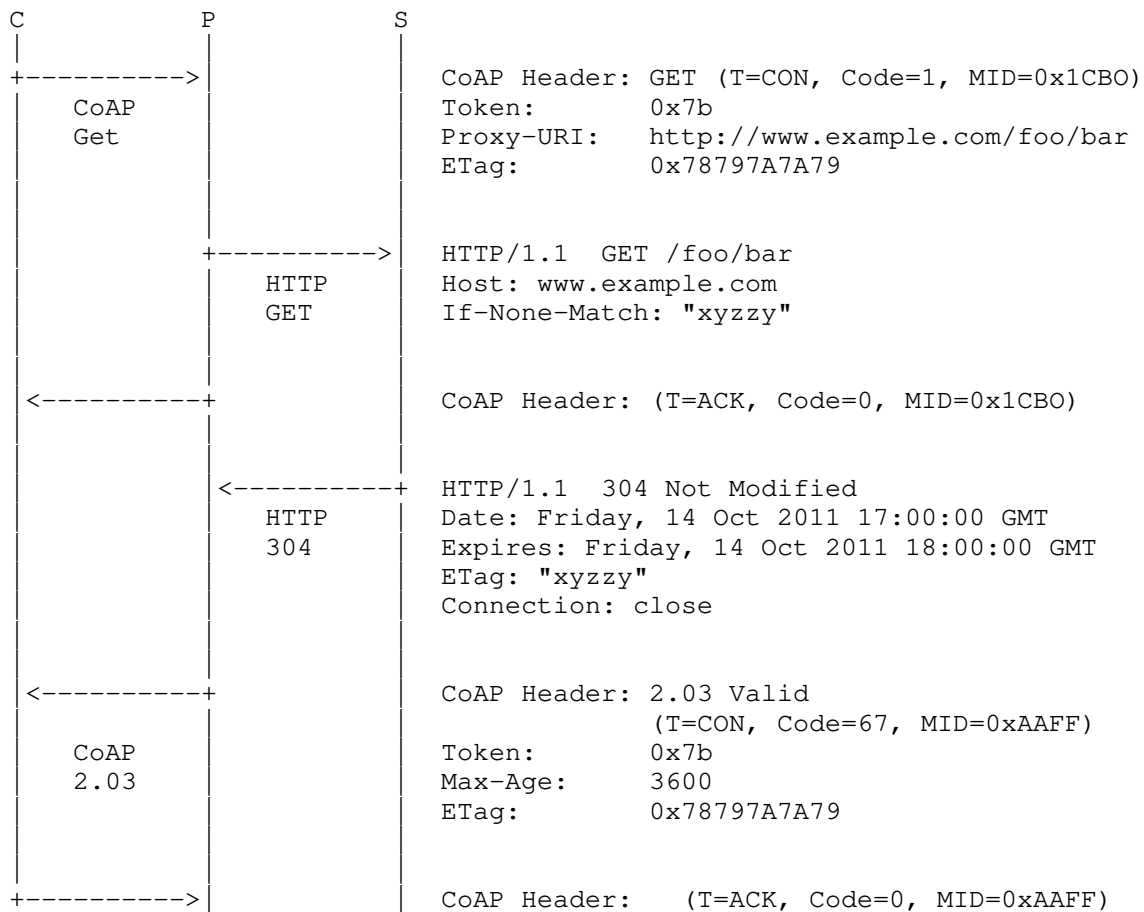


Figure 9: A CoAP-HTTP GET Request with an ETag Option

8. Acknowledgements

TBD.

9. IANA Considerations

This memo includes no request to IANA.

10. Security Considerations

10.1. Cross-protocol Security Policy Mapping

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

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

10.2. Subscription

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

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

11. References

11.1. Normative References

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

Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best practices for HTTP-CoAP mapping implementation", draft-castellani-core-http-mapping-04 (work in progress), April 2012.

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-08 (work in progress), February 2012.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-10 (work in progress), June 2012.

- [I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP",
draft-ietf-core-groupcomm-01 (work in progress),
March 2012.
- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP",
draft-ietf-core-observe-05 (work in progress), March 2012.
- [I-D.ietf-httpbis-p1-messaging]
Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part
1: URIs, Connections, and Message Parsing",
draft-ietf-httpbis-p1-messaging-19 (work in progress),
March 2012.
- [I-D.ietf-httpbis-p2-semantics]
Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part
2: Message Semantics", draft-ietf-httpbis-p2-semantics-19
(work in progress), March 2012.
- [I-D.thomson-hybi-http-timeout]
Thomson, M., Loreto, S., and G. Wilkins, "Hypertext
Transfer Protocol (HTTP) Keep-Alive Header",
draft-thomson-hybi-http-timeout-01 (work in progress),
March 2012.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail
Extensions (MIME) Part Two: Media Types", RFC 2046,
November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, January 2005.
- [RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom
Syndication Format", RFC 4287, December 2005.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

11.2. Informative References

- [I-D.bormann-core-simple-server-discovery]
Bormann, C., "CoRE Simple Server Discovery",
draft-bormann-core-simple-server-discovery-01 (work in
progress), March 2012.
- [I-D.shelby-core-resource-directory]
Shelby, Z. and S. Krco, "CoRE Resource Directory",
draft-shelby-core-resource-directory-03 (work in
progress), May 2012.
- [I-D.snell-http-prefer]
Snell, J., "Prefer Header for HTTP",
draft-snell-http-prefer-12 (work in progress),
February 2012.
- [I-D.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building
Control", draft-vanderstok-core-bc-05 (work in progress),
October 2011.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web
Replication and Caching Taxonomy", RFC 3040, January 2001.
- [RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-
Service Considerations", RFC 4732, December 2006.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins,
"Known Issues and Best Practices for the Use of Long
Polling and Streaming in Bidirectional HTTP", RFC 6202,
April 2011.
- [W3C.HTML5]
Hickson, I., "HTML5", World Wide Web Consortium WD (work
in progress) WD-html5-20111018, October 2011,
<<http://dev.w3.org/html5/spec/>>.

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

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

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

independent mapping source.

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

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

A.1. URL Map Algorithm

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

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

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

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

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

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

INPUT
* requested URL

OUTPUT
* target URL

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

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

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

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

A.2. Security Policy Map Algorithm

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

INPUT

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

OUTPUT

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

SYNTAX

```

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

```

EXAMPLE

```
<TBD>
```

A.3. Content-Type Map Algorithm

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

INPUT

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

OUTPUT

- * mapped content-type

SYNTAX

```

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

```

EXAMPLE

```

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

```

Authors' Addresses

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

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

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

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

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

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

Esko Dijk
Philips Research

Email: esko.dijk@philips.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: April 25, 2013

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
KoanLogic
E. Dijk
Philips Research
October 22, 2012

Best Practices for HTTP-CoAP Mapping Implementation
draft-castellani-core-http-mapping-06

Abstract

This draft provides reference information for HTTP-CoAP proxy implementors focusing primarily on the reverse proxy case. 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 April 25, 2013.

Copyright Notice

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

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

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

Table of Contents

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

1. Introduction

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

Section 10 of [I-D.ietf-core-coap] describes the fundamentals of the CoAP-HTTP (and vice-versa) cross-protocol mapping process. However, implementing such a cross-protocol proxy can be complex, and many details regarding its internal procedures and design choices require further elaboration. Therefore a first goal of this document is to provide more detailed information to proxy designers and implementers, to help implement proxies that correctly inter-work with other CoAP and HTTP client/server implementations that adhere to the 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 arbitrary (coincidental) variation in 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 different mapping approaches and the related proxy deployments;
- o Section 3 discusses impact of the mapping on URI and describes notable options;
- o Section 5 analyzes the mapping from HTTP to CoAP;
- o Section 7 discusses possible security impact related to cross-protocol mapping.

2. Terminology

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

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 [RFC3040].

Note: In this document we focus on the Reverse Proxy mode of the Cross-Protocol Proxy.

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", "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 means 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 is the first part of the URI, and it 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 has cross-protocol URIs referring to it.

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

4. HTTP to CoAP URI Mapping

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

As shown in the above example, if a cross-protocol URI exists,

scheme, authority and path parts of the URI may change. The process of providing cross URIs may be complex, since a mechanism to statically or dynamically (discover) map the URI is needed.

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

4.1. Homogeneous Mapping

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

Example: The CoAP resource `"/node.coap.something.net/foo"` can be accessed by an HTTP client by requesting `"http://node.coap.something.net/foo"`. The Cross-Protocol Proxy receiving the request is responsible to map the URI to `"coap://node.coap.something.net/foo"`

When homogeneous cross-protocol URIs are supported, HTTP to CoAP URI Mapping is easily implemented.

4.2. Embedded Mapping

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

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

When embedded mapping of URIs are supported, the complexity of a cross-protocol proxy is reduced.

4.3. Scheme Security Mapping

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

5. HTTP-CoAP Reverse Cross-Protocol 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.

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 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. Cross-Protocol Proxy Placement

Typically, a Cross-Protocol Proxy is expected to be located at the edge of the constrained network. See Figure 1. The arguments supporting this placement are the following:

TCP/UDP: Translation between HTTP and CoAP requires also TCP/UDP translation; TCP is preferred over UDP on the global Internet given their relative performance. To minimize required retransmissions and maximize overall reliability, TCP/UDP conversion SHOULD be performed as close to the server as possible.

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.

Multicast: To support CoAPs use of local-multicast functionalities that MAY be available in a constrained network, the Cross-Protocol Proxy MAY require a network interface directly attached to the constrained network.

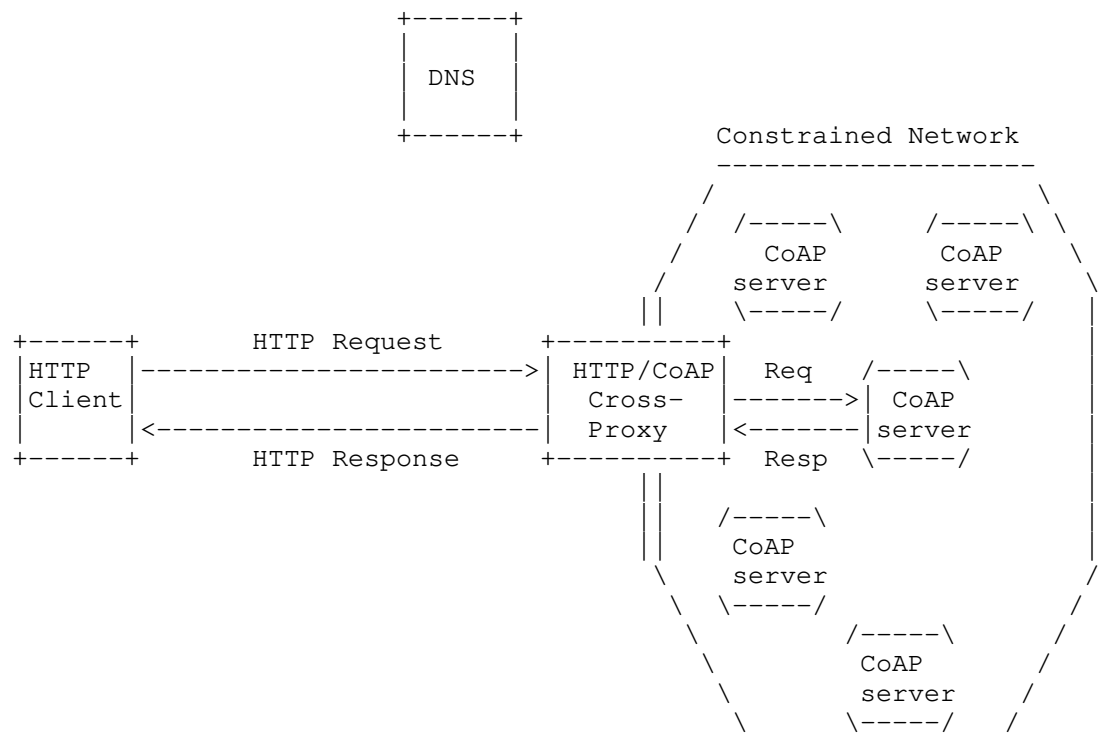


Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

5.2. Comparison of Proxy Placement Scenarios

Table 1 shows three relevant Cross-Protocol Proxy deployment scenarios and notes the advantages ('+') and disadvantages ('-') related to each scenario.

Feature	HCF CS	HCR SS	HCI SS
TCP/UDP usage	-	+	+
Multicast support	-	+	+
Caching efficiency	-	+	+
Scalability/Availability	+	+/-	+
Configuration needs	-	-	+

Table 1: Comparison of relevant Cross-Protocol Proxy deployments

It can be seen that SS deployment is typically preferred above CS. Scalability and Availability are usually improved using multiple redundant proxies. For HCR SS, Scalability/Availability can be provided but there is configuration overhead involved. For HCI this overhead is minimal. For HCF CS, there is the issue that static configuration of multiple forward proxies is typically not feasible in existing (legacy) HTTP clients.

5.3. 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 CoAP response code in the first 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 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 here, 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.

5.4. 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]).

5.5. Caching and Congestion Control

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

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

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

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

Resources experiencing a high access rate coupled with high volatility MAY be observed [I-D.ietf-core-observe] by the Cross-Protocol Proxy to keep their cached representation fresh while minimizing the number CoAP messages. See Section 5.6.

5.6. 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, sleeping 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.

5.7. Use of CoAP Blockwise Transfer

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

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

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

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

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

5.8. 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.

5.9. 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 5.5), 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 5.3 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-08 (work in progress), February 2012.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-12 (work in progress), October 2012.
- [I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-02 (work in progress), July 2012.
- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-06 (work in progress), September 2012.
- [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-21 (work in progress), October 2012.
- [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-21 (work in progress), October 2012.
- [I-D.ietf-httpbis-p7-auth]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", draft-ietf-httpbis-p7-auth-21 (work in progress), October 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

9.2. Informative References

- [I-D.bormann-core-simple-server-discovery]
Bormann, C., "CoRE Simple Server Discovery",
draft-bormann-core-simple-server-discovery-01 (work in
progress), March 2012.
- [I-D.shelby-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource
Directory", draft-shelby-core-resource-directory-04 (work
in progress), July 2012.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web
Replication and Caching Taxonomy", RFC 3040, January 2001.
- [RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-
Service Considerations", RFC 4732, December 2006.

Authors' Addresses

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

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

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

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

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

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

Esko Dijk
Philips Research

Email: esko.dijk@philips.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: April 22, 2013

E. Dijk, Ed.
Philips Research
A. Rahman, Ed.
InterDigital Communications, LLC
October 19, 2012

Miscellaneous CoAP Group Communication Topics
draft-dijk-core-groupcomm-misc-02

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on April 22, 2013.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	3
2. Requirements	3
2.1. Background	3
2.2. General Requirements	4
2.3. Security Requirements	5
3. Group Communication Solutions	7
3.1. IP Multicast Transmission Methods	7
3.1.1. Serial unicast	7
3.1.2. Unreliable IP Multicast	7
3.1.3. Reliable IP Multicast	7
3.2. Overlay Multicast	8
3.3. CoAP Application Layer Group Management	9
4. DNS-SD Based Group Resource Manipulation	12
5. Deployment Guidelines	12
5.1. Overview	12
5.2. Implementation in Target Network Topologies	12
5.2.1. Single LLN Topology	13
5.2.2. Single LLN with Backbone Topology	15
5.2.3. Multiple LLNs with Backbone Topology	17
5.2.4. LLN(s) with Multiple 6LBRs	17
5.2.5. Conclusions	17
5.3. Implementation Considerations	18
5.3.1. MLD Implementation on LLNs and MLD alternatives	18
5.3.2. 6LBR Implementation	19
5.3.3. Backbone IP Multicast Infrastructure	19
6. Miscellaneous Topics	20
7. Acknowledgements	20
8. IANA Considerations	20
9. Security Considerations	20
10. References	20
10.1. Normative References	20
10.2. Informative References	21
Appendix A. Multicast Listener Discovery (MLD)	22
Authors' Addresses	23

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 3) 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 6) 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. 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.

2.1. 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).

2.2. 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 2.3 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.

2.3. 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 secure 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.

3. 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].

3.1. IP Multicast Transmission Methods

3.1.1. 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.

3.1.2. 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.

3.1.3. 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)

3.2. 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.

3.3. CoAP Application Layer Group Management

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

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

The following group management primitives are in general useful:

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

In this proposal a (at least one) CoAP Proxy node is responsible for group membership management. A constrained node can specify which group it intends to join (or leave) using a CoAP request to the appropriate CoAP Proxy. To Join, the group name will be included in optional request header fields (explained below). These header fields will be included in a PUT request to the Proxy. The Proxy-URI is set to the Group Management URI of the Proxy (found previously through the "/.well-known/" resource discovery mechanism). Note that

in this solution also CoAP Proxies may exist in a network that are not capable of CoAP group operations.

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

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

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

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

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

Figure 1: CoAP Header Options for Group Management

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

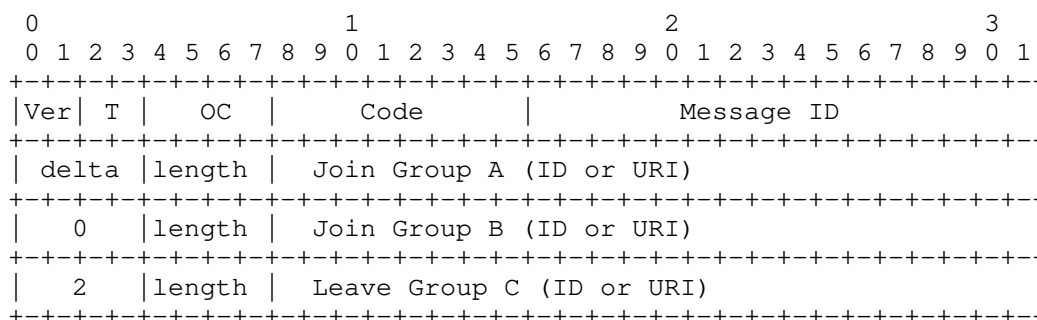


Figure 2: CoAP Message for Group Management

Header Fields for the above example:

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

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

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

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

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

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

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

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

4. 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.

5. Deployment Guidelines

5.1. 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.

5.2. 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.)

5.2.1. 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.

5.2.1.1. 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.

5.2.1.2. 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.

5.2.1.3. 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 5.3.1.

5.2.1.4. 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.

5.2.1.5. 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 5.3.1 for more efficient substitutes for MLD targeted towards a LLN context.

5.2.2. 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.

5.2.2.1. 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 5.3.1.

5.2.2.2. 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.

5.2.2.3. 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 5.2.1.3.

5.2.2.4. 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 5.2.2.1.

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 5.2.2.1) 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 5.2.2.1 or to use an MLD alternative suitable for LLNs as described in

Section 5.3.1. 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.

5.2.2.5. 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.

5.2.3. 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.

5.2.4. 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?]

5.2.5. 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 5.3.1.

5.3. Implementation Considerations

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

5.3.1. 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 [I-D.ietf-6lowpan-nd] 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 hosts'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]

5.3.2. 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.

5.3.3. 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.]

6. Miscellaneous Topics

This section is a placeholder to add miscellaneous text, topics or proposals related to CoAP group communication in future versions of this document.

7. Acknowledgements

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

8. IANA Considerations

This memo includes no request to IANA.

9. 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.

10. References

10.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-12 (work in progress), October 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.

10.2. Informative References

- [I-D.castellani-core-advanced-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation",
draft-castellani-core-advanced-http-mapping-00 (work in progress), July 2012.

- [I-D.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-nd]
Shelby, Z., Chakrabarti, S., and E. Nordmark, "Neighbor Discovery Optimization for Low Power and Lossy Networks (6LoWPAN)", draft-ietf-6lowpan-nd-21 (work in progress), August 2012.
- [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-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP",
draft-ietf-core-groupcomm-02 (work in progress), July 2012.
- [I-D.ietf-roll-trickle-mcast]
Hui, J. and R. Kelsey, "Multicast Forwarding Using Trickle", draft-ietf-roll-trickle-mcast-01 (work in progress), July 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.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-05 (work in progress), October 2011.

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.

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

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: August 18, 2013

M. Ersue, Ed.
Nokia Siemens Networks
D. Romascanu, Ed.
Avaya
J. Schoenwaelder, Ed.
Jacobs University Bremen
February 14, 2013

Management of Networks with Constrained Devices: Problem Statement, Use
Cases and Requirements
draft-ersue-constrained-mgmt-03

Abstract

This document provides a problem statement and discusses the use cases and requirements for the management of networks with constrained devices.

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 18, 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. Overview	4
1.2. Terminology	5
1.3. Class of Networks in Focus	7
1.4. Constrained Device Deployment Options	10
1.5. Management Topology Options	11
1.6. Managing the Constrainedness of a Device or Network	11
2. Problem Statement	15
3. Use Cases	17
3.1. Environmental Monitoring	17
3.2. Medical Applications	17
3.3. Industrial Applications	18
3.4. Home Automation	19
3.5. Building Automation	20
3.6. Energy Management	22
3.7. Transport Applications	23
3.8. Infrastructure Monitoring	24
3.9. Community Network Applications	25
3.10. Mobile Applications	27
3.11. Automated Metering Infrastructure (AMI)	29
3.12. MANET Concept of Operations (CONOPS) in Military	31
4. Requirements on the Management of Networks with Constrained Devices	36
4.1. Management Architecture/System	36
4.2. Management protocols and data model	41
4.3. Configuration management	44
4.4. Monitoring functionality	46
4.5. Self-management	51
4.6. Security and Access Control	52
4.7. Energy Management	54
4.8. SW Distribution	56
4.9. Traffic management	56
4.10. Transport Layer	57
4.11. Implementation Requirements	59
5. IANA Considerations	61
6. Security Considerations	62
7. Contributors	63
8. Acknowledgments	64
9. References	65
9.1. Normative References	65
9.2. Informative References	65
Appendix A. Related Development in other Bodies	67

A.1.	ETSI TC M2M	67
A.2.	OASIS	68
A.3.	OMA	69
A.4.	IPSO Alliance	69
Appendix B.	Related Research Projects	71
Appendix C.	Open issues	72
Appendix D.	Change Log	73
D.1.	02-03	73
D.2.	01-02	74
D.3.	00-01	74
Authors'	Addresses	76

1. Introduction

1.1. Overview

Small devices with limited CPU, memory, and power resources, so called constrained devices (aka. sensor, smart object, or smart device) can constitute a network. Such a network of constrained devices itself may be constrained or challenged, e.g. with unreliable or lossy channels, wireless technologies with limited bandwidth and a dynamic topology, needing the service of a gateway or proxy to connect to the Internet. In other scenarios, the constrained devices can be connected to a non-constrained network using off-the-shelf protocol stacks.

Constrained devices might be in charge of gathering information in diverse settings including natural ecosystems, buildings, and factories and send the information to one or more server stations. Constrained devices may work under severe resource constraints such as limited battery and computing power, little memory and insufficient wireless bandwidth, and communication capabilities. A central entity, e.g., a base station or controlling server, might have more computational and communication resources and can act as a gateway between the constrained devices and the application logic in the core network.

Today diverse size of small devices with different resources and capabilities are becoming connected. Mobile personal gadgets, building-automation devices, cellular phones, Machine-to-machine (M2M) devices, etc. benefit from interacting with other "things" in the near or somewhere in the Internet. With this the Internet of Things (IoT) becomes a reality build up of uniquely identifiable objects (things). And over the next decade, this could grow to trillions of constrained devices and will greatly increase the Internet's size and scope.

Network management is characterized by monitoring network status, detecting faults, and inferring their causes, setting network parameters, and carrying out actions to remove faults, maintain normal operation, and improve network efficiency and application performance. The traditional network management application periodically collects information from a set of elements that are needed to manage, processes the data, and presents them to the network management users. Constrained devices, however, often have limited power, low transmission range, and might be unreliable. They might also need to work in hostile environments with advanced security requirements or need to be used in harsh environments for a long time without supervision. Due to such constraints, the management of a network with constrained devices offers different

type of challenges compared to the management of a traditional IP network.

The IETF has already done a lot of standardization work to enable the communication in IP networks and to manage such networks as well as the manifold type of nodes in these networks [RFC6632]. However, the IETF so far has not developed any specific technologies for the management of constrained devices and the networks comprised by constrained devices. IP-based sensors or constrained devices in such an environment, i.e., devices with very limited memory and CPU resources, use today application-layer protocols in an ad-hoc manner to do simple resource management and monitoring.

This document raises the questions on and aims to understand the use cases and requirements for the management of a network with constrained devices. The document especially aims to avoid recommending any particular solutions. Section 1.3 and Section 1.5 describe different topology options for the networking and management of constrained devices. Section 1.4 explains different deployment options for the networking of constrained devices. Section 2 provides a problem statement on the issue of the management of networked constrained devices. Section 3 lists diverse use cases and scenarios for the management from the network as well as from the application point of view. Section 4 lists requirements on the management of applications and networks with constrained devices. Note that the requirements in Section 4 need to be seen as standalone requirements. As of today this document does not recommend the realization of a profile of requirements.

1.2. Terminology

Concerning constrained devices and networks this document generally builds on the terminology defined in [LWIG-TERMS]. As such the terms like Constrained Device, Constrained Network, etc. are defined in [LWIG-TERMS].

The following terms are additionally used throughout this documentation:

AMI: (Advanced Metering Infrastructure) A system including hardware, software, and networking technologies that measures, collects, and analyzes energy usage, and communicates with a hierarchically deployed network of metering devices, either on request or on a schedule.

C0: Class 0 constrained device as defined in Section 3. of [LWIG-TERMS].

C1: Class 1 constrained device as defined in Section 3. of [LWIG-TERMS].

C2: Class 2 constrained device as defined in Section 3. of [LWIG-TERMS].

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

Intermediary entity: As defined in the CoAP document an intermediary entity can be a CoAP endpoint that acts both as a server and as a client towards (possibly via further intermediaries) an origin server. An intermediary entity can be used to support hierarchical management.

Network of Constrained Devices: A network to which constrained devices are connected. It may or may not be a Constrained Network (see [LWIG-TERMS] for the definition of the term Constrained Network).

M2M: (Machine to Machine) stands for the automatic data transfer between devices of different kind. In M2M scenarios a device (such as a sensor or meter) captures an event, which is relayed through a network (wireless, wired or hybrid) to an application.

MANET: Mobile Ad-hoc Networks, a self-configuring and infrastructureless network of mobile devices connected by wireless technologies.

Mote: A sensor node in a wireless network that is capable of performing some limited processing, gathering sensory information and communicating with other connected nodes in the network.

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

Smart Grid: An electrical grid that uses communication technologies to gather and act on information in an automated fashion to improve the efficiency, reliability and sustainability of the production and distribution of electricity.

Smart Meter: An electrical meter (in the context of a Smart Grid) that records consumption of electric energy in intervals of an hour or less and communicates that information at least daily back to the utility network for monitoring and billing purposes.

For a detailed discussion on the constrained networks as well as classes of constrained devices and their capabilities please see [LWIG-TERMS].

1.3. Class of Networks in Focus

In this document we differentiate following type of networks concerning their transport and communication technologies:

(Note that a network in general can involve constrained and non-constrained devices.)

- o Wireline non-constrained networks (CN0), e.g. an Ethernet-LAN with non-constrained and constrained devices involved.
- o A combination of wireline and wireless networks (CN1), which may or may not be mesh-based but have a multi-hop connectivity between constrained devices, utilizing dynamic routing in both the wireless and wireline portions of the network. CN1 usually support highly distributed applications with many nodes (e.g. environmental monitoring). CN1 tend to deal with large-scale multipoint-to-point systems with massive data flows. Wireless Mesh Networks (WMN), as a specific type of CN1 networks, use off-the-shelf radio technology such as Wi-Fi, WiMax, and cellular 3G/4G. WMNs are reliable based on the redundancy they offer and have often a more planned deployment to provide dynamic and cost effective connectivity over a certain geographic area.
- o A combination of wireline and wireless networks with point-to-point or point-to-multipoint communication (CN2) generally with single-hop connectivity to constrained devices, utilizing static routing over the wireless network. CN2 support short-range, point-to-point, low-data-rate, source-to-sink type of applications such as RFID systems, light switches, fire and smoke detectors, and home appliances. CN2 usually support confined short-range spaces such as a home, a factory, a building, or the human body. IEEE 802.15.1 (Bluetooth) and IEEE 802.15.4 are well-known examples of applicable standards for CN2 networks.
- o Mobile Adhoc networks (MANET) are self-configuring infrastructureless networks of mobile devices connected by wireless technologies. MANETs are based on point-to-point communications of devices moving independently in any direction and changing the links to other devices frequently. MANET devices do act as a router to forward traffic unrelated to their own use.

A CN0 is used for specific applications like Building Automation or Infrastructure Monitoring. However, CN1 and CN2 networks are

especially in the interest of the analysis on the management of constrained devices in this document.

Furthermore different network characteristics are determined by multiple dimensions: dynamicity of the topology, bandwidth, and loss rate. In the following, each dimension is explained, and networks in scope for this document are outlined:

Network Topology:

The topology of a network can be represented as a graph, with edges (i.e., links) and vertices (routers and hosts). Examples of different topologies include "star" topologies (with one central node and multiple nodes in one hop distance), tree structures (with each node having exactly one parent), directed acyclic graphs (with each node having one or more parents), clustered topologies (where one or more "cluster heads" are responsible for a certain area of the network), mesh topologies (fully distributed), etc.

Management protocols may take advantage of specific network topologies, for example by distributing large-scale management tasks amongst multiple distributed network management stations (e.g., in case of a mesh topology), or by using a hierarchical management approach (e.g., in case of a tree topology). These different management topology options are described in Section 1.6.

Note that in certain network deployments, such as community ad hoc networks (as described in Section 3.9, the topology is not pre-planned, and thus may be unknown for management purposes. In other use cases, such as industrial applications (as described in Section 3.3, the topology may be designed in advance and therefore taken advantage of when managing the network.

Dynamicity of the network topology:

The dynamicity of the network topology determines the rate of change of the graph per time. Such changes can occur due to different factors, such as mobility of nodes (e.g., in MANETs or cellular networks), duty cycles (for low-power devices enabling their network interface only periodically to transmit or receive packets), or unstable links (in particular wireless links with strongly fluctuating link quality).

Examples of different levels of dynamicity of the topology are Ethernet networks (with typically a very static topology) on the one side, and low-power and lossy networks (LLNs) on the other side. LLNs nodes often using duty cycles, operate on unreliable wireless links and are potentially mobile (e.g. for sensor networks).

The more the topology is dynamic, the more routing, transport and application layer protocols have to cope with interrupted connectivity and/or longer delays. For example, management protocols (with a given underlying transport protocol) that expect continuous session flows without changes of routes during a communication flow, may fail to operate.

Networks with a very low dynamicity (e.g. Ethernet) with no or infrequent topology changes (e.g. less than once every 30 minutes), are in-scope of this document if they are used with constrained devices (see e.g. the use case "Building Automation" in Section 3.5).

Traffic flows:

The traffic flow in a network determines from which sources data traffic is sent to which destinations in the network. Several different traffic flows are defined in [I-D.ietf-roll-terminology], including "point-to-point" (P2P), "multipoint-to-point" (MP2P), and "point-to-multipoint" (P2MP) flows as:

- o P2P: Point To Point. This refers to traffic exchanged between two nodes (regardless of the number of hops between the two nodes).
- o P2MP: Point-to-Multipoint traffic refers to traffic between one node and a set of nodes. This is similar to the P2MP concept in Multicast or MPLS Traffic Engineering.
- o MP2P: Multipoint-to-Point is used to describe a particular traffic pattern (e.g. MP2P flows collecting information from many nodes flowing inwards towards a collecting sink).

If one of these traffic patterns is predominant in a network, protocols (routing, transport, application) may be optimized for the specific traffic flow. For example, in a network with a tree topology and MP2P traffic, collection tree protocols are efficient to send data from the leaves of the tree to the root of the tree, via each node's parent.

Bandwidth:

The bandwidth of the network is the amount of data that can be sent per time between two communication end-points. It is usually determined by the link with the minimum bandwidth on the path from the source to the destination of data packets. The bandwidth in networks can range from a few Kilobytes per second (such as on some 802.15.4 link layers) to many Gigabytes per second (e.g., on fiber optics).

For management purposes, the management protocol typically requires to send information between the network management station and the clients, for monitoring or control purposes. If the available bandwidth is insufficient for the management protocol, packets will be buffered and eventually dropped, and thus management is not possible with such a protocol.

Networks without bandwidth limitation (e.g. Ethernet) are in-scope of this document if they are used with constrained devices (see the use case "Building Automation" in Section 3.5).

Loss rate:

The loss rate (or bit error rate) is the number of bit errors divided by the total number of bits transmitted. For wired networks, loss rates are typically extremely low, e.g. around 10^{-12} or 10^{-13} for the latest 10Gbit Ethernet. For wireless networks, such as 802.15.4, the bit error rate can be as high as 10^{-1} to 10^0 in case of interferences. Even when using a reliable transport protocol, management operations can fail if the loss rate is too high, unless they are specifically designed to cope with these situations.

Note: The discussion on the management requirements of MANETs is currently not in the focus of this document. The use case in Section 3.4 has been provided to make it clear how a MANET-based application differs from others.

1.4. Constrained Device Deployment Options

We differentiate following Deployment options for the constrained devices:

- o a network of constrained devices, which communicate with each other,
- o Constrained devices, which are connected directly to the Internet or an IP network
- o A network of constrained devices which communicate with a gateway or proxy with more communication capabilities acting possibly as a representative of the device to entities in the non-constrained network
- o Constrained devices, which are connected to the Internet or an IP network via a gateway/proxy
- o A hierarchy of constrained devices, e.g., a network of C0 devices connected to one or more C1 devices - connected to one or more C2

devices - connected to one or more gateways - connected to some application servers or NMS system

- o The possibility of device grouping (possibly in a dynamic manner) such as that the grouped devices can act as one logical device at the edge of the network and one device in this group can act as the managing entity

1.5. Management Topology Options

We differentiate following options for the management of networks of constrained devices:

- o A network of constrained devices managed by one central manager. A logically centralized management might be implemented in a hierarchical fashion for scalability and robustness reasons. The manager and the management application logic might have a gateway/proxy in between or might be on different nodes in different networks, e.g., management application running on a cloud server.
- o Distributed management, where a constrained network is managed by more than one manager. Each manager controls a subnetwork and may communicate directly with other manager stations in a cooperative fashion. The distributed management may be weakly distributed, where functions are broken down and assigned to many managers dynamically, or strongly distributed, where almost all managed things have embedded management functionality and explicit management disappears, which usually comes with the price that the strongly distributed management logic now needs to be managed.
- o Hierarchical management, where a hierarchy of constrained networks are managed by the managers at their corresponding hierarchy level. I.e. each manager is responsible for managing the nodes in its sub-network. It passes information from its sub-network to its higher-level manager, and disseminates management functions received from the higher-level manager to its sub-network. Hierarchical management is essentially a scalability mechanism, logically the decision-making may be still centralized.

1.6. Managing the Constrainedness of a Device or Network

The capabilities of a constrained device or network and the constrainedness thereof influence and have an impact on the requirements for the management of such network or devices.

A constrained device:

- o might only support an unreliable radio with lossy links, i.e. the client and server of a management protocol need to gracefully ignore incomplete commands or repeat commands as necessary.
- o might only be able to go online from time-to-time, where it is reachable, i.e. a command might be necessary to repeat after a longer timeout or the timeout value with which one endpoint waits on a response needs to be sufficiently high.
- o might only be able to support a limited operating time (e.g. based on the available battery), i.e. the devices need to economize their energy usage with suitable mechanisms and the managing entity needs to monitor and control the energy status of the constrained devices it manages.
- o might only be able to support one simple communication protocol, i.e. the management protocol needs to be possible to downscale from constrained (C2) to very constrained (C0) devices with modular implementation and a very basic version with just a few simple commands.
- o might only be able to support limited or no user and/or transport security, i.e. the management system needs to support a less-costly and simple but sufficiently secure authentication mechanism.
- o might not be able to support compression and decompression of exchanged data based on limited CPU power, i.e. an intermediary entity which is capable of data compression should be able to communicate with both, devices, which support data compression (e.g. C2) and devices, which do not support data compression (e.g. C1 and C0).
- o might only be able to support very simple encryption, i.e. it would be efficient if the devices use cryptographic algorithms that are supported in hardware.
- o might only be able to communicate with one single managing entity and cannot support the parallel access of many managing entities.
- o might depend on a self-configuration feature, i.e. the managing entity might not know all devices in a network and the device needs to be able to initiate connection setup for the device configuration.
- o might depend on self- or neighbor-monitoring feature, i.e. the managing entity might not be able to monitor all devices in a network continuously.

- o might only be able to communicate with its neighbors, i.e. the device should be able to get its configuration from a neighbor.
- o might only be able to support parsing of data models with limited size, i.e. the device data models need to be compact containing the most necessary data and if possible parsable as a stream.
- o might only be able to support a limited or no failure detection, i.e. the managing entity needs to handle the situation, where a failure does not get detected or gets detected late gracefully e.g. with asking repeatedly.
- o might only be able to support the reporting of just one or a limited set failure types.
- o might only be able to support a limited set of notifications, possible only an "I-am-alive" message.
- o might only be able to support a soft-reset from failure recovery.
- o might possibly generate a huge amount of redundant reporting data, i.e. the intermediary management entity should be able to filter and aggregate redundant data.

A constrained network:

- o might only support an unreliable radio with lossy links, i.e. the client and server of a management protocol need to repeat commands as necessary or gracefully ignore incomplete commands.
- o might be necessary to manage based on multicast communication, i.e. the managing entity needs to be prepared to configure many devices at once based on the same data model.
- o might have a very large topology supporting 10.000 or more nodes for some applications and as such node naming is a specific issue for constrained networks.
- o must be able to self-organize, i.e. given the large number of nodes and their potential placement in hostile locations and frequently changing topology, manual configuration is typically not feasible. As such the network must be able to reconfigure itself so that it can continue to operate properly and support reliable connectivity.
- o needs a management solution, which is energy-efficient, using as little wireless bandwidth as possible since communication is highly energy demanding.

- o needs to support localization schemes to determine the location of devices since the devices might be moving and location information is important for some applications.
- o needs a management solution, which is scalable as the network may consist of thousands of nodes and may need to be extended continuously.
- o needs to provide fault tolerance. Faults in network operation including hardware and software errors, failures detected by the transport protocol and other self-monitoring mechanisms can be used to provide fault tolerance.
- o might require new management capabilities: for example, network coverage information and a constrained device power-distribution-map.
- o might require a new management function for data management, since the type and amount of data collected in constrained networks is different from those of the traditional networks.
- o might also need energy-efficient key management algorithms for security.

2. Problem Statement

The terminology for the "Internet of Things" is still nascent, and depending on the network type or layer in focus diverse technologies and terms are in use. Common to all these considerations is the "Things" or "Objects" are supposed to have physical or virtual identities using interfaces to communicate. In this context, we need to differentiate between the Constrained and Smart Devices identified by an IP address compared to virtual entities such as Smart Objects, which can be identified as a resource or a virtual object by using a unique identifier. Furthermore, the smart devices usually have a limited memory and CPU power as well as aim to be self-configuring and easy to deploy.

However, the tininess of the network nodes requires a rethinking of the protocol characteristics concerning power consumption, performance, memory, and CPU usage. As such, there is a demand for protocol simplification, energy-efficient communication, less CPU usage and small memory footprint.

On the application layer the IETF is already developing protocols like the Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] supporting constrained devices and networks e.g., for smart energy applications or home automation environments. The deployment of such an environment involves in fact many, in some scenarios up to million small devices (e.g. smart meters), which produce a huge amount of data. This data needs to be collected, filtered, and pre-processed for further use in diverse services.

Considering the high number of nodes to deploy, one has to think on the manageability aspects of the smart devices and plan for easy deployment, configuration, and management of the networks of constrained devices as well as the devices themselves. Consequently, seamless monitoring and self-configuration of such network nodes becomes more and more imperative. Self-configuration and self-management is already a reality in the standards of some of the bodies such as 3GPP. To introduce self-configuration of smart devices successfully a device-initiated connection establishment is required.

A simple application layer protocol, such as CoAP, is essential to address the issue of efficient object-to-object communication and information exchange. Such an information exchange should be done based on interoperable data models to enable the exchange and interpretation of diverse application and management related data.

In an ideal world, we would have only one network management protocol for monitoring, configuration, and exchanging management data,

independently of the type of the network (e.g., Smart Grid, wireless access, or core network). Furthermore, it would be desirable to derive the basic data models for constrained devices from the core models used today to enable reuse of functionality and end-to-end information exchange. However, the current management protocols seem to be too heavyweight compared to the capabilities the constrained devices have and are not applicable directly for the use in a network of constrained devices. Furthermore, the data models addressing the requirements of such smart devices need yet to be designed.

The IETF so far has not developed any specific technologies for the management of constrained devices and the networks comprised by constrained devices. IP-based sensors or constrained devices in such an environment, i.e., devices with very limited memory and CPU resources, use today, e.g., application-layer protocols to do simple resource management and monitoring. This might be sufficient for some basic cases, however, there is a need to reconsider the network management mechanisms based on the new, changed, as well as reduced requirements coming from smart devices and the network of such constrained devices. Albeit it is questionable whether we can take the same comprehensive approach we use in an IP network also for the management of constrained devices. Hence, the management of a network with constrained devices might become necessary to design as much as possible simplified and less complex.

As the Section 1.6 highlights, there are diverse characteristics of constrained devices or networks, which stem from their constrainedness and therefore have an impact on the requirements for the management of such a network with constrained devices. The use cases discussed in Section 3 show that the requirements on constrained networks are manifold and need to be analyzed from different angles, e.g. concerning the design of the management architecture, the selection of the appropriate protocol features as well as the specific issues which are new in the context of constrained devices. Examples of such issues are e.g. the careful management of the scarce energy resources, the necessity for self-organization and self-management of such devices but also the implementation considerations to enable the use of common communication technologies on a constrained hardware in an efficient manner. For an exhaustive list of issues and requirements, which need to be addressed for the management of a network with constrained devices please see Section 1.6 and Section 4.

3. Use Cases

This section discusses some application scenarios where networks of constrained devices are expected to be deployed. For each application scenario, we first briefly describe the characteristics followed by a discussion how network management can be provided, who is likely going to be responsible for it, and on which time-scale management operations are likely to be carried out.

3.1. Environmental Monitoring

Environmental monitoring applications are characterized by the deployment of a number of sensors to monitor emissions, water quality, or even the movements and habits of wildlife. Other applications in this category include earthquake or tsunami early-warning systems. The sensors often span a large geographic area, they can be mobile, and they are often difficult to replace. Furthermore, the sensors are usually not protected against tampering.

Management of environmental monitoring applications is largely concerned with the monitoring whether the system is still functional and the roll-out of new constrained devices in case the system loses too much of its structure. The constrained devices themselves need to be able to establish connectivity (auto-configuration) and they need to be able to deal with events such as losing neighbors or being moved to other locations.

Management responsibility typically rests with the organization running the environmental monitoring application. Since these monitoring applications must be designed to tolerate a number of failures, the time scale for detecting and recording failures is for some of these applications likely measured in hours and repairs might easily take days. However, for certain environmental monitoring applications, much tighter time scales may exist and might be enforced by regulations (e.g., monitoring of nuclear radiation).

3.2. Medical Applications

Constrained devices can be seen as an enabling technology for advanced and possibly remote health monitoring and emergency notification systems, ranging from blood pressure and heart rate monitors to advanced devices capable to monitor implanted technologies, such as pacemakers or advanced hearing aids. Medical sensors may not only be attached to human bodies, they might also exist in the infrastructure used by humans such as bathrooms or kitchens. Medical applications will also be used to ensure treatments are being applied properly and they might guide people losing orientation. Fitness and wellness applications, such as

connected scales or wearable heart monitors, encourage consumers to exercise and empower self-monitoring of key fitness indicators. Different applications use Bluetooth, Wi-Fi or Zigbee connections to access the patient's smartphone or home cellular connection to access the Internet.

Constrained devices that are part of medical applications are managed either by the users of those devices or by an organization providing medical (monitoring) services for physicians. In the first case, management must be automatic and or easy to install and setup by average people. In the second case, it can be expected that devices be controlled by specially trained people. In both cases, however, it is crucial to protect the privacy of the people to which medical devices are attached. Even though the data collected by a heart beat monitor might be protected, the pure fact that someone carries such a device may need protection. As such, certain medical appliances may not want to participate in discovery and self-configuration protocols in order to remain invisible.

Many medical devices are likely to be used (and relied upon) to provide data to physicians in critical situations since the biggest market is likely elderly and handicapped people. As such, fault detection of the communication network or the constrained devices becomes a crucial function that must be carried out with high reliability and, depending on the medical appliance and its application, within seconds.

3.3. Industrial Applications

Industrial Applications and smart manufacturing refer not only to production equipment, but also to a factory that carries out centralized control of energy, HVAC (heating, ventilation, and air conditioning), lighting, access control, etc. via a network. For the management of a factory it is becoming essential to implement smart capabilities. From an engineering standpoint, industrial applications are intelligent systems enabling rapid manufacturing of new products, dynamic response to product demand, and real-time optimization of manufacturing production and supply chain networks. Potential industrial applications e.g. for smart factories and smart manufacturing are:

- o Digital control systems with embedded, automated process controls, operator tools, as well as service information systems optimizing plant operations and safety.
- o Asset management using predictive maintenance tools, statistical evaluation, and measurements maximizing plant reliability.

- o Smart sensors detecting anomalies to avoid abnormal or catastrophic events.
- o Smart systems integrated within the industrial energy management system and externally with the smart grid enabling real-time energy optimization.

Sensor networks are an essential technology used for smart manufacturing. Measurements, automated controls, plant optimization, health and safety management, and other functions are provided by a large number of networked sectors. Data interoperability and seamless exchange of product, process, and project data are enabled through interoperable data systems used by collaborating divisions or business systems. Intelligent automation and learning systems are vital to smart manufacturing but must be effectively integrated with the decision environment. Wireless sensor networks (WSN) have been developed for machinery Condition-based Maintenance (CBM) as they offer significant cost savings and enable new functionalities. Inaccessible locations, rotating machinery, hazardous areas, and mobile assets can be reached with wireless sensors. WSNs can provide today wireless link reliability, real-time capabilities, and quality-of-service and enable industrial and related wireless sense and control applications.

Management of industrial and factory applications is largely focused on the monitoring whether the system is still functional, real-time continuous performance monitoring, and optimization as necessary. The factory network might be part of a campus network or connected to the Internet. The constrained devices in such a network need to be able to establish configuration themselves (auto-configuration) and might need to deal with error conditions as much as possible locally. Access control has to be provided with multi-level administrative access and security. Support and diagnostics can be provided through remote monitoring access centralized outside of the factory.

Management responsibility is typically owned by the organization running the industrial application. Since the monitoring applications must handle a potentially large number of failures, the time scale for detecting and recording failures is for some of these applications likely measured in minutes. However, for certain industrial applications, much tighter time scales may exist, e.g. in real-time, which might be enforced by the manufacturing process or the use of critical material.

3.4. Home Automation

Home automation includes the control of lighting, heating, ventilation, air conditioning, appliances, and entertainment devices

to improve convenience, comfort, energy efficiency, and security. It can be seen as a residential extension of building automation.

Home automation networks need a certain amount of configuration (associating switches or sensors to actors) that is either provided by electricians deploying home automation solutions or done by residents by using the application user interface to configure (parts of) the home automation solution. Similarly, failures may be reported via suitable interfaces to residents or they might be recorded and made available to electricians in charge of the maintenance of the home automation infrastructure.

The management responsibility lies either with the residents or it may be outsourced to electricians providing management of home automation solutions as a service. The time scale for failure detection and resolution is in many cases likely counted in hours to days.

3.5. Building Automation

Building automation comprises the distributed systems designed and deployed to monitor and control the mechanical, electrical and electronic systems inside buildings with various destinations (e.g., public and private, industrial, institutions, or residential). Advanced Building Automation Systems (BAS) may be deployed concentrating the various functions of safety, environmental control, occupancy, security. More and more the deployment of the various functional systems is connected to the same communication infrastructure (possibly Internet Protocol based), which may involve wired or wireless communications networks inside the building.

Building automation requires the deployment of a large number (10-100.000) of sensors that monitor the status of devices, and parameters inside the building and controllers with different specialized functionality for areas within the building or the totality of the building. Inter-node distances between neighboring nodes vary between 1 to 20 meters. Contrary to home automation in building management all devices are known to a set of commissioning tools and a data storage, such that every connected device has a known origin. The management includes verifying the presence of the expected devices and detecting the presence of unwanted devices.

Examples of functions performed by such controllers are regulating the quality, humidity, and temperature of the air inside the building and lighting. Other systems may report the status of the machinery inside the building like elevators, or inside the rooms like projectors in meeting rooms. Security cameras and sensors may be deployed and operated on separate dedicated infrastructures connected

to the common backbone. The deployment area of a BAS is typically inside one building (or part of it) or several buildings geographically grouped in a campus. A building network can be composed of subnets, where a subnet covers a floor, an area on the floor, or a given functionality (e.g. security cameras).

Some of the sensors in Building Automation Systems (for example fire alarms or security systems) register, record and transfer critical alarm information and therefore must be resilient to events like loss of power or security attacks. This leads to the need that some components and subsystems operate in constrained conditions and are separately certified. Also in some environments, the malfunctioning of a control system (like temperature control) needs to be reported in the shortest possible time. Complex control systems can misbehave, and their critical status reporting and safety algorithms need to be basic and robust and perform even in critical conditions.

Building Automation solutions are deployed in some cases in newly designed buildings, in other cases it might be over existing infrastructures. In the first case, there is a broader range of possible solutions, which can be planned for the infrastructure of the building. In the second case the solution needs to be deployed over an existing structure taking into account factors like existing wiring, distance limitations, the propagation of radio signals over walls and floors. As a result, some of the existing WLAN solutions (e.g. IEEE 802.11 or IEEE 802.15) may be deployed. In mission-critical or security sensitive environments and in cases where link failures happen often, topologies that allow for reconfiguration of the network and connection continuity may be required. Some of the sensors deployed in building automation may be very simple constrained devices for which class 0 or class 1 may be assumed.

For lighting applications, groups of lights must be defined and managed. Commands to a group of light must arrive within 200 ms at all destinations. The installation and operation of a building network has different requirements. During the installation, many stand-alone networks of a few to 100 nodes co-exist without a connection to the backbone. During this phase, the nodes are identified with a network identifier related to their physical location. Devices are accessed from an installation tool to connect them to the network in a secure fashion. During installation, the setting of parameters to common values to enable interoperability may occur (e.g. Trickle parameter values). During operation, the networks are connected to the backbone while maintaining the network identifier to physical location relation. Network parameters like address and name are stored in DNS. The names can assist in determining the physical location of the device.

3.6. Energy Management

EMAN working group developed [I-D.ietf-eman-framework], which defines a framework for providing Energy Management for devices within or connected to communication networks. This document observes that one of the challenges of energy management is that a power distribution network is responsible for the supply of energy to various devices and components, while a separate communication network is typically used to monitor and control the power distribution network. Devices that have energy management capability are defined as Energy Devices and identified components within a device (Energy Device Components) can be monitored for parameters like Power, Energy, Demand and Power Quality. If a device contains batteries, they can be also monitored and managed.

Energy devices differ in complexity and may include basic sensors or switches, specialized electrical meters, or power distribution units (PDU), and subsystems inside the network devices (routers, network switches) or home or industrial appliances. An Energy Management System is a combination of hardware and software used to administer a network with the primary purpose being Energy Management. The operators of such a system are either the utility providers or customers that aim to control and reduce the energy consumption and the associated costs. The topology in use differs and the deployment can cover areas from small surfaces (individual homes) to large geographical areas. EMAN requirements document [I-D.ietf-eman-requirements] discusses the requirements for energy management concerning monitoring and control functions.

It is assumed that Energy Management will apply to a large range of devices of all classes and networks topologies. Specific resource monitoring like battery utilization and availability may be specific to devices with lower physical resources (device classes C0 or C1).

Energy Management is especially relevant to Smart Grid. A Smart Grid is an electrical grid that uses data networks to gather and act on energy and power-related information, in an automated fashion with the goal to improve the efficiency, reliability, economics, and sustainability of the production and distribution of electricity. As such Smart Grid provides sustainable and reliable generation, transmission, distribution, storage and consumption of electrical energy based on advanced energy and ICT solutions and as such enables e.g. following specific application areas: Smart transmission systems, Demand Response/Load Management, Substation Automation, Advanced Distribution Management, Advanced Metering Infrastructure (AMI), Smart Metering, Smart Home and Building Automation, E-mobility, etc.

Smart Metering is a good example of a M2M application and can be realized as one of the vertical applications in an M2M environment. Different types of possibly wireless small meters produce all together a huge amount of data, which is collected by a central entity and processed by an application server. The M2M infrastructure can be provided by a mobile network operator as the meters in urban areas will have most likely a cellular or WiMAX radio.

Smart Grid is built on a distributed and heterogeneous network and can use a combination of diverse networking technologies, such as wireless Access Technologies (WiMAX, Cellular, etc.), wireline and Internet Technologies (e.g., IP/MPLS, Ethernet, SDH/PDH over Fiber optic, etc.) as well as low-power radio technologies enabling the networking of smart meters, home appliances, and constrained devices (e.g. BT-LE, ZigBee, Z-Wave, Wi-Fi, etc.). The operational effectiveness of the smart grid is highly dependent on a robust, two-way, secure, and reliable communications network with suitable availability.

The management of a distributed system like smart grid requires an end-to-end management of and information exchange through different type of networks. However, as of today there is no integrated smart grid management approach and no common smart grid information model available. Specific smart grid applications or network islands use their own management mechanisms. For example, the management of smart meters depends very much on the AMI environment they have been integrated to and the networking technologies they are using. In general, smart meters do only need seldom reconfiguration and they send a small amount of redundant data to a central entity. For a discussion on the management needs of an AMI network see Section 3.11. The management needs for Smart Home and Building Automation are discussed in Section 3.4 and Section 3.5.

3.7. Transport Applications

Transport Application is a generic term for the integrated application of communications, control, and information processing in a transportation system. Transport telematics or vehicle telematics are used as a term for the group of technologies that support transportation systems. Transport applications running on such a transportation system cover all modes of the transport and consider all elements of the transportation system, i.e. the vehicle, the infrastructure, and the driver or user, interacting together dynamically. The overall aim is to improve decision making, often in real time, by transport network controllers and other users, thereby improving the operation of the entire transport system. As such, transport applications can be seen as one of the important M2M

service scenarios with the involvement of manifold small devices.

The definition encompasses a broad array of techniques and approaches that may be achieved through stand-alone technological applications or as enhancements to other transportation communication schemes. Examples for transport applications are inter and intra vehicular communication, smart traffic control, smart parking, electronic toll collection systems, logistic and fleet management, vehicle control, and safety and road assistance.

As a distributed system, transport applications require an end-to-end management of different types of networks. It is likely that constrained devices in a network (e.g. a moving in-car network) have to be controlled by an application running on an application server in the network of a service provider. Such a highly distributed network including mobile devices on vehicles is assumed to include a wireless access network using diverse long distance wireless technologies such as WiMAX, 3G/LTE or satellite communication, e.g. based on an embedded hardware module. As a result, the management of constrained devices in the transport system might be necessary to plan top-down and might need to use data models obliged from and defined on the application layer. The assumed device classes in use are mainly C2 devices. In cases, where an in-vehicle network is involved, C1 devices with limited capabilities and a short-distance constrained radio network, e.g. IEEE 802.15.4 might be used additionally.

Management responsibility typically rests within the organization running the transport application. The constrained devices in a moving transport network might be initially configured in a factory and a reconfiguration might be needed only rarely. New devices might be integrated in an ad-hoc manner based on self-management and -configuration capabilities. Monitoring and data exchange might be necessary to do via a gateway entity connected to the back-end transport infrastructure. The devices and entities in the transport infrastructure need to be monitored more frequently and can be able to communicate with a higher data rate. The connectivity of such entities does not necessarily need to be wireless. The time scale for detecting and recording failures in a moving transport network is likely measured in hours and repairs might easily take days. It is likely that a self-healing feature would be used locally.

3.8. Infrastructure Monitoring

Infrastructure monitoring is concerned with the monitoring of infrastructures such as bridges, railway tracks, or (offshore) windmills. The primary goal is usually to detect any events or changes of the structural conditions that can impact the risk and

safety of the infrastructure being monitored. Another secondary goal is to schedule repair and maintenance activities in a cost effective manner.

The infrastructure to monitor might be in a factory or spread over a wider area but difficult to access. As such, the network in use might be based on a combination of fixed and wireless technologies, which use robust networking equipment and support reliable communication. It is likely that constrained devices in such a network are mainly C2 devices and have to be controlled centrally by an application running on a server. In case such a distributed network is widely spread, the wireless devices might use diverse long-distance wireless technologies such as WiMAX, or 3G/LTE, e.g. based on embedded hardware modules. In cases, where an in-building network is involved, the network can be based on Ethernet or wireless technologies suitable for in-building usage.

The management of infrastructure monitoring applications is primarily concerned with the monitoring of the functioning of the system. Infrastructure monitoring devices are typically rolled out and installed by dedicated experts and changes are rare since the infrastructure itself changes rarely. However, monitoring devices are often deployed in unsupervised environments and hence special attention must be given to protecting the devices from being modified.

Management responsibility typically rests with the organization owning the infrastructure or responsible for its operation. The time scale for detecting and recording failures is likely measured in hours and repairs might easily take days. However, certain events (e.g., natural disasters) may require that status information be obtained much more quickly and that replacements of failed sensors can be rolled out quickly (or redundant sensors are activated quickly). In case the devices are difficult to access, a self-healing feature on the device might become necessary.

3.9. Community Network Applications

Community networks are comprised of constrained routers in a multi-hop mesh topology, communicating over a lossy, and often wireless channel. While the routers are mostly non-mobile, the topology may be very dynamic because of fluctuations in link quality of the (wireless) channel caused by, e.g., obstacles, or other nearby radio transmissions. Depending on the routers that are used in the community network, the resources of the routers (memory, CPU) may be more or less constrained - available resources may range from only a few kilobytes of RAM to several megabytes or more, and CPUs may be small and embedded, or more powerful general-purpose processors.

Examples of such community networks are the FunkFeuer network (Vienna, Austria), Freifunk (Berlin, Germany), Seattle Wireless (Seattle, USA), and AWMN (Athens, Greece). These community networks are public and non-regulated, allowing their users to connect to each other and - through an uplink to an ISP - to the Internet. No fee, other than the initial purchase of a wireless router, is charged for these services. Applications of these community networks can be diverse, e.g., location based services, free Internet access, file sharing between users, distributed chat services, social networking etc, video sharing etc.

As an example of a community network, the FunkFeuer network comprises several hundred routers, many of which have several radio interfaces (with omnidirectional and some directed antennas). The routers of the network are small-sized wireless routers, such as the Linksys WRT54GL, available in 2011 for less than 50 Euros. These routers, with 16 MB of RAM and 264 MHz of CPU power, are mounted on the rooftops of the users. When new users want to connect to the network, they acquire a wireless router, install the appropriate firmware and routing protocol, and mount the router on the rooftop. IP addresses for the router are assigned manually from a list of addresses (because of the lack of autoconfiguration standards for mesh networks in the IETF).

While the routers are non-mobile, fluctuations in link quality require an ad hoc routing protocol that allows for quick convergence to reflect the effective topology of the network (such as NHDP [RFC6130] and OLSRv2 [I-D.ietf-manet-olsrv2] developed in the MANET WG). Usually, no human interaction is required for these protocols, as all variable parameters required by the routing protocol are either negotiated in the control traffic exchange, or are only of local importance to each router (i.e. do not influence interoperability). However, external management and monitoring of an ad hoc routing protocol may be desirable to optimize parameters of the routing protocol. Such an optimization may lead to a more stable perceived topology and to a lower control traffic overhead, and therefore to a higher delivery success ratio of data packets, a lower end-to-end delay, and less unnecessary bandwidth and energy usage.

Different use cases for the management of community networks are possible:

- o One single Network Management Station (NMS), e.g. a border gateway providing connectivity to the Internet, requires managing or monitoring routers in the community network, in order to investigate problems (monitoring) or to improve performance by changing parameters (managing). As the topology of the network is dynamic, constant connectivity of each router towards the

management station cannot be guaranteed. Current network management protocols, such as SNMP and Netconf, may be used (e.g., using interfaces such as the NHDp-MIB [RFC6779]). However, when routers in the community network are constrained, existing protocols may require too many resources in terms of memory and CPU; and more importantly, the bandwidth requirements may exceed the available channel capacity in wireless mesh networks. Moreover, management and monitoring may be unfeasible if the connection between the NMS and the routers is frequently interrupted.

- o A distributed network monitoring, in which more than one management station monitors or manages other routers. Because connectivity to a server cannot be guaranteed at all times, a distributed approach may provide a higher reliability, at the cost of increased complexity. Currently, no IETF standard exists for distributed monitoring and management.
- o Monitoring and management of a whole network or a group of routers. Monitoring the performance of a community network may require more information than what can be acquired from a single router using a network management protocol. Statistics, such as topology changes over time, data throughput along certain routing paths, congestion etc., are of interest for a group of routers (or the routing domain) as a whole. As of 2012, no IETF standard allows for monitoring or managing whole networks, instead of single routers.

3.10. Mobile Applications

M2M services are increasingly provided by mobile service providers as numerous devices, home appliances, utility meters, cars, video surveillance cameras, and health monitors, are connected with mobile broadband technologies. This diverse range of machines brings new network and service requirements and challenges. Different applications e.g. in a home appliance or in-car network use Bluetooth, Wi-Fi or Zigbee and connect to a cellular module acting as a gateway between the constrained environment and the mobile cellular network.

Such a gateway might provide different options for the connectivity of mobile networks and constrained devices, e.g.:

- o a smart phone with 3G/4G and WLAN radio might use BT-LE to connect to the devices in a home area network,
- o a femtocell might be combined with home gateway functionality acting as a low-power cellular base station connecting smart

devices to the application server of a mobile service provider.

- o an embedded cellular module with LTE radio connecting the devices in the car network with the server running the telematics service,
- o an M2M gateway connected to the mobile operator network supporting diverse IoT connectivity technologies including ZigBee and CoAP over 6LoWPAN over IEEE 802.15.4.

Common to all scenarios above is that they are embedded in a service and connected to a network provided by a mobile service provider. Usually there is a hierarchical deployment and management topology in place where different parts of the network are managed by different management entities and the count of devices to manage is high (e.g. many thousands). In general, the network is comprised by manifold type and size of devices matching to different device classes. As such, the managing entity needs to be prepared to manage devices with diverse capabilities using different communication or management protocols. In case the devices are directly connected to a gateway they most likely are managed by a management entity integrated with the gateway, which itself is part of the Network Management System (NMS) run by the mobile operator. Smart phones or embedded modules connected to a gateway might be themselves in charge to manage the devices on their level. The initial and subsequent configuration of such a device is mainly based on self-configuration and is triggered by the device itself.

The challenges in the management of devices in a mobile application are manifold. Firstly, the issues caused through the device mobility need to be taken into consideration. While the cellular devices are moving around or roaming between different regional networks, they should report their status to the corresponding management entities with regard to their proximity and management hierarchy. Secondly, a variety of device troubleshooting information needs to be reported to the management system in order to provide accurate service to the customer. Third but not least, the NMS and the used management protocol need to be tailored to keep the cellular devices lightweight and as energy efficient as possible.

The data models used in these scenario are mostly derived from the models of the operator NMS and might be used to monitor the status of the devices and to exchange the data sent by or read from the devices. The gateway might be in charge of filtering and aggregating the data received from the device as the information sent by the device might be mostly redundant.

3.11. Automated Metering Infrastructure (AMI)

An AMI network enables an electric utility to retrieve frequent electric usage data from each electric meter installed at a customer's home or business. With an AMI network, a utility can also receive immediate notification of power outages when they occur, directly from the electric meters that are experiencing those outages. In addition, if the AMI network is designed to be open and extensible, it could serve as the backbone for communicating with other distribution automation devices besides meters, which could include transformers and reclosers.

In this use case, each meter in the AMI network contains a constrained device. These devices are typically C2 devices. Each meter connects to a constrained mesh network with a low-bandwidth radio. These radios can be 50, 150, or 200 kbps at raw link speed, but actual network throughput may be significantly lower due to forward error correction, multihop delays, MAC delays, lossy links, and protocol overhead.

The constrained devices are used to connect the metering logic with the network, so that usage data and outage notifications can be sent back to the utility's headend systems over the network. These headend systems are located in a data center managed by the utility, and may include meter data collection systems, meter data management systems, and outage management systems.

The meters are connected to a mesh network, and each meter can act as both a source of traffic and as a router for other meters' traffic. In a typical AMI application, smaller amounts of traffic (read requests, configuration) flow "downstream" from the headend to the mesh, and larger amounts of traffic flow "upstream" from the mesh to the headend. However, during a firmware update operation, larger amounts of traffic might flow downstream while smaller amounts flow upstream. Other applications that make use of the AMI network may have their own distinct traffic flows.

The mesh network is anchored by a collection of higher-end devices, which contain a mesh radio that connects to the constrained network as well as a backhaul link that connects to a less-constrained network. The backhaul link could be cellular, WiMAX, or Ethernet, depending on the backhaul networking technology that the utility has chosen. These higher-end devices (termed "routers" in this use case) are typically installed on utility poles throughout the service territory. Router devices are typically less constrained than meters, and often contain the full routing table for all the endpoints routing through them.

In this use case, the utility typically installs on the order of 1000 meters per router. The collection of meters comprised in a local network that are routing through a specific router is called in this use case a Local Meter Network (LMN). When powered on, each meter is designed to discover the nearby LMNs, select the optimal LMN to join, and select the optimal meters in that LMN to route through when sending data to the headend. After joining the LMN, the meter is designed to continuously monitor and optimize its connection to the LMN, and it may change routes and LMNs as needed.

Each LMN may be configured e.g. to share an encryption key, providing confidentiality for all data traffic within the LMN. This key may be obtained by a meter only after an end-to-end authentication process based on certificates, ensuring that only authorized and authenticated meters are allowed to join the LMN, and by extension, the mesh network as a whole.

After joining the LMN, each endpoint obtains a routable and possibly private IPv6 address that enables end-to-end communication between the headend systems and each meter. In this use case, the meters are always-on. However, due to lossy links and network optimization, not every meter will be immediately accessible, though eventually every meter will be able to exchange data with the headend.

In a large AMI deployment, there may be 10 million meters supported by 10,000 routers, spread across a very large geographic area. Within a single LMN, the meters may range between 1 and approx. 20 hops from the router. During the deployment process, these meters are installed and turned on in large batches, and those meters must be authenticated, given addresses, and provisioned with any configuration information necessary for their operation. During deployment and after deployment is finished, the network must be monitored continuously and failures must be handled. Configuration parameters may need to be changed on large numbers of devices, but most of the devices will be running the same configuration. Moreover, eventually, the firmware in those meters will need to be upgraded, and this must also be done in large batches because most of the devices will be running the same firmware image.

Because there may be thousands of routers, this operational model (batch deployment, automatic provisioning, continuous monitoring, batch reconfiguration, batch firmware update) should also apply to the routers as well as the constrained devices. The scale is different (thousands instead of millions) but still large enough to make individual management impractical for routers as well.

3.12. MANET Concept of Operations (CONOPS) in Military

The use case on the Concept of Operations (CONOPS) focuses on the configuration and monitoring of networks that are currently being used in military and as such, it offers insights and challenges of network management that military agencies are facing.

As technology advances, military networks nowadays become large and consist of varieties of different types of equipments that run different protocols and tools that obviously increase complexity of the tactical networks. Moreover, lacks of open common interfaces and Application Programming Interface (API) are often a challenge to network management. Configurations are, most likely, manually performed. Some devices do not support IP networks. Integration and evaluation process are no longer trivial for a large set of protocols and tools. In addition, majority of protocols and tools developed by vendors that are being used are proprietary which makes integration more difficult. The main reason that leads to this problem is that there is no clearly defined standard for the MANET Concept of Operations (CONOPS). In the following, a set of scenarios of network operations are described, which might lead to the development of network management protocols and a framework that can potentially be used in military networks.

Note: The term "node" is used at IETF for either a host or router. The term "unit" or "mobile unit" in military (e.g. Humvees, tanks) is a unit that contains multiple routers, hosts, and/or other non-IP-based communication devices.

Scenario: Parking Lot Staging Area:

The Parking Lot Staging Area is the most common network operation that is currently widely used in military prior to deployment. MANET routers, which can be identical such as the platoon leader's or rifleman's radio, are shipped to a remote location along with a Fixed Network Operations Center (NOC), where they are all connected over traditional wired or wireless networks. The Fixed NOC then performs mass-configuration and evaluation of configuration processes. The same concept can be applied to mobile units. Once all units are successfully configured, they are ready to be deployed.

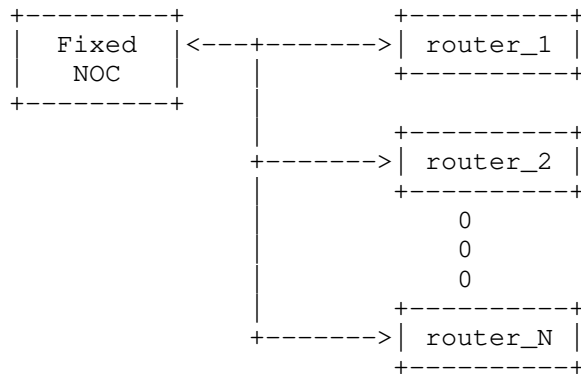


Figure 1: Parking Lot Staging Area

Scenario: Monitoring with SatCom Reachback:

The Monitoring with SatCom Reachback, which is considered another possible common scenario to military's network operations, is similar to the Parking Lot Staging Area. Instead, the Fixed NOC and MANET routers are connected through a Satellite Communications (SatCom) network. The Monitoring with SatCom Reachback is a scenario where MANET routers are augmented with SatCom Reachback capabilities while On-The-Move (OTM). Vehicles carrying MANET routers support multiple types of wireless interfaces, including High Capacity Short Range Radio interfaces as well as Low Capacity OTM SatCom interfaces. The radio interfaces are the preferred interfaces for carrying data traffic due to their high capacity, but the range is limiting with respect to connectivity to a Fixed NOC. Hence, OTM SatCom interfaces offer a more persistent but lower capacity reachback capability. The existence of a SatCom persistent Reachback capability offers the NOC the ability to monitor and manage the MANET routers over the air. Similarly to the Parking Lot Staging scenario, the same concept can be applied to mobile units.

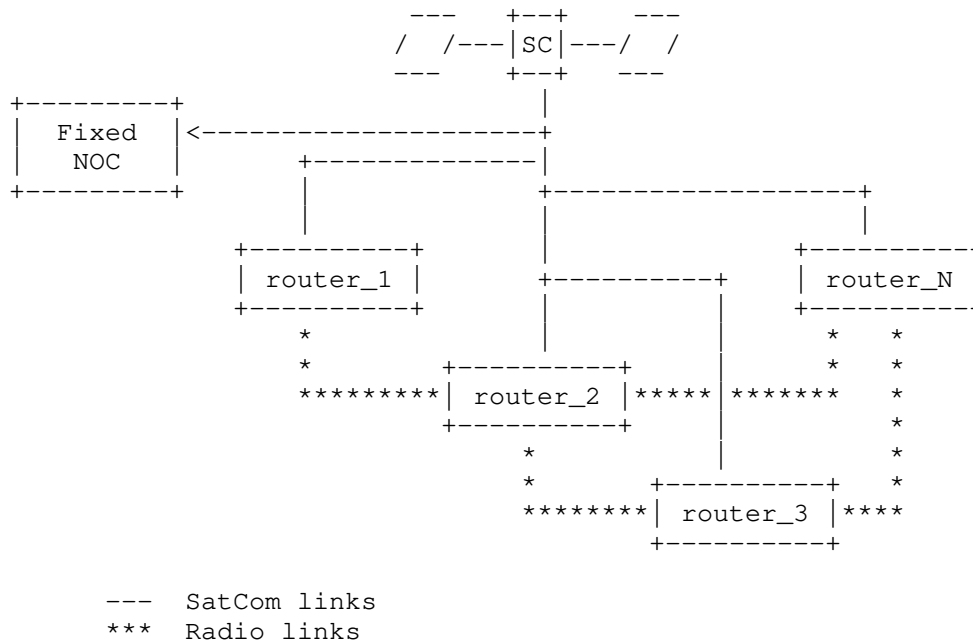
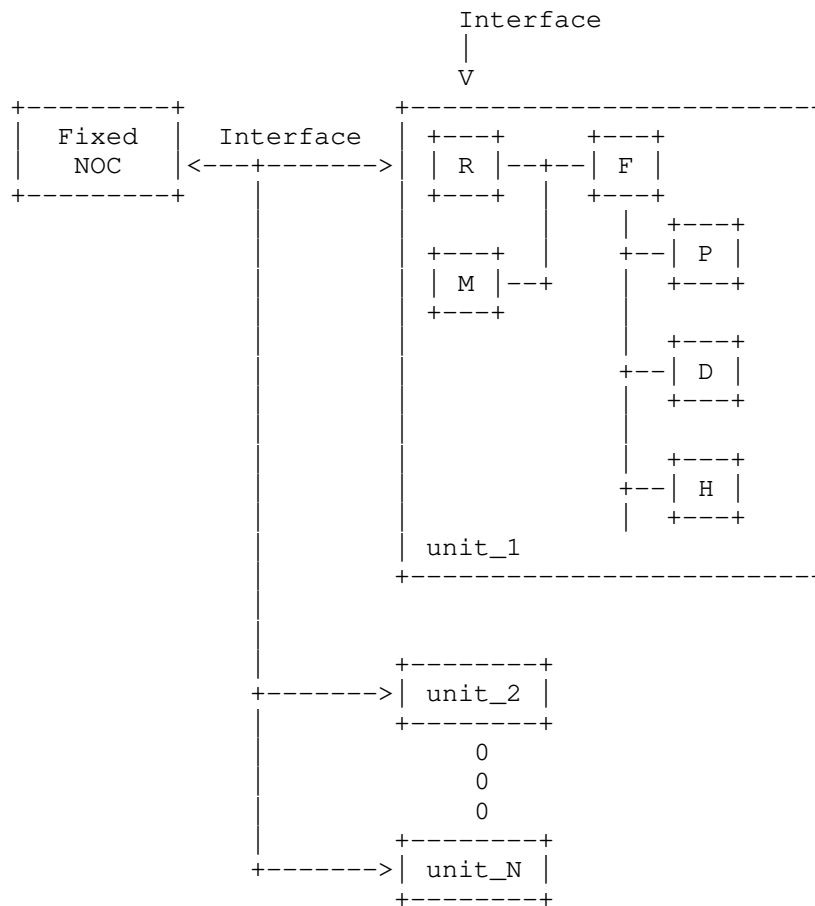


Figure 2: Monitoring with one-hop SatCom Reachback network

Scenario: Hierarchical Management:

Another reasonable scenario common to military operations in a MANET environment is the Hierarchical Management scenario. Vehicles carry a rather complex set of networking devices, including routers running MANET control protocols. In this hierarchical architecture, the MANET mobile unit has a rather complex internal architecture where a local manager within the unit is responsible for local management. The local management includes management of the MANET router and control protocols, the firewall, servers, proxies, hosts and applications. In addition, a standard management interface is required in this architecture. Moreover, in addition to requiring standard management interfaces into the components comprising the MANET nodal architecture, the local manager is responsible for local monitoring and the generation of periodic reports back to the Fixed NOC.



Key: R-Router
 F-Firewall
 P-PEP (Performance Enhancing Proxy)
 D-Servers, e.g., DNS
 H-hosts
 M-Local Manager

Figure 3: Hierarchical Management

Scenario: Management over Lossy/Intermittent Links:

In the future of military operations, the standard management will be done over lossy and intermittent links and ideally the Fixed NOC will become mobile. In this architecture, the nature and current quality

of each link are distinct. However, there are a number of issues that would arise and need to be addressed:

1. Common and specific configurations are undefined:
 - A. When mass-configuring devices, common set of configurations are undefined at this time.
 - B. Similarly, when performing a specific device, set of specific configurations is unknown.
2. Once the total number of units becomes quite large, scalability would be an issue and need to be addressed.
3. The state of the devices are different and may be in various states of operations, e.g., ON/OFF, etc.
4. Pushing large data files over reliable transport, e.g., TCP, would be problematic. Would a new mechanism of transmitting large configurations over the air in low bandwidth be implemented? Which protocol would be used at transport layer?
5. How to validate network configuration (and local configuration) is complex, even when to cutover is an interesting question.
6. Security as a general issue needs to be addressed as it could be problematic in military operations.

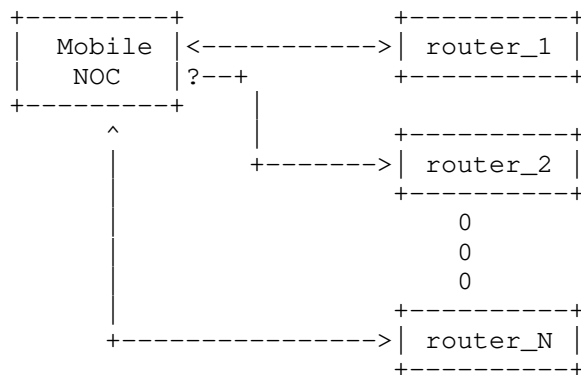


Figure 4: Management over Lossy/intermittent Links

4. Requirements on the Management of Networks with Constrained Devices

This section describes the requirements categorized by management areas listed in subsections.

Note that the requirements in this section need to be seen as standalone requirements. A device might be able to provide selected requirements but might not be capable to provide all requirements at once. On the other hand a device vendor might select a subset of the requirements to implement. As of today this document does not recommend the realization of a profile of requirements.

Following template is used for the definition of the requirements.

Req-ID: An ID uniquely identified by a three-digit number

Title: The title of the requirement.

Description: The rational and description of the requirement.

Source: The origin of the requirement and the matching use case or application.

Requirement Type: Functional Requirement, Non-Functional Requirement, Design Constraint

Device type: The device types by which this requirement can be supported: C0, C1 and/or C2.

Priority: The priority of the requirement showing the importance: Mandatory (M), Optional (O), Conditional (C).

4.1. Management Architecture/System

Req-ID: 4.1.001

Title: Support multiple device classes within a single network.

Description: Larger networks usually are made up of devices belonging to different device classes (e.g., constrained mesh endpoints and less constrained routers) that work together. Hence, the management architecture must be applicable to networks that have a mix of different device classes. See Section 3. of [LWIG-TERMS] for the definition of Constrained Device Classes.

Source: All use cases.

Requirement Type: Non-Functional Requirement

Device type: Managing and intermediary entities.

Priority: Mandatory

Req-ID: 4.1.002

Title: Management scalability.

Description: The management architecture must be able to scale with the number of devices involved and operate efficiently in any network size and topology. This implies that e.g. the managing entity is able to handle huge amount of device monitoring data and the management protocol is not sensitive to the decrease of the time between two client requests. To achieve good scalability, caching techniques, in-network data aggregation techniques, hierarchical management models may be used.

Source: General requirement for all use cases to enable large scale networks.

Requirement Type: Design Constraint

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.1.003

Title: Hierarchical management

Description: Provide a means of hierarchical management, i.e. provide intermediary management entities on different levels, which can take over the responsibility for the management of a sub-hierarchy of the network of constraint devices. The intermediary management entity can e.g. support management data aggregation to handle e.g. high-frequent monitoring data or provide a caching mechanism for the uplink and downlink communication. Hierarchical management contributes to management scalability.

Source: Use cases where a huge amount of devices are deployed with a hierarchical topology.

Requirement Type: Non-Functional Requirement

Device type: Managing and intermediary entities.

Priority: Optional

Req-ID: 4.1.004

Title: Minimize state maintained on constrained devices.

Description: The amount of state that needs to be maintained on constrained devices should be minimized. This is important in order to save memory (especially relevant for C0 and C1 devices) and in order to allow devices to restart for example to apply configuration changes or to recover from extended periods of inactivity. One way to achieve this is to adopt a RESTful architecture that minimizes the amount of state maintained by managed constrained devices and that makes resources of a device addressable via URIs.

Source: Basic requirement which concerns all use cases.

Requirement Type: Non-Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.1.005

Title: Automatic re-synchronization with eventual consistency.

Description: To support large scale networks, where some constrained devices may be offline at any point in time, it is necessary to distribute configuration parameters in a way that allows temporary inconsistencies but eventually converges, after a sufficiently long period of time without further changes, towards global consistency.

Source: Use cases with large scale networks with many devices.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.1.006

Title: Support for lossy links and unreachable devices.

Description: Some constrained devices will only be able to support lossy and unreliable links characterized by a limited data rate, a high latency, and a high transmission error rate. Furthermore constrained devices often duty cycle their radio or the whole device in order to save energy. In both cases the management system must not assume that constrained devices are always reachable. The management protocol(s) must act gracefully if a constrained device is not reachable and provide a high degree of resilience. Intermediaries may be used that provide information for devices currently inactive or that take responsibility to re-synchronize devices when they become reachable again after an extended offline period.

Source: Basic requirement for constrained networks with unreliable links and constrained devices which sleep to save energy.

Requirement Type: Design Constraint

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.1.007

Title: Network-wide configuration

Description: Provide means by which the behavior of the network can be specified at a level of abstraction (network-wide configuration) higher than a set of configuration information specific to individual devices. It is useful to derive the device specific configuration from the network-wide configuration. The identification of the relevant subset of the policies to be

provisioned is according to the capabilities of each device and can be obtained from a pre-configured data-repository. Such a repository can be used to configure pre-defined device or protocol parameters for the whole network. Furthermore, such a network-wide view can be used to monitor and manage a group of routers or a whole network. E.g. monitoring the performance of a network requires additional information other than what can be acquired from a single router using a management protocol.

Source: In general all use cases, which want to configure the network and its devices based on a network view in a top-down manner.

Requirement Type: Non-Functional Requirement

Device type: C0, C1, and C2

Priority: Optional

Req-ID: 4.1.008

Title: Distributed Management

Description: Provide a means of simple distributed management, where a constrained network can be managed or monitored by more than one manager. Since the connectivity to a server cannot be guaranteed at all times, a distributed approach may provide a higher reliability, at the cost of increased complexity. This requirement implies the handling of data consistency in case of concurrent read and write access to the device datastore. It might also happen that no management (configuration) server is accessible and the only reachable node is a peer device. In this case the device should be able to obtain its configuration from peer devices.

Source: Use cases where the count of devices to manage is high.

Requirement Type: Non-Functional Requirement

Device type: C1 and C2

Priority: Optional

4.2. Management protocols and data model

Req-ID: 4.2.001

Title: Modular implementation of management protocols

Description: Management protocols should allow modular implementations, i.e., it should be possible to implement only a basic set of protocol primitives on highly constrained devices while devices with additional resources may provide more support for additional protocol primitives. It should be possible to discover the management protocol primitives by a device.

Source: Basic requirement interesting for all use cases.

Requirement Type: Non-Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.2.002

Title: Compact encoding of management data

Description: The encoding of management data should be compact and space efficient, enabling small message sizes.

Source: General requirement to save memory for the receiver buffer and on-air bandwidth.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.2.003

Title: Compression of management data or complete messages

Description: Management data exchanges can be further optimized by applying data compression techniques or delta encoding techniques. Compression typically requires additional code size and some additional buffers and/or the maintenance of some additional state information. For C0 devices compression may not be feasible. As such, this requirement is marked as optional.

Source: Use cases where it is beneficial to reduce transmission time and bandwidth, e.g. mobile applications which require to save on-air bandwidth.

Requirement Type: Functional Requirement

Device type: C1 and C2

Priority: Optional

Req-ID: 4.2.004

Title: Mapping of management protocol interactions.

Description: It is desirable to have a loss-less automated mapping between the management protocol used to manage constrained devices and the management protocols used to manage regular devices. In the ideal case, the same core management protocol can be used with certain restrictions taking into account the resource limitations of constrained devices. However, for very resource constrained devices, this goal might not be achievable. Hence this requirement is marked optional for device class C2.

Source: Use cases where high-frequent interaction with the management system of a non-constrained network is required.

Requirement Type: Functional Requirement

Device type: C2

Priority: Optional

Req-ID: 4.2.005

Title: Consistency of data models with the underlying information model.

Description: The data models used by the management protocol must be consistent with the information model used to define data models for non-constrained networks. This is essential to facilitate the integration of the management of constrained networks with the management of non-constrained networks. Using an underlying information model for future data model design enables furthermore top-down model design and model reuse as well as data interoperability (i.e. exchange of management information between the constrained and non-constrained networks). This is a strong requirement, even despite the fact that the underlying information models are often not explicitly documented in the IETF.

Source: General requirement to support data interoperability, consistency and model reuse.

Requirement Type: Non-Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.2.006

Title: Loss-less mapping of management data models.

Description: It is desirable to have a loss-less automated mapping between the management data models used to manage regular devices and the management data models used for managing constrained devices. In the ideal case, the same core data models can be used with certain restrictions taking into account the resource limitations of constrained devices. However, for very resource constrained devices, this goal might not be achievable. Hence this requirement is marked optional for device class C2.

Source: Use cases where consistent data exchange with the management system of a non-constrained network is required.

Requirement Type: Functional Requirement

Device type: C2

Priority: Optional

Req-ID: 4.2.007

Title: Protocol extensibility

Description: Provide means of extensibility for the management protocol, i.e. by adding new protocol messages or mechanisms that can deal with the changing requirements on a supported message and data types effectively, without causing inter-operability problems or having to replace/update large amounts of deployed devices.

Source: Basic requirement useful for all use cases.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory

4.3. Configuration management

Req-ID: 4.3.001

Title: Self-configuration capability

Description: Automatic configuration and re-configuration of devices without manual intervention. Compared to the traditional management of devices where the management application is the central entity configuring the devices, in the auto-configuration scenario the device is the active part and initiates the configuration process. Self-configuration can be initiated during the initial configuration or for subsequent configurations, where the configuration data needs to be refreshed. Self-configuration should be also supported during the initialization phase or in the event of failures, where prior knowledge of the network topology is not available or the topology of the network is uncertain.

Source: In general all use cases requiring easy deployment and plug&play behavior as well as easy maintenance of many constrained devices.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory for C0 and C1, Optional for C2.

Req-ID: 4.3.002

Title: Capability Discovery

Description: Enable the discovery of supported optional management capabilities of a device and their exposure via at least one protocol and/or data model.

Source: Use cases where the device interaction with other devices or applications is a function of the level of support for its capabilities.

Requirement Type: Functional Requirement

Device type: C1 and C2

Priority: Optional

Req-ID: 4.3.003

Title: Asynchronous Transaction Support

Description: Provide configuration management with asynchronous transaction support. Configuration operations must support a transactional model, with asynchronous indications that the transaction was completed.

Source: Use cases, which require transaction-oriented processing because of reliability or distributed architecture functional requirements.

Requirement Type: Functional Requirement

Device type: C1 and C2

Priority: Conditional

Req-ID: 4.3.004

Title: Network reconfiguration

Description: Provide a means of iterative network reconfiguration in order to recover the network functionality from node and communication faults. The network reconfiguration can be failure-driven and self-initiated (automatic reconfiguration). The network reconfiguration can be also performed on the whole hierarchical structure of a network (network topology).

Source: Practically all use cases, as network connectivity is a basic requirement.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory, Conditional if the network has a hierarchical topology.

4.4. Monitoring functionality

Req-ID: 4.4.001

Title: Device status monitoring

Description: Provide a monitoring function to collect and expose information about device status and exposing it via at least one management interface. The device monitoring might make use of the hierarchical management through the intermediary entities and the data caching mechanism. The device monitoring might also make use of neighbor-monitoring (fault detection in local network) to support fast fault detection and recovery, e.g. in a scenario where a managing entity is unreachable and a neighbor can take over the monitoring responsibility.

Source: All use cases

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory, Conditional for neighbor-monitoring.

Req-ID: 4.4.002

Title: Energy status monitoring

Description: Provide a monitoring function to collect and expose information about device energy parameters and usage (e.g. battery level and communication power).

Source: Use case Energy Management

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory for energy reporting devices, Optional for the rest

Req-ID: 4.4.003

Title: Monitoring of current and estimated device availability

Description: Provide a monitoring function to collect and expose information about current device availability (energy, memory, computing power, forwarding plane utilization, queue buffers, etc.) and estimation of remaining available resources.

Source: All use cases. Note that monitoring energy resources (like battery status) may be required on all kinds of devices.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Optional

Req-ID: 4.4.004

Title: Network status monitoring

Description: Provide a monitoring function to collect and expose information related to the status of a network or network segments connected to the interfaces of the device.

Source: All use cases.

Requirement Type: Functional Requirement

Device type: C1 and C2

Priority: Optional

Req-ID: 4.4.005

Title: Self-monitoring

Description: Provide self-monitoring (local fault detection) feature for fast fault detection and recovery.

Source: Use cases where the devices cannot be monitored centrally in appropriate manner, e.g. self-healing is required.

Requirement Type: Functional Requirement

Device type: C1 and C2

Priority: Mandatory for C2, Optional for C1

Req-ID: 4.4.006

Title: Performance Monitoring

Description: The device will provide a monitoring function to collect and expose information about the basic TBD performance of the device. The performance management functionality might make use of the hierarchical management through the intermediary devices.

Source: Use cases Building automation, and Transport applications

Requirement Type: Functional Requirement

Device type: C1 and C2

Priority: Optional

Req-ID: 4.4.007

Title: Fault detection monitoring

Description: The device will provide fault detection monitoring. The system collects information about network states in order to identify whether faults have occurred. In some cases the detection of the faults might be based on the processing and analysis of the parameters retrieved from the network or other devices. In case of C0 devices the monitoring might be limited to the check whether the device is alive or not.

Source: Use cases Environmental Monitoring, Building Automation, Energy Management, Infrastructure Monitoring

Requirement Type: Functional Requirement

Device type: C0, C1 and C2

Priority: Optional

Req-ID: 4.4.008

Title: Passive and Reactive Monitoring

Description: The device will provide passive and reactive monitoring capabilities. The system or manager collects information about device components and network states (passive monitoring) and may perform postmortem analysis of collected data. In case events of interest have occurred the system or manager can adaptively react (reactive monitoring), e.g. reconfigure the network. Typically actions (re-actions) will be executed or sent as commands by the management applications.

Source: Diverse use cases relevant for device status and network state monitoring

Requirement Type: Functional Requirement

Device type: C2

Priority: Optional

Req-ID: 4.4.009

Title: Recovery

Description: Provide local, central and hierarchical recovery mechanisms (recovery is in some cases achieved by recovering the whole network of constrained devices).

Source: Use cases Industrial applications, Home and Building Automation, Mobile Applications that involve different forms of clustering or area managers.

Requirement Type: Functional Requirement

Device type: C2

Priority: Optional

Req-ID: 4.4.010

Title: Network topology discovery

Description: Provide a network topology discovery capability (e.g. use of topology extraction algorithms to retrieve the network state) and a monitoring function to collect and expose information about the network topology.

Source: Use cases Community Network Applications and Mobile Applications

Requirement Type: Functional Requirement

Device type: C1 and C2

Priority: Optional

Req-ID: 4.4.011

Title: Notifications

Description: The device will provide the capability of sending notifications on critical events and faults.

Source: All use cases.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory for C2, Optional for C1

Req-ID: 4.4.012

Title: Logging

Description: The device will provide the capability of building, keeping, and allowing retrieval of logs of events (including but not limited to critical faults and alarms).

Source: Use cases Industrial Applications, Building Automation, Infrastructure monitoring

Requirement Type: Functional Requirement

Device type: C2

Priority: Mandatory for some medical or industrial applications, Optional otherwise

4.5. Self-management

Req-ID: 4.5.001

Title: Self-management - Self-healing

Description: Enable event-driven and/or periodic self-management functionality in a device. The device should be able to react in case of a failure e.g. by initiating a fully or partly reset and initiate a self-configuration or management data update as necessary. A device might be further able to check for failures cyclically or schedule-controlled to trigger self-management as necessary. It is a matter of device design and subject for discussion how much self-management a C1 device can support. A minimal failure detection and self-management logic is assumed to be generally useful for the self-healing of a device.

Source: The requirement generally relates to all use cases in this document.

Requirement Type: Functional Requirement

Device type: C1 and C2

Priority: Optional

4.6. Security and Access Control

Req-ID: 4.6.001

Title: Authentication of management system and devices.

Description: Systems having a management role must be properly authenticated to the device such that the device can exercise proper access control and in particular distinguish rightful management systems from rogue systems. On the other hand managed devices must authenticate themselves to systems having a management role such that management systems can protect themselves from rogue devices. In certain application scenarios, it is possible that a large number of devices need to be (re)started at about the same time. Protocols and authentication systems should be designed such that a large number of devices (re)starting simultaneously does not negatively impact the device authentication process.

Source: Basic security requirement for all use cases.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory, Optional for the (re)start of a large number of devices

Req-ID: 4.6.002

Title: Support suitable security bootstrapping mechanisms

Description: Mechanisms should be supported that simplify the bootstrapping of device that is the discovery of newly deployed devices in order to add them to access control lists.

Source: Basic security requirement for all use cases.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.6.003

Title: Access control on management system and devices

Description: Systems acting in a management role must provide an access control mechanism that allows the security administrator to restrict which devices can access the managing system (e.g., using an access control white list of known devices). On the other hand managed constrained devices must provide an access control mechanism that allows the security administrator to restrict how systems in a management role can access the device (e.g., no-access, read-only access, and read-write access).

Source: Basic security requirement for use cases where access control is essential.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.6.004

Title: Select cryptographic algorithms that are efficient in both code space and execution time.

Description: Cryptographic algorithms have a major impact in terms of both code size and overall execution time. It is therefore necessary to select mandatory to implement cryptographic algorithms (like some elliptic curve algorithm) that are reasonable to implement with the available code space and that have a small impact at runtime. Furthermore some wireless technologies (e.g., IEEE 802.15.4) require the support of certain cryptographic algorithms. It might be useful to choose algorithms that are likely to be supported in wireless chipsets for certain

wireless technologies.

Source: Generic requirement to reduce the footprint and CPU usage of a constrained device.

Requirement Type: Non-Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory, Optional for hardware-supported algorithms.

4.7. Energy Management

Req-ID: 4.7.001

Title: Management of Energy Resources

Description: Enable managing power resources in the network, e.g. reduce the sampling rate of nodes with critical battery and reduce node transmission power, put nodes to sleep, put single interfaces to sleep, reject a management job based on available energy, criteria e.g. importance levels pre-defined by the management application, etc. (e.g. a task marked as essential can be executed even if the energy level is low). The device may further implement standard data models for energy management and expose it through a management protocol interface, e.g. EMAN MIB modules and extensions. It might be necessary to downscale EMAN MIBs for the use in C1 and C2 devices.

Source: Use case Energy Management

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory for the use case Energy Management, Optional otherwise.

Req-ID: 4.7.002

Title: Support of energy-optimized communication protocols

Description: Use of an optimized communication protocol to minimize energy usage for the device (radio) receiver/transmitter, on-air bandwidth (protocol efficiency), reduced amount of data communication between nodes (implies data aggregation and

filtering but also a compact format for the transferred data).

Source: Use cases Energy Management and Mobile Applications.

Requirement Type: Functional Requirement

Device type: C2

Priority: Optional

Req-ID: 4.7.003

Title: Support for layer 2 energy-aware protocols

Description: The device will support layer 2 energy management protocols (e.g. energy-efficient Ethernet IEEE 802.3az) and be able to report on these.

Source: Use case Energy Management

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Optional

Req-ID: 4.7.004

Title: Dying gasp

Description: When energy resources draw below the red line level, the device will send a dying gasp notification and perform if still possible a graceful shutdown including conservation of critical device configuration and status information.

Source: Use case Energy Management

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Optional

4.8. SW Distribution

Req-ID: 4.8.001

Title: Group-based provisioning

Description: Support group-based provisioning, i.e. firmware update and configuration management, of a large set of constrained devices with eventual consistency and coordinated reload times. The device should accept group-based configuration management based on bulk commands, which aim similar configurations of a large set of constrained devices of the same type in a given group. Activation of configuration may be based on pre-loaded sets of default values.

Source: All use cases

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Optional

4.9. Traffic management

Req-ID: 4.9.001

Title: Congestion avoidance

Description: Provide the ability to avoid congestion by modifying the device's reporting rate for periodical data (which is usually redundant) based on the importance and reliability level of the management data. This functionality is usually controlled by the managing entity, where the managing entity marks the data as important or relevant for reliability. However reducing a device's reporting rate can also be initiated by a device if it is able to detect congestion or has insufficient buffer memory.

Source: Use cases with high reporting rate and traffic e.g. AMI or M2M.

Requirement Type: Design Constraint

Device type: C1 and C2

Priority: Optional

Req-ID: 4.9.002

Title: Redirect traffic

Description: Provide the ability for network nodes to redirect traffic from overloaded intermediary nodes in a network to another path in order to prevent congestion on a central server and in the primary network.

Source: Use cases with high reporting rate and traffic e.g. AMI or M2M.

Requirement Type: Design Constraint

Device type: Intermediary entity in the network.

Priority: Optional

Req-ID: 4.9.003

Title: Traffic delay schemes.

Description: Provide the ability to apply delay schemes to incoming and outgoing links on an overloaded intermediary node as necessary in order to reduce the amount of traffic in the network.

Source: Use cases with high reporting rate and traffic e.g. AMI or M2M.

Requirement Type: Design Constraint

Device type: Intermediary entity in the network.

Priority: Optional

4.10. Transport Layer

Req-ID: 4.10.001

Title: Scalable transport layer

Description: Enable the use of a scalable transport layer, i.e. not sensitive to the decrease of the time between two client requests, which is useful for applications requiring frequent access to device data.

Source: Applications with high frequent access to the device data.

Requirement Type: Design Constraint

Device type: C0, C1 and C2

Priority: Conditional, in case such scalability is a prerequisite.

Req-ID: 4.10.002

Title: Reliable unicast transport.

Description: Provide reliable unicast transport of messages.

Source: Generally all applications benefit from the reliability of the message transport.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.10.003

Title: Best-effort multicast

Description: Provide best-effort multicast of messages, which is generally useful when devices need to discover a service provided by a server or many devices need to be configured by a managing entity at once based on the same data model.

Source: Use cases where a device needs to discover services as well as use cases with high amount of devices to manage, which are hierarchically deployed, e.g. AMI or M2M.

Requirement Type: Functional Requirement

Device type: C0, C1, and C2

Priority: Optional

Req-ID: 4.10.004

Title: Secure message transport.

Description: Enable secure message transport providing authentication, data integrity, confidentiality by using existing transport layer technologies with small footprint such as TLS/DTLS.

Source: All use cases.

Requirement Type: Non-Functional Requirements

Device type: C1 and C2

Priority: Mandatory

4.11. Implementation Requirements

Req-ID: 4.11.001

Title: Avoid complex application layer transactions requiring large application layer messages.

Description: Complex application layer transactions tend to require large memory buffers that are typically not available on C0 or C1 devices and only by limiting functionality on C2 devices. Furthermore, the failure of a single large transaction requires repeating the whole transaction. On constrained devices, it is often more desirable to a large transaction down into a sequence of smaller transactions, which require less resources and allow to make progress using a sequence of smaller steps.

Source: Basic requirement which concerns all use cases with memory constrained devices.

Requirement Type: Design Constraint

Device type: C0, C1, and C2

Priority: Mandatory

Req-ID: 4.11.002

Title: Avoid reassembly of messages at multiple layers in the
 protocol stack.

Description: Reassembly of messages at multiple layers in the
 protocol stack requires buffers at multiple layers, which leads to
 inefficient use of memory resources. This can be avoided by
 making sure the application layer, the security layer, the
 transport layer, the IPv6 layer and any adaptation layers are
 aware of the limitations of each other such that unnecessary
 fragmentation and reassembly can be avoided. In addition, message
 size constraints must be announced to protocol peers such that
 they can adapt and avoid sending messages that can't be processed
 due to resource constraints on the receiving device.

Source: Basic requirement which concerns all use cases with memory
 constrained devices.

Requirement Type: Design Constraint

Device type: C0, C1, and C2

Priority: Mandatory

5. IANA Considerations

This document does not introduce any new code-points or namespaces for registration with IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

6. Security Considerations

This document discusses the use cases and requirements on the network of constrained devices. If specific requirements for security will be identified, they will be described in future versions of this document.

7. Contributors

Following persons made significant contributions to and reviewed this document:

- o Ulrich Herberg (Fujitsu Laboratories of America) contributed the Section 3.9 on Community Network Applications and to the Section 1.3 on Class of Networks in Focus.
- o Peter van der Stok contributed to Section 3.5 on Building Automation.
- o Zhen Cao contributed to Section 3.10 on Mobile Applications.
- o Gilman Tolle contributed the Section 3.11 on Automated Metering Infrastructure.
- o James Nguyen and Ulrich Herberg contributed the Section 3.12 on MANET Concept of Operations (CONOPS) in Military.

8. Acknowledgments

The editors would like to thank the contributors and the participants on the Coman maillist for their valuable contributions and comments.

9. References

9.1. Normative References

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

9.2. Informative References

- [RFC6632] Ersue, M. and B. Claise, "An Overview of the IETF Network Management Standards", RFC 6632, June 2012.
- [RFC6130] Clausen, T., Dearlove, C., and J. Dean, "Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP)", RFC 6130, April 2011.
- [RFC6779] Herberg, U., Cole, R., and I. Chakeres, "Definition of Managed Objects for the Neighborhood Discovery Protocol", RFC 6779, October 2012.
- [I-D.ietf-manet-olsrv2] Clausen, T., Dearlove, C., Jacquet, P., and U. Herberg, "The Optimized Link State Routing Protocol version 2", draft-ietf-manet-olsrv2-17 (work in progress), October 2012.
- [I-D.ietf-lwig-guidance] Bormann, C., "Guidance for Light-Weight Implementations of the Internet Protocol Suite", draft-ietf-lwig-guidance-02 (work in progress), August 2012.
- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.ietf-eman-framework] Claise, B., Parello, J., Schoening, B., Quittek, J., and B. Nordman, "Energy Management Framework", draft-ietf-eman-framework-06 (work in progress), October 2012.
- [I-D.ietf-eman-requirements] Quittek, J., Chandramouli, M., Winter, R., Dietz, T., and B. Claise, "Requirements for Energy Management", draft-ietf-eman-requirements-11 (work in progress), January 2013.

[I-D.ietf-roll-terminology]

Vasseur, J., "Terminology in Low power And Lossy Networks", draft-ietf-roll-terminology-10 (work in progress), January 2013.

[M2MDEVCLASS]

Open Mobile Alliance, "OMA M2M Device Classification v1.0", October 2012, <http://technical.openmobilealliance.org/Technical/release_program/m2m_Device_class_v1_0.aspx>.

[EU-IOT-A]

EU Commission Seventh Framework Programme, "EU FP7 Project Internet-of-Things Architecture", <<http://www.iot-a.eu/>>.

[EU-SENSEI]

EU Commission Seventh Framework Programme, "EU Project SENSEI", <<http://www.sensei-project.eu/>>.

[EU-FI-WARE]

EU Commission Future Internet Public Private Partnership (FI-PPP), "EU Project Future Internet-Core Platform", <<http://www.iot-butler.eu/>>.

[EU-IOT-BUTLER]

EU Commission Seventh Framework Programme, "EU FP7 Project Butler Smartlife", <<http://www.iot-butler.eu/>>.

[LWIG-TERMS]

Bormann, C., "Terminology for Constrained Node Networks", draft-bormann-lwig-terms (work in progress), November 2012.

Appendix A. Related Development in other Bodies

Note that over time the summary on the related work in other bodies might become outdated.

A.1. ETSI TC M2M

ETSI Technical Committee Machine-to-Machine (ETSI TC M2M) aims to provide an end-to-end view of M2M standardization, which enables the integration of multiple vertical M2M applications. The main goal is to overcome the current M2M market fragmentation and to reuse existing mechanisms from telecom standards such as from OMA or 3GPP.

ETSI Release 1 is functionally frozen. The main focus is on use cases for Smart Metering (Technical Report (TR) 102 691) but it also includes eHealth use cases (TR 102 732) and some others. The Service requirements (Technical Standard (TS) 102 689) derived from the use cases, and the functional architecture specification (TS 102 690), will together define the M2M platform. The architecture consists of Service Capabilities (SC), which are basic functional building blocks for building the M2M platform.

Smart Metering is seen as the important showcase for M2M. It is believed that the Service Enablers that were defined based on the work done for Smart Metering and eHealth segments will also allow the building of other services like vending machines, alarm systems etc.

The functional architecture includes following management-related definitions:

- o Network Management Functions: consists of all functions required to manage the Access, Transport and Core networks: these include Provisioning, Supervision, Fault Management, etc.
- o M2M Management Functions: consists of functions required to manage generic functionalities of M2M Applications and M2M Service Capabilities in the Network and Applications Domain. The management of the M2M Devices and Gateways may use specific M2M Service Capabilities.

The Release 2 work of ETSI TC M2M has started beginning of 2012. Following is a list of networking- and management-related topics under work:

- o Interworking with 3GPP networks. This is a new work item, and no discussion has been held on technical details. The intent is to define which ETSI TC M2M functions are applicable when 3GPP NW is used as transport. It is possible that this work would also cover

details on how to use 3GPP interfaces, e.g. those defined in the SIMTC work, but also for charging and policy control.

- o Creating a Semantic Model or Data Abstraction layer for vertical industries and interworking. This would provide some high level information description that would be usable for interworking with local networks (e.g. ZigBee), and also for verticals, and it would allow the ETSI Service Enablement layer to also understand the data, instead of being just a bit storage and bit pipe. All technical details are still under discussion, but it has been agreed that a function for this exists in the architecture at least for interworking.

A.2. OASIS

Developments in OASIS related to management of constrained networks are following:

- o The Energy Interoperation TC works to define interaction between Smart Grids and their end nodes, including Smart Buildings, Enterprises, Industry, Homes, and Vehicles. The TC develops data and communication models that enable the interoperable and standard exchange of signals for dynamic pricing, reliability, and emergencies. The TC's agenda also extends to the communication of market participation data (such as bids), load predictability, and generation information. The first version of the Energy Interoperation specification is in final review.
- o OASIS Open Data Protocol (OData) aims to simplify the querying and sharing of data across disparate applications and multiple stakeholders for re-use in the enterprise, Cloud, and mobile devices. As a REST-based protocol, OData builds on HTTP, AtomPub, and JSON using URIs to address and access data feed resources. It enables information to be accessed from a variety of sources including (but not limited to) relational databases, file systems, content management systems, and traditional Web sites.
- o Open Building Information Exchange (oBIX) aims to enable the mechanical and electrical control systems in buildings to communicate with enterprise applications, and to provide a platform for developing new classes of applications that integrate control systems with other enterprise functions. Enterprise functions include processes such as Human Resources, Finance, Customer Relationship Management (CRM), and Manufacturing.

A.3. OMA

OMA is currently working on Lightweight M2M Enabler, OMA Device Management (OMA DM) Next Generation, and a white paper on M2M Device Classification.

The Lightweight M2M Enabler covers both M2M device management and service management for constrained devices. In the case of less constrained devices, OMA DM Next Generation Enabler may be more appropriate. OMA DM is structured around Management Objects (MO), each specified for a specific purpose. There is also ongoing work with various other MOs such as the Gateway Management Object (GwMO). A draft for the "Lightweight M2M Requirements" is available.

OMA Lightweight M2M and OMA DM Next Generation are important to M2M device management, provisioning and service managements in both the protocol and management objects. OMA Lightweight M2M work seems to have grown from its original scope of being targeted for very simple devices only, i.e. such that could not handle all those protocols that ETSI M2M requires.

The white paper on the M2M Device Classification [M2MDEVCLASS] provides an M2M device classification framework based on the horizontal attributes (e.g., wide or local area communication interface, IP stack, I/O capabilities) of interest to communication service providers and M2M service providers, independent of vertical markets, such as smart grid, connected cars, e-health, etc. The white paper can be used as a tool to analyze the applicability of existing requirements and specifications developed by OMA and other cooperative standards development organizations.

A.4. IPSO Alliance

IPSO Alliance developed a profile for Device Functions supporting devices such as sensors with a limited user interface, where the configuration of even basic parameters is impossible to do manually. This is a challenge especially for consumer devices that are managed by non-professional users. The configuration of a web service application running on a constrained device goes beyond the autoconfiguration of the IP stack and local information (e.g. proxy address). Constrained devices need additionally service provider and user account related configuration, such as an address/locator and the username for a web server.

IPSO discusses the use cases and requirements for user friendly configuration of such information on a constrained device, and specifies how IPSO profile Device Function Set can be used in the process. It furthermore defines a standard format for the basic

application configuration information.

Appendix B. Related Research Projects

- o The EU project IoT-A (Internet-of-Things Architecture) develops an architectural reference model together with the definition of an initial set of key building blocks. These enable the integration of IoT into the service layer of the Future Internet, and realize a novel resolution infrastructure, as well as a network infrastructure that allows the seamless communication flow between IoT devices and services. The development includes a conceptual model of a smart object as well as a basic Internet of Things reference model defining the interaction and communication between IoT devices and relevant entities. The requirements document includes also network and information management requirements (see [EU-IOT-A]).
- o The EU project SENSEI specified the document on 'End to End Networking and Management' for Wireless Sensor and Actuator Networks. This report presents several research results carried out in SENSEI's tasks related to End-to-End Networking and Management. Particular analyses have been addressed related to naming and addressing of resources, management of resources, resource plug and play, resource level mobility and traffic modelling. The detailed analysis on each of these topics is intended to identify possible gaps between their specific mechanisms and the functional requirements in the SENSEI reference architecture (see [EU-SENSEI]).
- o The EU project FI-WARE is developing the Things Management GE (generic enabler), which uses a data model derived from the OMA DM NGSI data model. Using the abstraction level of things which include non-technical things like rooms, places and people, Things Management GE aims to discover and look up IoT resources that can provide information about things or actuate on these things. The system aims to manage the dynamic associations between IoT resources and things in order to allow internal components as well as external applications to interact with the system using the thing abstraction as the core concept (see [EU-FI-WARE]).
- o EU project BUTLER Smart Life discusses different IoT management aspects and collects requirements for smart life use cases (e.g. smart home or smart city) mainly from service management pov. (see [EU-IOT-BUTLER]).

Appendix C. Open issues

- o Section 4 on the management requirements, as the core section in the document, needs further discussion and consolidation.

Appendix D. Change Log

D.1. 02-03

- o Extended the terminology section and removed some of the terminology addressed in the new LWIG terminology draft. Referenced the LWIG terminology draft.
- o Moved Section 1.3. on Constrained Device Classes to the new LWIG terminology draft.
- o Class of networks considering the different type of radio and communication technologies in use and dimensions extended.
- o Extended the Problem Statement in Section 2. following the requirements listed in Section 4.
- o Following requirements, which belong together and can be realized with similar or same kind of solutions, have been merged.
 - * Distributed Management and Peer Configuration,
 - * Device status monitoring and Neighbor-monitoring,
 - * Passive Monitoring and Reactive Monitoring,
 - * Event-driven self-management - Self-healing and Periodic self-management,
 - * Authentication of management systems and Authentication of managed devices,
 - * Access control on devices and Access control on management systems,
 - * Management of Energy Resources and Data models for energy management,
 - * Software distribution (group-based firmware update) and Group-based provisioning.
- o Deleted the empty section on the gaps in network management standards, as it will be written in a separate draft.
- o Added links to mentioned external pages.
- o Added text on OMA M2M Device Classification in appendix.

D.2. 01-02

- o Extended the terminology section.
- o Added additional text for the use cases concerning deployment type, network topology in use, network size, network capabilities, radio technology, etc.
- o Added examples for device classes in a use case.
- o Added additional text provided by Cao Zhen (China Mobile) for Mobile Applications and by Peter van der Stok for Building Automation.
- o Added the new use cases 'Advanced Metering Infrastructure' and 'MANET Concept of Operations in Military'.
- o Added the section 'Managing the Constrainedness of a Device or Network' discussing the needs of very constrained devices.
- o Added a note that the requirements in Section 4 need to be seen as standalone requirements and the current document does not recommend any profile of requirements.
- o Added Section 4 on the detailed requirements on constrained management matched to management tasks like fault, monitoring, configuration management, Security and Access Control, Energy Management, etc.
- o Solved nits and added references.
- o Added Appendix A on the related development in other bodies.
- o Added Appendix B on the work in related research projects.

D.3. 00-01

- o Splitted the section on 'Networks of Constrained Devices' into the sections 'Network Topology Options' and 'Management Topology Options'.
- o Added the use case 'Community Network Applications' and 'Mobile Applications'.
- o Provided a Contributors section.
- o Extended the section on 'Medical Applications'.

- o Solved nits and added references.

Authors' Addresses

Mehmet Ersue (editor)
Nokia Siemens Networks

Email: mehmet.ersue@nsn.com

Dan Romascanu (editor)
Avaya

Email: dromasca@avaya.com

Juergen Schoenwaelder (editor)
Jacobs University Bremen

Email: j.schoenwaelder@jacobs-university.de

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

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

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

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on September 12, 2013.

Copyright Notice

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

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

Table of Contents

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

1. Conventions and Terminology Used in this Document

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

2. Introduction

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

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

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

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

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

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

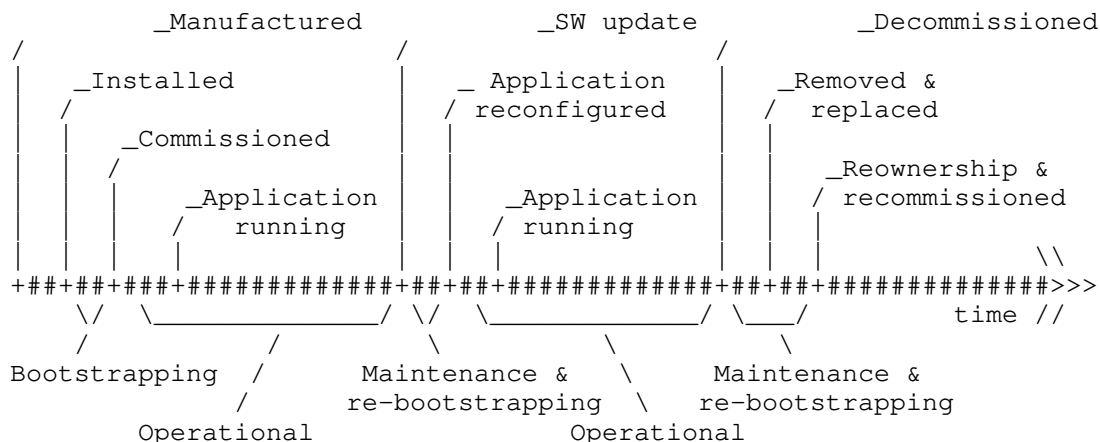
3. The Thing Lifecycle and Architectural Considerations

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

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

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

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



The lifecycle of a thing in the Internet of Things.

Figure 1

3.1. Threat Analysis

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

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

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

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

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

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

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

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

The security threat analysis

Figure 2

3.2. Security Aspects

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

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

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

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

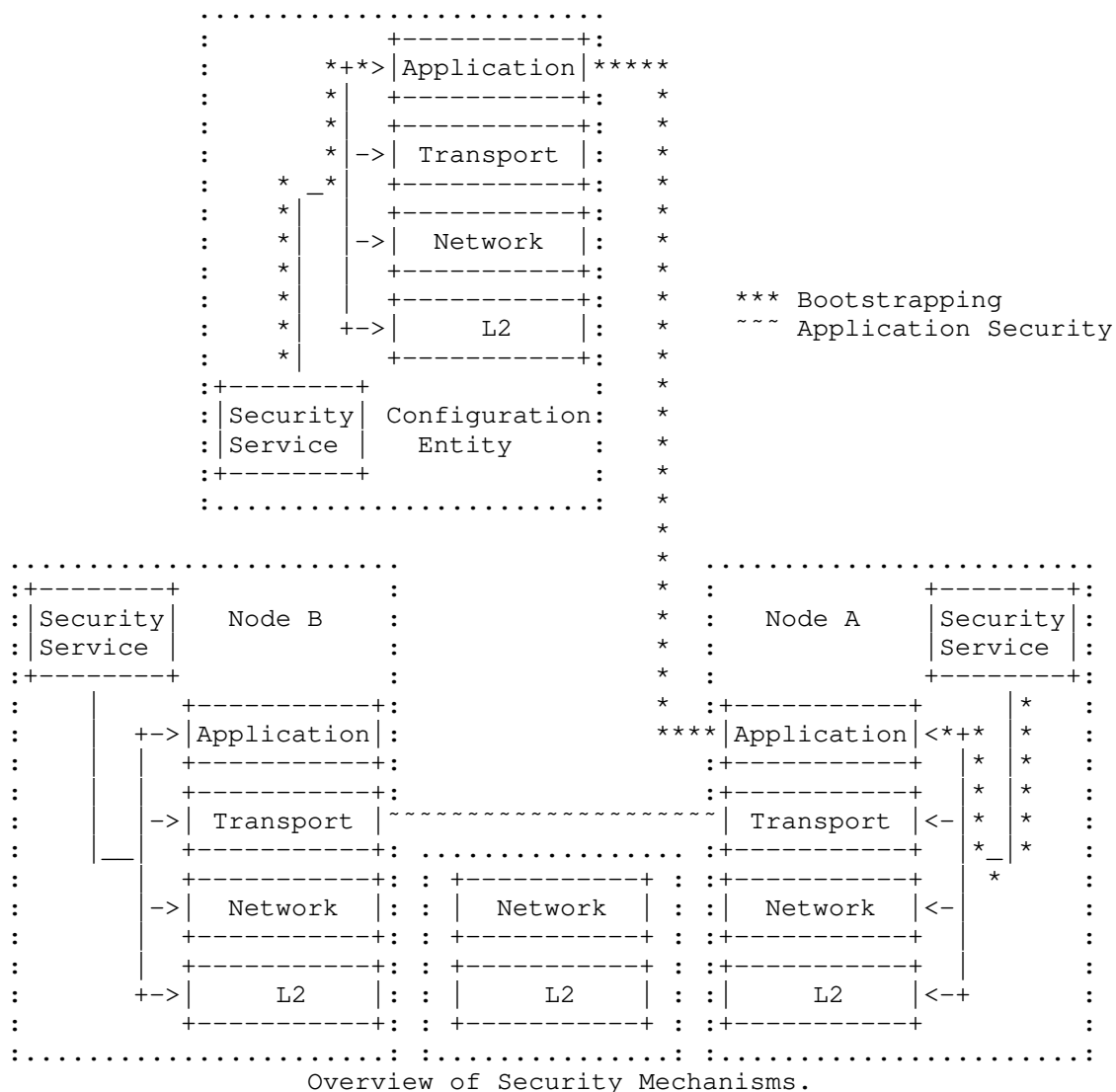


Figure 3

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

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

4. State of the Art

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

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

4.1. IP-based Security Solutions

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

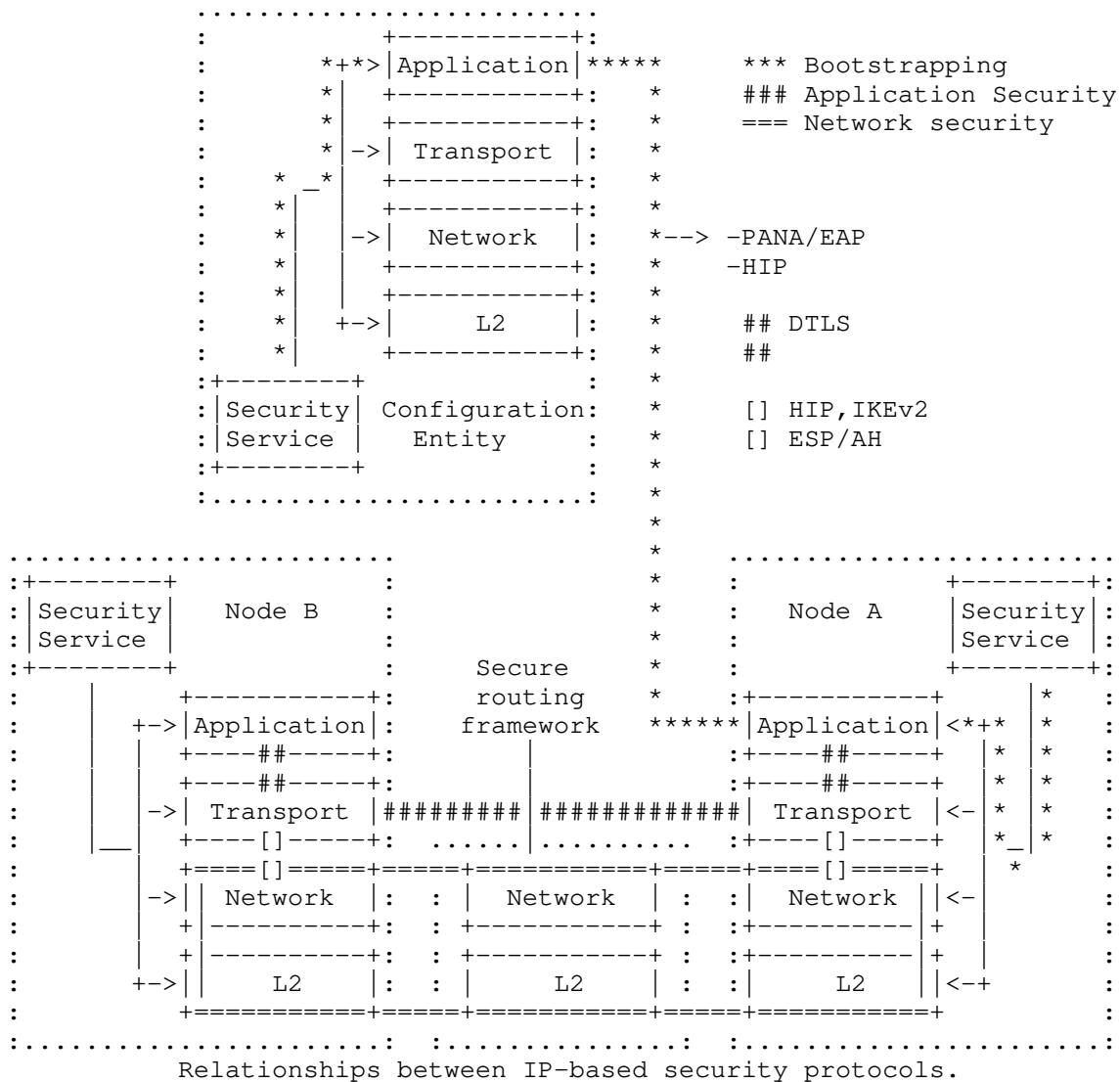


Figure 4

The Internet Key Exchange (IKEv2)/IPsec and the Host Identity protocol (HIP) reside at or above the network layer in the OSI model. Both protocols are able to perform an authenticated key exchange and set up the IPsec transforms for secure payload delivery. Currently, there are also ongoing efforts to create a HIP variant coined Diet HIP [ID-HIP] that takes lossy low-power networks into account at the authentication and key exchange level.

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

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

4.2. Wireless Sensor Network Security and Beyond

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

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

5. Challenges for a Secure Internet of Things

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

5.1. Constraints and Heterogeneous Communication

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

5.1.1. Tight Resource Constraints

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

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

Relationships between IP-based security protocols.

Figure 5

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

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

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

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

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

5.1.2. Denial-of-Service Resistance

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

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

5.1.3. Protocol Translation and End-to-End Security

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

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

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

There are essentially four solutions for this problem:

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

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

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

5.2. Bootstrapping of a Security Domain

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

5.2.1. Distributed vs. Centralized Architecture and Operation

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

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

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

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

5.2.2. Bootstrapping a thing's identity and keying materials

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

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

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

provides the joining thing with fresh network and security parameters.

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

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

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

5.2.3. Privacy-aware Identification

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

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

only reveal the responder's identity to possible eavesdroppers.

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

5.3. Operation

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

5.3.1. End-to-End Security

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

5.3.2. Group Membership and Security

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

on symmetric group keys.

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

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

5.3.3. Mobility and IP Network Dynamics

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

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

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

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

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

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

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

5 Application security.

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

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

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

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

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

Security profiles and application areas.

Figure 6

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

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

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

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

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

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

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

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

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

6.1. Security Architecture

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

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

Security architectures in different security profiles.

Figure 7

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

6.2. Security Model

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

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

Thing security models in different security profiles.

Figure 8

6.3. Security Bootstrapping and Management

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

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

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

Security bootstrapping methods in different security profiles

Figure 9

6.4. Network Security

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

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

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

Network security needs in different security profiles

Figure 10

6.5. Application Security

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

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

Application security methods in different security profiles

Figure 11

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

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

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

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

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

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

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

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

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

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

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

protocols.

8. Security Considerations

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

9. IANA Considerations

This document contains no request to IANA.

10. Acknowledgements

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

11. References

11.1. Informative References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Authors' Addresses

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

Email: oscar.garcia@philips.com

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

Email: sandeep.kumar@philips.com

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

Email: SyeLoong.Keoh@glasgow.ac.uk

Rene Hummen
RWTH Aachen University
Templergraben 55
Aachen, 52056
Germany

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

Rene Struik
Struik Security Consultancy
Toronto,
Canada

Email: rstruik.ext@gmail.com

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

B. Greevenbosch
Huawei Technologies
September 25, 2012

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

Abstract

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

Note

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

Status of this Memo

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

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

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

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

Copyright Notice

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

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

Table of Contents

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

1. Introduction

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

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

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

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

2. Requirements notation

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

3. Definitions

transaction

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

block transaction

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

two subsequent requests

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

request interval

The time between two subsequent requests.

request speed

The multiplicative inverse of the request interval.

ms

Milliseconds or mibiseconds, depending on the implementation.

mibisecond

1/1024 of a second.

4. Motivation

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

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

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

5. The "MinimumRequestInterval" option

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

Table 1: The "MinimumRequestInterval" option

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

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

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

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

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

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

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

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

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

6. Legacy behaviour

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

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

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

7. Example

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

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

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

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

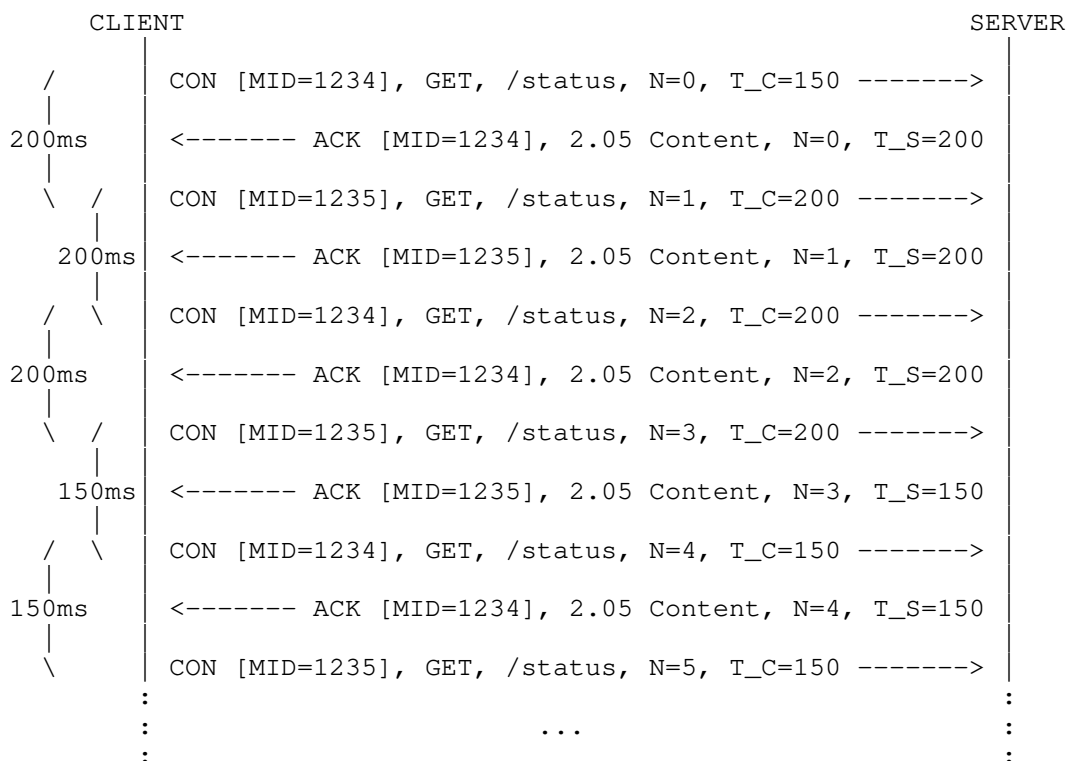


Figure 1: Example of transaction with "MinimumRequestInterval"

8. Security Considerations

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

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

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

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

9. IANA Considerations

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

Number	Name	Reference
TBD (elective)	MinimumRequestInterval	[RFCXXXX]

Table 2: CoAP option numbers

10. Acknowledgements

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

11. Normative References

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

Author's Address

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

Phone: +86-755-28978088

Email: bert.greevenbosch@huawei.com

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

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

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

Abstract

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

Note

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

Status of this Memo

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

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

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

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

Copyright Notice

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

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

Table of Contents

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

1. Requirements notation

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

2. Introduction

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

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

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

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

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

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

3. Obtaining the profile information

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

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

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

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

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

Section 5 covers this in more detail.

4. The Profile Format

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

4.1. The path "path" profile field

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

4.2. The options "op" profile field

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

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

Figure 1: "op" syntax

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

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

4.3. The Content-Formats "cf" profile field

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

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

Figure 2: "cf" syntax

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

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

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

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

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

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

Figure 3: "bls" and "b2s" syntax

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

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

5. Usage of URI Queries

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

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

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

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

6. Forward compatibility

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

7. Examples

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

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

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

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

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

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

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

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

Res: 2.05 Content (application/json)

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

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

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

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

Res: 2.05 Content (application/json)

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


8. Open topics

8.1. Open since v00

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

8.2. Open since v01

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

9. Change log

9.1. Changes in v01

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

10. Security Considerations

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

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

11. IANA Considerations

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

12. Acknowledgements

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

13. Normative References

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

Authors' Addresses

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

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

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

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

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

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

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

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

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

Abstract

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

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

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

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

Status of this Memo

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

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

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

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

Copyright Notice

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

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

Table of Contents

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

1. Introduction

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

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

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

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

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

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

options include:

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

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

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

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

2. Block-wise transfers

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

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

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

In the following, the term "payload" will be used for the actual content of a single CoAP message, i.e. a single block being transferred, while the term "body" will be used for the entire resource representation that is being transferred in a block-wise fashion.

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

2.1. The Block Options

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

Table 1: Block Option Numbers

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

payload.

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

(As a memory aid: Block_1_ pertains to the payload of the _1st_ part of the request-response exchange, i.e. the request, and Block_2_ pertains to the payload of the _2nd_ part of the request-response exchange, i.e. the response.)

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

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

2.2. Structure of a Block Option

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

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

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

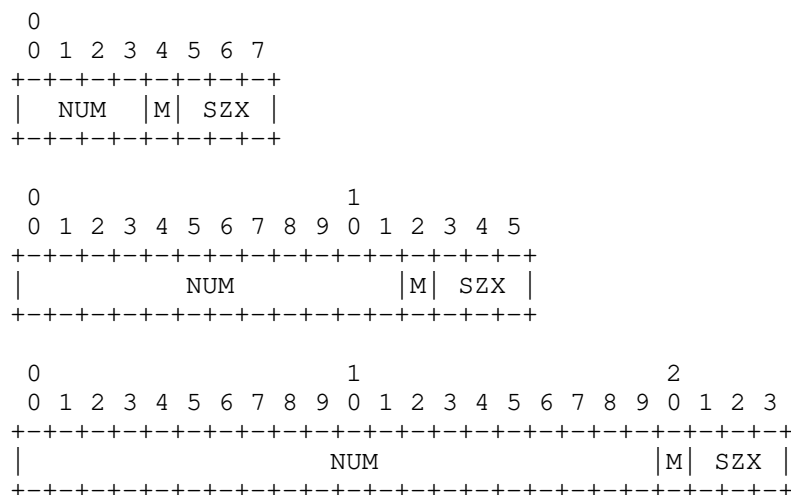


Figure 1: Block option value

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

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

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

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

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

NUM: Block Number. The block number is a variable-size (4, 12, or 20 bit) unsigned integer (uint, see Appendix A of [I-D.ietf-core-coap]) indicating the block number being requested or provided. Block number 0 indicates the first block of a body.

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

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

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

2.3. Block Options in Requests and Responses

The Block options are used in one of three roles:

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

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

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

2.4. Using the Block Options

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

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

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

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

Block-wise transfers can be used to GET resources the representations of which are entirely static (not changing over time at all, such as in a schema describing a device), or for dynamically changing resources. In the latter case, the Block2 Option **SHOULD** be used in conjunction with the ETag Option, to ensure that the blocks being reassembled are from the same version of the representation: The server **SHOULD** include an ETag option in each response. If an ETag option is available, the client's reassembler, when reassembling the representation from the blocks being exchanged, **MUST** compare ETag Options. If the ETag Options do not match in a GET transfer, the

requester has the option of attempting to retrieve fresh values for the blocks it retrieved first. To minimize the resulting inefficiency, the server MAY cache the current value of a representation for an ongoing sequence of requests. (The server may identify the sequence by the combination of the requesting end-point and the URI being the same in each block-wise request.) Note well that this specification makes no requirement for the server to establish any state; however, servers that offer quickly changing resources may thereby make it impossible for a client to ever retrieve a consistent set of blocks.

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

In response to a request with a payload (e.g., a PUT or POST transfer), the block size given in the Block1 Option indicates the block size preference of the server for this resource (control usage). Obviously, at this point the first block has already been transferred by the client without benefit of this knowledge. Still, the client SHOULD heed the preference and, for all further blocks, use the block size preferred by the server or a smaller one. Note that any reduction in the block size may mean that the second request starts with a block number larger than one, as the first request already transferred multiple blocks as counted in the smaller size.

To counter the effects of adaptation layer fragmentation on packet delivery probability, a client may want to give up retransmitting a request with a relatively large payload even before MAX_RETRANSMIT has been reached, and try restating the request as a block-wise transfer with a smaller payload. Note that this new attempt is then a new message-layer transaction and requires a new Message ID. (Because of the uncertainty whether the request or the acknowledgement was lost, this strategy is useful mostly for idempotent requests.)

In a blockwise transfer of a request payload (e.g., a PUT or POST) that is intended to be implemented in an atomic fashion at the server, the actual creation/replacement takes place at the time the final block, i.e. a block with the M bit unset in the Block1 Option, is received. If not all previous blocks are available at the server at this time, the transfer fails and error code 4.08 (Request Entity Incomplete) MUST be returned. The error code 4.13 (Request Entity Too Large) can be returned at any time by a server that does not currently have the resources to store blocks for a block-wise request payload transfer that it would intend to implement in an atomic fashion. (Note that a 4.13 response to a request that does not employ Block1 is a hint for the client to try sending Block1, and a 4.13 response with a smaller SZX in the Block1 than requested is a

hint to try a smaller SZX.)

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

3. Examples

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

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

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

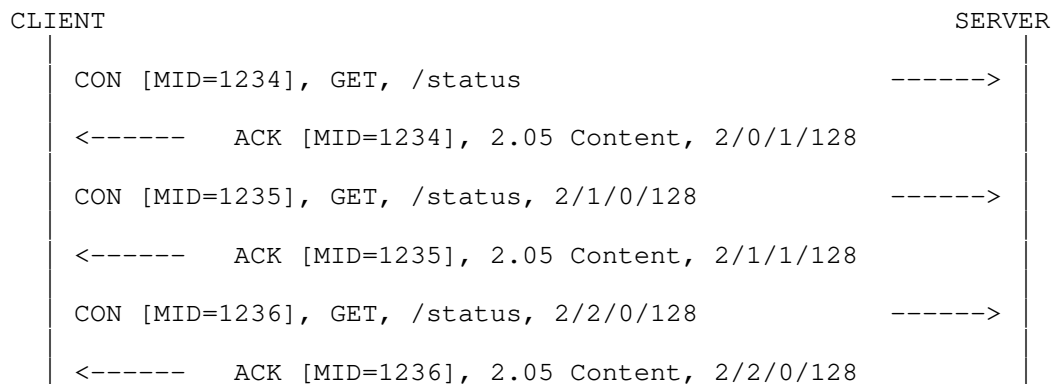


Figure 2: Simple blockwise GET

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.

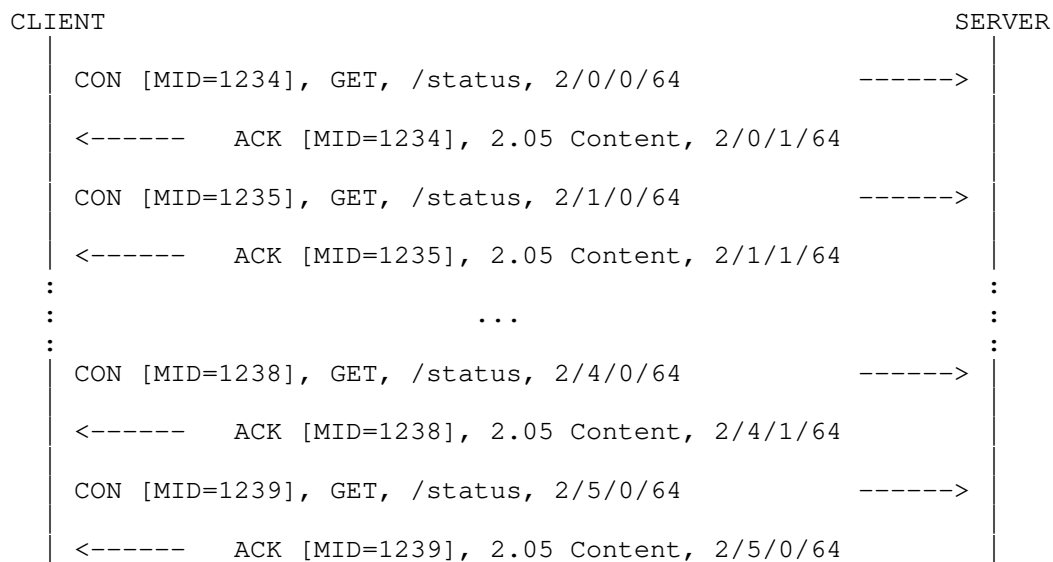


Figure 3: Blockwise GET with early negotiation

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.

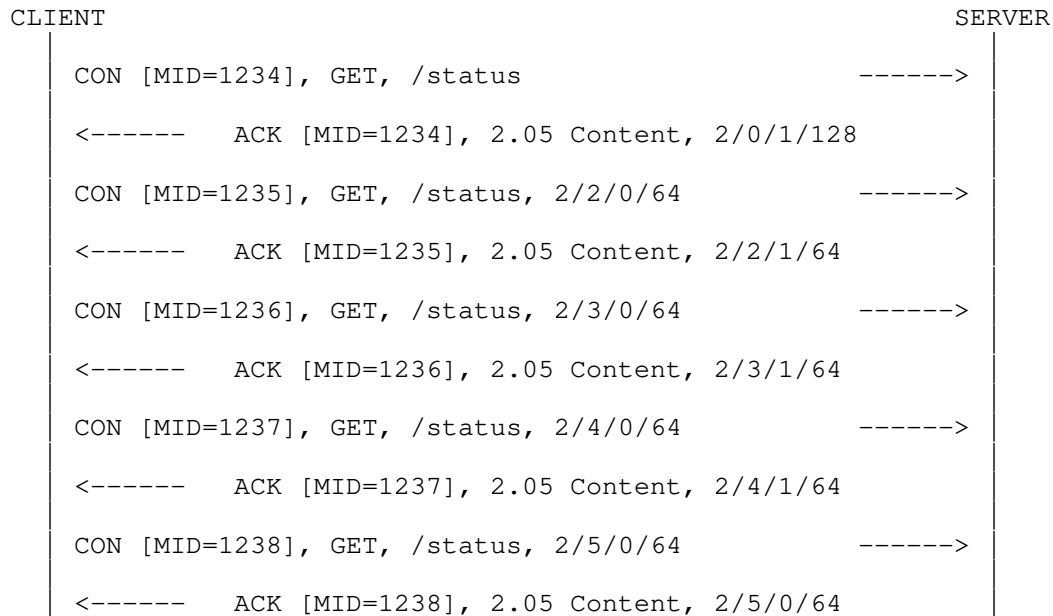


Figure 4: Blockwise GET with late negotiation

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).

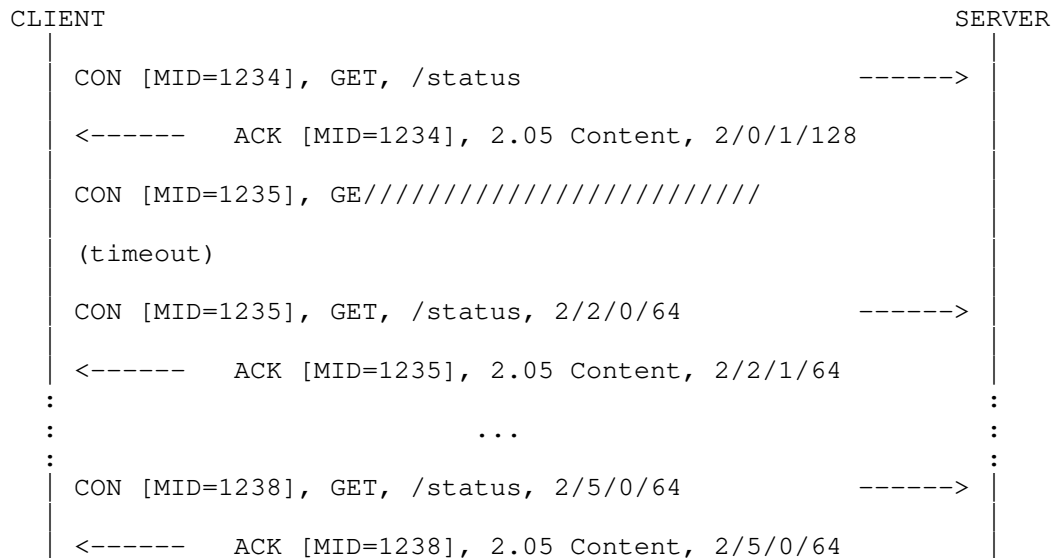


Figure 5: Blockwise GET with late negotiation and lost CON

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

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

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

Figure 7: Simple atomic blockwise PUT

A stateless server that simply builds/updates the resource in place (statelessly) may indicate this by not setting the more bit in the response (Figure 8); in this case, the response codes are valid separately for each block being updated. This is of course only an acceptable behavior of the server if the potential inconsistency present during the run of the message exchange sequence does not lead to problems, e.g. because the resource being created or changed is not yet or not currently in use.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1/0/1/128 ----->	
<----- ACK [MID=1234], 2.04 Changed, 1/0/0/128	
CON [MID=1235], PUT, /options, 1/1/1/128 ----->	
<----- ACK [MID=1235], 2.04 Changed, 1/1/0/128	
CON [MID=1236], PUT, /options, 1/2/0/128 ----->	
<----- ACK [MID=1236], 2.04 Changed, 1/2/0/128	

Figure 8: Simple stateless blockwise PUT

Finally, a server receiving a blockwise PUT or POST may want to indicate a smaller block size preference (Figure 9). In this case, the client SHOULD continue with a smaller block size; if it does, it MUST adjust the block number to properly count in that smaller size.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1/0/1/128 ----->	
<----- ACK [MID=1234], 2.04 Changed, 1/0/1/32	
CON [MID=1235], PUT, /options, 1/4/1/32 ----->	
<----- ACK [MID=1235], 2.04 Changed, 1/4/1/32	
CON [MID=1236], PUT, /options, 1/5/1/32 ----->	
<----- ACK [MID=1235], 2.04 Changed, 1/5/1/32	
CON [MID=1237], PUT, /options, 1/6/0/32 ----->	
<----- ACK [MID=1236], 2.04 Changed, 1/6/0/32	

Figure 9: Simple atomic blockwise PUT with negotiation

Block options may be used in both directions of a single exchange. The following example demonstrates a blockwise POST request, resulting in a separate blockwise response.

CLIENT	SERVER
CON [MID=1234], POST, /soap, 1/0/1/128	----->
<----- ACK [MID=1234], 2.01 Created, 1/0/1/128	
CON [MID=1235], POST, /soap, 1/1/1/128	----->
<----- ACK [MID=1235], 2.01 Created, 1/1/1/128	
CON [MID=1236], POST, /soap, 1/2/0/128	----->
<----- ACK [MID=1236], 0, 1/2/0/128	
<----- CON [MID=4712], 2.01 Created, 2/0/1/128	
ACK [MID=4712], 0, 2/0/1/128	----->
<----- CON [MID=4713], 2.01 Created, 2/1/1/128	
ACK [MID=4713], 0, 2/1/1/128	----->
<----- CON [MID=4714], 2.01 Created, 2/2/1/128	
ACK [MID=4714], 0, 2/2/1/128	----->
<----- CON [MID=4715], 2.01 Created, 2/3/0/128	
ACK [MID=4715], 0, 2/3/0/128	----->

Figure 10: Atomic blockwise POST with separate blockwise response

4. The Size Option

In many cases when transferring a large resource representation block by block, it is advantageous to know the total size early in the process. Some indication may be available from the maximum size estimate attribute "sz" provided in a resource description [RFC6690]. However, the size may vary dynamically, so a more up-to-date indication may be useful.

The Size Option may be used for three purposes:

- o in a request, to ask the server to provide a size estimate along with the usual response ("size request"). For this usage, the value MUST be set to 0.
- o in a response carrying a Block2 Option, to indicate the current estimate the server has of the total size of the resource representation.
- o in a request carrying a Block1 Option, to indicate the current estimate the client has of the total size of the resource representation.

In the latter two cases ("size indication"), the value of the option is the current estimate, measured in bytes.

A size request can be easily distinguished from a size indication, as the third case is not useful for a GET or DELETE, and an actual size indication of 0 would either be overridden by the actual size of the payload for a PUT or POST or would not be useful.

Apart from conveying/asking for size information, the Size option has no other effect on the processing of the request or response. If the client wants to minimize the size of the payload in the resulting response, it should add a Block2 option to the request with a small block size (e.g., setting SZX=0).

The Size Option is "elective", i.e., a client MUST be prepared for the server to ignore the size estimate request. The Size Option MUST NOT occur more than once.

Type	C	U	N	R	Name	Format	Length	Default
28	-	-	N	-	Size	uint	0-4 B	(none)

Implementation Notes:

- o As a quality of implementation consideration, blockwise transfers for which the total size considerably exceeds the size of one block are expected to include size indications, whenever those can be provided without undue effort (preferably with the first block exchanged). If the size estimate does not change, the indication does not need to be repeated for every block.
- o The end of a blockwise transfer is governed by the M bits in the Block Options, not by exhausting the size estimates exchanged.
- o As usual for an option of type uint, the value 0 is best expressed as an empty option (0 bytes). There is no default value.
- o Size is neither critical nor unsafe, and is marked as No-Cache-Key.

5. HTTP Mapping Considerations

In this subsection, we give some brief examples for the influence the Block options might have on intermediaries that map between CoAP and HTTP.

For mapping CoAP requests to HTTP, the intermediary may want to map the sequence of block-wise transfers into a single HTTP transfer. E.g., for a GET request, the intermediary could perform the HTTP request once the first block has been requested and could then fulfill all further block requests out of its cache. A constrained implementation may not be able to cache the entire object and may use a combination of TCP flow control and (in particular if timeouts occur) HTTP range requests to obtain the information necessary for the next block transfer at the right time.

For PUT or POST requests, there is more variation in how HTTP servers might implement ranges. Some WebDAV servers do, but in general the CoAP-to-HTTP intermediary will have to try sending the payload of all the blocks of a block-wise transfer within one HTTP request. If enough buffering is available, this request can be started when the last CoAP block is received. A constrained implementation may want to relieve its buffering by already starting to send the HTTP request at the time the first CoAP block is received; any HTTP 408 status code that indicates that the HTTP server became impatient with the resulting transfer can then be mapped into a CoAP 4.08 response code (similarly, 413 maps to 4.13).

For mapping HTTP to CoAP, the intermediary may want to map a single HTTP transfer into a sequence of block-wise transfers. If the HTTP client is too slow delivering a request body on a PUT or POST, the CoAP server might time out and return a 4.08 response code, which in turn maps well to an HTTP 408 status code (again, 4.13 maps to 413). HTTP range requests received on the HTTP side may be served out of a cache and/or mapped to GET requests that request a sequence of blocks overlapping the range.

(Note that, while the semantics of CoAP 4.08 and HTTP 408 differ, this difference is largely due to the different way the two protocols are mapped to transport. HTTP has an underlying TCP connection, which supplies connection state, so a HTTP 408 status code can immediately be used to indicate that a timeout occurred during transmitting a request through that active TCP connection. The CoAP 4.08 response code indicates one or more missing blocks, which may be due to timeouts or resource constraints; as there is no connection state, there is no way to deliver such a response immediately; instead, it is delivered on the next block transfer. Still, HTTP 408 is probably the best mapping back to HTTP, as the timeout is the most

likely cause for a CoAP 4.08. Note that there is no way to distinguish a timeout from a missing block for a server without creating additional state, the need for which we want to avoid.)

6. IANA Considerations

This draft adds the following option numbers to the CoAP Option Numbers registry of [I-D.ietf-core-coap]:

Number	Name	Reference
23	Block2	[RFCXXXX]
28	Size	[RFCXXXX]
27	Block1	[RFCXXXX]

Table 2: CoAP Option Numbers

This draft adds the following response code to the CoAP Response Codes registry of [I-D.ietf-core-coap]:

Code	Description	Reference
136	4.08 Request Entity Incomplete	[RFCXXXX]

Table 3: CoAP Response Codes

7. Security Considerations

Providing access to blocks within a resource may lead to surprising vulnerabilities. Where requests are not implemented atomically, an attacker may be able to exploit a race condition or confuse a server by inducing it to use a partially updated resource representation. Partial transfers may also make certain problematic data invisible to intrusion detection systems; it is RECOMMENDED that an intrusion detection system (IDS) that analyzes resource representations transferred by CoAP implement the Block options to gain access to entire resource representations. Still, approaches such as transferring even-numbered blocks on one path and odd-numbered blocks on another path, or even transferring blocks multiple times with different content and obtaining a different interpretation of temporal order at the IDS than at the server, may prevent an IDS from seeing the whole picture. These kinds of attacks are well understood from IP fragmentation and TCP segmentation; CoAP does not add fundamentally new considerations.

Where access to a resource is only granted to clients making use of a specific security association, all blocks of that resource MUST be subject to the same security checks; it MUST NOT be possible for unprotected exchanges to influence blocks of an otherwise protected resource. As a related consideration, where object security is employed, PUT/POST should be implemented in the atomic fashion, unless the object security operation is performed on each access and the creation of unusable resources can be tolerated.

A stateless server might be susceptible to an attack where the adversary sends a Block1 (e.g., PUT) block with a high block number: A naive implementation might exhaust its resources by creating a huge resource representation.

Misleading size indications may be used by an attacker to induce buffer overflows in poor implementations, for which the usual considerations apply.

7.1. Mitigating Resource Exhaustion Attacks

Certain blockwise requests may induce the server to create state, e.g. to create a snapshot for the blockwise GET of a fast-changing resource to enable consistent access to the same version of a resource for all blocks, or to create temporary resource representations that are collected until pressed into service by a final PUT or POST with the more bit unset. All mechanisms that induce a server to create state that cannot simply be cleaned up create opportunities for denial-of-service attacks. Servers SHOULD avoid being subject to resource exhaustion based on state created by

untrusted sources. But even if this is done, the mitigation may cause a denial-of-service to a legitimate request when it is drowned out by other state-creating requests. Wherever possible, servers should therefore minimize the opportunities to create state for untrusted sources, e.g. by using stateless approaches.

Performing segmentation at the application layer is almost always better in this respect than at the transport layer or lower (IP fragmentation, adaptation layer fragmentation), e.g. because there is application layer semantics that can be used for mitigation or because lower layers provide security associations that can prevent attacks. However, it is less common to apply timeouts and keepalive mechanisms at the application layer than at lower layers. Servers MAY want to clean up accumulated state by timing it out (cf. response code 4.08), and clients SHOULD be prepared to run blockwise transfers in an expedient way to minimize the likelihood of running into such a timeout.

7.2. Mitigating Amplification Attacks

[I-D.ietf-core-coap] discusses the susceptibility of CoAP end-points for use in amplification attacks.

A CoAP server can reduce the amount of amplification it provides to an attacker by offering large resource representations only in relatively small blocks. With this, e.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

8. Acknowledgements

Much of the content of this draft is the result of discussions with the [I-D.ietf-core-coap] authors, and via many CoRE WG discussions.

Charles Palmer provided extensive editorial comments to a previous version of this draft, some of which the authors hope to have covered in this version. Esko Dijk reviewed a more recent version, leading to a number of further editorial improvements as well as a solution to the 4.13 ambiguity problem. Markus Becker proposed getting rid of an ill-conceived default value for the Block2 and Block1 options.

Kepeng Li, Linyi Tian, and Barry Leiba wrote up an early version of the Size Option, which has informed this draft.

9. References

9.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-12 (work in progress), October 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

9.2. Informative References

- [REST] Fielding, R., "Architectural Styles and the Design of
Network-based Software Architectures", Ph.D. Dissertation,
University of California, Irvine, 2000, <[http://
www.ics.uci.edu/~fielding/pubs/dissertation/
fielding_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6
over Low-Power Wireless Personal Area Networks (6LoWPANs):
Overview, Assumptions, Problem Statement, and Goals",
RFC 4919, August 2007.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link
Format", RFC 6690, August 2012.

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Zach Shelby (editor)
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 4, 2013

Z. Shelby
Sensinode
K. Hartke
C. Bormann
Universitaet Bremen TZI
B. Frank
SkyFoundry
October 1, 2012

Constrained Application Protocol (CoAP)
draft-ietf-core-coap-12

Abstract

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as 6LoWPAN often have high packet error rates and a typical throughput of 10s of kbit/s. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP easily interfaces with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 4, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	6
1.1. Features	6
1.2. Terminology	7
2. Constrained Application Protocol	10
2.1. Messaging Model	11
2.2. Request/Response Model	12
2.3. Intermediaries and Caching	15
2.4. Resource Discovery	15
3. Message Format	15
3.1. Header Format	16
3.2. Option Format	17
3.3. Option Jump	19
3.4. Option Value Formats	20
3.4.1. uint	21
3.4.2. string	21
3.4.3. opaque	21
3.4.4. empty	22
4. Message Transmission	22
4.1. Messages and Endpoints	22
4.2. Messages Transmitted Reliably	23
4.3. Messages Transmitted Without Reliability	24
4.4. Message Correlation	24
4.5. Message Deduplication	25
4.6. Message Size	26
4.7. Congestion Control	27
4.8. Transmission Parameters	27
4.8.1. Changing The Parameters	28
4.8.2. Time Values derived from Transmission Parameters	29
5. Request/Response Semantics	31
5.1. Requests	31

5.2.	Responses	31
5.2.1.	Piggy-backed	32
5.2.2.	Separate	33
5.2.3.	Non-Confirmable	34
5.3.	Request/Response Matching	34
5.4.	Options	35
5.4.1.	Critical/Elective	37
5.4.2.	Proxy Unsafe/Safe and Cache-Key	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.	Payload	39
5.5.1.	Representation	39
5.5.2.	Diagnostic Message	39
5.6.	Caching	39
5.6.1.	Freshness Model	40
5.6.2.	Validation Model	40
5.7.	Proxying	41
5.7.1.	Proxy Operation	42
5.7.2.	Forward-Proxies	43
5.7.3.	Reverse-Proxies	43
5.8.	Method Definitions	44
5.8.1.	GET	44
5.8.2.	POST	44
5.8.3.	PUT	45
5.8.4.	DELETE	45
5.9.	Response Code Definitions	45
5.9.1.	Success 2.xx	45
5.9.2.	Client Error 4.xx	47
5.9.3.	Server Error 5.xx	48
5.10.	Option Definitions	49
5.10.1.	Token	50
5.10.2.	Uri-Host, Uri-Port, Uri-Path and Uri-Query	50
5.10.3.	Proxy-Uri	51
5.10.4.	Content-Format	51
5.10.5.	Accept	52
5.10.6.	Max-Age	52
5.10.7.	ETag	52
5.10.8.	Location-Path and Location-Query	53
5.10.9.	If-Match	53
5.10.10.	If-None-Match	54
6.	CoAP URIs	54
6.1.	coap URI Scheme	55
6.2.	coaps URI Scheme	56
6.3.	Normalization and Comparison Rules	56
6.4.	Decomposing URIs into Options	57
6.5.	Composing URIs from Options	58

7.	Discovery	59
7.1.	Service Discovery	59
7.2.	Resource Discovery	60
7.2.1.	'ct' Attribute	60
8.	Multicast CoAP	60
8.1.	Messaging Layer	61
8.2.	Request/Response Layer	61
8.2.1.	Caching	62
8.2.2.	Proxying	62
9.	Securing CoAP	63
9.1.	DTLS-secured CoAP	64
9.1.1.	Messaging Layer	65
9.1.2.	Request/Response Layer	65
9.1.3.	Endpoint Identity	66
9.2.	Using CoAP with IPsec	68
10.	Cross-Protocol Proxying between CoAP and HTTP	68
10.1.	CoAP-HTTP Proxying	69
10.1.1.	GET	70
10.1.2.	PUT	70
10.1.3.	DELETE	70
10.1.4.	POST	71
10.2.	HTTP-CoAP Proxying	71
10.2.1.	OPTIONS and TRACE	71
10.2.2.	GET	71
10.2.3.	HEAD	72
10.2.4.	POST	72
10.2.5.	PUT	73
10.2.6.	DELETE	73
10.2.7.	CONNECT	73
11.	Security Considerations	73
11.1.	Protocol Parsing, Processing URIs	74
11.2.	Proxying and Caching	74
11.3.	Risk of amplification	75
11.4.	IP Address Spoofing Attacks	76
11.5.	Cross-Protocol Attacks	76
12.	IANA Considerations	78
12.1.	CoAP Code Registry	78
12.1.1.	Method Codes	79
12.1.2.	Response Codes	80
12.2.	Option Number Registry	81
12.3.	Content-Format Registry	83
12.4.	URI Scheme Registration	84
12.5.	Secure URI Scheme Registration	85
12.6.	Service Name and Port Number Registration	86
12.7.	Secure Service Name and Port Number Registration	87
12.8.	Multicast Address Registration	88
13.	Acknowledgements	88
14.	References	89

14.1. Normative References	89
14.2. Informative References	91
Appendix A. Examples	93
Appendix B. URI Examples	98
Appendix C. Changelog	99
Authors' Addresses	107

1. Introduction

The use of web services on the Internet has become ubiquitous in most applications, and depends on the fundamental Representational State Transfer [REST] architecture of the web.

The Constrained RESTful Environments (CoRE) work aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN, [RFC4944]). Constrained networks like 6LoWPAN support the expensive fragmentation of IPv6 packets into small link-layer frames. One design goal of CoAP has been to keep message overhead small, thus limiting the use of fragmentation.

One of the main goals of CoAP is to design a generic web protocol for the special requirements of this constrained environment, especially considering energy, building automation and other machine-to-machine (M2M) applications. The goal of CoAP is not to blindly compress HTTP [RFC2616], but rather to realize a subset of REST common with HTTP but optimized for M2M applications. Although CoAP could be used for compressing simple HTTP interfaces, it more importantly also offers features for M2M such as built-in discovery, multicast support and asynchronous message exchanges.

This document specifies the Constrained Application Protocol (CoAP), which easily translates to HTTP for integration with the existing web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments and M2M applications.

1.1. Features

CoAP has the following main features:

- o Constrained web protocol fulfilling M2M requirements.
- o UDP binding with optional reliability supporting unicast and multicast requests.
- o Asynchronous message exchanges.
- o Low header overhead and parsing complexity.
- o URI and Content-type support.
- o Simple proxy and caching capabilities.

- o A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.
- o Security binding to Datagram Transport Layer Security (DTLS).

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC2616]. In addition, this specification defines the following terminology:

Endpoint

An entity participating in the CoAP protocol. Colloquially, an endpoint lives on a "Node", although "Host" would be more consistent with Internet standards usage, and is further identified by transport layer multiplexing information that can include a UDP port number and a security association (Section 4.1).

Sender

The originating endpoint of a message. When the aspect of identification of the specific sender is in focus, also "source endpoint".

Recipient

The destination endpoint of a message. When the aspect of identification of the specific recipient is in focus, also "destination endpoint".

Client

The originating endpoint of a request; the destination endpoint of a response.

Server

The destination endpoint of a request; the originating endpoint of a response.

Origin Server

The server on which a given resource resides or is to be created.

Intermediary

A CoAP endpoint that acts both as a server and as a client towards (possibly via further intermediaries) an origin server. A common form of an intermediary is a proxy; several classes of such proxies are discussed in this specification.

Proxy

An intermediary that mainly is concerned with forwarding requests and relaying back responses, possibly performing caching, namespace translation, or protocol translation in the process. As opposed to intermediaries in the general sense, proxies generally do not implement specific application semantics. Based on the position in the overall structure of the request forwarding, there are two common forms of proxy: forward-proxy and reverse-proxy. In some cases, a single endpoint might act as an origin server, forward-proxy, or reverse-proxy, switching behavior based on the nature of each request.

Forward-Proxy

A "forward-proxy" is an endpoint selected by a client, usually via local configuration rules, to perform requests on behalf of the client, doing any necessary translations. Some translations are minimal, such as for proxy requests for "coap" URIs, whereas other requests might require translation to and from entirely different application-layer protocols.

Reverse-Proxy

A "reverse-proxy" is an endpoint that stands in for one or more other server(s) and satisfies requests on behalf of these, doing any necessary translations. Unlike a forward-proxy, the client may not be aware that it is communicating with a reverse-proxy; a reverse-proxy receives requests as if it was the origin server for the target resource.

Cross-Proxy

A cross-protocol proxy, or "cross-proxy" for short, is a proxy that translates between different protocols, such as a CoAP-to-HTTP proxy or an HTTP-to-CoAP proxy. While this specification makes very specific demands of CoAP-to-CoAP proxies, there is more variation possible in cross-proxies.

Confirmable Message

Some messages require an acknowledgement. These messages are called "Confirmable". When no packets are lost, each confirmable message elicits exactly one return message of type Acknowledgement

or type Reset.

Non-Confirmable Message

Some other messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual success is sufficient.

Acknowledgement Message

An Acknowledgement message acknowledges that a specific Confirmable Message arrived. It does not indicate success or failure of any encapsulated request.

Reset Message

A Reset message indicates that a specific message (confirmable or non-confirmable) was received, but some context is missing to properly process it. This condition is usually caused when the receiving node has rebooted and has forgotten some state that would be required to interpret the message.

Piggy-backed Response

A Piggy-backed Response is included right in a CoAP Acknowledgement (ACK) message that is sent to acknowledge receipt of the Request for this Response (Section 5.2.1).

Separate Response

When a Confirmable message carrying a Request is acknowledged with an empty message (e.g., because the server doesn't have the answer right away), a Separate Response is sent in a separate message exchange (Section 5.2.2).

Critical Option

An option that would need to be understood by the endpoint receiving the message in order to properly process the message (Section 5.4.1). Note that the implementation of critical options is, as the name "Option" implies, generally optional: unsupported critical options lead to an error response or summary rejection of the message.

Elective Option

An option that is intended to be ignored by an endpoint that does not understand it. Processing the message even without understanding the option is acceptable (Section 5.4.1).

Unsafe Option

An option that would need to be understood by a proxy receiving the message in order to safely forward the message (Section 5.4.2).

Safe Option

An option that is intended to be safe for forwarding by a proxy that does not understand it. Forwarding the message even without understanding the option is acceptable (Section 5.4.2).

Resource Discovery

The process where a CoAP client queries a server for its list of hosted resources (i.e., links, Section 7).

Content-Format

The combination of an Internet media type, potentially with specific parameters given, and a content-coding (which is often the identity content-coding), identified by a numeric identifier defined by the CoAP Content-Format registry.

In this specification, the term "byte" is used in its now customary sense as a synonym for "octet".

All multi-byte integers in this protocol are interpreted in network byte order.

Where arithmetic is used, this specification uses the notation familiar from the programming language C, except that the operator "***" stands for exponentiation.

2. Constrained Application Protocol

The interaction model of CoAP is similar to the client/server model of HTTP. However, machine-to-machine interactions typically result in a CoAP implementation acting in both client and server roles. A CoAP request is equivalent to that of HTTP, and is sent by a client to request an action (using a method code) on a resource (identified by a URI) on a server. The server then sends a response with a response code; this response may include a resource representation.

Unlike HTTP, CoAP deals with these interchanges asynchronously over a datagram-oriented transport such as UDP. This is done logically using a layer of messages that supports optional reliability (with exponential back-off). CoAP defines four types of messages: Confirmable, Non-Confirmable, Acknowledgement, Reset; method codes and response codes included in some of these messages make them carry requests or responses. The basic exchanges of the four types of messages are somewhat orthogonal to the request/response interactions; requests can be carried in Confirmable and Non-Confirmable messages, and responses can be carried in these as well as piggy-backed in Acknowledgement messages.

One could think of CoAP logically as using a two-layer approach, a CoAP messaging layer used to deal with UDP and the asynchronous nature of the interactions, and the request/response interactions using Method and Response codes (see Figure 1). CoAP is however a single protocol, with messaging and request/response just features of the CoAP header.

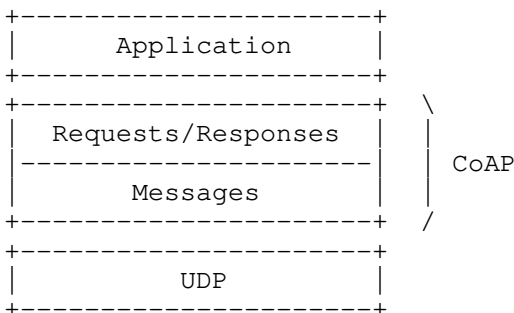


Figure 1: Abstract layering of CoAP

2.1. Messaging Model

The CoAP messaging model is based on the exchange of messages over UDP between endpoints.

CoAP uses a short fixed-length binary header (4 bytes) that may be followed by compact binary options and a payload. This message format is shared by requests and responses. The CoAP message format is specified in Section 3. Each message contains a Message ID used to detect duplicates and for optional reliability.

Reliability is provided by marking a message as Confirmable (CON). A Confirmable message is retransmitted using a default timeout and exponential back-off between retransmissions, until the recipient sends an Acknowledgement message (ACK) with the same Message ID (for example, 0x7d34) from the corresponding endpoint; see Figure 2. When a recipient is not at all able to process a Confirmable message (i.e., not even able to provide a suitable error response), it replies with a Reset message (RST) instead of an Acknowledgement (ACK).

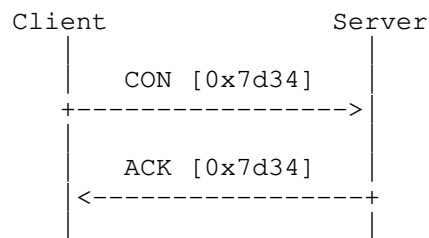


Figure 2: Reliable message transmission

A message that does not require reliable transmission, for example each single measurement out of a stream of sensor data, can be sent as a Non-confirmable message (NON). These are not acknowledged, but still have a Message ID for duplicate detection; see Figure 3. When a recipient is not able to process a Non-confirmable message, it may reply with a Reset message (RST).

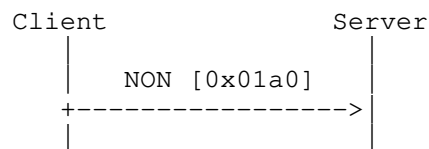


Figure 3: Unreliable message transmission

See Section 4 for details of CoAP messages.

As CoAP is based on UDP, it also supports the use of multicast IP destination addresses, enabling multicast CoAP requests. Section 8 discusses the proper use of CoAP messages with multicast addresses and precautions for avoiding response congestion.

Several security modes are defined for CoAP in Section 9 ranging from no security to certificate-based security. The use of IPsec along with a binding to DTLS are specified for securing the protocol.

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 Option is used to match responses to requests independently from the underlying messages (Section 5.3).

A request is carried in a Confirmable (CON) or Non-confirmable (NON) message, and if immediately available, the response to a request

carried in a Confirmable message is carried in the resulting Acknowledgement (ACK) message. This is called a piggy-backed response, detailed in Section 5.2.1. Two examples for a basic GET request with piggy-backed response are shown in Figure 4, one successful, one resulting in a 4.04 (Not Found) response.

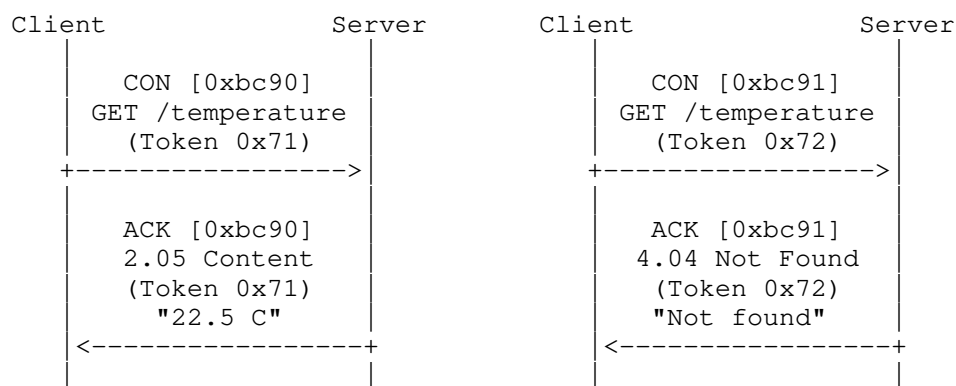


Figure 4: Two GET requests with piggy-backed responses

If the server is not able to respond immediately to a request carried in a Confirmable message, it simply responds with an empty Acknowledgement message so that the client can stop retransmitting the request. When the response is ready, the server sends it in a new Confirmable message (which then in turn needs to be acknowledged by the client). This is called a separate response, as illustrated in Figure 5 and described in more detail in Section 5.2.2.

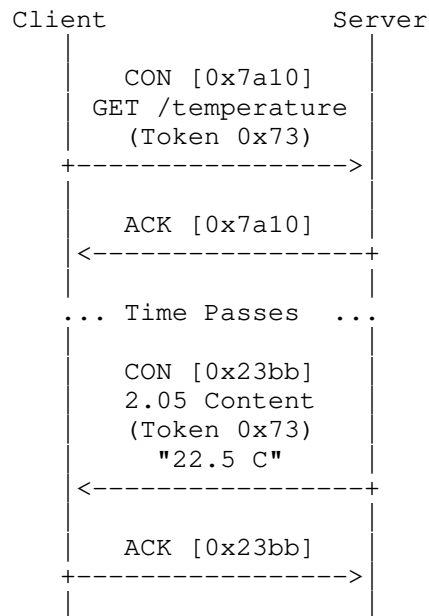


Figure 5: A GET request with a separate response

Likewise, if a request is sent in a Non-Confirmable message, then the response is usually sent using a new Non-Confirmable message, although the server may send a Confirmable message. This type of exchange is illustrated in Figure 6.

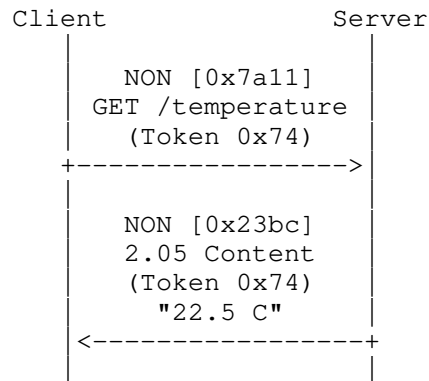


Figure 6: A NON request and response

CoAP makes use of GET, PUT, POST and DELETE methods in a similar manner to HTTP, with the semantics specified in Section 5.8. (Note that the detailed semantics of CoAP methods are "almost, but not

entirely unlike" those of HTTP methods: Intuition taken from HTTP experience generally does apply well, but there are enough differences that make it worthwhile to actually read the present specification.)

URI support in a server is simplified as the client already parses the URI and splits it into host, port, path and query components, making use of default values for efficiency. Response codes correspond to a small subset of HTTP response codes with a few CoAP specific codes added, as defined in Section 5.9.

2.3. Intermediaries and Caching

The protocol supports the caching of responses in order to efficiently fulfill requests. Simple caching is enabled using freshness and validity information carried with CoAP responses. A cache could be located in an endpoint or an intermediary. Caching functionality is specified in Section 5.6.

Proxying is useful in constrained networks for several reasons, including network traffic limiting, to improve performance, to access resources of sleeping devices or for security reasons. The proxying of requests on behalf of another CoAP endpoint is supported in the protocol. When using a proxy, the URI of the resource to request is included in the request, while the destination IP address is set to the address of the proxy. See Section 5.7 for more information on proxy functionality.

As CoAP was designed according to the REST architecture and thus exhibits functionality similar to that of the HTTP protocol, it is quite straightforward to map from CoAP to HTTP and from HTTP to CoAP. Such a mapping may be used to realize an HTTP REST interface using CoAP, or for converting between HTTP and CoAP. This conversion can be carried out by a cross-protocol proxy ("cross-proxy"), which converts the method or response code, media type, and options to the corresponding HTTP feature. Section 10 provides more detail about HTTP mapping.

2.4. Resource Discovery

Resource discovery is important for machine-to-machine interactions, and is supported using the CoRE Link Format [RFC6690] as discussed in Section 7.

3. Message Format

CoAP is based on the exchange of short messages which, by default,

are transported over UDP (i.e. each CoAP message occupies the data section of one UDP datagram). CoAP may also be used over Datagram Transport Layer Security (DTLS) (see Section 9.1). It could also be used over other transports such as SMS, TCP or SCTP, the specification of which is out of this document's scope.

CoAP messages are encoded in a simple binary format. A message consists of a fixed-sized CoAP Header followed by options in Type-Length-Value (TLV) format and a payload. The number of options is determined by the header. The payload is made up of the bytes after the options, if any; its length is calculated from the datagram length.

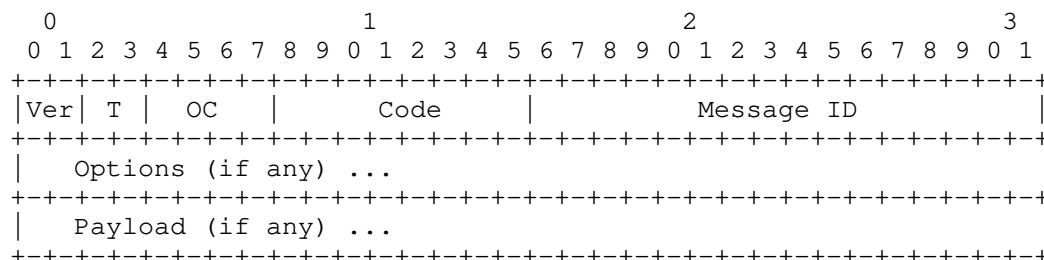


Figure 7: Message Format

3.1. Header Format

The fields in the header are defined as follows:

Version (Ver): 2-bit unsigned integer. Indicates the CoAP version number. Implementations of this specification **MUST** set this field to 1. Other values are reserved for future versions.

Type (T): 2-bit unsigned integer. Indicates if this message is of type Confirmable (0), Non-Confirmable (1), Acknowledgement (2) or Reset (3). See Section 4 for the semantics of these message types.

Option Count (OC): 4-bit unsigned integer. Indicates the number of options after the header (0-14). If set to 0, there are no options and the payload (if any) immediately follows the header. If set to 15, then an end-of-options marker is used to indicate the end of options and the start of the payload. The format of options is defined below.

Code: 8-bit unsigned integer. Indicates if the message carries a request (1-31) or a response (64-191), or is empty (0). (All other code values are reserved.) In case of a request, the Code field indicates the Request Method; in case of a response a Response Code. Possible values are maintained in the CoAP Code Registry (Section 12.1). See Section 5 for the semantics of requests and responses.

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 and messages of type Confirmable/Non-confirmable. See Section 4 for Message ID generation rules and how messages are matched.

3.2. Option Format

Options MUST appear in order of their Option Number (see Section 5.4.6). A delta encoding is used between options: The Option Number for each Option is calculated as the sum of its Option Delta field and the Option Number of the preceding Option in the message, if any. For the first Option in the message, the Option Delta becomes the Option Number (i.e., an implementation can simply initialize the number variable as zero). Multiple options with the same Option Number can be included by using an Option Delta of zero. The Option Jump mechanism (Section 3.3) is used when the delta to the next option number is greater than 14.

Following the Option Delta, each option has a Length field which specifies the length of the Option Value, in bytes. The Length field can be extended for options with values longer than 14 bytes by adding extension bytes. The maximum length for an option is 1034 bytes. The Option Value immediately follows the Length field.

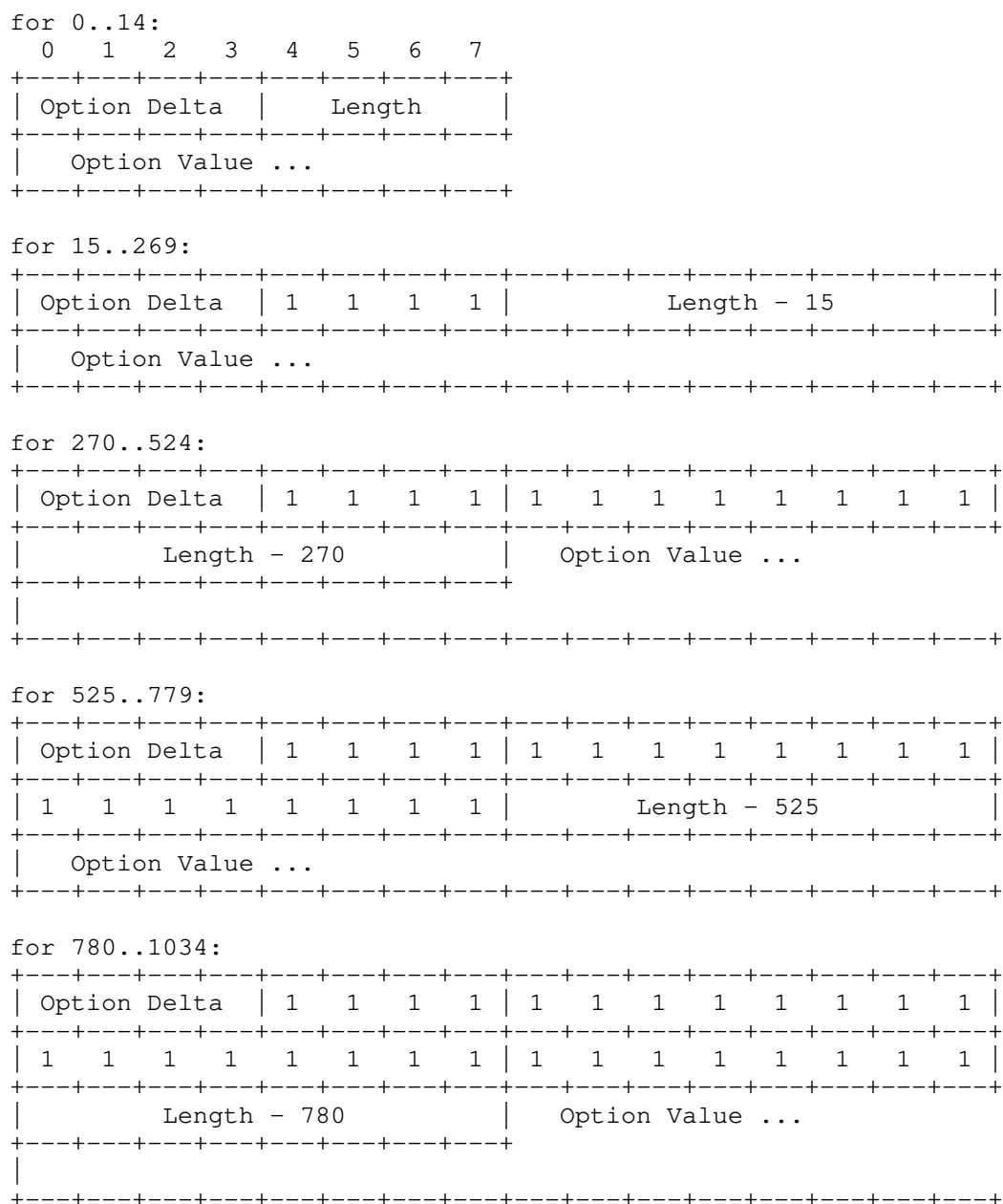


Figure 8: Option Format

The fields in an option are defined as follows:

Option Delta: 4-bit unsigned integer. Indicates the difference between the Option Number of this option and the previous option (or zero for the first option). In other words, the Option Number is calculated by simply summing the Option Delta fields of this and previous options before it. The Option Delta 15 is reserved for special constructs such as the end-of-options marker (see below) and Option Jumps. The Option Jump mechanism (Section 3.3) is used when the delta to the next option number is larger than 14.

Length: Indicates the length of the Option Value, in bytes. Normally Length is a 4-bit unsigned integer allowing value lengths of 0-14 bytes. When the Length field is set to 15, another byte is added as an 8-bit unsigned integer whose value is added to the 15, allowing option value lengths of 15-270 bytes. For option lengths beyond 270 bytes, we reserve the value 255 of an extension byte to mean "add 255, read another extension byte". Options that are longer than 1034 bytes MUST NOT be sent; an option that has 255 (all one bits) in the field called "Length - 780" MUST be rejected upon reception as an encoding error.

Value: The length and format of the Option Value depends on the respective option, which MAY define variable length values. See Section 3.4 for the formats the options defined in this document make use of; other options MAY make use of other option value formats.

If the Option Count field in the CoAP header is 15 and the Option Header byte is 0xf0 (the Option Delta is 15 and the Option Length is 0), the option is interpreted as the end-of-options marker instead of the option with the resulting Option Number. (In other words, the end-of-options marker always is just a single byte valued 0xf0.) When this marker is encountered, it is immediately followed by the payload (if any). (Note that, by this special meaning, the Option Delta of 15 is made special, not any specific Option Number.) The sender MUST NOT include the end-of-options marker in an Option in a message with an Option Count other than 15; recipients MUST treat this as an encoding error.

Option Numbers are maintained in the CoAP Option Number Registry (Section 12.2). See Section 5.10 for the semantics of the options defined in this document.

3.3. Option Jump

The following construct can occur in front of any Option:

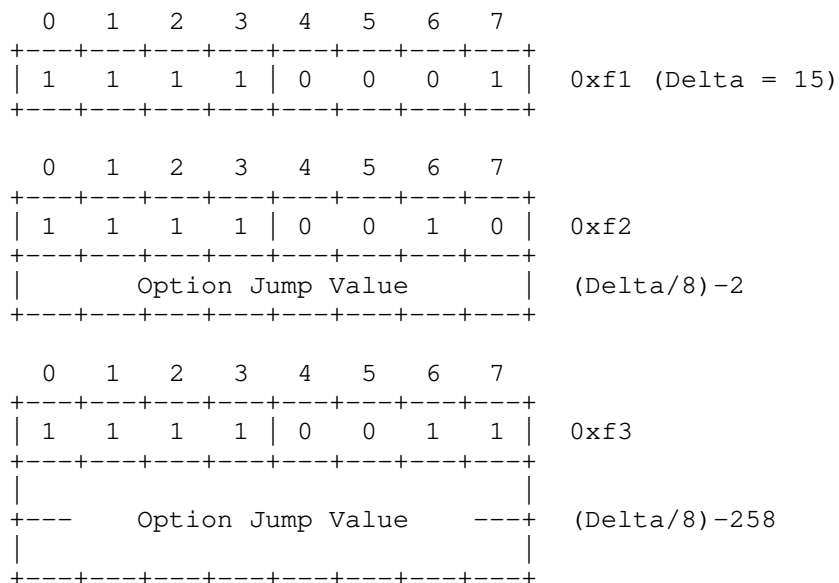


Figure 9: Option Jump Format

This construct is not by itself an Option. It can occur in front of any Option to increase the current Option number that then goes into its Option number calculation. The increase is done by 15 or in multiples of eight. For the formats that include an Option Jump Value, the actual addition to the current Option number is computed as follows:

$$\text{Delta} = ((\text{Option Jump Value}) + N) * 8$$

where N is 2 for the one-byte version and N is 258 for the two-byte version.

An Option Jump MUST be followed by an actual Option, i.e., it MUST NOT be followed by another Option Jump or an end-of-options indicator. A message violating this MUST be treated as an encoding error.

Option Jumps do NOT count as Options in the Option Count field of the header (i.e., they cannot by themselves end the Option sequence).

3.4. Option Value Formats

The options defined in this document make use of the following option value formats.

3.4.1. uint

A non-negative integer which is represented in network byte order using the given number of bytes. An option definition may specify a range of permissible numbers of bytes; if it has a choice, a sender SHOULD represent the integer with as few bytes as possible, i.e., without leading zeros. A recipient MUST be prepared to process values with leading zeros.

Implementation Note: The exceptional behavior permitted above is for highly constrained templated implementations (e.g. hardware implementations) that use fixed size options in the templates.

Length = 0 (implies value of 0)

```

      0
      0 1 2 3 4 5 6 7
+---+---+---+---+---+---+
Length = 1 |           0-255           |
+---+---+---+---+---+---+
```

```

      0                                     1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
Length = 2 |           0-65535           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Length = 3 is 24 bits, Length = 4 is 32 bits etc.

3.4.2. 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 unless 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.

3.4.3. opaque

An opaque sequence of bytes.

3.4.4. empty

A zero-length sequence of bytes.

4. Message Transmission

CoAP messages are exchanged asynchronously between CoAP endpoints. They are used to transport CoAP requests and responses, the semantics of which are defined in Section 5.

As CoAP is bound to non-reliable transports such as UDP, CoAP messages may arrive out of order, appear duplicated, or go missing without notice. For this reason, CoAP implements a lightweight reliability mechanism, without trying to re-create the full feature set of a transport like TCP. It has the following features:

- o Simple stop-and-wait retransmission reliability with exponential back-off for Confirmable messages.
- o Duplicate detection for both Confirmable and Non-confirmable messages.

4.1. Messages and Endpoints

A CoAP endpoint is the source or destination of a CoAP message. It is identified depending on the security mode used (see Section 9): With no security, the endpoint is solely identified by an IP address and a UDP port number. With other security modes, the endpoint is identified as defined by the security mode.

There are different types of messages. The type of a message is specified by the T 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 Code field in the CoAP header and is relevant to the request/response model. Possible values for the Code field are maintained by the CoAP Code Registry (Section 12.1).

An empty message has the Code field set to 0. The OC field SHOULD be set to 0 and no bytes SHOULD be present after the Message ID field. The OC field and any bytes trailing the header MUST be ignored by any recipient.

4.2. Messages Transmitted Reliably

The reliable transmission of a message is initiated by marking the message as Confirmable in the CoAP header. A Confirmable message always carries either a request or response and MUST NOT be empty. A recipient MUST acknowledge such a message with an Acknowledgement message or, if it lacks context to process the message properly (including the case where the message is empty or has an encoding error), MUST reject it; rejecting a Confirmable message is effected by sending a matching Reset message and otherwise ignoring it. The Acknowledgement message MUST echo the Message ID of the Confirmable message, and MUST carry a response or be empty (see Section 5.2.1 and Section 5.2.2). The Reset message MUST echo the Message ID of the confirmable message, and MUST be empty. Rejecting an Acknowledgement or Reset message is effected by silently ignoring it.

The sender retransmits the Confirmable message at exponentially increasing intervals, until it receives an acknowledgement (or Reset message), or runs out of attempts.

Retransmission is controlled by two things that a CoAP endpoint MUST keep track of for each Confirmable message it sends while waiting for an acknowledgement (or reset): a timeout and a retransmission counter. For a new Confirmable message, the initial timeout is set to a random number between ACK_TIMEOUT and (ACK_TIMEOUT * ACK_RANDOM_FACTOR) (see Section 4.8), and the retransmission counter is set to 0. When the timeout is triggered and the retransmission counter is less than MAX_RETRANSMIT, the message is retransmitted, the retransmission counter is incremented, and the timeout is doubled. If the retransmission counter reaches MAX_RETRANSMIT on a timeout, or if the endpoint receives a Reset message, then the attempt to transmit the message is canceled and the application process informed of failure. On the other hand, if the endpoint receives an acknowledgement message in time, transmission is considered successful.

A CoAP endpoint that sent a Confirmable message MAY give up in attempting to obtain an ACK even before the MAX_RETRANSMIT counter value is reached: E.g., the application has canceled the request as it no longer needs a response, or there is some other indication that the CON message did arrive. In particular, a CoAP request message may have elicited a separate response, in which case it is clear to the requester that only the ACK was lost and a retransmission of the request would serve no purpose. However, a responder MUST NOT in turn rely on this cross-layer behavior from a requester, i.e. it SHOULD retain the state to create the ACK for the request, if needed, even if a Confirmable response was already acknowledged by the requester.

4.3. Messages Transmitted Without Reliability

Some messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual success is sufficient.

As a more lightweight alternative, a message can be transmitted less reliably by marking the message as Non-confirmable. A Non-confirmable message always carries either a request or response and MUST NOT be empty. A Non-confirmable message MUST NOT be acknowledged by the recipient. If a recipient lacks context to process the message properly (including the case where the message is empty or has an encoding error), it MUST reject the message; rejecting a Non-Confirmable message MAY involve sending a matching Reset message, and apart from the Reset message the rejected message MUST be silently ignored.

At the CoAP level, there is no way for the sender to detect if a Non-confirmable message was received or not. A sender MAY choose to transmit multiple copies of a Non-confirmable message within MAX_TRANSMIT_SPAN, or the network may duplicate the message in transit. To enable the receiver to act only once on the message, Non-confirmable messages specify a Message ID as well. (This Message ID is drawn from the same number space as the Message IDs for Confirmable messages.)

4.4. Message Correlation

An Acknowledgement or Reset message is related to a Confirmable message or Non-confirmable message by means of a Message ID along with additional address information of the corresponding endpoint. The Message ID is a 16-bit unsigned integer that is generated by the sender of a Confirmable or Non-confirmable message and included in the CoAP header. The Message ID MUST be echoed in the Acknowledgement or Reset message by the recipient.

The same Message ID MUST NOT be re-used (in communicating with the same endpoint) within the EXCHANGE_LIFETIME (Section 4.8.2).

Implementation Note: Several implementation strategies can be employed for generating Message IDs. In the simplest case a CoAP endpoint generates Message IDs by keeping a single Message ID variable, which is changed each time a new confirmable or non-confirmable message is sent regardless of the destination address or port. Endpoints dealing with large numbers of transactions could keep multiple Message ID variables, for example per prefix or destination address. The initial variable value should be

randomized.

For an Acknowledgement or Reset message to match a Confirmable or Non-confirmable message, the Message ID and source endpoint of the Acknowledgement or Reset message MUST match the Message ID and destination endpoint of the Confirmable or Non-confirmable message.

4.5. Message Deduplication

A recipient MUST be prepared to receive the same Confirmable message (as indicated by the Message ID and source endpoint) multiple times within the EXCHANGE_LIFETIME (Section 4.8.2), for example, when its Acknowledgement went missing or didn't reach the original sender before the first timeout. The recipient SHOULD acknowledge each duplicate copy of a Confirmable message using the same Acknowledgement or Reset message, but SHOULD process any request or response in the message only once. This rule MAY be relaxed in case the Confirmable message transports a request that is idempotent (see Section 5.1) or can be handled in an idempotent fashion. Examples for relaxed message deduplication:

- o A server MAY relax the requirement to answer all retransmissions of an idempotent request with the same response (Section 4.2), so that it does not have to maintain state for Message IDs. For example, an implementation might want to process duplicate transmissions of a GET, PUT or DELETE request as separate requests if the effort incurred by duplicate processing is less expensive than keeping track of previous responses would be.
- o A constrained server MAY even want to relax this requirement for certain non-idempotent requests if the application semantics make this trade-off favorable. For example, if the result of a POST request is just the creation of some short-lived state at the server, it may be less expensive to incur this effort multiple times for a request than keeping track of whether a previous transmission of the same request already was processed.

A recipient MUST be prepared to receive the same Non-confirmable message (as indicated by the Message ID and source endpoint) multiple times within NON_LIFETIME (Section 4.8.2). As a general rule that may be relaxed based on the specific semantics of a message, the recipient SHOULD silently ignore any duplicated Non-confirmable message, and SHOULD process any request or response in the message only once.

4.6. Message Size

While specific link layers make it beneficial to keep CoAP messages small enough to fit into their link layer packets (see Section 1), this is a matter of implementation quality. The CoAP specification itself provides only an upper bound to the message size. Messages larger than an IP fragment result in undesired packet fragmentation. A CoAP message, appropriately encapsulated, SHOULD fit within a single IP packet (i.e., avoid IP fragmentation) and (by fitting into one UDP payload) obviously MUST fit within a single IP datagram. If the Path MTU is not known for a destination, an IP MTU of 1280 bytes SHOULD be assumed; if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size.

Implementation Note: CoAP's choice of message size parameters works well with IPv6 and with most of today's IPv4 paths. (However, with IPv4, it is harder to absolutely ensure that there is no IP fragmentation. If IPv4 support on unusual networks is a consideration, implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes; worse, the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload. Implementations extremely focused on this problem set might also set the IPv4 DF bit and perform some form of path MTU discovery; this should generally be unnecessary in most realistic use cases for CoAP, however.) A more important kind of fragmentation in many constrained networks is that on the adaptation layer (e.g., 6LoWPAN L2 packets are limited to 127 bytes including various overheads); this may motivate implementations to be frugal in their packet sizes and to move to block-wise transfers [I-D.ietf-core-block] when approaching three-digit message sizes.

Message sizes are also of considerable importance to implementations on constrained nodes. Many implementations will need to allocate a buffer for incoming messages. If an implementation is too constrained to allow for allocating the above-mentioned upper bound, it could apply the following implementation strategy: Implementations receiving a datagram into a buffer that is too small are usually able to determine if the trailing portion of a datagram was discarded and to retrieve the initial portion. So, if not all of the payload, at least the CoAP header and options are likely to fit within the buffer. A server can thus fully interpret a request and return a 4.13 (Request Entity Too Large) response code if the payload was truncated. A client sending an idempotent request and receiving a response larger than would fit in the buffer can repeat the request with a

suitable value for the Block Option [I-D.ietf-core-block].

4.7. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.2.

In order not to cause congestion, Clients (including proxies) MUST strictly limit the number of simultaneous outstanding interactions that they maintain to a given server (including proxies) to NSTART. An outstanding interaction is either a CON for which an ACK has not yet been received but is still expected (message layer) or a request for which neither a response nor an Acknowledgment message has yet been received but is still expected (which may both occur at the same time, counting as one outstanding interaction). The default value of NSTART for this specification is 1.

Further congestion control optimizations and considerations are expected in the future, which may for example provide automatic initialization of the CoAP transmission parameters defined in Section 4.8, and thus may allow a value for NSTART greater than one.

A client stops expecting a response to a Confirmable request for which no acknowledgment message was received, after EXCHANGE_LIFETIME. The specific algorithm by which a client stops to "expect" a response to a Confirmable request that was acknowledged, or to a Non-confirmable request, is not defined. Unless this is modified by additional congestion control optimizations, it MUST be chosen in such a way that an endpoint does not exceed an average data rate of PROBING_RATE in sending to another endpoint that does not respond.

Note: CoAP places the onus of congestion control mostly on the clients. However, clients may malfunction or actually be attackers, e.g. to perform amplification attacks (Section 11.3). To limit the damage (to the network and to its own energy resources), a server SHOULD implement some rate limiting for its response transmission based on reasonable assumptions about application requirements. This is most helpful if the rate limit can be made effective for the misbehaving endpoints, only.

4.8. Transmission Parameters

Message transmission is controlled by the following parameters:

name	default value
ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1
DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 Byte/second

4.8.1. Changing The Parameters

The values for ACK_TIMEOUT, ACK_RANDOM_FACTOR, MAX_RETRANSMIT, NSTART, DEFAULT_LEISURE, and PROBING_RATE may be configured to values specific to the application environment (including dynamically adjusted values), however the configuration method is out of scope of this document. It is recommended that an application environment use consistent values for these parameters.

The transmission parameters have been chosen to achieve a behavior in the presence of congestion that is safe in the Internet. If a configuration desires to use different values, the onus is on the configuration to ensure these congestion control properties are not violated. In particular, a decrease of ACK_TIMEOUT below 1 second would violate the guidelines of [RFC5405].

([I-D.allman-tcpm-rto-consider] provides some additional background.) CoAP was designed to enable implementations that do not maintain round-trip-time (RTT) measurements. However, where it is desired to decrease the ACK_TIMEOUT significantly or increase NSTART, this can only be done safely when maintaining such measurements. Configurations MUST NOT decrease ACK_TIMEOUT or increase NSTART without using mechanisms that ensure congestion control safety, either defined in the configuration or in future standards documents.

ACK_RANDOM_FACTOR MUST NOT be decreased below 1.0, and it SHOULD have a value that is sufficiently different from 1.0 to provide some protection from synchronization effects.

MAX_RETRANSMIT can be freely adjusted, but a too small value will reduce the probability that a confirmable message is actually received, while a larger value than given here will require further adjustments in the time values (see discussion below).

If the choice of transmission parameters leads to an increase of derived time values (see below), the configuration mechanism MUST ensure the adjusted value is also available to all the endpoints in communicating with which these adjusted values are to be used.

4.8.2. Time Values derived from Transmission Parameters

The combination of ACK_TIMEOUT, ACK_RANDOM_FACTOR and MAX_RETRANSMIT influences the timing of retransmissions, which in turn influences how long certain information items need to be kept by an implementation. To be able to unambiguously reference these derived time values, we give them names as follows:

- o MAX_TRANSMIT_SPAN is the maximum time from the first transmission of a confirmable message to its last retransmission. For the default transmission parameters, the value is $(2+4+8+16)*1.5 = 45$ seconds, or more generally:

$$\text{ACK_TIMEOUT} * (2 ** \text{MAX_RETRANSMIT} - 1) * \text{ACK_RANDOM_FACTOR}$$

- o MAX_TRANSMIT_WAIT is the maximum time from the first transmission of a confirmable message to the time when the sender gives up on receiving an acknowledgement or reset. For the default transmission parameters, the value is $(2+4+8+16+32)*1.5 = 93$ seconds, or more generally:

$$\text{ACK_TIMEOUT} * (2 ** (\text{MAX_RETRANSMIT} + 1) - 1) * \text{ACK_RANDOM_FACTOR}$$

In addition, some assumptions need to be made on the characteristics of the network and the nodes.

- o MAX_LATENCY is the maximum time a datagram is expected to take from the start of its transmission to the completion of its reception. This constant is related to the MSL (Maximum Segment Lifetime) of [RFC0793], which is "arbitrarily defined to be 2 minutes" ([RFC0793] glossary, page 81). Note that this is not necessarily smaller than MAX_TRANSMIT_WAIT, as MAX_LATENCY is not intended to describe a situation when the protocol works well, but the worst case situation against which the protocol has to guard. We, also arbitrarily, define MAX_LATENCY to be 100 seconds. Apart from being reasonably realistic for the bulk of configurations as well as close to the historic choice for TCP, this value also allows message ID lifetime timers to be represented in 8 bits (when measured in seconds). In these calculations, there is no assumption that the direction of the transmission is irrelevant (i.e. that the network is symmetric), just that the same value can reasonably be used as a maximum value for both directions. If that is not the case, the following calculations become only slightly more complex.
- o PROCESSING_DELAY is the time a node takes to turn around a confirmable message into an acknowledgement. We assume the node

will attempt to send an ACK before having the sender time out, so as a conservative assumption we set it equal to ACK_TIMEOUT.

- o MAX_RTT is the maximum round-trip time, or:

$$2 * MAX_LATENCY + PROCESSING_DELAY$$

From these values, we can derive the following values relevant to the protocol operation:

- o EXCHANGE_LIFETIME is the time from starting to send a confirmable message to the time when an acknowledgement is no longer expected, i.e. message layer information about the message exchange can be purged. EXCHANGE_LIFETIME includes a MAX_TRANSMIT_SPAN, a MAX_LATENCY forward, PROCESSING_DELAY, and a MAX_LATENCY for the way back. Note that there is no need to consider MAX_TRANSMIT_WAIT if the configuration is chosen such that the last waiting period (ACK_TIMEOUT * (2 ** MAX_RETRANSMIT) or the difference between MAX_TRANSMIT_SPAN and MAX_TRANSMIT_WAIT) is less than MAX_LATENCY -- which is a likely choice, as MAX_LATENCY is a worst case value unlikely to be met in the real world. In this case, EXCHANGE_LIFETIME simplifies to:

$$(ACK_TIMEOUT * (2 ** MAX_RETRANSMIT - 1) * ACK_RANDOM_FACTOR) + (2 * MAX_LATENCY) + PROCESSING_DELAY$$

or 248 seconds with the default transmission parameters.

- o NON_LIFETIME is the time from sending a non-confirmable message to the time its message-ID can be safely reused. If multiple transmission of a NON message is not used, its value is MAX_LATENCY, or 100 seconds. However, a CoAP sender might send a NON message multiple times, in particular for multicast applications. While the period of re-use is not bounded by the specification, an expectation of reliable detection of duplication at the receiver is in the timescales of MAX_TRANSMIT_SPAN. Therefore, for this purpose, it is safer to use the value:

$$MAX_TRANSMIT_SPAN + MAX_LATENCY$$

or 145 seconds with the default transmission parameters; however, an implementation that just wants to use a single timeout value for retiring message-IDs can safely use the larger value for EXCHANGE_LIFETIME.

5. Request/Response Semantics

CoAP operates under a similar request/response model as HTTP: a CoAP endpoint in the role of a "client" sends one or more CoAP requests to a "server", which services the requests by sending CoAP responses. Unlike HTTP, requests and responses are not sent over a previously established connection, but exchanged asynchronously over CoAP messages.

5.1. Requests

A CoAP request consists of the method to be applied to the resource, the identifier of the resource, a payload and Internet media type (if any), and optional meta-data about the request.

CoAP supports the basic methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP. They have the same properties of safe (only retrieval) and idempotent (you can invoke it multiple times with the same effects) as HTTP (see Section 9.1 of [RFC2616]). The GET method is safe, therefore it MUST NOT take any other action on a resource other than retrieval. The GET, PUT and DELETE methods MUST be performed in such a way that they are idempotent. POST is not idempotent, because its effect is determined by the origin server and dependent on the target resource; it usually results in a new resource being created or the target resource being updated.

A request is initiated by setting the Code field in the CoAP header of a Confirmable or a Non-confirmable message to a Method Code and including request information.

The methods used in requests are described in detail in Section 5.8.

5.2. Responses

After receiving and interpreting a request, a server responds with a CoAP response, which is matched to the request by means of a client-generated token.

A response is identified by the Code field in the CoAP header being set to a Response Code. Similar to the HTTP Status Code, the CoAP Response Code indicates the result of the attempt to understand and satisfy the request. These codes are fully defined in Section 5.9. The Response Code numbers to be set in the Code field of the CoAP header are maintained in the CoAP Response Code Registry (Section 12.1.2).

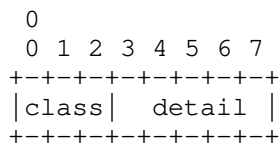


Figure 10: Structure of a Response Code

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 10). There are 3 classes:

- 2 - Success: The request was successfully received, understood, and accepted.
- 4 - Client Error: The request contains bad syntax or cannot be fulfilled.
- 5 - Server Error: The server failed to fulfill an apparently valid request.

The response codes are designed to be extensible: Response Codes in the Client Error and Server Error class that are unrecognized by an endpoint MUST be treated as being equivalent to the generic Response Code of that class (4.00 and 5.00, respectively). However, there is no generic Response Code indicating success, so a Response Code in the Success class that is unrecognized by an endpoint can only be used to determine that the request was successful without any further details.

As a human readable notation for specifications and protocol diagnostics, the numeric value of a response code is indicated by giving the upper three bits in decimal, followed by a dot and then the lower five bits in a two-digit decimal. E.g., "Not Found" is written as 4.04 -- indicating a value of hexadecimal 0x84 or decimal 132. In other words, the dot "." functions as a short-cut for "*32+".

The possible response codes are described in detail in Section 5.9.

Responses can be sent in multiple ways, which are defined below.

5.2.1. Piggy-backed

In the most basic case, the response is carried directly in the Acknowledgement message that acknowledges the request (which requires that the request was carried in a Confirmable message). This is

called a "Piggy-backed" Response.

The response is returned in the Acknowledgement message independent of whether the response indicates success or failure. In effect, the response is piggy-backed on the Acknowledgement message, so no separate message is required to both acknowledge that the request was received and return the response.

Implementation Note: The protocol leaves the decision whether to piggy-back a response or not (i.e., send a separate response) to the server. The client **MUST** be prepared to receive either. On the quality of implementation level, there is a strong expectation that servers will implement code to piggy-back whenever possible -- saving resources in the network and both at the client and at the server.

5.2.2. Separate

It may not be possible to return a piggy-backed response in all cases. For example, a server might need longer to obtain the representation of the resource requested than it can wait sending back the Acknowledgement message, without risking the client to repeatedly retransmit the request message. Responses to requests carried in a Non-Confirmable message are always sent separately (as there is no Acknowledgement message).

The server maybe initiates the attempt to obtain the resource representation and times out an acknowledgement timer, or it immediately sends an acknowledgement knowing in advance that there will be no piggy-backed response. The acknowledgement effectively is a promise that the request will be acted upon.

When the server finally has obtained the resource representation, it sends the response. When it is desired that this message is not lost, it is sent as a Confirmable message from the server to the client and answered by the client with an Acknowledgement, echoing the new Message ID chosen by the server. (It may also be sent as a Non-Confirmable message; see Section 5.2.3.)

Implementation Notes: Note that, as the underlying datagram transport may not be sequence-preserving, the Confirmable message carrying the response may actually arrive before or after the acknowledgement message for the request. Note also that, while the CoAP protocol itself does not make any specific demands here, there is an expectation that the response will come within a time frame that is reasonable from an application point of view; as there is no underlying transport protocol that could be instructed to run a keep-alive mechanism, the requester **MAY** want to set up a

timeout that is unrelated to CoAP's retransmission timers in case the server is destroyed or otherwise unable to send the response.)

An exchange is separate by definition when the Acknowledgement to the Confirmable request is an empty message. The Acknowledgement to the Confirmable response MUST also be an empty message, i.e. one that carries neither a request nor a response. However, a server MUST stop retransmitting its response on any matching Acknowledgement (silently ignoring any response code or payload) or Reset message.

5.2.3. Non-Confirmable

If the request message is Non-confirmable, then the response SHOULD be returned in a Non-confirmable message as well. However, an endpoint MUST be prepared to receive a Non-confirmable response (preceded or followed by an empty acknowledgement message) in reply to a Confirmable request, or a Confirmable response in reply to a Non-confirmable request.

5.3. Request/Response Matching

Regardless of how a response is sent, it is matched to the request by means of a token that is included by the client in the request as one of the options along with additional address information of the corresponding endpoint. The token MUST be echoed by the server in any resulting response without modification.

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.

The client SHOULD generate tokens in a way that tokens currently in use for a given source/destination pair are unique. (Note that a client can use the same token for any request if it uses a different source port number each time.)

An endpoint that did not generate a token MUST treat it as opaque and make no assumptions about its format. (Note that there is a default value for the Token Option, so every message carries a token, even if it is not explicitly expressed in a CoAP option.)

In case a message carrying a response is unexpected (i.e. the client is not waiting for a response at the endpoint addressed and/or with the given token), the response is rejected (Section 4.2, Section 4.3).

Implementation Note: A client that receives a response in a CON message may want to clean up the message state right after sending the ACK. If that ACK is lost and the server retransmits the CON, the client may no longer have any state to correlate this response to, making the retransmission an unexpected message; the client may send a Reset message so it does not receive any more retransmissions. This behavior is normal and not an indication of an error. (Clients that are not aggressively optimized in their state memory usage will still have message state that will identify the second CON as a retransmission. Clients that actually expect more messages from the server [I-D.ietf-core-observe] will have to keep state in any case.)

5.4. Options

Both requests and responses may include a list of one or more options. For example, the URI in a request is transported in several options, and meta-data that would be carried in an HTTP header in HTTP is supplied as options as well.

CoAP defines a single set of options that are used in both requests and responses:

- o Content-Format
- o ETag
- o Location-Path
- o Location-Query
- o Max-Age
- o Proxy-Uri
- o Token
- o Uri-Host
- o Uri-Path
- o Uri-Port

- o Uri-Query
- o Accept
- o If-Match
- o If-None-Match

The semantics of these options along with their properties are defined in detail in Section 5.10.

Not all options are defined for use with all methods and response codes. The possible options for methods and response codes are defined in Section 5.8 and Section 5.9 respectively. In case an option is not defined for a method or response code, it **MUST NOT** be included by a sender and **MUST** be treated like an unrecognized option by a recipient.

An Option number is constructed with a bit mask to indicate if an option is Critical/Elective, Unsafe/Safe and in the case of Safe, also a Cache-Key as indicated by the following figure. When bit 7 (the least significant bit) is 1, an option is Critical (and likewise Elective when 0). When bit 6 is 1, an option is Unsafe (and likewise Safe when 0). When an 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.

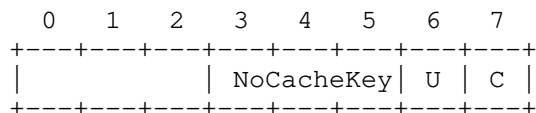


Figure 11: Option Number Mask

An endpoint may use an equivalent of the following C code to derive the characteristics of an option number "onum":

```
Critical = (onum & 1);
Unsafe = (onum & 2);
NoCacheKey = ((onum & 0x1e) == 0x1c);
```

Figure 12

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 message describing the unrecognized option(s) (see Section 5.5.2).
- o Unrecognized options of class "critical" that occur in a confirmable response, or piggy-backed in an acknowledgement, MUST cause the response to be rejected (Section 4.2).
- o Unrecognized options of class "critical" that occur in a non-confirmable message MUST cause the message to be rejected (Section 4.3).

Note that, whether critical or elective, an option is never "mandatory" (it is always optional): These rules are defined in order to enable implementations to stop processing options they do not understand or implement.

Critical/Elective rules apply to non-proxying endpoints. A proxy processes options based on Unsafe/Safe classes as defined in Section 5.7.

5.4.2. Proxy Unsafe/Safe and Cache-Key

In addition to an option being marked as Critical or Elective, options are also classified based on how a proxy is to deal with the option if it does not recognize it. For this purpose, an option can either be considered Unsafe to Forward (Unsafe is set) or Safe to Forward (Unsafe is clear).

In addition, for options that are marked Safe to Forward, the option indicates whether it is intended to be part of the Cache-Key in a request (NoCacheKey is not all set) or not (NoCacheKey is set).

Note: The Cache-Key indication is relevant only for proxies that do not implement the given option as a request option and instead rely on the Safe/Unsafe indication only. E.g., for ETag, actually using the request option as a cache key is grossly inefficient, but it is the best thing one can do if ETag is not implemented by

a proxy, as the response is going to differ based on the presence of the request option. A more useful proxy that does implement the ETag request option is not using ETag as a cache key.

Proxy behavior with regard to these classes is defined in Section 5.7.

5.4.3. Length

Option values are defined to have a specific length, often in the form of an upper and lower bound. If the length of an option value in a request is outside the defined range, that option MUST be treated like an unrecognized option (see Section 5.4.1).

5.4.4. Default Values

Options may be defined to have a default value. If the value of option is intended to be this default value, the option SHOULD NOT be included in the message. If the option is not present, the default value MUST be assumed.

Where a critical option has a default value, this is chosen in such a way that the absence of the option in a message can be processed properly both by implementations unaware of the critical option and by implementations that interpret this absence as the presence of the default value for the option.

5.4.5. Repeatable Options

The definition of an option MAY specify the option to be repeatable. An option that is repeatable MAY be included one or more times in a message. An option that is not repeatable MUST NOT be included more than once in a message.

If a message includes an option with more occurrences than the option is defined for, the additional option occurrences MUST be treated like an unrecognized option (see Section 5.4.1).

5.4.6. Option Numbers

Options are identified by an option number. 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.)

The option numbers for the options defined in this document are listed in the CoAP Option Number Registry (Section 12.2).

5.5. Payload

Both requests and responses may include payload, depending on the method or response code respectively. If a method or response code is not defined to have a payload, then a sender **MUST NOT** include one, and a recipient **MUST** ignore it.

5.5.1. Representation

The payload of requests or of responses indicating success is typically a representation of a resource or the result of the requested action. Its format is specified by the Internet media type and content coding given by the Content-Format Option. In the absence of this option, no default value is assumed and the format must be inferred by the application (e.g., from the application context or by "sniffing" the payload).

5.5.2. Diagnostic Message

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 Content-Format Option **MUST NOT** be included by the sender and **MUST** be treated like an unrecognized option by the recipient.

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 message **SHOULD** be empty if there is no additional information beyond the response code.

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 the Token, Max-Age, or ETag request option(s), or any request options marked as NoCacheKey (Section 5.4), and
- o the stored response is either fresh or successfully validated as defined below.

5.6.1. Freshness Model

When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.

The mechanism for determining freshness is for an origin server to provide an explicit expiration time in the future, using the Max-Age Option (see Section 5.10.6). 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 5.10.7) in the GET request to give the

origin server an opportunity to both select a stored response to be used, and to update its freshness. This process is known as "validating" or "revalidating" the stored response.

When sending such a request, the endpoint SHOULD add an ETag Option specifying the entity-tag of each stored response that is applicable.

A 2.03 (Valid) response indicates the stored response identified by the entity-tag given in the response's ETag Option can be reused, after updating its freshness with the value of the Max-Age Option that is included (explicitly, or implicitly as a default value) with the response (see Section 5.9.1.3).

Any other response code indicates that none of the stored responses nominated in the request is suitable. Instead, the response SHOULD be used to satisfy the request and MAY replace the stored response.

5.7. Proxying

A proxy is a CoAP endpoint that can be tasked by CoAP clients to perform requests on their behalf. This may be useful, for example, when the request could otherwise not be made, or to service the response from a cache in order to reduce response time and network bandwidth or energy consumption.

In an overall architecture for a Constrained RESTful Environment, proxies can serve quite different purposes. Proxies can be explicitly selected by clients, a role that we term "forward-proxy". Proxies can also be inserted to stand in for origin servers, a role that we term "reverse-proxy". Orthogonal to this distinction, a proxy can map from a CoAP request to a CoAP request (CoAP-to-CoAP proxy) or translate from or to a different protocol ("cross-proxy"). Full definitions of these terms are provided in Section 1.2.

Notes: The terminology in this specification has been selected to be culturally compatible with the terminology used in the wider Web application environments, without necessarily matching it in every detail (which may not even be relevant to Constrained RESTful Environments). Not too much semantics should be ascribed to the components of the terms (such as "forward", "reverse", or "cross").

HTTP proxies, besides acting as HTTP proxies, often offer a transport protocol proxying function ("CONNECT") to enable end-to-end transport layer security through the proxy. No such function is defined for CoAP-to-CoAP proxies in this specification, as forwarding of UDP packets is unlikely to be of much value in Constrained RESTful environments. See also Section 10.2.7 for the

cross-proxy case.

5.7.1. Proxy Operation

A proxy generally needs a way to determine potential request parameters for a request to a destination based on the request it received. This way is fully specified for a forward-proxy, but may depend on the specific configuration for a reverse-proxy. In particular, the client of a reverse-proxy generally does not indicate a locator for the destination, necessitating some form of namespace translation in the reverse-proxy. However, some aspects of the operation of proxies are common to all its forms.

If a proxy does not employ a cache, then it simply forwards the translated request to the determined destination. Otherwise, if it does employ a cache but does not have a stored response that matches the translated request and is considered fresh, then it needs to refresh its cache according to Section 5.6. For options in the request that the proxy recognizes, it knows whether the option is intended to act as part of the key used in looking up the cached value or not. E.g., since requests for different Uri-Path values address different resources, Uri-Path values are always parts of the cache key, while, e.g., Token values are never part of the cache key. For options that the proxy does not recognize but that are marked Safe in the option number, the option also indicates whether it is to be included in the cache key (NoCacheKey is not all set) or not (NoCacheKey is all set). (Options that are unrecognized and marked Unsafe lead to 4.02 Bad Option.)

If the request to the destination times out, then a 5.04 (Gateway Timeout) response MUST be returned. If the request to the destination returns a response that cannot be processed by the proxy (e.g, due to unrecognized critical options, encoding errors), then a 5.02 (Bad Gateway) response MUST be returned. Otherwise, the proxy returns the response to the client.

If a response is generated out of a cache, it MUST be generated with a Max-Age Option that does not extend the max-age originally set by the server, considering the time the resource representation spent in the cache. E.g., the Max-Age Option could be adjusted by the proxy for each response using the formula:

$$\text{proxy-max-age} = \text{original-max-age} - \text{cache-age}$$

For example if a request is made to a proxied resource that was refreshed 20 seconds ago and had an original Max-Age of 60 seconds, then that resource's proxied max-age is now 40 seconds. Considering potential network delays on the way from the origin server, a proxy

SHOULD be conservative in the max-age values offered.

All options present in a proxy request MUST be processed at the proxy. Unsafe options in a request that are not recognized by the proxy MUST lead to a 4.02 (Bad Option) response being returned by the proxy. A CoAP-to-CoAP proxy MUST forward to the origin server all Safe options that it does not recognize. Similarly, Unsafe options in a response that are not recognized by the CoAP-to-CoAP proxy server MUST lead to a 5.02 (Bad Gateway) response. Again, Safe options that are not recognized MUST be forwarded.

Additional considerations for cross-protocol proxying between CoAP and HTTP are discussed in Section 10.

5.7.2. Forward-Proxies

CoAP distinguishes between requests made (as if) to an origin server and a request made through a forward-proxy. CoAP requests to a forward-proxy are made as normal confirmable or non-confirmable requests to the forward-proxy endpoint, but specify the request URI in a different way: The request URI in a proxy request is specified as a string in the Proxy-Uri Option (see Section 5.10.3), 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 5.10.2).

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 5.10.3), 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 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 option, but need to determine the destination (next hop) of a request from information in the request and information in their configuration. E.g., a reverse-proxy might offer various resources the existence of which it has

learned through resource discovery as if they were its own resources. The reverse-proxy is free to build a namespace for the URIs that identify these resources. A reverse-proxy may also build a namespace that gives the client more control over where the request goes, e.g. by embedding host identifiers and port numbers into the URI path of the resources offered.

In processing the response, a reverse-proxy has to be careful about namespacing the ETag option. In many cases, it can be forwarded unchanged. If the mapping from a resource offered by the reverse-proxy to resources offered by its various origin servers is not unique, the reverse-proxy may need to generate a new ETag, making sure the semantics of this option are properly preserved.

5.8. Method Definitions

In this section each method is defined along with its behavior. A request with an unrecognized or unsupported Method Code MUST generate a 4.05 (Method Not Allowed) piggy-backed response.

5.8.1. GET

The GET method retrieves a representation for the information that currently corresponds to the resource identified by the request URI. If the request includes one or more Accept Options, they indicate the preferred content-format of a response. If the request includes an ETag Option, the GET method requests that ETag be validated and that the representation be transferred only if validation failed. Upon success a 2.05 (Content) or 2.03 (Valid) response code SHOULD be present in the response.

The GET method is safe and idempotent.

5.8.2. POST

The POST method requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and dependent on the target resource. It usually results in a new resource being created or the target resource being updated.

If a resource has been created on the server, the response returned by the server SHOULD have a 2.01 (Created) response code and SHOULD include the URI of the new resource in a sequence of one or more Location-Path and/or Location-Query Options (Section 5.10.8). 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 5.10.9) or If-None-Match (see Section 5.10.10) options in the request.

PUT is not safe, but is idempotent.

5.8.4. DELETE

The DELETE method requests that the resource identified by the request URI be deleted. A 2.02 (Deleted) response code SHOULD be used on success or in case the resource did not exist before the request.

DELETE is not safe, but is idempotent.

5.9. Response Code Definitions

Each response code is described below, including any options required in the response. Where appropriate, some of the codes will be specified in regards to related response codes in HTTP [RFC2616]; this does not mean that any such relationship modifies the HTTP mapping specified in Section 10.

5.9.1. Success 2.xx

This class of status code indicates that the clients request was successfully received, understood, and accepted.

5.9.1.1. 2.01 Created

Like HTTP 201 "Created", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

If the response includes one or more Location-Path and/or Location-Query Options, the values of these options specify the location at which the resource was created. Otherwise, the resource was created at the request URI. A cache receiving this response MUST mark any stored response for the created resource as not fresh.

This response is not cacheable.

5.9.1.2. 2.02 Deleted

Like HTTP 204 "No Content", but only used in response to DELETE requests. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache SHOULD 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.

When a cache receives a 2.03 (Valid) response, it MUST update the stored response with the value of the Max-Age Option included in the response (see Section 5.6.2).

5.9.1.4. 2.04 Changed

Like HTTP 204 "No Content", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache MUST mark any stored response for the changed resource as not fresh.

5.9.1.5. 2.05 Content

Like HTTP 200 "OK", but only used in response to GET requests.

The payload returned with the response is a representation of the

target resource.

This response is cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1) and (if present) the ETag Option for validation (see Section 5.6.2).

5.9.2. Client Error 4.xx

This class of response code is intended for cases in which the client seems to have erred. These response codes are applicable to any request method.

The server SHOULD include a diagnostic message as detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

5.9.2.1. 4.00 Bad Request

Like HTTP 400 "Bad Request".

5.9.2.2. 4.01 Unauthorized

The client is not authorized to perform the requested action. The client SHOULD NOT repeat the request without previously improving its authentication status to the server. Which specific mechanism can be used for this is outside this document's scope; see also Section 9.

5.9.2.3. 4.02 Bad Option

The request could not be understood by the server due to one or more unrecognized or malformed options. The client SHOULD NOT repeat the request without modification.

5.9.2.4. 4.03 Forbidden

Like HTTP 403 "Forbidden".

5.9.2.5. 4.04 Not Found

Like HTTP 404 "Not Found".

5.9.2.6. 4.05 Method Not Allowed

Like HTTP 405 "Method Not Allowed", but with no parallel to the "Allow" header field.

5.9.2.7. 4.06 Not Acceptable

Like HTTP 406 "Not Acceptable", but with no response entity.

5.9.2.8. 4.12 Precondition Failed

Like HTTP 412 "Precondition Failed".

5.9.2.9. 4.13 Request Entity Too Large

Like HTTP 413 "Request Entity Too Large".

5.9.2.10. 4.15 Unsupported Content-Format

Like HTTP 415 "Unsupported Media Type".

5.9.3. Server Error 5.xx

This class of response code indicates cases in which the server is aware that it has erred or is incapable of performing the request. These response codes are applicable to any request method.

The server SHOULD include a diagnostic message as 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 (see Section 5.10.3).

5.10. Option Definitions

The individual CoAP options are summarized in Table 1 and explained below.

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x			Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x			Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x		x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x			Max-Age	uint	0-4	60
15	x	x		x	Uri-Query	string	1-255	(none)
16				x	Accept	uint	0-2	(none)
19	x	x			Token	opaque	1-8	(empty)
20				x	Location-Query	string	0-255	(none)
35	x	x			Proxy-Uri	string	1-1034	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Table 1: Options

Temporary Note: At the time of submission, the great renumbering is not yet reflected in [I-D.ietf-core-block] and [I-D.ietf-core-observe]. The new numbers are: 6 for Observe, 23 for Block2, 27 for Block1, and 28 for Size. This note to be removed when the satellite drafts are updated.

5.10.1. Token

The Token Option is used to match a response with a request. Every request has a client-generated token which the server **MUST** echo in any response. The token value is a sequence of 0 to 8 bytes. A default value of the zero-length token is assumed in the absence of the option. A value of 1 to 8 bytes can be sent as an option value. Thus when the token value is empty, the Token Option **MUST** be elided.

A token is intended for use as a client-local identifier for differentiating between concurrent requests (see Section 5.3). A client **SHOULD** generate tokens in a way that tokens currently in use for a given source/destination pair are unique. 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. There are however multiple possible implementation strategies to fulfill this. An endpoint receiving a token **MUST** treat it as opaque and make no assumptions about its format.

5.10.2. 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.

5.10.3. Proxy-Uri

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).

5.10.4. 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).

5.10.5. Accept

The CoAP Accept option indicates when included one or more times in a request, one or more Content-Formats, each of which is an acceptable Content-Format for the client, in the order of preference (most preferred first). The representation format is given as a numeric Content-Format identifier that is defined in the CoAP Content-Format registry (Section 12.3). If no Accept options are given, the client does not express a preference (thus no default value is assumed). The client prefers the representation returned by the server to be in one of the Content-Formats indicated. The server SHOULD return one of the preferred Content-Formats if available. If none of the preferred Content-Formats can be returned, then a 4.06 "Not Acceptable" SHOULD be sent as a response.

Note that as a server might not support the Accept option (and thus would ignore it as it is elective), the client needs to be prepared to receive a representation in a different Content-Format. The client can simply discard a representation it can not make use of.

5.10.6. Max-Age

The Max-Age Option indicates the maximum time a response may be cached before it MUST be considered not fresh (see Section 5.6.1).

The option value is an integer number of seconds between 0 and $2^{32}-1$ inclusive (about 136.1 years). A default value of 60 seconds is assumed in the absence of the option in a response.

The value is intended to be current at the time of transmission. Servers that provide resources with strict tolerances on the value of Max-Age SHOULD update the value before each retransmission. (See also Section 5.7.1.)

5.10.7. ETag

The ETag Option in a response provides the current value of the entity-tag for the enclosed representation of the target resource.

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 may be generated in any number of ways including a version, checksum, hash or time. An endpoint receiving an entity-tag MUST treat it as opaque and make no assumptions about its format. (Endpoints generating 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.)

An endpoint that has one or more representations previously obtained from the resource can specify the ETag Option in a request for each stored response to determine if any of those representations is current (see Section 5.6.2).

The ETag Option **MUST NOT** occur more than once in a response, and **MAY** occur one or more times in a request.

5.10.8. 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 and the implied URI identifies one or more currently stored responses, those entries **SHOULD** 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.

More Location-* options may be defined in the future, and have been reserved option numbers 128, 132 and 136. 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.

5.10.9. 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 server performs the request method as if the set of If-Match Options were not present.

If there is one or more If-Match Option, but none of the options match, the server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

If the request would, without the If-Match Options, result in anything other than a 2.xx or 4.12 response code, then any If-Match Options MUST be ignored.

5.10.10. 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 server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

6. CoAP URIs

CoAP uses the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. Resources are organized hierarchically and governed by a potential CoAP origin server listening for CoAP requests ("coap") or DTLS-secured CoAP requests ("coaps") on a given UDP port. The CoAP server is identified via the generic syntax's authority component, which includes a host component and optional UDP port number. The remainder of the URI is considered to be identifying a resource which can be operated on by the methods defined by the CoAP protocol. The "coap" and "coaps" URI schemes can thus be compared to the "http" and

"https" URI schemes respectively.

The syntax of the "coap" and "coaps" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", "query", "segment", "IP-literal", "IPv4address" and "reg-name" are adopted from [RFC3986].

Implementation Note: Unfortunately, over time the URI format has acquired significant complexity. Implementers are encouraged to examine [RFC3986] closely. E.g., the ABNF for IPv6 addresses is more complicated than maybe expected. Also, implementers should take care to perform the processing of percent decoding/encoding exactly once on the way from a URI to its decoded components or back. Percent encoding is crucial for data transparency, but may lead to unusual results such as a slash in a path component.

6.1. coap URI Scheme

coap-URI = "coap:" "://" host [":" port] path-abempty ["?" query]

If the host component is provided as an IP-literal or IPv4address, then the CoAP server can be reached at that IP address. If host is a registered name, then that name is considered an indirect identifier and the endpoint might use a name resolution service, such as DNS, to find the address of that host. The host MUST NOT be empty; if a URI is received with a missing authority or an empty host, then it MUST be considered invalid. The port subcomponent indicates the UDP port at which the CoAP server is located. If it is empty or not given, then the default port 5683 is assumed.

The path identifies a resource within the scope of the host and port. It consists of a sequence of path segments separated by a slash character (U+002F SOLIDUS "/").

The query serves to further parameterize the resource. It consists of a sequence of arguments separated by an ampersand character (U+0026 AMPERSAND "&"). An argument is often in the form of a "key=value" pair.

The "coap" URI scheme supports the path prefix "/.well-known/" defined by [RFC5785] for "well-known locations" in the name-space of a host. This enables discovery of policy or other information about a host ("site-wide metadata"), such as hosted resources (see Section 7).

Application designers are encouraged to make use of short, but descriptive URIs. As the environments that CoAP is used in are usually constrained for bandwidth and energy, the trade-off between

these two qualities should lean towards the shortness, without ignoring descriptiveness.

6.2. coaps URI Scheme

```
coaps-URI = "coaps:" "://" host [ ":" port ] path-abempty
           [ "?" query ]
```

All of the requirements listed above for the "coap" scheme are also requirements for the "coaps" scheme, except that a default UDP port of [IANA_TBD_PORT] is assumed if the port subcomponent is empty or not given, and the UDP datagrams MUST be secured for privacy through the use of DTLS as described in Section 9.1.

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 (Section 11.2). They can, however, be reused in a private cache if the message is cacheable by default in CoAP.

Resources made available via the "coaps" scheme have no shared identity with the "coap" scheme even if their resource identifiers indicate the same authority (the same host listening to the same UDP port). They are distinct name spaces and are considered to be distinct origin servers.

6.3. Normalization and Comparison Rules

Since the "coap" and "coaps" schemes conform to the URI generic syntax, such URIs are normalized and compared according to the algorithm defined in [RFC3986], Section 6, using the defaults described above for each scheme.

If the port is equal to the default port for a scheme, the normal form is to elide the port subcomponent. Likewise, an empty path component is equivalent to an absolute path of "/", so the normal form is to provide a path of "/" instead. The scheme and host are case-insensitive and normally provided in lowercase; IP-literals are in recommended form [RFC5952]; all other components are compared in a case-sensitive manner. Characters other than those in the "reserved" set are equivalent to their percent-encoded octets (see [RFC3986], Section 2.1): the normal form is to not encode them.

For example, the following three URIs are equivalent, and cause the same options and option values to appear in the CoAP messages:

```
coap://example.com:5683/~sensors/temp.xml
coap://EXAMPLE.com/%7Esensors/temp.xml
coap://EXAMPLE.com:/%7esensors/temp.xml
```

6.4. Decomposing URIs into Options

The steps to parse a request's options from a string `/url/` are as follows. These steps either result in zero or more of the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in the request, or they fail.

1. If the `/url/` string is not an absolute URI ([RFC3986]), then fail this algorithm.
2. Resolve the `/url/` string using the process of reference resolution defined by [RFC3986], with the URL character encoding set to UTF-8 [RFC3629].

NOTE: It doesn't matter what it is resolved relative to, since we already know it is an absolute URL at this point.

3. If `/url/` does not have a `<scheme>` component whose value, when converted to ASCII lowercase, is "coap" or "coaps", then fail this algorithm.
4. If `/url/` has a `<fragment>` component, then fail this algorithm.
5. If the `<host>` component of `/url/` does not represent the request's destination IP address as an IP-literal or IPv4address, include a Uri-Host Option and let that option's value be the value of the `<host>` component of `/url/`, converted to ASCII lowercase, and then converting all percent-encodings ("% followed by two hexadecimal digits) to the corresponding characters.

NOTE: In the usual case where the request's destination IP address is derived from the host part, this ensures that a Uri-Host Option is only used for a `<host>` component of the form `reg-name`.

6. If `/url/` has a `<port>` component, then let `/port/` be that component's value interpreted as a decimal integer; otherwise, let `/port/` be the default port for the scheme.
7. If `/port/` does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be `/port/`.
8. If the value of the `<path>` component of `/url/` is empty or consists of a single slash character (U+002F SOLIDUS "/"), then

move to the next step.

Otherwise, for each segment in the <path> component, include a Uri-Path Option and let that option's value be the segment (not including the delimiting slash characters) after converting all percent-encodings ("% followed by two hexadecimal digits) to the corresponding characters.

9. If /url/ has a <query> component, then, for each argument in the <query> component, include a Uri-Query Option and let that option's value be the argument (not including the question mark and the delimiting ampersand characters) after converting all percent-encodings to the corresponding characters.

Note that these rules completely resolve any percent-encoding.

6.5. Composing URIs from Options

The steps to construct a URI from a request's options are as follows. These steps either result in a URI, or they fail. In these steps, percent-encoding a character means replacing each of its (UTF-8 encoded) bytes by a "%" character followed by two hexadecimal digits representing the byte, where the digits A-F are in upper case (as defined in [RFC3986] Section 2.1; to reduce variability, the hexadecimal notation for percent-encoding in CoAP URIs MUST use uppercase letters). The definitions of "unreserved" and "sub-delims" are adopted from [RFC3986].

1. If the request is secured using DTLS, let /url/ be the string "coaps://". Otherwise, let /url/ be the string "coap://".
2. If the request includes a Uri-Host Option, let /host/ be that option's value, where any non-ASCII characters are replaced by their corresponding percent-encoding. If /host/ is not a valid reg-name or IP-literal or IPv4address, fail the algorithm. If the request does not include a Uri-Host Option, let /host/ be the IP-literal (making use of the conventions of [RFC5952]) or IPv4address representing the request's destination IP address.
3. Append /host/ to /url/.
4. If the request includes a Uri-Port Option, let /port/ be that option's value. Otherwise, let /port/ be the request's destination UDP port.
5. If /port/ is not the default port for the scheme, then append a single U+003A COLON character (:) followed by the decimal representation of /port/ to /url/.

6. Let /resource name/ be the empty string. For each Uri-Path Option in the request, append a single character U+002F SOLIDUS (/) followed by the option's value to /resource name/, after converting any character that is not either in the "unreserved" set, "sub-delims" set, a U+003A COLON (:) or U+0040 COMMERCIAL AT (@) character, to its percent-encoded form.
7. If /resource name/ is the empty string, set it to a single character U+002F SOLIDUS (/).
8. For each Uri-Query Option in the request, append a single character U+003F QUESTION MARK (?) (first option) or U+0026 AMPERSAND (&) (subsequent options) followed by the option's value to /resource name/, after converting any character that is not either in the "unreserved" set, "sub-delims" set (except U+0026 AMPERSAND (&)), a U+003A COLON (:), U+0040 COMMERCIAL AT (@), U+002F SOLIDUS (/) or U+003F QUESTION MARK (?) character, to its percent-encoded form.
9. Append /resource name/ to /url/.
10. Return /url/.

Note that these steps have been designed to lead to a URI in normal form (see Section 6.3).

7. Discovery

7.1. Service Discovery

A server is discovered by a client by the client knowing or learning a URI that references a resource in the namespace of the server. Alternatively, clients can use Multicast CoAP (see Section 8) and the "All CoAP Nodes" multicast address to find CoAP servers.

Unless the port subcomponent in a "coap" or "coaps" URI indicates the UDP port at which the CoAP server is located, the server is assumed to be reachable at the default port.

The CoAP default port number 5683 MUST be supported by a server that offers resources for resource discovery (see Section 7.2 below) and SHOULD be supported for providing access to other resources. The default port number [IANA_TBD_PORT] for DTLS-secured CoAP MAY be supported by a server for resource discovery and for providing access to other resources. In addition other endpoints may be hosted at other ports, e.g. in the dynamic port space.

Implementation Note: When a CoAP server is hosted by a 6LoWPAN node, header compression efficiency is improved when it also supports a port number in the 61616-61631 compressed UDP port space defined in [RFC4944] (note that, as its UDP port differs from the default port, it is a different endpoint from the server at the default port).

7.2. Resource Discovery

The discovery of resources offered by a CoAP endpoint is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. A CoAP endpoint SHOULD support the CoRE Link Format of discoverable resources as described in [RFC6690]. It is up to the server which resources are made discoverable (if any).

7.2.1. 'ct' Attribute

This section defines a new Web Linking [RFC5988] attribute for use with [RFC6690]. The Content-Format code "ct" attribute provides a hint about the Content-Formats this resource returns. Note that this is only a hint, and does not override the Content-Format Option of a CoAP response obtained by actually requesting the representation of the resource. The value is in the CoAP identifier code format as a decimal ASCII integer and MUST be in the range of 0-65535 (16-bit unsigned integer). For example application/xml would be indicated as "ct=41". If no Content-Format code attribute is present then nothing about the type can be assumed. The Content-Format code attribute MAY include a space-separated sequence of Content-Format codes, indicating that multiple content-formats are available. The syntax of the attribute value is summarized in the production ct-value in Figure 13, where cardinal, SP and DQUOTE are defined as in [RFC6690].

```
ct-value = cardinal
          / DQUOTE cardinal *( 1*SP cardinal ) DQUOTE
```

Figure 13

8. Multicast CoAP

CoAP supports making requests to a IP multicast group. This is defined by a series of deltas to Unicast CoAP.

CoAP endpoints that offer services that they want other endpoints to be able to find using multicast service discovery, join one or more of the appropriate all-CoAP-nodes multicast addresses Section 12.8 and listen on the default CoAP port. Note that an endpoint might

receive multicast requests on other multicast addresses, including the all-nodes IPv6 address (or via broadcast on IPv4); an endpoint MUST therefore be prepared to receive such messages but MAY ignore them if multicast service discovery is not desired.

8.1. Messaging Layer

A multicast request is characterized by being transported in a CoAP message that is addressed to an IP multicast address instead of a CoAP endpoint. Such multicast requests MUST be Non-Confirmable.

A server SHOULD be aware that a request arrived via multicast, e.g. by making use of modern APIs such as IPV6_RECVPKTINFO [RFC3542], if available.

When a server is aware that a request arrived via multicast, it MUST NOT return a RST in reply to NON. If it is not aware, it MAY return a RST in reply to NON as usual. Because such a Reset message will look identical to an RST for a unicast message from the sender, the sender MUST avoid using a Message ID that is also still active from this endpoint with any unicast endpoint that might receive the multicast message.

8.2. Request/Response Layer

When a server is aware that a request arrived via multicast, the server MAY always pretend it did not receive the request, in particular if it doesn't have anything useful to respond (e.g., if it only has an empty payload or an error response). The decision for this may depend on the application. (For example, in [RFC6690] query filtering, a server should not respond to a multicast request if the filter does not match.)

If a server does decide to respond to a multicast request, it should not respond immediately. Instead, it should pick a duration for the period of time during which it intends to respond. For purposes of this exposition, we call the length of this period the Leisure. The specific value of this Leisure may depend on the application, or MAY be derived as described below. The server SHOULD then pick a random point of time within the chosen Leisure period to send back the unicast response to the multicast request. If further responses need to be sent based on the same multicast address membership, a new leisure period starts at the earliest after the previous one finishes.

To compute a value for Leisure, the server should have a group size estimate G , a target data transfer rate R (which both should be chosen conservatively) and an estimated response size S ; a rough

lower bound for Leisure can then be computed as
$$lb_Leisure = S * G / R$$

E.g., for a multicast request with link-local scope on an 2.4 GHz IEEE 802.15.4 (6LoWPAN) network, G could be (relatively conservatively) set to 100, S to 100 bytes, and the target rate to a conservative 8 kbit/s = 1 kB/s. The resulting lower bound for the Leisure is 10 seconds.

If a CoAP endpoint does not have suitable data to compute a value for Leisure, it MAY resort to DEFAULT_LEISURE.

When matching a response to a multicast request, only the token MUST match; the source endpoint of the response does not need to (and will not) be the same as the destination endpoint of the original request.

8.2.1. Caching

When a client makes a multicast request, it always makes a new request to the multicast group (since there may be new group members that joined meanwhile or ones that did not get the previous request). It MAY update the cache with the received responses. Then it uses both cached-still-fresh and 'new' responses as the result of the request.

A response received in reply to a GET request to a multicast group MAY be used to satisfy a subsequent request on the related unicast request URI. The unicast request URI is obtained by replacing the authority part of the request URI with the transport layer source address of the response message.

A cache MAY revalidate a response by making a GET request on the related unicast request URI.

A GET request to a multicast group MUST NOT contain an ETag option. A mechanism to suppress responses the client already has is left for further study.

8.2.2. Proxying

When a forward-proxy receives a request with a Proxy-Uri 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.

9. Securing CoAP

This section defines the DTLS binding for CoAP, and the alternative use of IPsec.

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 Section 9.2.

PreSharedKey: DTLS is enabled and there is a list of pre-shared keys [RFC4279] and each key includes a list of which nodes it can be used to communicate with as described in Section 9.1.3.1. At the extreme there may be one key for each node this CoAP node needs to communicate with (1:1 node/key ratio).

RawPublicKey: DTLS is enabled and the device has an asymmetric key pair without a certificate (a raw public key) that is validated using an out-of-band mechanism [I-D.ietf-tls-oob-pubkey] as described in Section 9.1.3.2. The device also has an identity calculated from the public key and a list of identities of the nodes it can communicate with.

Certificate: DTLS is enabled and the device has an asymmetric key pair with an X.509 certificate [RFC5280] that binds it to its Authority Name and is signed by some common trust root as described in Section 9.1.3.3. The device also has a list of root trust anchors that can be used for validating a certificate.

In the "NoSec" mode, the system simply sends the packets over normal UDP over IP and is indicated by the "coap" scheme and the CoAP default port. The system is secured only by keeping attackers from being able to send or receive packets from the network with the CoAP nodes; see Section 11.5 for an additional complication with this approach.

The other three security modes are achieved using DTLS and are indicated by the "coaps" scheme and DTLS-secured CoAP default port. The result is a security association that can be used to authenticate (within the limits of the security model) and, based on this

authentication, authorize the communication partner. CoAP itself does not provide protocol primitives for authentication or authorization; where this is required, it can either be provided by communication security (i.e., IPsec or DTLS) or by object security (within the payload). Devices that require authorization for certain operations are expected to require one of these two forms of security. Necessarily, where an intermediary is involved, communication security only works when that intermediary is part of the trust relationships; CoAP does not provide a way to forward different levels of authorization that clients may have with an intermediary to further intermediaries or origin servers -- it therefore may be required to perform all authorization at the first intermediary.

9.1. DTLS-secured CoAP

Just as HTTP is secured using Transport Layer Security (TLS) over TCP, CoAP is secured using Datagram TLS (DTLS) [RFC6347] over UDP (see Figure 14). 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.

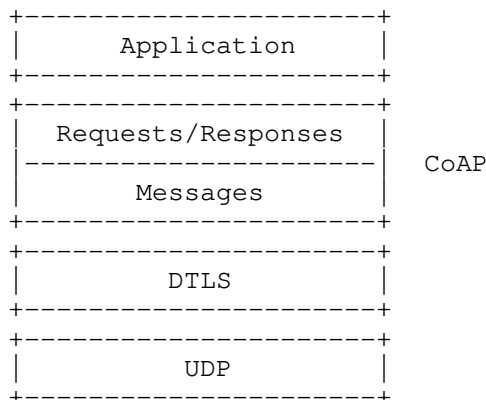


Figure 14: Abstract layering of DTLS-secured CoAP

In some constrained nodes (limited flash and/or RAM) and networks (limited bandwidth or high scalability requirements), and depending on the specific cipher suites in use, all modes of DTLS may not be applicable. Some DTLS cipher suites can add significant implementation complexity as well as some initial handshake overhead needed when setting up the security association. Once the initial

handshake is completed, DTLS adds a limited per-datagram overhead of approximately 13 bytes, not including any initialization vectors/nonces (e.g., 8 bytes with TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]), integrity check values (e.g., 8 bytes with TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]) and padding required by the cipher suite. Whether and which mode of using DTLS is applicable for a CoAP-based application should be carefully weighed considering the specific cipher suites that may be applicable, and whether the session maintenance makes it compatible with application flows and sufficient resources are available on the constrained nodes and for the added network overhead. DTLS is not applicable to group keying (multicast communication); however, it may be a component in a future group key management protocol.

9.1.1. Messaging Layer

The endpoint acting as the CoAP client should also act as the DTLS client. It should initiate a session to the server on the appropriate port. When the DTLS handshake has finished, the client may initiate the first CoAP request. All CoAP messages MUST be sent as DTLS "application data".

The following rules are added for matching an ACK or RST to a CON message or a RST to a NON message: The DTLS session MUST be the same and the epoch MUST be the same.

A message is the same when it is sent within the same DTLS session and same epoch and has the same Message ID.

Note: When a confirmable message is retransmitted, a new DTLS sequence_number is used for each attempt, even though the CoAP Message ID stays the same. So a recipient still has to perform deduplication as described in Section 4.5. Retransmissions MUST NOT be performed across epochs.

DTLS connections in RawPublicKey and Certificate mode are set up using mutual authentication so they can remain up and be reused for future message exchanges in either direction. Devices can close a DTLS connection when they need to recover resources but in general they should keep the connection up for as long as possible. Closing the DTLS connection after every CoAP message exchange is very inefficient.

9.1.2. Request/Response Layer

The following rules are added for matching a response to a request: The DTLS session MUST be the same and the epoch MUST be the same.

9.1.3. Endpoint Identity

Devices SHOULD support the Server Name Indication (SNI) to indicate their Authority Name in the SNI HostName field as defined in Section 3 of [RFC6066]. This is needed so that when a host that acts as a virtual server for multiple Authorities receives a new DTLS connection, it knows which keys to use for the DTLS session.

9.1.3.1. Pre-Shared Keys

When forming a connection to a new node, the system selects an appropriate key based on which nodes it is trying to reach and then forms a DTLS session using a PSK (Pre-Shared Key) mode of DTLS. Implementations in these modes MUST support the mandatory to implement cipher suite TLS_PSK_WITH_AES_128_CCM_8 as specified in [RFC6655].

The security considerations of [RFC4279] (Section 7) apply. In particular, applications should carefully weigh whether they need Perfect Forward Secrecy (PFS) or not and select an appropriate cipher suite (7.1). The entropy of the PSK must be sufficient to mitigate against brute-force and (where the PSK is not chosen randomly but by a human) dictionary attacks (7.2). The cleartext communication of client identities may leak data or compromise privacy (7.3).

9.1.3.2. Raw Public Key Certificates

In this mode the device has an asymmetric key pair but without an X.509 certificate (called a raw public key). A device MAY be configured with multiple raw public keys. The type and length of the raw public key depends on the cipher suite used. Implementations in RawPublicKey mode MUST support the mandatory to implement cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as specified in [I-D.mcgregw-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. The mechanism for using raw public keys with TLS is specified in [I-D.ietf-tls-oob-pubkey].

9.1.3.2.1. Provisioning

The RawPublicKey mode was designed to be easily provisioned in M2M deployments. It is assumed that each device has an appropriate asymmetric public key pair installed. An identifier is calculated from the public key as described in Section 2 of [I-D.farrell-decade-ni]. All implementations that support checking RawPublicKey identities MUST support at least the sha-256-120 mode (SHA-256 truncated to 120 bits). Implementations SHOULD support also longer length identifiers and MAY support shorter lengths. Note that the shorter lengths provide less security against attacks and their

use is NOT RECOMMENDED.

Depending on how identifiers are given to the system that verifies them, support for URI, binary, and/or human-speakable format [I-D.farrell-decade-ni] needs to be implemented. All implementations SHOULD support the binary mode and implementations that have a user interface SHOULD also support the human-speakable format.

During provisioning, the identifier of each node is collected, for example by reading a barcode on the outside of the device or by obtaining a pre-compiled list of the identifiers. These identifiers are then installed in the corresponding endpoint, for example an M2M data collection server. The identifier is used for two purposes, to associate the endpoint with further device information and to perform access control. During provisioning, an access control list of identifiers the device may start DTLS sessions with SHOULD also be installed.

9.1.3.3. X.509 Certificates

Implementations in Certificate Mode MUST support the mandatory to implement cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as specified in [RFC5246].

The Authority Name in the certificate is the name that would be used in the Authority part of a CoAP URI. It is worth noting that this would typically not be either an IP address or DNS name but would instead be a long term unique identifier for the device such as the EUI-64 [EUI64]. The discovery process used in the system would build up the mapping between IP addresses of the given devices and the Authority Name for each device. Some devices could have more than one Authority and would need more than a single certificate.

When a new connection is formed, the certificate from the remote device needs to be verified. If the CoAP node has a source of absolute time, then the node SHOULD check that the validity dates of the certificate are within range. The certificate MUST also be signed by an appropriate chain of trust. If the certificate contains a SubjectAltName, then the Authority Name MUST match at least one of the authority names of any CoAP URI found in a URI type field in the SubjectAltName set. If there is no SubjectAltName in the certificate, then the Authoritative Name must match the CN found in the certificate using the matching rules defined in [RFC2818] with the exception that certificates with wildcards are not allowed.

If the system has a shared key in addition to the certificate, then a cipher suite that includes the shared key such as TLS_RSA_PSK_WITH_AES_128_CBC_SHA [RFC4279] SHOULD be used.

9.2. Using CoAP with IPsec

One mechanism to secure CoAP in constrained environments is the IPsec Encapsulating Security Payload (ESP) [RFC4303] when CoAP is used without DTLS in NoSec Mode. Using IPsec ESP with the appropriate configuration, it is possible for many constrained devices to support encryption with built-in link-layer encryption hardware. For example, some IEEE 802.15.4 radio chips are compatible with AES-CBC (with 128-bit keys) [RFC3602] as defined for use with IPsec in [RFC4835]. Alternatively, particularly on more common IEEE 802.15.4 hardware that supports AES encryption but not decryption, and to avoid the need for padding, nodes could directly use the more widely supported AES-CCM as defined for use with IPsec in [RFC4309], if the security considerations in Section 9 of that specification can be fulfilled.

Necessarily for AES-CCM, but much preferably also for AES-CBC, static keying should be avoided and the initial keying material be derived into transient session keys, e.g. using a low-overhead mode of IKEv2 [RFC5996] as described in [I-D.kivinen-ipsecme-ikev2-minimal]; such a protocol for managing keys and sequence numbers is also the only way to achieve anti-replay capabilities. However, no recommendation can be made at this point on how to manage group keys (i.e., for multicast) in a constrained environment. Once any initial setup is completed, IPsec ESP adds a limited overhead of approximately 10 bytes per packet, not including initialization vectors, integrity check values and padding required by the cipher suite.

When using IPsec to secure CoAP, both authentication and confidentiality SHOULD be applied as recommended in [RFC4303]. The use of IPsec between CoAP endpoints is transparent to the application layer and does not require special consideration for a CoAP implementation.

IPsec may not be appropriate for all environments. For example, IPsec support is not available for many embedded IP stacks and even in full PC operating systems or on back-end web servers, application developers may not have sufficient access to configure or enable IPsec or to add a security gateway to the infrastructure. Problems with firewalls and NATs may furthermore limit the use of IPsec.

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 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.

10.1. CoAP-HTTP Proxying

If a request contains a Proxy-URI Option with an 'http' or 'https' URI [RFC2616], then the receiving CoAP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated HTTP resource and return the result to the client.

This section specifies for any CoAP request the CoAP response that the proxy should return to the client. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to an HTTP origin server.

Since HTTP and CoAP share the basic set of request methods, performing a CoAP request on an HTTP resource is not so different from performing it on a CoAP resource. The meanings of the individual CoAP methods when performed on HTTP resources are explained below.

If the proxy is unable or unwilling to service a request with an HTTP URI, a 5.05 (Proxying Not Supported) response is returned to the client. If the proxy services the request by interacting with a

third party (such as the HTTP origin server) and is unable to obtain a result within a reasonable time frame, a 5.04 (Gateway Timeout) response is returned; if a result can be obtained but is not understood, a 5.02 (Bad Gateway) response is returned.

10.1.1. GET

The GET method requests the proxy to return a representation of the HTTP resource identified by the request URI.

Upon success, a 2.05 (Content) response code SHOULD be returned. The payload of the response MUST be a representation of the target HTTP resource, and the Content-Format Option be set accordingly. The response MUST indicate a Max-Age value that is no greater than the remaining time the representation can be considered fresh. If the HTTP entity has an entity tag, the proxy SHOULD include an ETag Option in the response and process ETag Options in requests as described below.

A client can influence the processing of a GET request by including the following option:

Accept: The request MAY include one or more Accept Options, identifying the preferred response content-format.

ETag: The request MAY include one or more ETag Options, identifying responses that the client has stored. This requests the proxy to send a 2.03 (Valid) response whenever it would send a 2.05 (Content) response with an entity tag in the requested set otherwise. Note that CoAP ETags are always strong ETags in the HTTP sense; CoAP does not have the equivalent of HTTP weak ETags, and there is no good way to make use of these in a cross-proxy.

10.1.2. 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. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to a CoAP origin server. The meanings of the individual HTTP methods when performed on CoAP resources are explained below.

If the proxy is unable or unwilling to service a request with a CoAP URI, a 501 (Not Implemented) response SHOULD be 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 SHOULD be returned; if a result can be obtained but is not understood, a 502 (Bad Gateway) response SHOULD be 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 SHOULD be returned. The payload of the response MUST be a representation of the target CoAP resource, and the Content-Type and Content-Encoding header fields be set accordingly. The response MUST indicate a max-age directive that indicates a value no greater than the remaining time the representation can be considered fresh. If the CoAP response has an ETag option, the proxy SHOULD include an ETag header field in the response.

A client can influence the processing of a GET request by including the following options:

Accept: Each individual Media-type of the HTTP Accept header in a request is mapped to a CoAP Accept option. HTTP Accept Media-type ranges, parameters and extensions are not supported by the CoAP Accept option. If the proxy cannot send a response which is acceptable according to the combined Accept field value, then the proxy SHOULD send a 406 (not acceptable) response.

Conditional GETs: Conditional HTTP GET requests that include an "If-Match" or "If-None-Match" request-header field can be mapped to a corresponding CoAP request. The "If-Modified-Since" and "If-Unmodified-Since" request-header fields are not directly supported by CoAP, but SHOULD be 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 SHOULD be 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 MUST be returned to the client. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be 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 SHOULD be 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 SHOULD be returned to the client.

11. Security Considerations

This section analyzes the possible threats to the protocol. It is meant to inform protocol and application developers about the security limitations of CoAP as described in this document. As CoAP realizes a subset of the features in HTTP/1.1, the security considerations in Section 15 of [RFC2616] are also pertinent to CoAP. This section concentrates on describing limitations specific to CoAP.

11.1. Protocol Parsing, Processing URIs

A network-facing application can exhibit vulnerabilities in its processing logic for incoming packets. Complex parsers are well-known as a likely source of such vulnerabilities, such as the ability to remotely crash a node, or even remotely execute arbitrary code on it. CoAP attempts to narrow the opportunities for introducing such vulnerabilities by reducing parser complexity, by giving the entire range of encodable values a meaning where possible, and by aggressively reducing complexity that is often caused by unnecessary choice between multiple representations that mean the same thing. Much of the URI processing has been moved to the clients, further reducing the opportunities for introducing vulnerabilities into the servers. Even so, the URI processing code in CoAP implementations is likely to be a large source of remaining vulnerabilities and should be implemented with special care. The most complex parser remaining could be the one for the CoRE Link Format, although this also has been designed with a goal of reduced implementation complexity [RFC6690]. (See also section 15.2 of [RFC2616].)

11.2. Proxying and Caching

As mentioned in 15.7 of [RFC2616], proxies are by their very nature men-in-the-middle, breaking any IPsec or DTLS protection that a direct CoAP message exchange might have. They are therefore interesting targets for breaking confidentiality or integrity of CoAP message exchanges. As noted in [RFC2616], they are also interesting targets for breaking availability.

The threat to confidentiality and integrity of request/response data is amplified where proxies also cache. Note that CoAP does not define any of the cache-suppressing Cache-Control options that HTTP/1.1 provides to better protect sensitive data.

For a caching implementation, any access control considerations that would apply to making the request that generated the cache entry also need to be applied to the value in the cache. This is relevant for clients that implement multiple security domains, as well as for proxies that may serve multiple clients. Also, a caching proxy **MUST NOT** make cached values available to requests that have lesser transport security properties than to which it would make available the process of forwarding the request in the first place.

Finally, a proxy that fans out Separate Responses (as opposed to Piggy-backed Responses) to multiple original requesters may provide additional amplification (see below).

11.3. Risk of amplification

CoAP servers generally reply to a request packet with a response packet. This response packet may be significantly larger than the request packet. An attacker might use CoAP nodes to turn a small attack packet into a larger attack packet, an approach known as amplification. There is therefore a danger that CoAP nodes could become implicated in denial of service (DoS) attacks by using the amplifying properties of the protocol: An attacker that is attempting to overload a victim but is limited in the amount of traffic it can generate, can use amplification to generate a larger amount of traffic.

This is particularly a problem in nodes that enable NoSec access, that are accessible from an attacker and can access potential victims (e.g. on the general Internet), as the UDP protocol provides no way to verify the source address given in the request packet. An attacker need only place the IP address of the victim in the source address of a suitable request packet to generate a larger packet directed at the victim.

As a mitigating factor, many constrained networks will only be able to generate a small amount of traffic, which may make CoAP nodes less attractive for this attack. However, the limited capacity of the constrained network makes the network itself a likely victim of an amplification attack.

A CoAP server can reduce the amount of amplification it provides to an attacker by using slicing/blocking modes of CoAP [I-D.ietf-core-block] and offering large resource representations only in relatively small slices. E.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

CoAP also supports the use of multicast IP addresses in requests, an important requirement for M2M. Multicast CoAP requests may be the source of accidental or deliberate denial of service attacks, especially over constrained networks. This specification attempts to reduce the amplification effects of multicast requests by limiting when a response is returned. To limit the possibility of malicious use, CoAP servers SHOULD NOT accept multicast requests that can not be authenticated. If possible a CoAP server SHOULD limit the support for multicast requests to specific resources where the feature is required.

On some general purpose operating systems providing a Posix-style API, it is not straightforward to find out whether a packet received

was addressed to a multicast address. While many implementations will know whether they have joined a multicast group, this creates a problem for packets addressed to multicast addresses of the form FF0x::1, which are received by every IPv6 node. Implementations SHOULD make use of modern APIs such as IPV6_RECVPKTINFO [RFC3542], if available, to make this determination.

11.4. IP Address Spoofing Attacks

Due to the lack of a handshake in UDP, a rogue endpoint which is free to read and write messages carried by the constrained network (i.e. NoSec or PreSharedKey deployments with nodes/key ratio > 1:1), may easily attack a single endpoint, a group of endpoints, as well as the whole network e.g. by:

1. spoofing RST in response to a CON or NON message, thus making an endpoint "deaf"; or
2. spoofing the entire response with forged payload/options (this has different levels of impact: from single response disruption, to much bolder attacks on the supporting infrastructure, e.g. poisoning proxy caches, or tricking validation / lookup interfaces in resource directories and, more generally, any component that stores global network state and uses CoAP as the messaging facility to handle state set/update's is a potential target.); or
3. spoofing a multicast request for a target node which may result in both network congestion/collapse and victim DoS'ing / forced wakeup from sleeping; or
4. spoofing observe messages, etc.

In principle, spoofing can be detected by CoAP only in case CON semantics is used, because of unexpected ACK/RSTs coming from the deceived endpoint. But this imposes keeping track of the used Message IDs which is not always possible, and moreover detection becomes available usually after the damage is already done. This kind of attack can be prevented using security modes other than NoSec.

11.5. Cross-Protocol Attacks

The ability to incite a CoAP endpoint to send packets to a fake source address can be used not only for amplification, but also for cross-protocol attacks against a victim listening to UDP packets at a given address (IP address and port):

- o the attacker sends a message to a CoAP endpoint with the given address as the fake source address,
- o the CoAP endpoint replies with a message to the given source address,
- o the victim at the given address receives a UDP packet that it interprets according to the rules of a different protocol.

This may be used to circumvent firewall rules that prevent direct communication from the attacker to the victim, but happen to allow communication from the CoAP endpoint (which may also host a valid role in the other protocol) to the victim.

Also, CoAP endpoints may be the victim of a cross-protocol attack generated through an endpoint of another UDP-based protocol such as DNS. In both cases, attacks are possible if the security properties of the endpoints rely on checking IP addresses (and firewalling off direct attacks sent from outside using fake IP addresses). In general, because of their lack of context, UDP-based protocols are relatively easy targets for cross-protocol attacks.

Finally, CoAP URIs transported by other means could be used to incite clients to send messages to endpoints of other protocols.

One mitigation against cross-protocol attacks is strict checking of the syntax of packets received, combined with sufficient difference in syntax. As an example, it might help if it were difficult to incite a DNS server to send a DNS response that would pass the checks of a CoAP endpoint. Unfortunately, the first two bytes of a DNS reply are an ID that can be chosen by the attacker, which map into the interesting part of the CoAP header, and the next two bytes are then interpreted as CoAP's Message ID (i.e., any value is acceptable). The DNS count words may be interpreted as multiple instances of a (non-existent, but elective) CoAP option 0. 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.

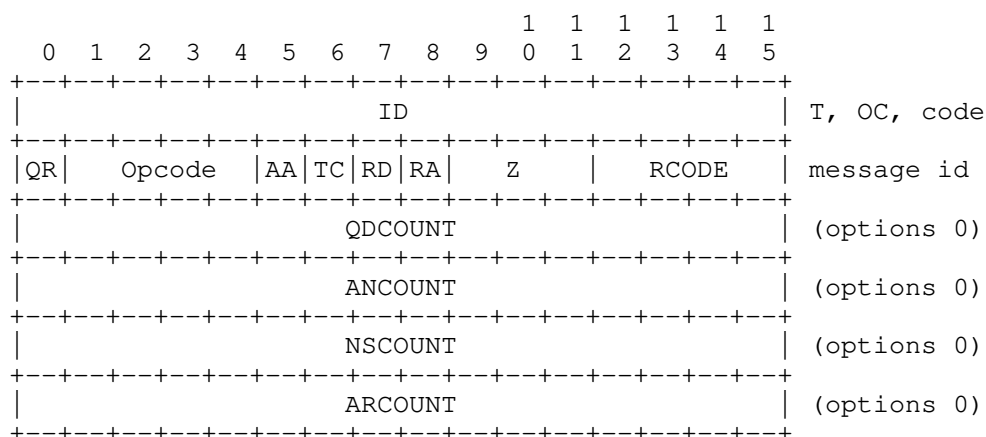


Figure 15: DNS Header vs. CoAP Message

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.

12. IANA Considerations

12.1. CoAP Code Registry

This document defines a registry for the values of the Code field in the CoAP header. The name of the registry is "CoAP Codes".

All values are assigned by sub-registries according to the following ranges:

- 0 Indicates an empty message (see Section 4.1).
- 1-31 Indicates a request. Values in this range are assigned by the "CoAP Method Codes" sub-registry (see Section 12.1.1).
- 32-63 Reserved
- 64-191 Indicates a response. Values in this range are assigned by the "CoAP Response Codes" sub-registry (see Section 12.1.2).
- 192-255 Reserved

12.1.1. Method Codes

The name of the sub-registry is "CoAP Method Codes".

Each entry in the sub-registry must include the Method Code in the range 1-31, the name of the method, and a reference to the method's documentation.

Initial entries in this sub-registry are as follows:

Code	Name	Reference
1	GET	[RFCXXXX]
2	POST	[RFCXXXX]
3	PUT	[RFCXXXX]
4	DELETE	[RFCXXXX]

Table 2: CoAP Method Codes

All other Method Codes are Unassigned.

The IANA policy for future additions to this registry is "IETF Review" 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 64-191, a description of the Response Code, and a reference to the Response Code's documentation.

Initial entries in this sub-registry are as follows:

Code	Description	Reference
65	2.01 Created	[RFCXXXX]
66	2.02 Deleted	[RFCXXXX]
67	2.03 Valid	[RFCXXXX]
68	2.04 Changed	[RFCXXXX]
69	2.05 Content	[RFCXXXX]
128	4.00 Bad Request	[RFCXXXX]
129	4.01 Unauthorized	[RFCXXXX]
130	4.02 Bad Option	[RFCXXXX]
131	4.03 Forbidden	[RFCXXXX]
132	4.04 Not Found	[RFCXXXX]
133	4.05 Method Not Allowed	[RFCXXXX]
134	4.06 Not Acceptable	[RFCXXXX]
140	4.12 Precondition Failed	[RFCXXXX]
141	4.13 Request Entity Too Large	[RFCXXXX]
143	4.15 Unsupported Content-Format	[RFCXXXX]
160	5.00 Internal Server Error	[RFCXXXX]
161	5.01 Not Implemented	[RFCXXXX]
162	5.02 Bad Gateway	[RFCXXXX]
163	5.03 Service Unavailable	[RFCXXXX]
164	5.04 Gateway Timeout	[RFCXXXX]
165	5.05 Proxying Not Supported	[RFCXXXX]

Table 3: CoAP Response Codes

The Response Codes 96-127 are Reserved for future use. All other Response Codes are Unassigned.

The IANA policy for future additions to this registry is "IETF Review" 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 message.
- o The format of the payload. For example, the format in a 2.05 (Content) response is indicated by the Content-Format Option; the format of the payload in an error response is always Net-Unicode text.
- o Whether the response is cacheable according to the freshness model.
- o Whether the response is validatable according to the validation model.
- o Whether the response causes a cache to mark responses stored for the request URI as not fresh.

12.2. Option Number Registry

This document defines a registry for the Option Numbers used in CoAP options. The name of the registry is "CoAP Option Numbers".

Each entry in the registry must include the Option Number, the name of the option and a reference to the option's documentation.

Initial entries in this registry are as follows:

Number	Name	Reference
0	(Reserved)	
1	If-Match	[RFCXXXX]
3	Uri-Host	[RFCXXXX]
4	ETag	[RFCXXXX]
5	If-None-Match	[RFCXXXX]
7	Uri-Port	[RFCXXXX]
8	Location-Path	[RFCXXXX]
11	Uri-Path	[RFCXXXX]
12	Content-Format	[RFCXXXX]
14	Max-Age	[RFCXXXX]
15	Uri-Query	[RFCXXXX]
16	Accept	[RFCXXXX]
19	Token	[RFCXXXX]
20	Location-Query	[RFCXXXX]
35	Proxy-Uri	[RFCXXXX]
128	(Reserved)	[RFCXXXX]
132	(Reserved)	[RFCXXXX]
136	(Reserved)	[RFCXXXX]

Table 4: CoAP Option Numbers

The IANA policy for future additions to this registry is split into three tiers as follows. The range of 0..255 is reserved for options defined by the IETF (IETF Review). The range of 256..2047 is reserved for commonly used options with public specifications (Specification Required). The range of 2048..65535 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.

Option Number	Policy [RFC5226]
0..255	IETF Review
256..2047	Specification Required
2048..65535	Designated Expert

The documentation of an Option Number should specify the semantics of an option with that number, including the following properties:

- o The meaning of the option in a request.

- o The meaning of the option in a response.
- o Whether the option is critical or elective, as determined by the Option Number.
- o Whether the option is Safe, and whether it is part of the Cache-Key, as determined by the Option Number (see Section 5.4.2).
- o The format and length of the option's value.
- o Whether the option must occur at most once or whether it can occur multiple times.
- o The default value, if any. For a critical option with a default value, a discussion on how the default value enables processing by implementations not implementing the critical option (Section 5.4.4).

12.3. Content-Format Registry

Internet media types are identified by a string, such as "application/xml" [RFC2046]. In order to minimize the overhead of using these media types to indicate the format of payloads, this document defines a registry for a subset of Internet media types to be used in CoAP and assigns each, in combination with a content-coding, a numeric identifier. The name of the registry is "CoAP Content-Formats".

Each entry in the registry must include the media type registered with IANA, the numeric identifier in the range 0-65535 to be used for that media type in CoAP, the content-coding associated with this identifier, and a reference to a document describing what a payload with that media type means semantically.

CoAP does not include a way to convey content-encoding information with a request or response, and for that reason the content-encoding is also specified for each identifier (if any). If multiple content-encodings will be used with a media type, then a separate Content-Format identifier for each is to be registered. Similarly, other parameters related to an Internet media type, such as level, can be defined for a CoAP Content-Format entry.

Initial entries in this registry are as follows:

Media type	Encoding	Id.	Reference
text/plain; charset=utf-8	-	0	[RFC2046] [RFC3676] [RFC5147]
application/ link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/ octet-stream	-	42	[RFC2045] [RFC2046]
application/exi	-	47	[EXIMIME]
application/json	-	50	[RFC4627]

Table 5: CoAP Content-Formats

The identifiers between 201 and 255 inclusive are reserved for Private Use. 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 0-200 inclusive to the registry is "Expert Review" as described in [RFC5226]. The IANA policy for additions in the range 256-65535 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.

[The explanatory text to be removed upon allocation of the addresses, except for the note about the distinct multicast addresses.]

13. Acknowledgements

Special thanks to Peter Bigot, Esko Dijk and Cullen Jennings for substantial contributions to the ideas and text in the document, along with countless detailed reviews and discussions.

Thanks to Ed Beroaset, Angelo P. Castellani, Gilbert Clark, Robert Cragie, Esko Dijk, Lisa Dussealt, Thomas Fossati, Tom Herbst, Richard Kelsey, Ari Keranen, Matthias Kovatsch, Salvatore Loreto, Kerry Lynn, Alexey Melnikov, Guido Moritz, Petri Mutka, Colin O'Flynn, Charles Palmer, Adriano Pezzuto, Robert Quattlebaum, Akbar Rahman, Eric Rescorla, David Ryan, Szymon Sasin, Michael Scharf, Dale Seed, Robby Simpson, Peter van der Stok, Michael Stuber, Linyi Tian, Gilman Tolle, Matthieu Vial and Alper Yegin for helpful comments and discussions that have shaped the document.

Some of the text has been borrowed from the working documents of the IETF httpbis working group.

14. References

14.1. Normative References

- [I-D.farrell-decade-ni]
Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", draft-farrell-decade-ni-10 (work in progress), August 2012.
- [I-D.ietf-tls-oob-pubkey]
Wouters, P., Gilmore, J., Weiler, S., Kivinen, T., and H. Tschofenig, "Out-of-Band Public Key Validation for Transport Layer Security", draft-ietf-tls-oob-pubkey-04 (work in progress), July 2012.
- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.
- [RFC3602] Frankel, S., Glenn, R., and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec", RFC 3602, September 2003.
- [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.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.
- [RFC4309] Housley, R., "Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)", RFC 4309, December 2005.
- [RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.
- [RFC4835] Manral, V., "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 4835, April 2007.
- [RFC5147] Wilde, E. and M. Duerst, "URI Fragment Identifiers for the text/plain Media Type", RFC 5147, April 2008.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

- [RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen,
 "Internet Key Exchange Protocol Version 2 (IKEv2)",
 RFC 5996, September 2010.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions:
 Extension Definitions", RFC 6066, January 2011.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer
 Security Version 1.2", RFC 6347, January 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link
 Format", RFC 6690, August 2012.

14.2. Informative References

- [EUI64] "GUIDELINES FOR 64-BIT GLOBAL IDENTIFIER (EUI-64)
 REGISTRATION AUTHORITY", April 2010, <<http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>>.
- [EXIMIME] "Efficient XML Interchange (EXI) Format 1.0",
 December 2009, <<http://www.w3.org/TR/2009/CR-exi-20091208/#mediaTypeRegistration>>.
- [I-D.allman-tcpm-rto-consider]
 Allman, M., "Retransmission Timeout Considerations",
 draft-allman-tcpm-rto-consider-01 (work in progress),
 May 2012.
- [I-D.ietf-core-block]
 Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",
 draft-ietf-core-block-09 (work in progress), August 2012.
- [I-D.ietf-core-observe]
 Hartke, K., "Observing Resources in CoAP",
 draft-ietf-core-observe-06 (work in progress),
 September 2012.
- [I-D.kivinen-ipsecme-ikev2-minimal]
 Kivinen, T., "Minimal IKEv2",
 draft-kivinen-ipsecme-ikev2-minimal-00 (work in progress),
 February 2011.
- [I-D.mcgrew-tls-aes-ccm-ecc]
 McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-
 CCM ECC Cipher Suites for TLS",
 draft-mcgrew-tls-aes-ccm-ecc-05 (work in progress),
 July 2012.

- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.
- [RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.
- [RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July 2012.

Appendix A. Examples

This section gives a number of short examples with message flows for GET requests. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and multicast.

Figure 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 at its default value (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 (implicitly empty) Token value. The response includes a Payload of "22.3 C" and is 10 bytes long.

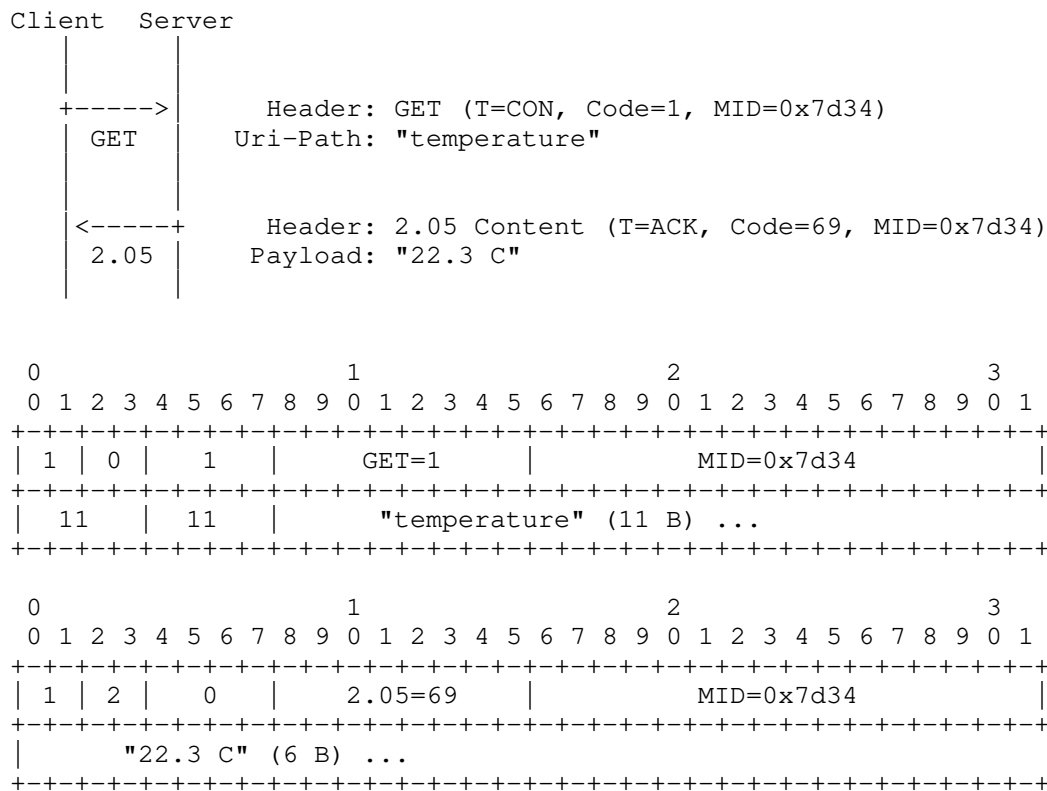


Figure 16: Confirmable request; piggy-backed response

Figure 17 shows a similar example, but with the inclusion of an

explicit Token Option (Delta 11 + 8 = 19, Length 1, Value 0x20) in the request and (Jump 15 + 4 = 19) in the response, increasing the sizes to 18 and 12 bytes, respectively.

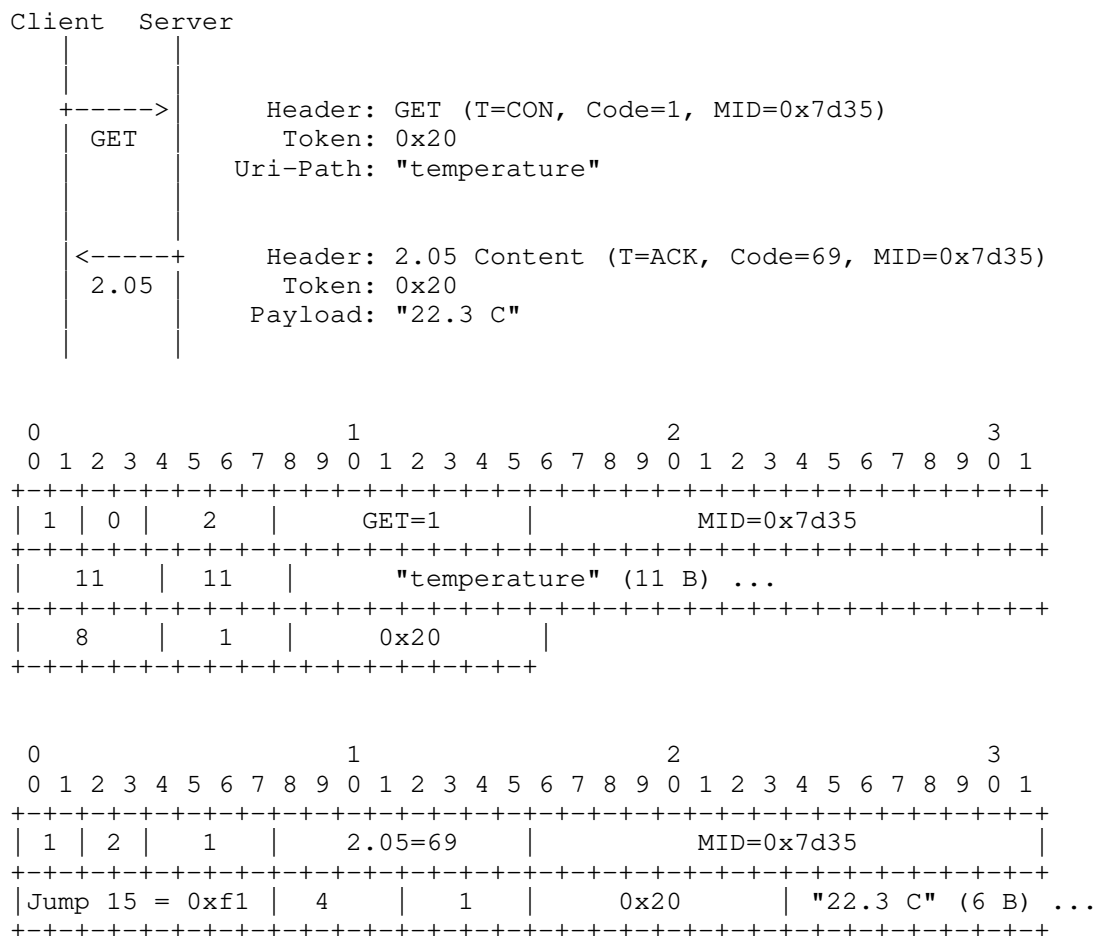


Figure 17: Confirmable request; piggy-backed response

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.

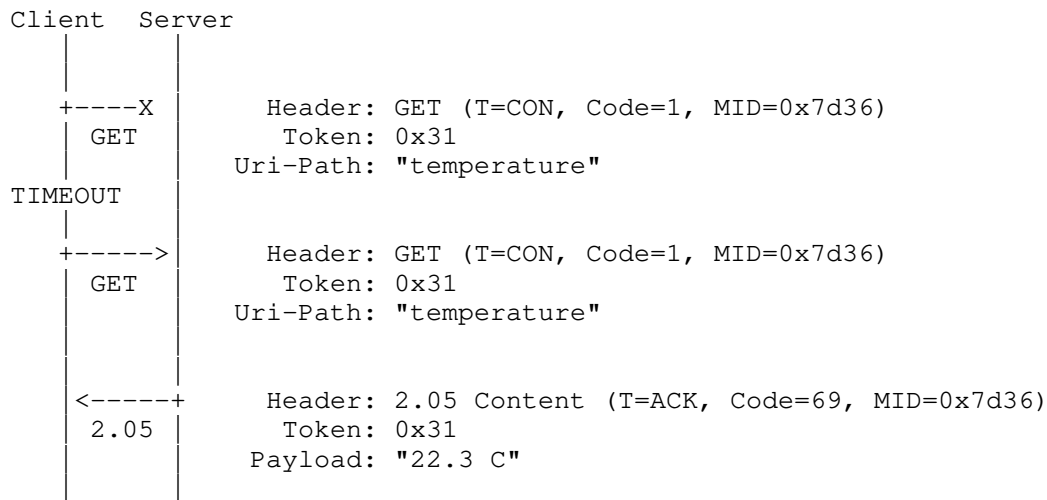


Figure 18: Confirmable request (retransmitted); piggy-backed response

In Figure 19, the first Acknowledgement message from the server to the client is lost. After ACK_TIMEOUT seconds, the client retransmits the request.

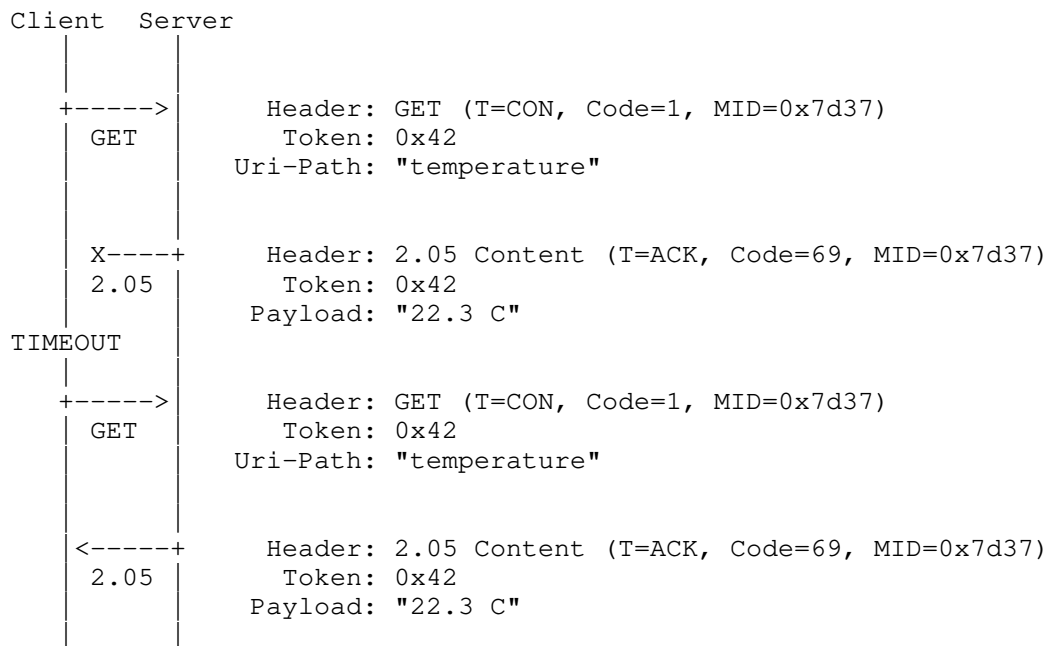


Figure 19: Confirmable request; piggy-backed response (retransmitted)

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.

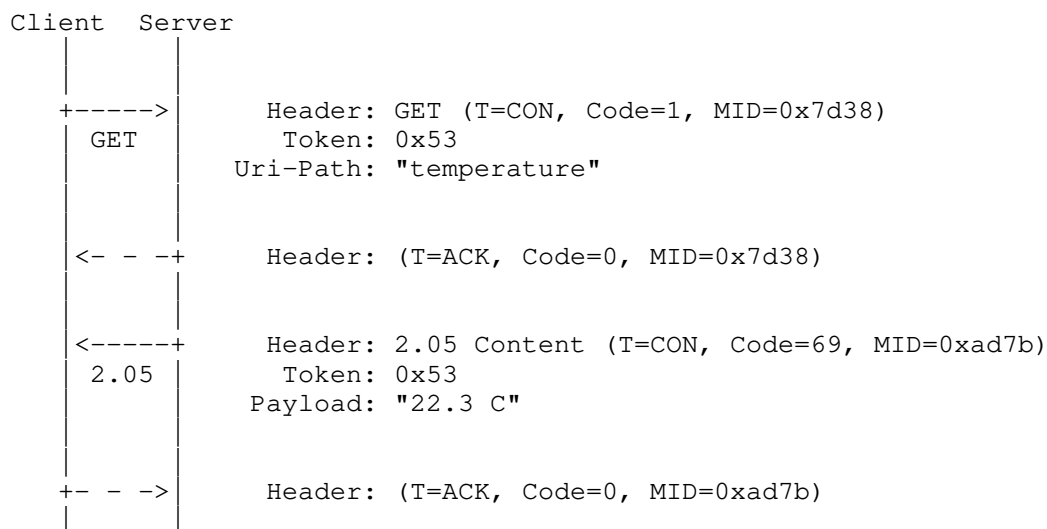


Figure 20: Confirmable request; separate response

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.

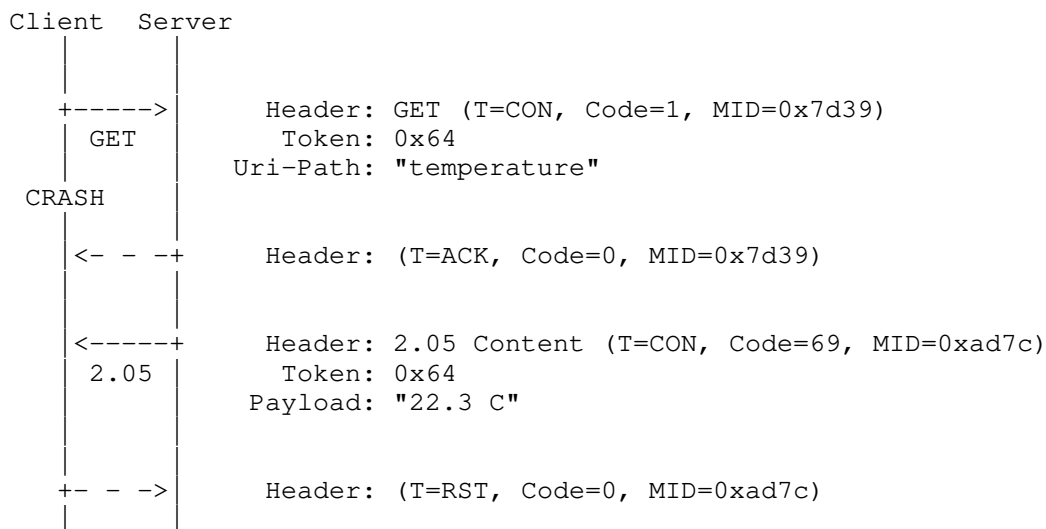


Figure 21: Confirmable request; separate response (unexpected)

Figure 22 shows a basic GET request where the request and the response are non-confirmable, so both may be lost without notice.

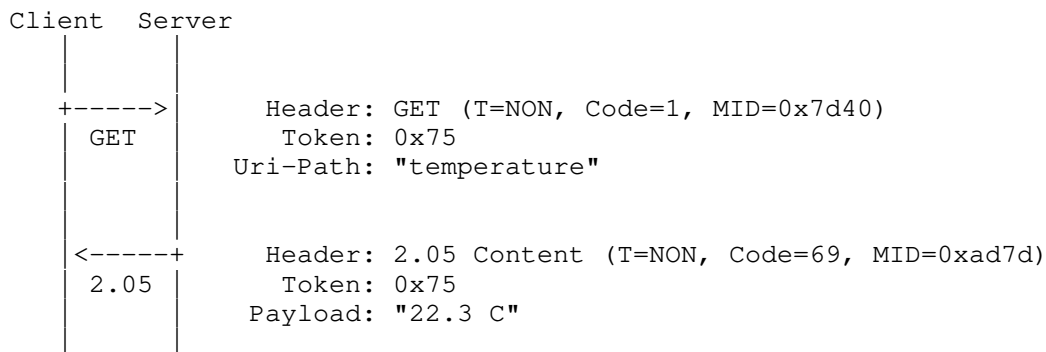


Figure 22: Non-confirmable request; Non-confirmable response

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.

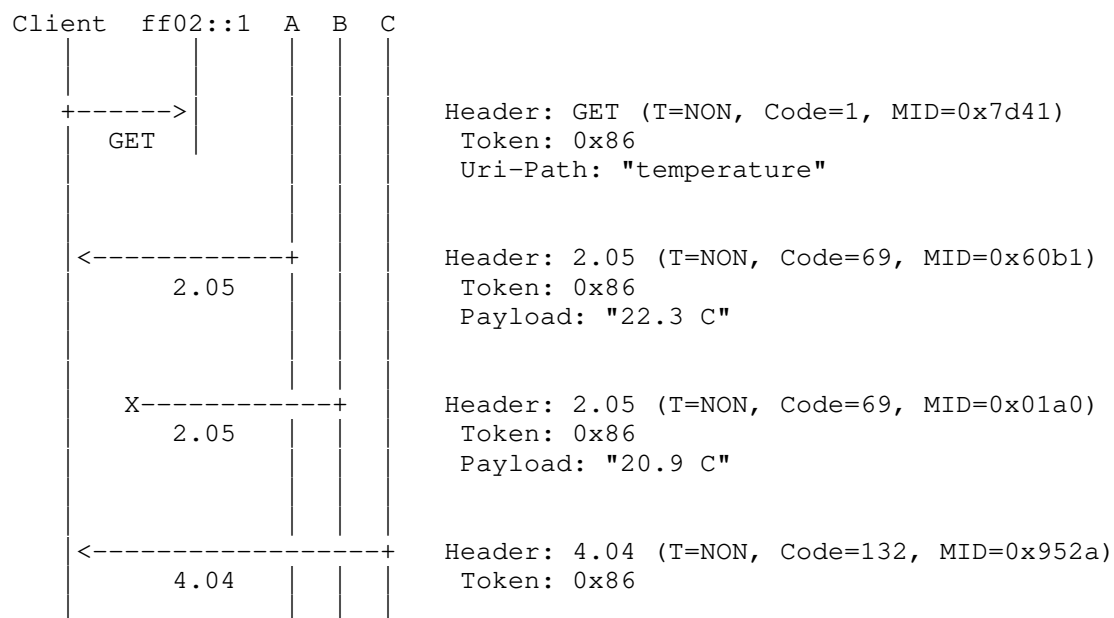


Figure 23: Non-confirmable request (multicast); Non-confirmable response

Appendix B. URI Examples

The following examples demonstrate different sets of Uri options, and the result after constructing an URI from them.

- o `coap://[2001:db8::2:1]/`
 Destination IP Address = `[2001:db8::2:1]`
 Destination UDP Port = `5683`
- o `coap://example.net/`
 Destination IP Address = `[2001:db8::2:1]`
 Destination UDP Port = `5683`
 Uri-Host = `"example.net"`
- o `coap://example.net/.well-known/core`

Destination IP Address = [2001:db8::2:1]

Destination UDP Port = 5683

Uri-Host = "example.net"

Uri-Path = ".well-known"

Uri-Path = "core"

- o coap://xn--18j4d.example/%E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF

Destination IP Address = [2001:db8::2:1]

Destination UDP Port = 5683

Uri-Host = "xn--18j4d.example"

Uri-Path = the string composed of the Unicode characters U+3053 U+3093 U+306b U+3061 U+306f, usually represented in UTF-8 as E38193E38293E381ABE381A1E381AF hexadecimal

- o coap://198.51.100.1:61616//%2F//?%2F%2F&?%26

Destination IP Address = 198.51.100.1

Destination UDP Port = 61616

Uri-Path = ""

Uri-Path = "/"

Uri-Path = ""

Uri-Path = ""

Uri-Query = "/"

Uri-Query = "?&"

Appendix C. Changelog

Changed from ietf-11 to ietf-12:

- o Extended options to support lengths of up to 1034 bytes (#202).

- o Added new Jump mechanism for options and removed Fenceposting (#214).
- o Added new IANA option number registration policy (#214).
- o Added Proxy Unsafe/Safe and Cache-Key masking to option numbers (#241).
- o Re-numbered option numbers to use Unsafe/Safe and Cache-Key compliant numbers (#241).
- o Defined NSTART and restricted the value to 1 with a MUST (#215).
- o Defined PROBING_RATE and set it to 1 Byte/second (#215).
- o Defined DEFAULT_LEISURE (#246).
- o Renamed Content-Type into Content-Format, and Media Type registry into Content-Format registry.
- o A large number of small editorial changes, clarifications and improvements have been made.

Changed from ietf-10 to ietf-11:

- o Expanded section 4.8 on Transmission Parameters, and used the derived values defined there (#201). Changed parameter names to be shorter and more to the point.
- o Several more small editorial changes, clarifications and improvements have been made.

Changed from ietf-09 to ietf-10:

- o Option deltas are restricted to 0 to 14; the option delta 15 is used exclusively for the end-of-options marker (#239).
- o Option numbers that are a multiple of 14 are not reserved, but are required to have an empty default value (#212).
- o Fixed misleading language that was introduced in 5.10.2 in coap-07 re Uri-Host and Uri-Port (#208).
- o Segments and arguments can have a length of zero characters (#213).
- o The Location-* options describe together describe one location. The location is a relative URI, not an "absolute path URI" (#218).

- o The value of the Location-Path Option must not be '.' or '..' (#218).
- o Added a sentence on constructing URIs from Location-* options (#231).
- o Reserved option numbers for future Location-* options (#230).
- o Fixed response codes with payload inconsistency (#233).
- o Added advice on default values for critical options (#207).
- o Clarified use of identifiers in RawPublicKey Mode Provisioning (#222).
- o Moved "Securing CoAP" out of the "Security Considerations" (#229).
- o Added "All CoAP Nodes" multicast addresses to "IANA Considerations" (#216).
- o Over 100 small editorial changes, clarifications and improvements have been made.

Changed from ietf-08 to ietf-09:

- o Improved consistency of statements about RST on NON: RST is a valid response to a NON message (#183).
- o Clarified that the protocol constants can be configured for specific application environments.
- o Added implementation note recommending piggy-backing whenever possible (#182).
- o Added a content-encoding column to the media type registry (#181).
- o Minor improvements to Appendix D.
- o Added text about multicast response suppression (#177).
- o Included the new End-of-options Marker (#176).
- o Added a reference to draft-ietf-tls-oob-pubkey and updated the RPK text accordingly.

Changed from ietf-07 to ietf-08:

- o Clarified matching rules for messages (#175)
- o Fixed a bug in Section 8.2.2 on Etags (#168)
- o Added an IP address spoofing threat analysis contribution (#167)
- o Re-focused the security section on raw public keys (#166)
- o Added an 4.06 error to Accept (#165)

Changed from ietf-06 to ietf-07:

- o application/link-format added to Media types registration (#160)
- o Moved content-type attribute to the document from link-format.
- o Added coaps scheme and DTLS-secured CoAP default port (#154)
- o Allowed 0-length Content-type options (#150)
- o Added congestion control recommendations (#153)
- o Improved text on PUT/POST response payloads (#149)
- o Added an Accept option for content-negotiation (#163)
- o Added If-Match and If-None-Match options (#155)
- o Improved Token Option explanation (#147)
- o Clarified mandatory to implement security (#156)
- o Added first come first server policy for 2-byte Media type codes (#161)
- o Clarify matching rules for messages and tokens (#151)
- o Changed OPTIONS and TRACE to always return 501 in HTTP-CoAP mapping (#164)

Changed from ietf-05 to ietf-06:

- o HTTP mapping section improved with the minimal protocol standard text for CoAP-HTTP and HTTP-CoAP forward proxying (#137).
- o Eradicated percent-encoding by including one Uri-Query Option per &-delimited argument in a query.

- o Allowed RST message in reply to a NON message with unexpected token (#135).
- o Cache Invalidation only happens upon successful responses (#134).
- o 50% jitter added to the initial retransmit timer (#142).
- o DTLS cipher suites aligned with ZigBee IP, DTLS clarified as default CoAP security mechanism (#138, #139)
- o Added a minimal reference to draft-kivinen-ipsecme-ikev2-minimal (#140).
- o Clarified the comparison of UTF-8s (#136).
- o Minimized the initial media type registry (#101).

Changed from ietf-04 to ietf-05:

- o Renamed Immediate into Piggy-backed and Deferred into Separate -- should finally end the confusion on what this is about.
- o GET requests now return a 2.05 (Content) response instead of 2.00 (OK) response (#104).
- o Added text to allow 2.02 (Deleted) responses in reply to POST requests (#105).
- o Improved message deduplication rules (#106).
- o Section added on message size implementation considerations (#103).
- o Clarification made on human readable error payloads (#109).
- o Definition of CoAP methods improved (#108).
- o Max-Age removed from requests (#107).
- o Clarified uniqueness of tokens (#112).
- o Location-Query Option added (#113).
- o ETag length set to 1-8 bytes (#123).
- o Clarified relation between elective/critical and option numbers (#110).

- o Defined when to update Version header field (#111).
- o URI scheme registration improved (#102).
- o Added review guidelines for new CoAP codes and numbers.

Changes from ietf-03 to ietf-04:

- o Major document reorganization (#51, #63, #71, #81).
- o Max-age length set to 0-4 bytes (#30).
- o Added variable unsigned integer definition (#31).
- o Clarification made on human readable error payloads (#50).
- o Definition of POST improved (#52).
- o Token length changed to 0-8 bytes (#53).
- o Section added on multiplexing CoAP, DTLS and STUN (#56).
- o Added cross-protocol attack considerations (#61).
- o Used new Immediate/Deferred response definitions (#73).
- o Improved request/response matching rules (#74).
- o Removed unnecessary media types and added recommendations for their use in M2M (#76).
- o Response codes changed to base 32 coding, new Y.XX naming (#77).
- o References updated as per AD review (#79).
- o IANA section completed (#80).
- o Proxy-Uri Option added to disambiguate between proxy and non-proxy requests (#82).
- o Added text on critical options in cached states (#83).
- o HTTP mapping sections improved (#88).
- o Added text on reverse proxies (#72).
- o Some security text on multicast added (#54).

- o Trust model text added to introduction (#58, #60).
- o AES-CCM vs. AES-CCB text added (#55).
- o Text added about device capabilities (#59).
- o DTLS section improvements (#87).
- o Caching semantics aligned with RFC2616 (#78).
- o Uri-Path Option split into multiple path segments.
- o MAX_RETRANSMIT changed to 4 to adjust for RESPONSE_TIME = 2.

Changes from ietf-02 to ietf-03:

- o Token Option and related use in asynchronous requests added (#25).
- o CoAP specific error codes added (#26).
- o Erroring out on unknown critical options changed to a MUST (#27).
- o Uri-Query Option added.
- o Terminology and definitions of URIs improved.
- o Security section completed (#22).

Changes from ietf-01 to ietf-02:

- o Sending an error on a critical option clarified (#18).
- o Clarification on behavior of PUT and idempotent operations (#19).
- o Use of Uri-Authority clarified along with server processing rules; Uri-Scheme Option removed (#20, #23).
- o Resource discovery section removed to a separate CoRE Link Format draft (#21).
- o Initial security section outline added.

Changes from ietf-00 to ietf-01:

- o New cleaner transaction message model and header (#5).
- o Removed subscription while being designed (#1).

- o Section 2 re-written (#3).
- o Text added about use of short URIs (#4).
- o Improved header option scheme (#5, #14).
- o Date option removed while being designed (#6).
- o New text for CoAP default port (#7).
- o Completed proxying section (#8).
- o Completed resource discovery section (#9).
- o Completed HTTP mapping section (#10).
- o Several new examples added (#11).
- o URI split into 3 options (#12).
- o MIME type defined for link-format (#13, #16).
- o New text on maximum message size (#15).
- o Location Option added.

Changes from shelby-01 to ietf-00:

- o Removed the TCP binding section, left open for the future.
- o Fixed a bug in the example.
- o Marked current Sub/Notify as (Experimental) while under WG discussion.
- o Fixed maximum datagram size to 1280 for both IPv4 and IPv6 (for CoAP-CoAP proxying to work).
- o Temporarily removed the Magic Byte header as TCP is no longer included as a binding.
- o Removed the Uri-code Option as different URI encoding schemes are being discussed.
- o Changed the rel= field to desc= for resource discovery.
- o Changed the maximum message size to 1024 bytes to allow for IP/UDP headers.

- o Made the URI slash optimization and method idempotence MUSTs
- o Minor editing and bug fixing.

Changes from shelby-00 to shelby-01:

- o Unified the message header and added a notify message type.
- o Renamed methods with HTTP names and removed the NOTIFY method.
- o Added a number of options field to the header.
- o Combines the Option Type and Length into an 8-bit field.
- o Added the magic byte header.
- o Added new ETag Option.
- o Added new Date Option.
- o Added new Subscription Option.
- o Completed the HTTP Code - CoAP Code mapping table appendix.
- o Completed the Content-type Identifier appendix and tables.
- o Added more simplifications for URI support.
- o Initial subscription and discovery sections.
- o A Flag requirements simplified.

Authors' Addresses

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Brian Frank
SkyFoundry
Richmond, VA
USA

Phone:
Email: brian@skyfoundry.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2013

K. Hartke
Universitaet Bremen TZI
October 22, 2012

Observing Resources in CoAP
draft-ietf-core-observe-07

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 a server to replicate a resource state to interested clients. The protocol follows a best-effort approach when transmitting new resource states to clients, and provides eventual consistency between the state observed by each client and the actual resource state.

Editor's Note

This is an interim revision which will receive further modifications during the resolution of open tickets, in particular #204, #235 and #242.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Background	3
1.2. Protocol Overview	3
1.3. Design Philosophy	6
1.4. Requirements Notation	6
2. The Observe Option	7
3. Client-side Requirements	7
3.1. Request	7
3.2. Notifications	8
3.3. Caching	8
3.4. Reordering	9
3.5. Cancellation	10
4. Server-side Requirements	11
4.1. Request	11
4.2. Notifications	11
4.3. Caching	12
4.4. Reordering	13
4.5. Retransmission	14
5. Intermediaries	15
6. Block-wise Transfers	15
7. Discovery	16
8. Security Considerations	16
9. IANA Considerations	17
10. Acknowledgements	17
11. References	18
11.1. Normative References	18
11.2. Informative References	18
Appendix A. Examples	19
A.1. Proxying	22
A.2. Block-wise Transfer	24
Appendix B. Modeling Resources to Tailor Notifications	24
Appendix C. Changelog	25
Author's Address	28

1. Introduction

1.1. Background

CoAP [I-D.ietf-core-coap] is an Application Protocol for Constrained Nodes/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 communication model of REST is that of a client exchanging resource states with an origin server using representations. The origin server is the definitive source for representations of the resources in its namespace. A client interested in the state of a resource sends a request to the origin server; the server then returns a response with a representation that is current at the time of the request.

This model does not work well when a client is interested in knowing the state of a resource over a period of time. Existing approaches when using HTTP, such as repeated polling or long-polls [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 to replicate a resource state from a server to interested clients over a period of time, while still keeping the properties of REST.

There is no intention for this mechanism 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, an observer must register separately for all of them. The pattern is typically used when a clean separation between related components is required, such as data storage and user interface.

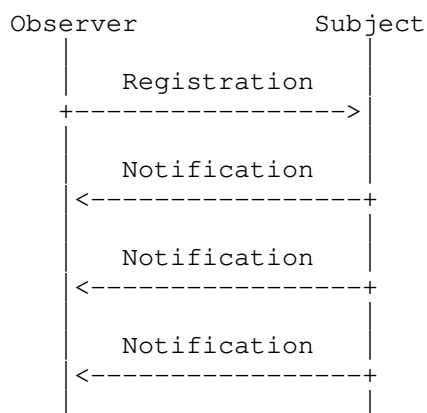


Figure 1: Observer Design Pattern

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 the current state of the resource at any given time.

Registration: A client registers its interest by sending an extended GET request to the server. In addition to returning a representation of the target resource, this request causes the server to add the client to the list of observers for the resource.

Notification: Whenever the state of a resource changes, the server notifies each client registered as observer for the resource. Each notification is an additional CoAP response sent by the server in reply to the GET request and includes a complete representation of the new resource state.

Figure 2 shows an example of a CoAP client registering its interest in a resource and receiving three notifications: the first with the current state upon registration and then two notifications when the state of the resource changes. Registration request and notifications are identified by the presence of the Observe Option defined in this document. All notifications echo the token specified by the client in the request, so the client can easily correlate them to the request.

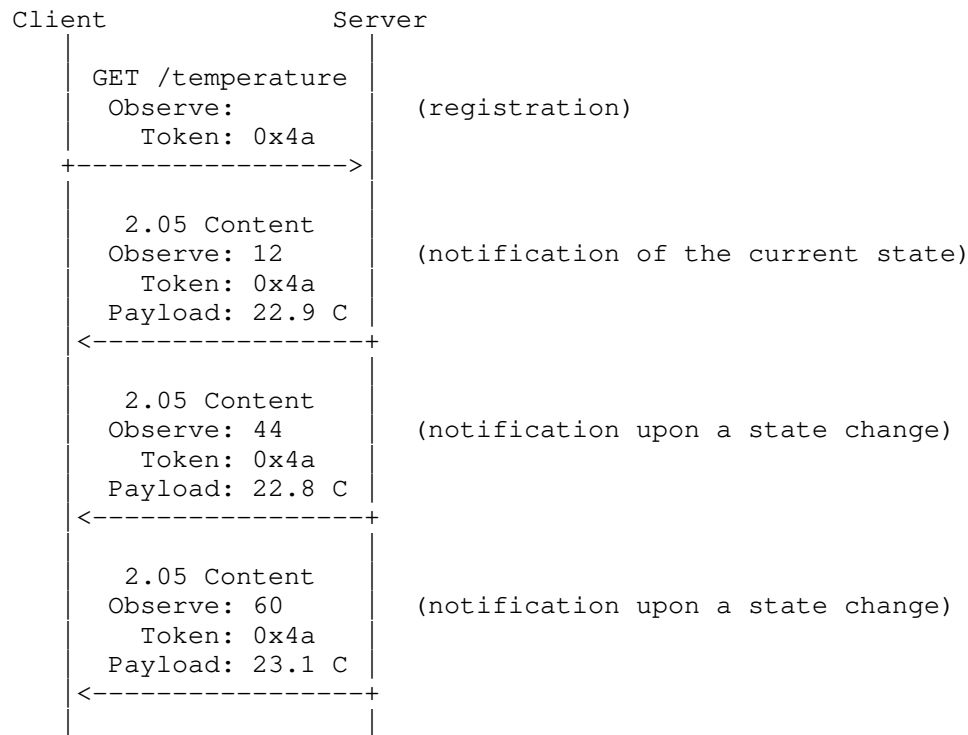


Figure 2: Observing a Resource in CoAP

A client remains on the list of observers as long as the server can determine the client's continued interest in the resource. The interest is determined by the server from the client's acknowledgement of notifications sent in confirmable messages. If the client rejects a notification or if the transmission of a notification ultimately fails, then the client is assumed to be no longer interested and is removed by the server from the list of observers.

A notification is cacheable like any other response and can be used until the next notification arrives. Each notification includes an indication of when the server will send the next notification at latest (Max-Age). This helps a client that does not receive a notification for a while, to decide if the resource simply did not undergo a change of state yet or if the next notification is overdue and the server is apparently no longer aware of the client's interest in the resource.

When a client wants to be notified after it has determined that no further notifications can be expected, it needs to register again.

1.3. Design Philosophy

The protocol builds on the architectural elements of REST, which include: a server that is responsible for the state and representation of the resources in its namespace, a client that is responsible for keeping the application state, and the stateless exchange of resource representations. (Beyond stateless REST, a server needs to keep track of the observers though, somewhat similar to how HTTP servers need to keep track of the TCP connections from their clients.) The protocol enables high scalability and efficiency through the support of caches and intermediaries that multiplex the interest of multiple clients in the same resource into a single association.

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 meaningful in the application context. Resources can be parameterized to achieve similar effects though; see Appendix B for examples.

Since bandwidth is in short supply in constrained environments, a server must adapt the rate of notifications to each client. This implies that a client cannot rely on observing every single state a resource goes through. Instead, the protocol follows a best-effort approach when transmitting the new resource state 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.

Furthermore, the protocol is designed on the principle of eventual consistency: it guarantees that if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the last resource state.

1.4. 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	x		Observe	empty/uint	0 B/0-2 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

The Observe Option, when present, modifies the GET method so it does not only retrieve a representation of the current state of the resource identified by the request URI, but also requests the server to add the client to the list of observers of the resource. The exact semantics are defined in the sections below. The value of the option in a request MUST be empty on transmission and MUST be ignored on reception.

In a response, the Observe Option identifies the message as a notification, which implies that the client has been added to the list of observers and that the server will notify the client of further changes to the resource state. The option's value is a sequence number that can be used for reordering detection (see Section 3.4 and Section 4.4). The value is encoded as a variable-length unsigned integer as defined in Section 3.4.1 of RFC XXXX [I-D.ietf-core-coap].

Since the Observe Option is not critical, a GET request that includes the Observe Option will automatically fall back to a normal GET request if the server is unwilling or unable to add the client to the list of observers.

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 for as long as the server can assume the client's interest.

3.2. Notifications

Notifications are additional responses sent by the server in reply to the GET request. Each notification includes an Observe Option with a sequence number (see Section 3.4), a Token Option that matches the token specified by the client in the GET request, and a payload of the same Content-Format as the initial response.

A notification can be confirmable or non-confirmable (i.e. sent in a confirmable or non-confirmable message). If a client does not recognize the token in a confirmable notification, it **MUST NOT** acknowledge the message and **SHOULD** reject it with a RST message. Otherwise, the client **MUST** acknowledge the message with an ACK message as usual. If a client does not recognize the token in a non-confirmable notification, it **MAY** reject it with a RST message.

An acknowledgement 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 it from the list of observers.

Notifications will have a 2.05 (Content) response code in most cases. They may also have a 2.03 (Valid) response code if the client includes an ETag Option in its request (see Section 3.3). In the event that the state of an observed resource is changed in a way that would cause a normal GET request not to return success (for example, when the resource is deleted), the server will send a notification with a non-success response code (such as 4.xx/5.xx) and empty the list of observers of the resource.

3.3. Caching

As notifications are just additional responses, notifications partake in caching as defined by Section 5.6 of RFC XXXX [I-D.ietf-core-coap]. Both the freshness model and the validation model are supported. The freshness model also serves as the model for the client to determine if it's still on the list of observers or if it needs to re-register its interest in the resource.

A client **MAY** store a notification like a response in its cache and use a stored notification/response that is fresh without contacting the origin server. A notification/response is considered fresh while its age is not greater than its Max-Age and no newer notification has been received.

The server will do its best to keep the client up to date with a fresh representation of the current resource state. It will send a

notification whenever the resource changes, or at latest when the age of the last notification becomes greater than its Max-Age. (Note that this notification may not arrive in time due to network latency.)

The client SHOULD assume that it's on the list of observers while the age of the last notification is not greater than Max-Age. If the client does not receive a notification before the age becomes greater than Max-Age, it can assume that it has been removed from the list of observers (e.g., due to a loss of server state). In this case, it may need to re-register its interest.

To make sure it has a fresh representation and/or to re-register its interest, a client MAY issue a new GET request with an Observe Option at any time. The GET request SHOULD specify a new token to avoid ambiguity, because the token serves as epoch identifier for the sequence numbers in the Observe Option (see Section 3.4).

It is RECOMMENDED that the client does not issue the request while it still has a fresh notification and, beyond that, while a new notification from the server is still likely to arrive. I.e. the client should wait until the age of the last notification becomes greater than its Max-Age plus MAX_LATENCY (the maximum time a datagram is expected to take from the start of its transmission to the completion of its reception; see Section 4.8 of RFC XXXX [I-D.ietf-core-coap]).

When a client has one or more notifications stored, it can use the ETag Option in the GET request to give the server an opportunity to select a stored response to be used. The client MAY include an ETag Option for each stored response that is applicable. It needs to keep those responses in the cache until it is no longer interested in receiving notifications for the target resource or it issues a new GET request with a new set of entity-tags. Whenever the observed resource changes its state 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.

3.4. Reordering

Messages that carry notifications can arrive in a different order than they were sent. Since the goal is eventual consistency (see Section 1.3), a client MAY safely skip a notification that arrives later than a newer notification. For this purpose, the server sets the value of the Observe Option in each notification to a 24-bit sequence number.

A notification is older than the latest notification received and thus can be skipped when the following condition is met:

$$(V1 - V2) \% (2^{**}24) < (2^{**}23) \quad \text{and} \quad T2 < (T1 + \text{EXCHANGE_LIFETIME})$$

where V1 is the value of the Observe Option of the latest valid notification received, V2 the value of the Observe Option of the present notification, T1 a client-local timestamp of the latest valid notification received (in seconds), and T2 a client-local timestamp of the present notification.

Design Note: The first condition essentially verifies that $V2 > V1$ holds in 24-bit sequence number arithmetic [RFC1982]. The second condition checks that the time expired between the two incoming messages is not so large that the sequence number might have wrapped around and the first check is therefore invalid. (In other words, after about EXCHANGE_LIFETIME seconds elapse without any notification, the client does not need to check the sequence numbers in order to assume an incoming notification is new.) The constant of $2^{**}23$ is non-critical, as is the even speed or precision of the clock involved.

3.5. Cancellation

When a client rejects a confirmable notification with a RST message or when it performs a GET request without an Observe Option for a currently observed resource, the server will remove the client from the list of observers for this resource. The client MAY use either method at any time to indicate that it is no longer interested in receiving notifications about a resource.

When a client rejects non-confirmable notification with a RST, there is also a chance that the server will remove the client from the list of observers for this resource. So the client MAY try this method as well. A client MAY rate-limit the RST messages it sends if the server appears to persistently ignore them.

Implementation Note: A client that does not mediate all its requests through its cache might inadvertantly cancel an observation relationship by sending an unrelated GET to the same resource. To avoid this, without incurring a need for synchronization, such clients can use a different source transport address for these unrelated GET requests.

4. Server-side Requirements

4.1. Request

A GET request that includes an Observe Option requests the server not only to return a representation of the resource identified by the request URI, but also to add the client to the list of observers of the target resource. If no error occurs, the server **MUST** return a response with the representation of the current resource state and **MUST** notify the client of subsequent changes to the state as long as the client is on the list of observers.

A server that is unable or unwilling to add the client to the list of observers of the target resource **MAY** silently ignore the Observe Option and process the GET request as usual. The resulting response **MUST NOT** include an Observe Option, the absence of which signals to the client that it will not be notified of changes to the resource state and, e.g., needs to poll the resource instead.

If the client is already on the list of observers, the server **MUST NOT** add it a second time but **MUST** replace or update the existing entry. If the server receives a GET request for the same resource that does not include an Observe Option or a GET request that includes an unrecognized critical option, the server **MUST** remove the client from the list of observers.

Two requests relate to the same list entry if and only if both the request URI and the source endpoint of the requests match. Message IDs and tokens are not taken into account.

Any request with a method other than GET **MUST NOT** have a direct effect on a list of observers of a resource. However, such a request can have the indirect consequence of causing the server to send a non-success notification which does affect the list of observers (e.g., when a DELETE request is successful and an observed resource no longer exists).

4.2. Notifications

A client is notified of resource state changes by additional responses sent by the server in reply to the GET request. Each such notification response **MUST** include an Observe Option and **MUST** echo the token specified by the client in the GET request. If there are multiple clients on the list of observers, the order in which they are notified is not defined; the server is free to use any method to determine the order.

A notification **SHOULD** have a 2.05 (Content) or 2.03 (Valid) response

code. However, in the event that the state of a resource changes in a way that would cause a normal GET request to return a non-success response code (for example, if the resource is deleted), the server SHOULD notify the client by sending a notification with an appropriate non-success response code (such as 4.xx/5.xx) and MUST empty the list of observers of the resource.

The Content-Format used in a notification MUST be the same as the one used in the initial response to the GET request. If the server is unable to continue sending notifications using this Content-Format, it SHOULD send a notification with a 5.00 (Internal Server Error) response code and MUST empty the list of observers of the resource.

A notification can be sent as a confirmable or a non-confirmable message. The message type used is typically application-dependent and MAY be determined by the server for each notification individually. For example, for resources that change in a somewhat predictable or regular fashion, notifications can be sent in non-confirmable messages; for resources that change infrequently, notifications can be sent in confirmable messages. The server can combine these two approaches depending on the frequency of state changes and the importance of individual notifications.

The acknowledgement of a confirmable notification implies the client's continued interest in being notified. If the client rejects a confirmable notification with a RST message, the server MUST remove the client from the list of observers. If the client rejects a non-confirmable notification with a RST message, the server MAY remove the client from the list of observers, i.e., it is expected that the server removes the client if it still has the state available that is needed to match the RST message to the notification, but the server is not required to keep this state.

If CoAP is used over a connection-oriented or session-based transport such as DTLS, the server MUST remove the client from the list of observers when the connection or session is closed.

4.3. Caching

The Max-Age Option of a notification SHOULD be set to a value that indicates when the server will send the next notification. For example, if the server sends a notification every 30 seconds, a Max-Age Option with value 30 should be included. The server MAY send a new notification before Max-Age ends and MUST send a new notification at latest when Max-Age ends. If the client does not receive a new notification before Max-Age ends, it will assume that it was removed from the list of observers (e.g., due to a loss of server state) and may issue a new GET request to re-register its interest.

It may not always be possible to predict when the server will send the next notification, for example, when a resource does not change its state in regular intervals. In this case, the server SHOULD set Max-Age to a good approximation. The value is a trade-off between increased usage of bandwidth and the risk of stale information. Smaller values lead to more notifications and more GET requests, while greater values result in network or device failures being detected later and data becoming stale.

The client can include a set of entity-tags in its request using the ETag Option. When the observed resource changes its state and the origin server is about to send a 2.05 (Content) notification, then, whenever that notification has an entity-tag in the set of entity-tags specified by the client, the server MAY send a 2.03 (Valid) response with an appropriate ETag Option instead. The server MUST NOT assume that the recipient has any response stored other than those identified by the entity-tags in the most recent GET request for the resource.

4.4. Reordering

Because messages can get reordered, the client needs a way to determine if a notification arrived later than a newer notification. For this purpose, the server MUST set the value of the Observe Option in each notification to the 24 least-significant bits of a strictly increasing sequence number. The sequence number MAY start at any value. The server MUST NOT reuse the same option value with the same client, token and resource within approximately $2 \times \text{EXCHANGE_LIFETIME}$ seconds (roughly 8.5 minutes with default CoAP parameters).

Implementation Note: A simple implementation that satisfies the requirements is to use a timestamp (in seconds) provided by the device's clock, or a 24-bit unsigned integer variable that is incremented periodically and does not wrap around more often than every $2 \times \text{EXCHANGE_LIFETIME}$ seconds. It is not necessary that the clock reflects the correct local time or that it ticks in a precisely periodical way.

The client is comparing sequence numbers only between notifications that arrive within EXCHANGE_LIFETIME seconds. However, a server should not assume that it is free to completely forget the sequence number right after EXCHANGE_LIFETIME -- the client may have a slower clock. If there is a need to discard sequence number state after some inactivity, this should be done only after $2 \times \text{EXCHANGE_LIFETIME}$ or later. Similarly, the sequence number should not increase so fast to span more than half the sequence number space within less than $2 \times \text{EXCHANGE_LIFETIME}$.

The 24-bit size for the sequence number has been chosen to enable very fast notification: If EXCHANGE_LIFETIME is the default value and 2*EXCHANGE_LIFETIME therefore approximately 2*9 seconds, the sequence number can increase every 2*14 times per second or approximately every 61 us before there is any danger of misdetection of wrap-around. On average, a server therefore is limited to about 15000 notifications per second per client and resource. This number may seem high in today's constrained node/networks, but it allows some leeway for both increased EXCHANGE_LIFETIME and high notification frequencies.

4.5. Retransmission

In CoAP, confirmable messages are retransmitted in exponentially increasing intervals for a certain number of attempts until they are acknowledged by the client. In the context of observing a resource, it is undesirable to continue transmitting the representation of a resource state when the state has changed in the meantime.

When a server is in the process of delivering a confirmable notification and is waiting for an acknowledgement, and it wants to notify the client of a state change using a new confirmable message, it MUST stop retransmitting the old notification and SHOULD attempt to deliver the new notification with the number of attempts remaining from the old notification. When the last attempt to retransmit a confirmable message with a notification for a resource times out, the server SHOULD remove the client from the list of observers and additionally MAY remove the client from the lists of observers of all resources in its namespace.

The server SHOULD use a number of retransmit attempts (MAX_RETRANSMIT) such that removing a client from the list of observers before Max-Age ends is avoided.

A server MAY choose to skip a notification if it knows that it will send another notification soon (e.g., when the state is changing frequently). Similarly, it MAY choose to send a notification more than once. For example, when state changes occur in bursts, the server can skip some notifications, send the notifications in non-confirmable messages, and make sure that the client observes the latest state change after the burst by repeating the last notification in a confirmable message.

When a notification is transmitted multiple times (either as caused by a retransmission attempt or repeating it), the server MUST update value of the Observe Option. Otherwise, the client might discard the notification as too old.

5. Intermediaries

A client may be interested in a resource in the namespace of an origin server that is reached through one or more CoAP-to-CoAP intermediaries. In this case, the client registers its interest with the first intermediary towards the origin server, acting as if it was communicating with the origin server itself as specified in Section 3. It is the task of this intermediary to provide the client with a current representation of the target resource and send notifications upon changes to the target resource state, much like an origin server as specified in Section 4.

To perform this task, the intermediary SHOULD make use of the protocol specified in this document, taking the role of the client and registering its own interest in the target resource with the next hop. 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, i.e. each hop in the server role MUST determine individually how many notifications to send, of which type, and so on. Each hop MUST generate its own values for the Observe Option, and MUST set the value of the Max-Age Option according to the age of the local current representation.

Because a client (or an intermediary in the client role) can only be once in the list of observers of a resource at a server (or an intermediary in the server role) -- it is useless to observe the same resource multiple times -- an intermediary MUST observe a resource only once, even if there are multiple clients for which it observes the resource.

An intermediary is not required to act on behalf of a client to observe a resource; an intermediary MAY observe a resource, for instance, just to keep its own cache up to date.

See Appendix A.1 for examples.

6. Block-wise Transfers

Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. CoAP provides a block-wise transfer mechanism to address this problem [I-D.ietf-core-block]. The following rules apply to the combination of block-wise 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 block-wise transfers, it SHOULD take note of the block size for all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the server receives a new GET request 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 value. The server performs the block-wise 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, or blocks that have been received more than approximately 2**14 seconds apart.

See Appendix A.2 for an example.

7. Discovery

A web link [RFC5988] to a resource accessible by the CoAP protocol MAY indicate that the server encourages the observation of this resource by including the link target attribute "obs". This is particularly useful in link-format documents [RFC6690].

The presence of this attribute can, for example, be used to indicate, via a graphical representation in a user interface, that this resource is changing its value and is useful for monitoring. The presence of this attribute is not a promise, though, that the Observe Option can actually be used to perform this observation. A client may need to resort to polling the resource if the Observe Option is not returned in the reply to the GET request.

The "obs" attribute is used as a flag, and thus has no value component -- a value given for the attribute MUST NOT be given for this version of the specification and MUST be ignored if present. The attribute MUST NOT be given more than once for this version of the specification.

8. Security Considerations

The security considerations of RFC XXXX [I-D.ietf-core-coap] apply.

The considerations about amplification attacks are somewhat amplified when observing resources. Without client authentication, a server MUST therefore strictly limit the number of notifications that it sends between receiving acknowledgements that confirm the actual

interest of the client in the data; i.e., any notifications sent in non-confirmable messages MUST be interspersed with confirmable messages. (An attacker may still spoof the acknowledgements if the confirmable messages are sufficiently predictable.)

As with any protocol that creates state, attackers may attempt to exhaust the resources that the server has available for maintaining the list of observers for each resource. Servers may want to access-control this creation of state. As degraded behavior, the server can always fall back to processing the request as a normal GET request (without an Observe Option) if it is unwilling or unable to add a client to the list of observers of a resource, including if system resources are exhausted or nearing exhaustion.

Intermediaries must be careful to ensure that notifications cannot be employed to create a loop. A simple way to break any loops is to employ caches for forwarding notifications in intermediaries.

9. IANA Considerations

The following entries are added to the CoAP Option Numbers registry:

Number	Name	Reference
6	Observe	[RFCXXXX]

10. Acknowledgements

Carsten Bormann was an original author of this draft and is acknowledged for significant contribution to this document.

Thanks to Daniele Alessandrelli, Jari Arkko, Peter Bigot, Angelo Castellani, Gilbert Clark, Esko Dijk, Thomas Fossati, Brian Frank, Cullen Jennings, Matthias Kovatsch, Salvatore Loreto, Charles Palmer and Zach Shelby for helpful comments and discussions that have shaped the document.

Klaus Hartke was funded by the Klaus Tschira Foundation.

11. References

11.1. Normative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",
draft-ietf-core-block-10 (work in progress), October 2012.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-12 (work in progress), October 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

11.2. Informative References

- [GOF] Gamma, E., Helm, R., Johnson, R., and J. Vlissides,
"Design Patterns: Elements of Reusable Object-Oriented
Software", Addison-Wesley, Reading, MA, USA,
November 1994.
- [REST] Fielding, R., "Architectural Styles and the Design of
Network-based Software Architectures", Ph.D. Dissertation,
University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982,
August 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.
- [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

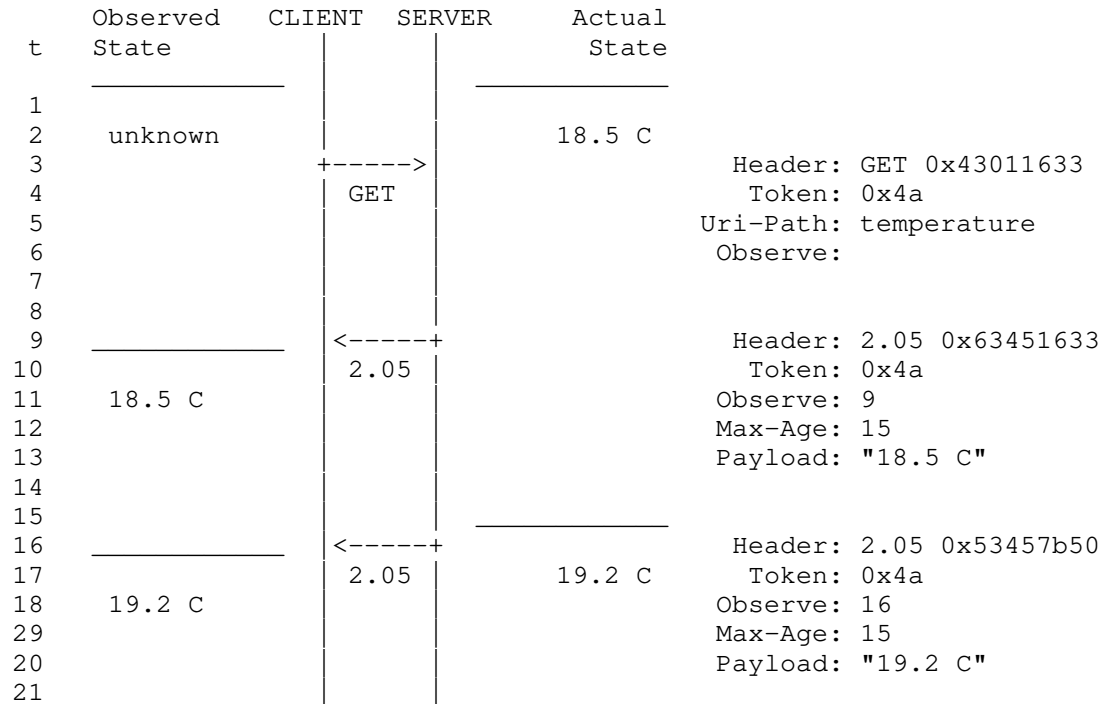


Figure 3: A client registers and receives a notification of the current state and upon a state change

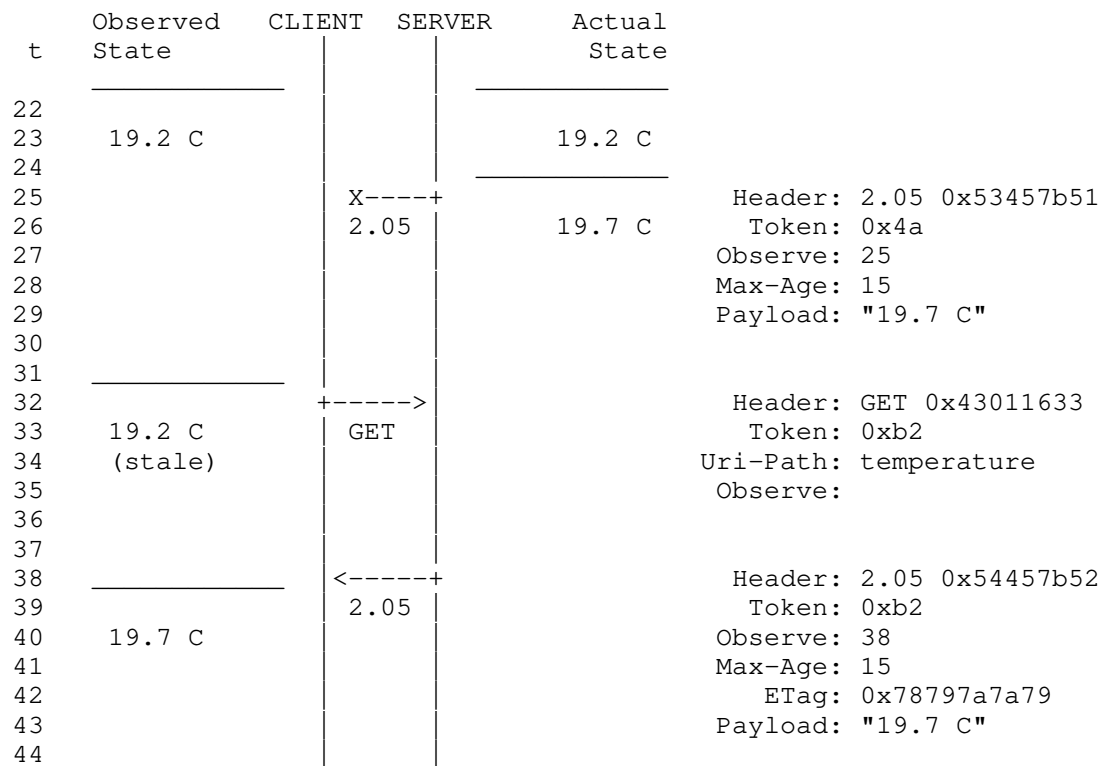


Figure 4: The client re-registers after Max-Age ends

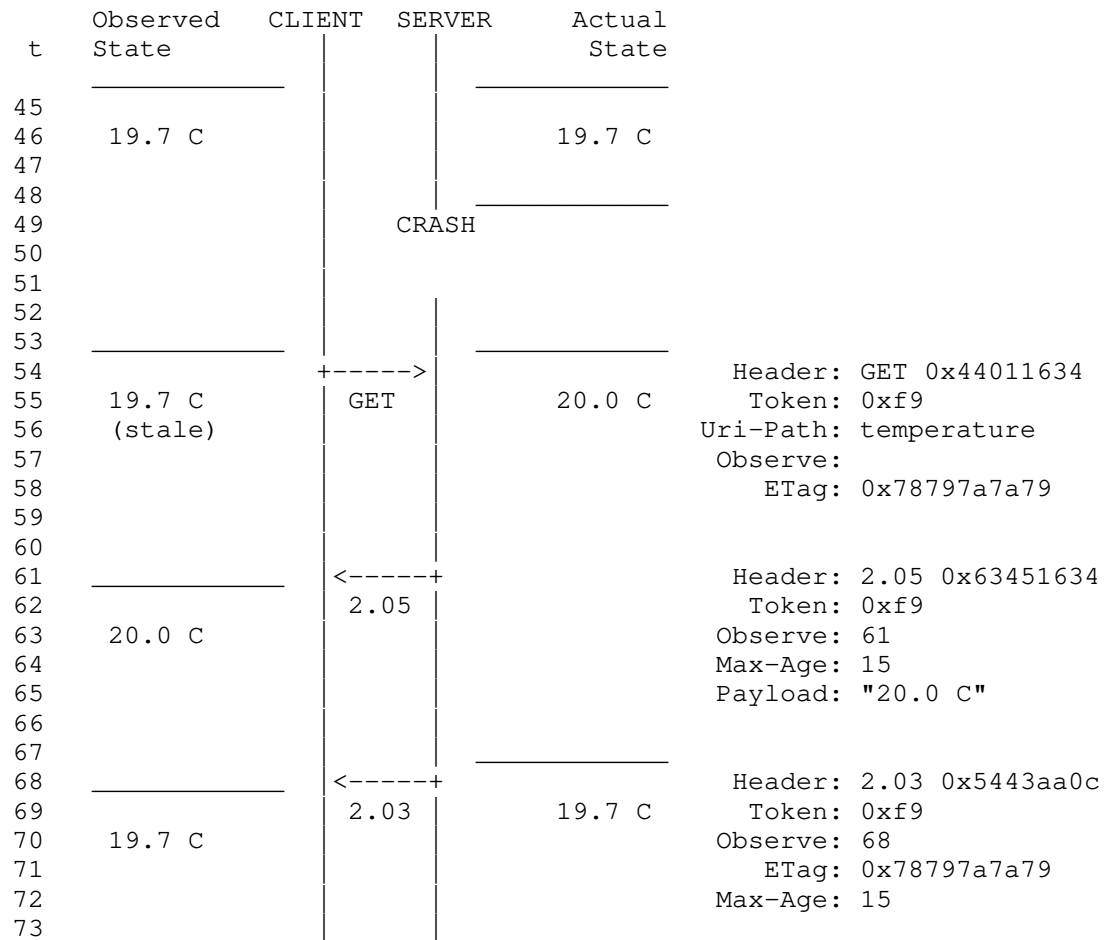


Figure 5: The client re-registers and gives the server the opportunity to select a stored response

A.1. Proxying

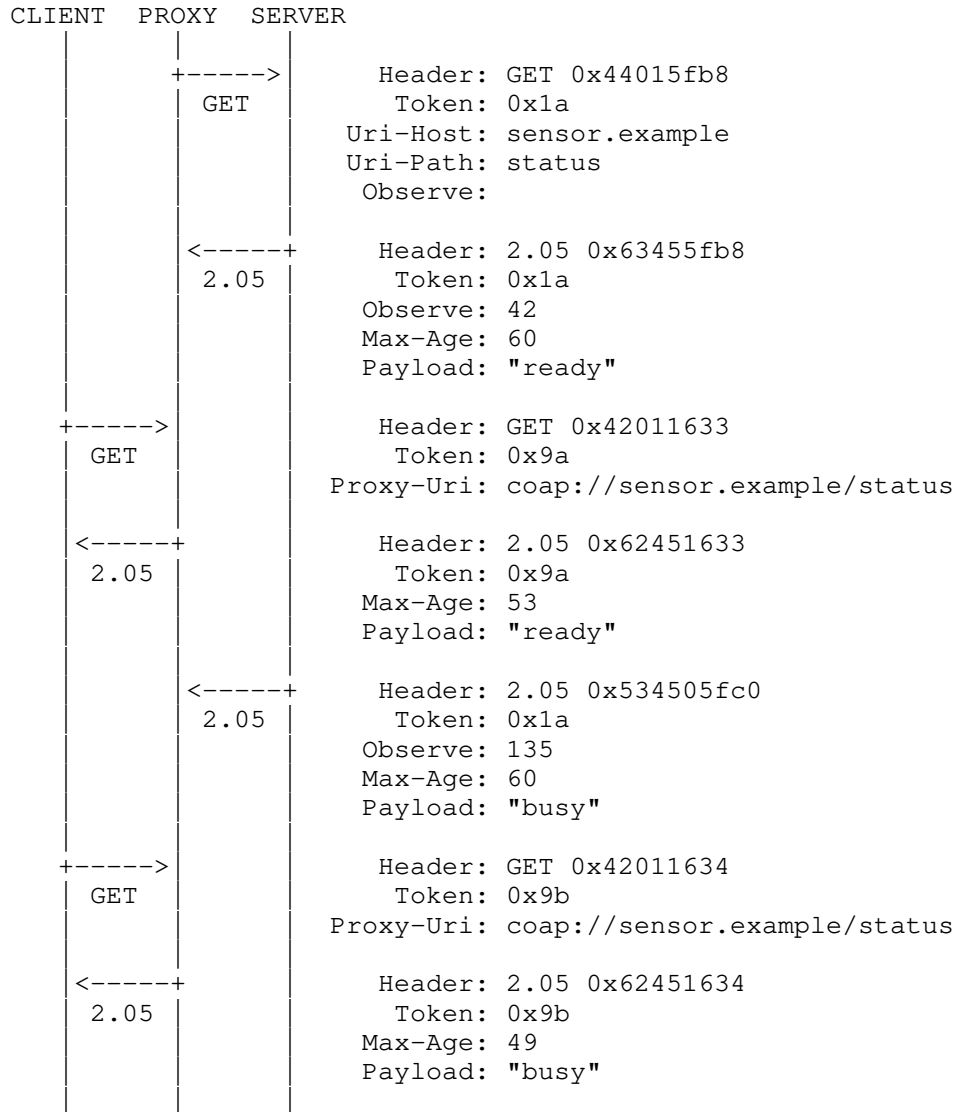


Figure 6: A proxy observes a resource to keep its cache up to date

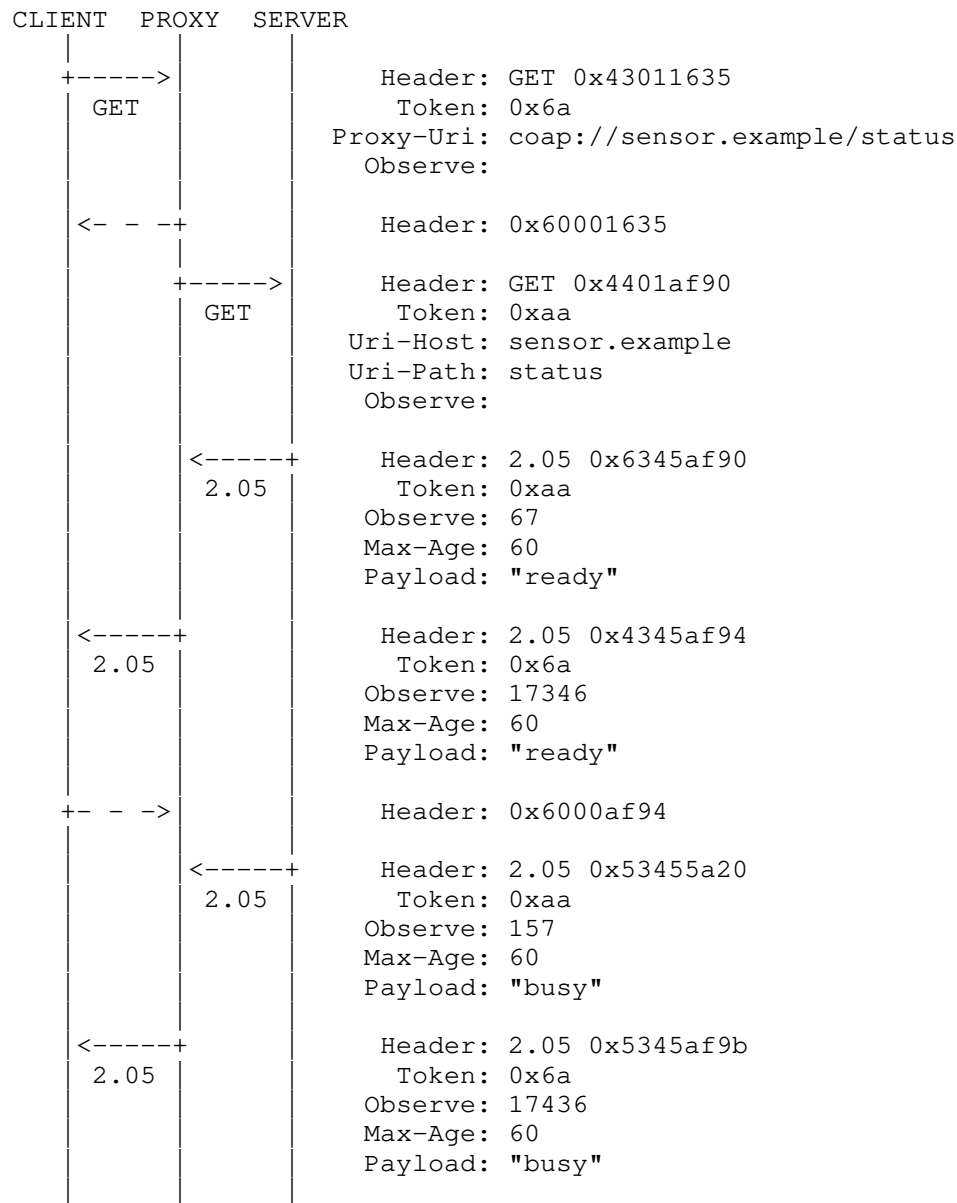


Figure 7: A client observes a resource through a proxy

A.2. Block-wise Transfer

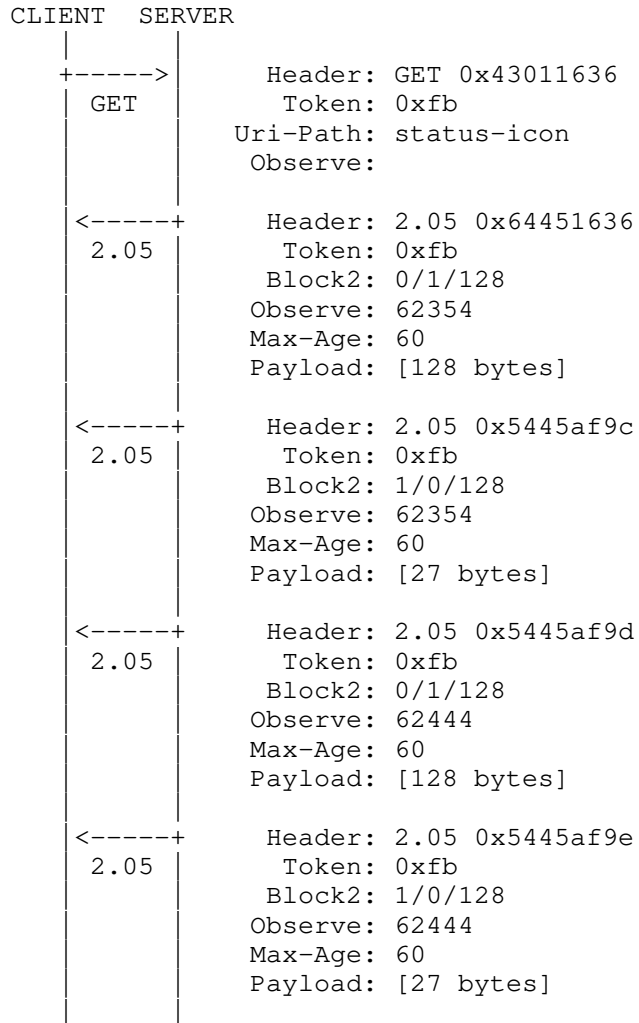


Figure 8: A server sends two notifications of two blocks each

Appendix B. Modeling Resources to Tailor Notifications

A server may want to provide notifications that respond to very specific conditions on some state. This is best done by modeling the resources that the server exposes according to these needs.

For example, for a CoAP server with an attached temperature sensor,

- o the server could, in the simplest form, expose a resource `<coap://server/temperature>` that changes its state every second to the current temperature measured by the sensor;
- o the server could, however, also expose a resource `<coap://server/temperature/felt>` that changes its state to "cold" when the temperature drops below a preconfigured threshold, and to "warm" when the temperature exceeds a second, higher threshold;
- o the server could expose a parameterized resource `<coap://server/temperature/critical?above=45>` that changes its state to the current temperature if the temperature exceeds the specified value, and 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. With parametrized resources, this is not limited to conditions defined by the server, but can be extended to arbitrarily complex conditions defined by the client. Thus, the server designer can choose exactly the right level of complexity for the application envisioned and devices used, and is not constrained to a "one size fits all" mechanism built into the protocol.

Appendix C. Changelog

Changes from ietf-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 RST message that is 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 RST in reply to non-confirmable notifications.
- o Added an implementation note about careless GETs (#184).
- o Updated examples.

Changes from ietf-03 to ietf-04:

- o Removed the "Max-OFE" Option.
- o Allowed RST 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 block-wise 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 block-wise transfers (#36).
- o Added Resource Discovery section (#99).
- o Added IANA Considerations.
- o Added Security Considerations (#40).
- o Added examples (#38).

Author's Address

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: April 16, 2013

C. Jennings
Cisco
October 13, 2012

Transitive Trust Enrollment for Constrained Devices
draft-jennings-core-transitive-trust-enrollment-01

Abstract

This document provides a sketch of a rendezvous protocol that allows constrained internet devices such as sensors to securely connect into a system that uses them. The solution is based on the idea that each device will be manufactured with a one time password that can be used by the customer to tell the device which controller to enroll with, and the device will be manufactured to contact a given Transfer Server that is used to tell the device which system to connect to. The administrator of the device will be able to get this one time password from the device using a QR code, and then will be able to use that one time password to inform a Transfer Server which system the device should connect to. The device will contact the Transfer Agent, get this information, and then connect to the appropriate system.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 16, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents
(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Enrollment Information Flow	5
3. Terminology	6
4. QR Code	6
5. Transfer Agent API	7
5.1. Create	7
5.2. Setup	8
5.3. Bind	8
5.4. Fetch	9
6. Controller API	9
6.1. Test Alive	10
6.2. Add	10
6.3. Sensor Update	10
7. Security Considerations	10
8. Variations	11
8.1. LED Based Enrollment	11
8.2. Bulk Enrollment	12
8.3. Public Key Crypto	12
9. Implementation Notes	12
9.1. Random Numbers	12
10. Open Issues	13
11. IANA Considerations	13
12. Acknowledgments	13
13. Appendix A: JOSE SHA224-CFB	13
13.1. Example	14
14. References	14
14.1. Normative References	14
14.2. Informative References	15
Author's Address	15

1. Introduction

Secure enrollment of devices into internet-based systems has never been easy. The constrained devices that need to be enrolled into systems today face many challenges. Typically, simple devices such as keyboards and screens have no user interface; they may have only a single button or LED. At the time they are installed, there may not be a working network or even power. However, these devices are being used for applications that are increasingly important and safety-critical, so they need to have reasonable security and privacy characteristics. This document specifies an enrollment system for such devices.

In many systems, there is a need to configure a Device, such as a sensor or actuator, so that it is controlled by some specific Controller. With Devices like a switch and light, it may be that all the Controller does is later configure the switch to control the light. To make this happen, both Devices need to be under the control of a common Controller that is authorized to make changes to the Devices.

The simplified high-level information flow is illustrated in the following figure. The goal is to get to the point where the Device knows that it should be talking to the Controller.

TODO - See PDF Version of draft

When the Manufacturer builds the Device, it includes a One Time Password (OTP) that the Introducer can use to enroll the Device with the Controller. The Manufacturer also runs a website known as the Transfer Agent that knows the OTP for every device that uses that Transfer Agent. The Device can include the OTP as a QR code on the outside of the Device. When the Device is installed, the network administrator or installer uses a software agent known as the Introducer. The Introducer would typically be simply a normal browser running on a smart phone with a camera that can read QR codes. When the Device is installed, the Introducer can scan the QR code on the Device. This QR code includes a URL to the Transfer Agent along with the OTP and a separate Device secret DevSecret. (Message 1). The Introducer then contacts the Transfer Agent and uses the OTP to tell the Transfer Agent which Controller this Device should use (Message 2). The Introducer can also tell the Controller the DevSecret (Message 3) so that the Controller and Device can authenticate each other. Later, the first time the Device boots up and gets network connectivity, it contacts the Transfer Agent, and the Transfer Agent tells the Device which Controller to talk to (Message 4). From that point on, any time the Device boots, the Device can communicate directly with the Controller (Message 5). The

actual message flow is slightly more complicated and shown in Section 2, but it uses the same basic idea as this simplified flow.

The system is designed to achieve several desirable properties:

- o Can work for Devices with very limited memory and processing power.
- o Does not require network or power to be available when the Device is installed.
- o Is fairly secure (see more in the security section).
- o Minimal addition to manufacturing costs.
- o The installer can detect if the OTP has already been used.
- o Provides a work flow in which a Device does not need to be taken out of the box to be enrolled. This can be very important to enable consumers themselves to enroll devices they buy from a service provider.
- o Works with common firewall and NAT network topologies.

One of the key steps in making this system work is getting the OTP from the Device to the Introducer. The approach used here is to use a QR code representing the URL. The QR code is printed on the Device and/or box it comes in.

The Device uses HTTP or COAP [I-D.ietf-core-coap] to communicate with the Controller. The Transfer Agent and Introducer use HTTPS to communicate with each other. There are three pieces of keying material used for cryptographic operations. The first is the One Time Password (OTP) that is passed via a QR code from the device to the Introducer and that the Introducer then uses to authorize itself to the Transfer Agent. There is also a DevSecret that is used to secure communications between the Device and the Controller. Finally there is a TaSecret that is used to secure communications between the Device and the TransferAgent. The Transfer Agent needs a normal certificate to use HTTPS.

It is assumed that the Device may have a NAT between it and the Controller and that the Device is on the inside of the NAT. The Transfer Agent is assumed to be a generally accessible server on the internet, but the Controller and Device can be on the inside of a firewall or NAT between them and the Transfer Agent.

The semantic level information in each message is discussed in Section 2 and the syntax of the messages is discussed in Section 5 and Section 6. The security properties of the system are described in Section 7.

2. Enrollment Information Flow

In the following message flow we use the following definitions:

TaURL An http URL that can be used to reach the root resource on the Transfer Agent.

DevURN A globally unique URN assigned by the Manufacturer to uniquely identify this Device.

OTP The One Time Password created by the Manufacturer for enrolling the Device.

TaSecret The secret created by the Manufacturer for the Device to communicate with the Transfer Agent.

DevSecret The secret created by the Manufacturer for the device to communicate with the Controller.

ContURL This is a URL that provides the address to reach the controller. It can have a scheme of http, https, coap, or coaps.

DevLabel A locally significant string that the Introducer can assign to a Device. For example, the convention for a thermostat in building 30, floor 2, office 361 might be assign the string "BLD30/2/361 - Thermostat". This string is provided purely as a way to let the Introducer and Controller exchange information that may be useful for the user installing the system.

The information flow is illustrated in the following figure. The goal is get to the point where the Device knows that it should be talking to the Controller, the Controller knows it should be talking the Device, and the Device and Controller can communicate and authenticate each other using the DevSecret.

TODO - See PDF Version of draft

When the Manufacturer builds the Device, it includes a TaSecret on the Device, a DevSecret, and the URN for the Device (DevURN). It also creates a QR code on the Device that contains the URL to the transfer agent (TaURL), the URN for the Device (DevURL), the OTP, and the DevSecret. This QR codes is described in detail in section TODO. The Manufacturer also tells the Transfer Agent the OTP, TaSecret and DevURN for this Device.

When the Device is installed, the Introducer reads OTP, DevSecret, DeviceURN, and the URL for the Transfer Agent (TaURL) from the Device by scanning the QR code on the device (Message 1). If the Introducer is a web browser, it uses the Transfer Agent URL to fetch an HTML user interface to perform the next steps (Message 2). The user interface on the Introducer allows the user to user to enter the URL for the Controller (ContURL) and an optional label for the device (DevLabel).

Next the controller tells the Transfer Agent the Controller URL to

use for this DeviceURN. This request is authenticated by the Transfer Agent using the OTP (Message 3). Open Issue: right now the OTP is transferred in the request (which is over HTTPS). A better design might be to have the Introducer prove possession of the OTP to the Transfer Agent and not send the OTP over the wire.

The Introducer also tells the controller the DevSecret for this Device and the optional DevLabel (message 4).

Later the Device contacts the Transfer Agent and the Transfer Agent tells the Device the URL of the Controller to talk to (ContURL) in message 5 and 6). The information from the Transfer Agent to Device is encrypted and signed with the TaSecret.

From that point on, any time the Device boots, it can directly communicate with the Controller (Messages 7 and 8). The Controller and Device both know the DevSecret for the Device and can use that to authenticate and encrypt communications between them. It is suggested that the first thing the Controller and Device should do is to use this DevSecret to securely replace it with some different secret that is not known to anyone that saw the QR code.

Open Issue: should we add in an additional ContSecret that is picked by the Controller, passed to Introducer, then passed to the Trust Agent, and finally passed to Device?

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

When this draft says Base64, it means the URL safe Base64 encoding from TODO.

4. QR Code

The QR code for the Device MUST be an HTTPS URL that points at the appropriate Transfer Agent. The authority MUST be formed by using the authority from the TaURL. The path is formed by concatenating ".well-known/tt1/" followed the DevURN followed by "/s". The DevURN SHOULD be one of the URNs defined in [I-D.arkko-core-dev-urn]. It MUST include the OTP as a Base64 encoded value for a parameter called otp. The secret MUST be encoded in Base64 and used as the fragment identifier of the URL. The secret is put as a fragment so that if the Introducer scans the QR code and dereferences the URL with a web

browser, the fragment identifier will not be transferred in the request to the Transfer Agent.

As an example, if the Transfer Agent's domain is example.net, a valid URL for the QR code would be:

```
TODO - change hex to base64
https://example.net/.well-known/tte1/urn:dev:mac:90a2da001a0c/s
?otp=88F5EC91493E473B758159C7792C#00004DCFD CDBD9F54C1B2E71FC22
```

The QR code SHOULD use an error coding level of "H". This would generate the following QR code:

QR code in ASCII art left as an exercise to the reader but there is one in the PDF version.

5. Transfer Agent API

Note that future version of the API that needed to increment a version number would do it by changing the tte1 to tte2.

TODO - still need to define all the error responses but basic approach will be a simple JSON object with the error responses.

5.1. Create

The HTTP REST API allows the manufacturer to tell the Transfer Device about the DevURN and OTP for a given device the Manufacturer has created.

```
Path: .well-known/tte1/d/{DevURN}
Methods: POST
Parameters:
  otp: Base64 encoded version of the OTP
Return Values: TODO
```

This creates an entry for the device in the database and stores the OTP associated with this Device. The Transfer Agent SHOULD authenticate this request to authorize it. Note the "d" in the path is short for "device"; having this path segment allows for future extensibility.

Example Request:

```
POST https://example.net/.well-known/tte1/d/urn:dev:mac:90a2da001a0c
?otp=88F5EC91493E473B758159C7792C
```

Example Response:

TODO

5.2. Setup

The Transfer Agent MUST return a web page that allows the user to provide the information needed for the bind, and then the Introducer must call the bind and add methods.

Path: .well-known/ttcl/d/{DevURN}/s

Methods: GET

Parameters:

otp: Base64 encoded version of the OTP

Return Values: HTML5 web page

This MUST return a HTML web page that has a suitable user interface to allow the user to enter the address of the the Controller. It is suggested that the page validate that this controller entry is correct using the "alive" API in Section TODO. Once the Controller is verified, the web page MUST tell the Transfer Agent the Controller address using the "bind" API in Section TODO. The page MUST tell the controller the DevURN and DevSecret for the Device using the "add" API in Section TODO. MUST be done over HTTPS.

Example Request:

GET https://example.net/.well-known/ttcl/d/urn:dev:mac:90a2da001a0c/s
?otp=88F5EC91493E473B758159C7792C

5.3. Bind

TODO MUST be sent over TLS, and the Introducer MUST verify that the HTTPS certificate of the Transfer Agent matches the URL.

Once the Transfer Agent has successfully stored the Controller's address for a given OTP, it MUST NOT allow that OTP to be used again to store an address for that Device.

Path: .well-known/ttcl/d/{DevURN}/c

Methods: PUT

Parameters:

otp: Base64 encoded version of the OTP

conturl: URL to controller escaped as necessary for placement in
a URL

Return Values: TODO

This request using the

Example Request:

TODO

PUT https://example.net/.well-known/tte1/d/urn:dev:mac:90a2da001a0c/c
?otp=88F5EC91493E473B758159C7792C

5.4. Fetch

This API allows a Device to get the information about the controller it should connect to. It is provided in a JSON object which is encrypted using the OTP.

Path: .well-known/tte1/{DevURN}/c

Methods: GET

Parameters: None

Success Return Values: JSON object as defined in TODO that contains the encrypted URL to the Controller.

Error Return Values: Returns HTTP 404 if the DevID can not be found or if the controller URL has not yet been set for this DevURN.

The Transfer Agent looks up the OTP and ContURL for the requested DevURN. If the DevURN cannot be found, or the ContURL has not yet been set for this DevURN, then the Transfer Agent returns an HTTP 404 error. If they are found, it then the Authenticated Encryption with Associated Data (AEAD) algorithm described in Appendix A (TODO ref) to form the JSON object that is returned. The TaSecret is used as the key for the AEAD, the ContURL is the input data to be encrypted, and the DevURN is used as Associated Data for the authentication.

Example Request:

GET https://example.net/.well-known/tte1/d/urn:dev:mac:90a2da001a0c/c

Example Response:

TODO

6. Controller API

The Alive and Add API need to include a CORS (TODO REF) header to allow AJAX from the Transfer Agent to call these APIs. They MUST include an HTTP header in the response that sets the header field Access-Control-Allow-Origin to a value of "*". TODO security section

needs to discuss implications.

6.1. Test Alive

This method allows the Introducer to validate that a valid Controller address has been entered. It simply returns an HTTP 200 if the controller is operational.

Path: .well-known/ttel/alive
Methods: GET
Parameters: None
Return Values: TODO

6.2. Add

This method is used by the Introducer to add a new Device to the Controller. (Open issues - should it result in redirect to a controller web page to configure device?)

Path: .well-known/ttel/c/{DevURN}/k
Methods: PUT
Parameters:
 devSecret: Base64 encoded version of the secret
 devLabel:
Return Values: TODO

6.3. Sensor Update

TODO

Path: .well-known/ttel/s/{DevURN}/v
Methods: PUT
Parameters: None
Body: Encrypted SENML
Return Values: TODO

The body is a Encrypted JOSE object, as specified in Appendix A (TODO REF). The secret for this Device is used as the key to encrypt the data. The DevURN is used as the associated data. The decrypted data is a SENML sensor reading for this Device as described in [I-D.jennings-senml].

7. Security Considerations

This section has not really been started and needs lots of work.

TODO - Discuss how one can replace a dead Controller with a new one

in an operational network. The short answer is likely that one needs to back up the keys of the old Controller and move these to the new Controller.

What happens if the OTP is stolen during Device transit? The short answer is that the Device is compromised at this point and needs to be discarded or returned to the manufacturer to get a new OTP. The Introducer needs to detect that this has happened and warn the user.

There are additional concerns about Devices that may be operational without ever being introduced to a Controller. For example, if a light switch supported this protocol but could also be used just as a stand alone light switch, there would be a risk that the OTP could be stolen by an attacker, with the attacker enrolling the Device to the attacker's Controller. If the correct user installed the light switch but did not Introduce it to anything, the fact it had been compromised would go undetected. One way to mitigate this risk might be to include some manual configuration on the Device to indicate that it is to be used in stand-alone mode, such as a jumper that can be cut.

Network topology consideration - Introducer can install firewall rules that allow Devices to contact Transfer Agent.

Explain why works with NATs / FWs.

8. Variations

8.1. LED Based Enrollment

An alternative to QR codes is to have an LED on the Device flash out the relevant information to the Introducer. The output string is formed by concatenating a 16-bit start of message constant value of 0x0001, followed by the 8 bit length of TaURN, TaURN, 8 bit length of DevURN, the DevURN, 8 bit length of OTP, OTP, 8 bit length of DevSect, DevSecret and then an 8-bit two's compliment checksum value computed over the previous bytes, including the start of message constant. All values are in network byte order. The resulting string is output using Non-Return-to-Zero Inverted (NRZI) encoding on the LED at a baud rate of 15 bps. This allows a Device such as a smartphone with video capture to detect the signal and recover the information.

TODO - see if this works at 30 bps. See about encoding multiple intensity levels or colors in the LED. Initial experiments indicate this does not work very well, as auto contrast in the video camera tends to saturate LED range. Would an Adler-32 checksum be better?

Could multiple colors of intensity improve the speed of this as this is very slow.

8.2. Bulk Enrollment

Imagine one wants to enroll a whole box of sensors. We should define some scheme where one could simply bar code something on the outside of a box so that all the sensors in the box could be bulk enrolled. Perhaps there could be a master secret and start and end DevURN on the outside of the box bar code. Then the OTP for a given Device would be generated using the master secret and DevURN of that Device. Work is needed to sort out details of a scheme like this.

8.3. Public Key Crypto

This specification assumes that COAP is being used with DTLS in Pre Shared Key (PSK) mode. It would also be possible to use DTLS with self signed certificates with a very similar flow, where the Introducer provided the Transfer Agent with the fingerprint of the certificate or public key of the Controller.

9. Implementation Notes

The system described here can be implemented on a very small device. An implementation for Arduino with ethernet was done that includes all the parts described here, including SENML, JSON, the encryption and signing, HTTP, DNS, and DHCP. It also included libraries for reading a 1-wire temperature sensor. This fits in under 32k of flash, and uses less than 4k of ram on an 8 bit AVR processor. That means that the cost for an embedded processor in volume with this much flash, ram, etc. is very roughly \$1.50 USD in 2012. A key part of getting this to be small is the extremely small crypto footprint from using SHA224-CFB.

9.1. Random Numbers

Note: This section would be better in a separate draft.

TODO - Explain how to use SHA224_DRBG as defined in NIST SP800-90A. TODO REF. Store reseed counter in eeprom every 24 hours. Explain what to store in eeprom on reseed. TODO REF RFC 4086 and XKCD 221.

Todo - Discuss sources of randomness in use: 16 bytes of random data created during manufacturing. A 32 bit boot counter that increments every time the device boots. 8 byte pool of random data from sensor readings. 8 byte pool of random data from timing of receiving or sending network packets. A 32 bit counter that increments each time

a random number is generated but resets to 0 on reboot.

10. Open Issues

The references section is in serious need of work - let me know stuff that should be added to it.

Does QR encoding of L work out better than H?

Is there any advantage in having the HTTP URL in well-known space?

Is there some clever way (perhaps zeroconf) for the Introducer to discover the ContURL?

11. IANA Considerations

TODO register .well-known/ttel

12. Acknowledgments

Some of the fundamental ideas in this draft were inspired by Max Pritikin's work on Transitive Trust Introduction. Randy Bush provided crisp and excellent advice on what the security properties of the solutions should be. I'd like to thank the following people for review comments: Eric Rescorla, Jari Arkko, Lyndsay Campbell, and Zach Shelby.

13. Appendix A: JOSE SHA224-CFB

Note: This section will eventually be moved to an experimental draft submitted to JOSE WG.

This section describes how to create a JOSE object as described by [I-D.barnes-jose-jsms] that is encrypted and signed with SHA224-CFB as specified in [HashCFB].

This adds a new ENCRYPTION algorithm called sha224-cfb to [I-D.barnes-jose-jsms]. This takes one parameter named "n" which represents the nonce as defined in [HashCFB]. It is RECOMMENDED that the key be 14 bytes long and that the nonce be 24 bytes long. The authentication information from the algorithm is stored in the "mac" field.

13.1. Example

TODO example. Todo fix to base64 instead of hex encoding. TOOD talk to Barnes about keyID and case with no key wrap. TODO - state of sha224-cfb and this is all experimental.

Input Key (Hex) = 88F5EC91493E473B758159C7792C
Input Associated Data = "urn:dev:mac:90a2da001a0c"
Input plain text = "http://example.com" - TODO

Result =
{
 TODO
}

14. References

14.1. Normative References

- [HashCFB] Forler, C., McGrew, D., Lucks, S., and J. Wenzel, "Hash-CFB: Authenticated Encryptions without a Block Cipher", Directions in Authenticated Ciphers Workshop , July 2012.
- [I-D.barnes-jose-jsms]
Barnes, R., "JavaScript Message Security Format",
draft-barnes-jose-jsms-00 (work in progress), June 2012.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-08 (work in progress), October 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

14.2. Informative References

[I-D.arkko-core-dev-urn]

Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-01 (work in progress), October 2011.

[I-D.jennings-senml]

Jennings, C., Shelby, Z., and J. Arkko, "Media Types for Sensor Markup Language (SENML)", draft-jennings-senml-07 (work in progress), October 2011.

Author's Address

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@iii.ca

CORE WG
Internet-Draft
Intended status: Standards Track
Expires: April 19, 2013

A. Rahman
InterDigital Communications, LLC
October 16, 2012

Enhanced Sleepy Node Support for CoAP
draft-rahman-core-sleepy-01

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 April 19, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Terminology and Conventions	3
2. Introduction	3
3. Proposal	4
3.1. RD Based Sleep Tracking	4
3.2. Example of Synchronous RD Based Sleep Tracking	5
3.3. Example of Asynchronous RD Based Sleep Tracking	7
3.4. RD Caching Proxy	11
4. Acknowledgements	12
5. IANA Considerations	12
6. Security Considerations	12
7. References	12
7.1. Normative References	12
7.2. Informative References	12
Author's Address	13

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 [I-D.ietf-core-link-format]. In addition, this document defines the following terminology:

Sleepy Node

A sleepy node is a CoAP client or server that may sometimes go into a sleep mode (i.e. go into a low power state to conserve power) and temporarily suspend CoAP protocol communication. A sleepy node may also sometimes remain in a fully powered on state where it has the capability to perform full CoAP protocol communication.

Non-Sleepy Node

A non-sleepy node is a CoAP client or server that always remains in a fully powered on state (i.e. always awake) where it has the capability to perform full CoAP protocol communication. The general operation of non-sleepy nodes are assumed to be well known and so are not explicitly spelled out in this document except where needed for clarity.

2. Introduction

The current CoAP approach assumes a minimal support of sleepy nodes as follows:

- o [I-D.ietf-core-coap] defines CoAP proxies which can cache and service requests for sleepy CoAP servers. A client explicitly sends a CoAP request (GET) to a proxy (identified by its IP address) while indicating the URI (of the resource of interest) associated to a sleepy CoAP origin server. If the proxy has a valid representation of the resource in its cache it can then respond directly to the client regardless of the current sleep state of the origin server. Otherwise the proxy has to attempt to retrieve (GET) the resource from the sleepy origin server. The attempt may or may not be successful depending on the sleep state of the origin server.
- o [I-D.ietf-core-link-format] and [I-D.shelby-core-resource-directory] defines a Resource Directory (RD) mechanism where sleepy CoAP servers can register/update

(POST/PUT to `"/.well-known/core"`) their list of resources on a central (non-sleepy) RD server. This allows clients to discover the list of resources from the RD (GET `/rd-lookup/...`) for a sleepy server, regardless of its current sleep state. Unlike a proxy, the RD stores only the URIs (i.e. CORE Link Format) for other nodes, and not the actual resource representation. The client then may attempt to retrieve (GET) the actual representation of the desired resource from the sleepy origin server. The attempt may or may not be successful depending on the sleep state of the origin server.

- o Lower layer (i.e. below the IP layer) support for sleepy nodes exist in most wireless technologies (e.g. IEEE 802.11 (WiFi), and IEEE 802.15.4 (ZigBee)). For example, most wireless technologies support limited functionality such as packet scheduling to account for sleepy nodes in their physical and MAC layer protocols. These lower layer functionalities are not aware of any specific timing or operational aspects of application layer protocols like CoAP.

3. Proposal

3.1. RD Based Sleep Tracking

The current CoAP approach to support sleepy nodes can be significantly improved by introducing RD based mechanisms for a CoAP client to determine whether:

- o A targeted resource is located on a sleepy server.
- o A sleepy server is currently in sleep mode or not.

We define the following new parameters to characterize a sleepy node:

- o SleepState - Indicates whether the node is currently in sleep mode or not (i.e. Sleeping or Awake).
- o SleepDuration - Indicates the maximum duration of time that the node stays in sleep mode.
- o TimeSleeping - Indicates the length of time the node has been sleeping (i.e. if Sleep State = Sleeping).
- o NextSleep - Indicates the next time the node will go to sleep (i.e. if Sleep State = Awake).

These parameters are all server (node) level and are new parameters added to the RD URI Template Variables defined in

[I-D.shelby-core-resource-directory].

We also define a new lookup-type ("ss") for the RD lookup interface specified in [I-D.shelby-core-resource-directory]. This new lookup-type supports looking up the SleepState of a specified end-point.

The three time based parameters (SleepDuration, TimeSleeping, NextSleep) can be based on either an absolute network time (for a time synchronized network) or a relative local time (measured at the local node).

Following the approach of [I-D.ietf-core-link-format] and [I-D.shelby-core-resource-directory], sleep parameters for sleepy servers can be stored by the server in the RD and accessed by all interested clients. Examples of using these parameters in a synchronous or asynchronous manner are shown in the following sections.

3.2. Example of Synchronous RD Based Sleep Tracking

Figure 1 shows an example of using RD based sleep tracking in a synchronous fashion:

(1) SleepyNode-1 is awake and having previously discovered the local RD, stores its CORE link format in the RD (POST/rd) identified by its entry point (?ep=SleepyNode-1). The sleep parameters are also updated as part of this step.

(2)-(3) RD services the POST and stores the CORE link format and starts the sleep timers for this node.

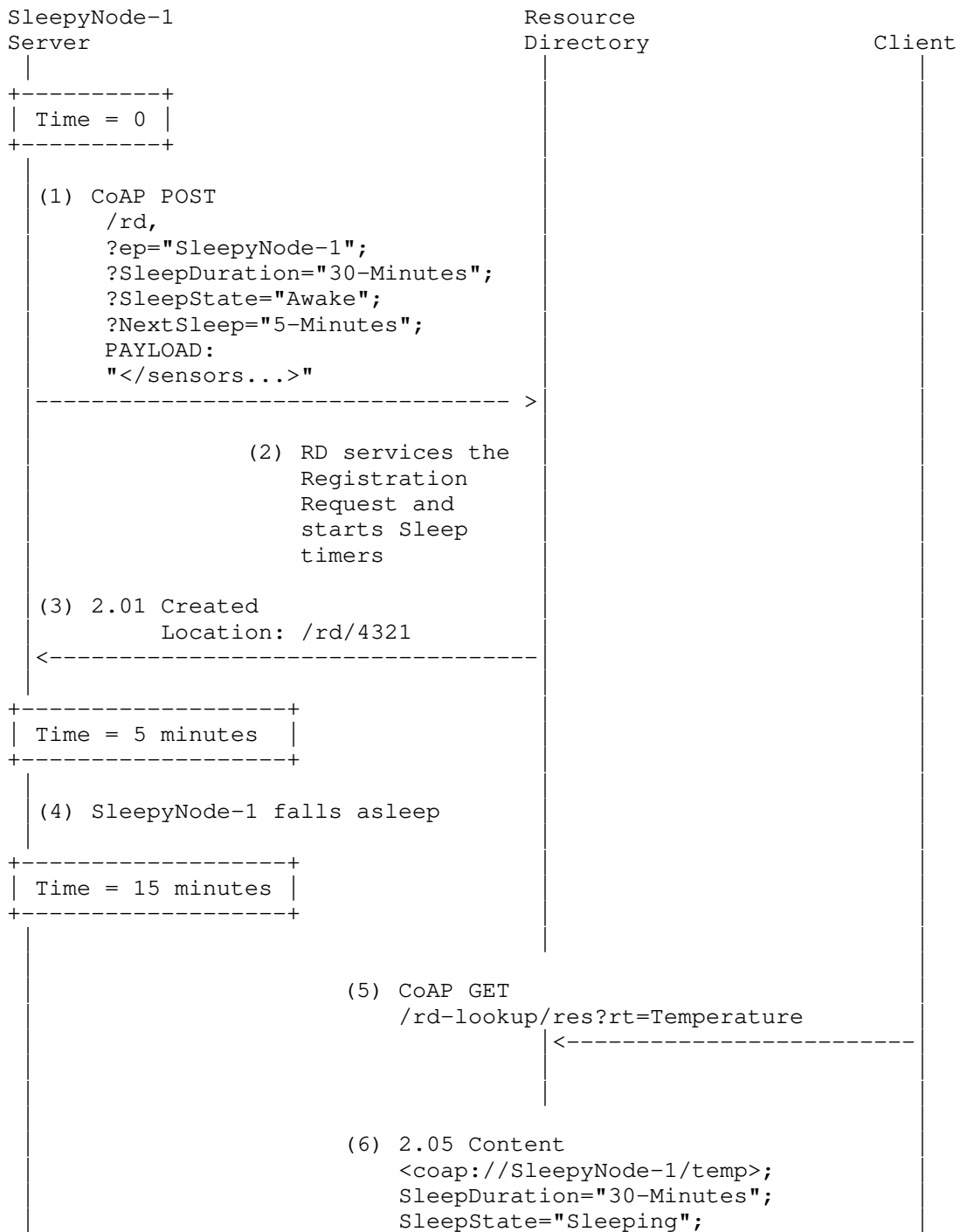
(4) SleepyNode-1 falls asleep.

(5) A client is interested in temperature sensors in this domain and does a lookup on the RD.

(6) The RD does a lookup and finds that SleepyNode-1 is the only node meeting the match and sends back the required info including the Sleep parameters.

(7) From the sleep parameters, the client knows that the node is currently asleep and so internally schedules to send the GET request when the node wakes up (plus a small safety hysteresis).

(8)-(9) Client sends GET request for temperature sensors and successfully receives the content as SleepyNode-1 is awake.



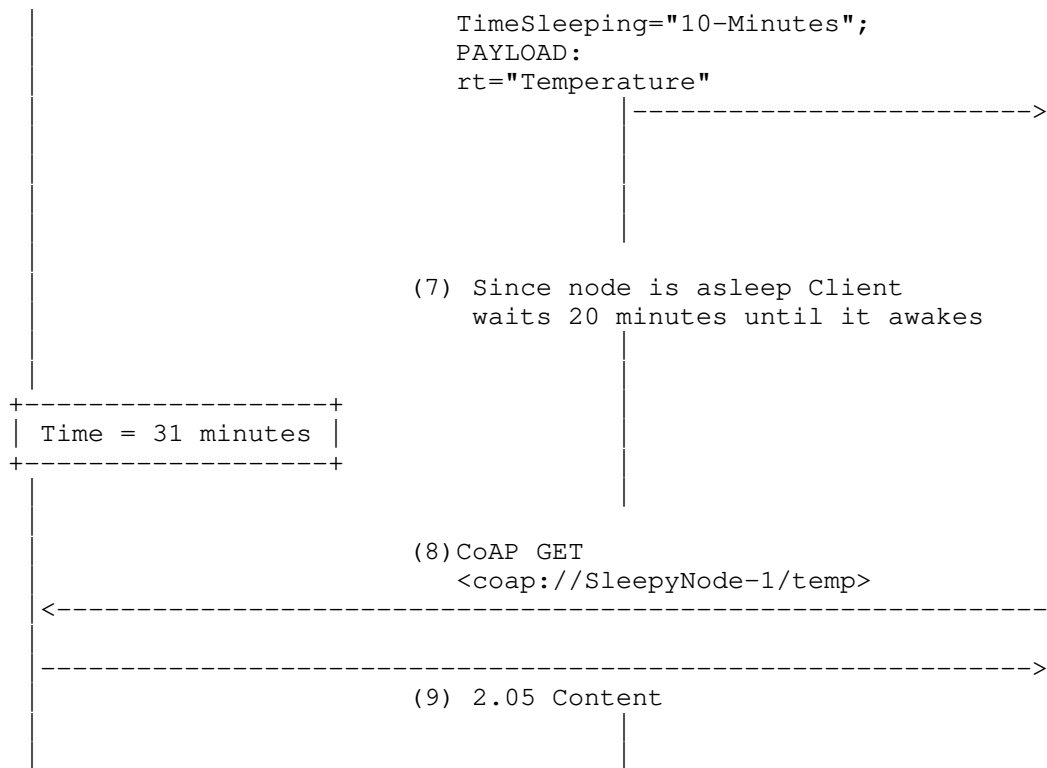


Figure 1: Synchronous Resource Directory Based Sleep Tracking

3.3. Example of Asynchronous RD Based Sleep Tracking

Figure 2 shows an example of using RD based sleep tracking in an asynchronous fashion:

(1) SleepyNode-1 is awake and having previously discovered the local RD, stores its CORE link format in the RD (POST/rd) identified by its entry point (?ep=SleepyNode-1).

(2)-(3) RD services the POST and stores the CORE link format.

(4) A client is interested in temperature sensors in this domain and does a lookup on the RD for all sensors that are currently awake.

(5) The RD does a lookup and finds that SleepyNode-1 is the only node meeting the match and sends back the required info.

(6)-(7) Using the sleep state lookup functionality (lookup-type :=

"ss"), the client adds itself to the list of observers to get SleepState updates from RD for SleepyNode-1 [I-D.ietf-core-observe].

(8)-(9) Client performs RD 'resource' lookup to find URI of temperature sensor of resource hosted on SleepyNode-1.

(10)-(13) SleepyNode-1 prepares to go to sleep and updates the SleepState in the RD.

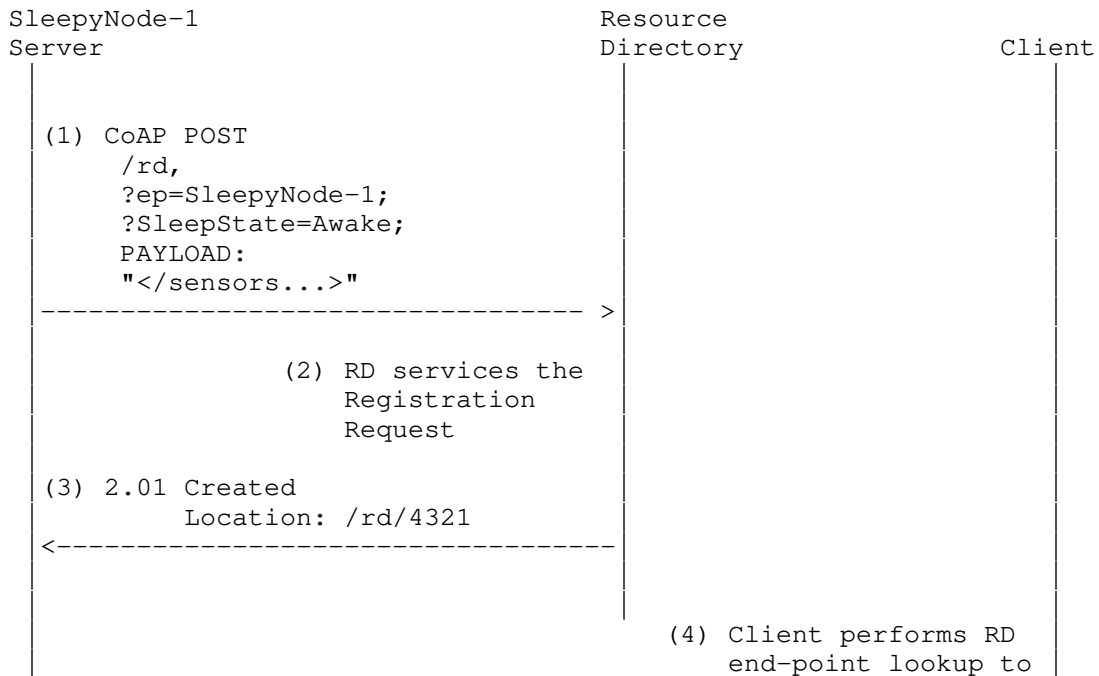
(14) RD notifies the client through previously established observe relationship.

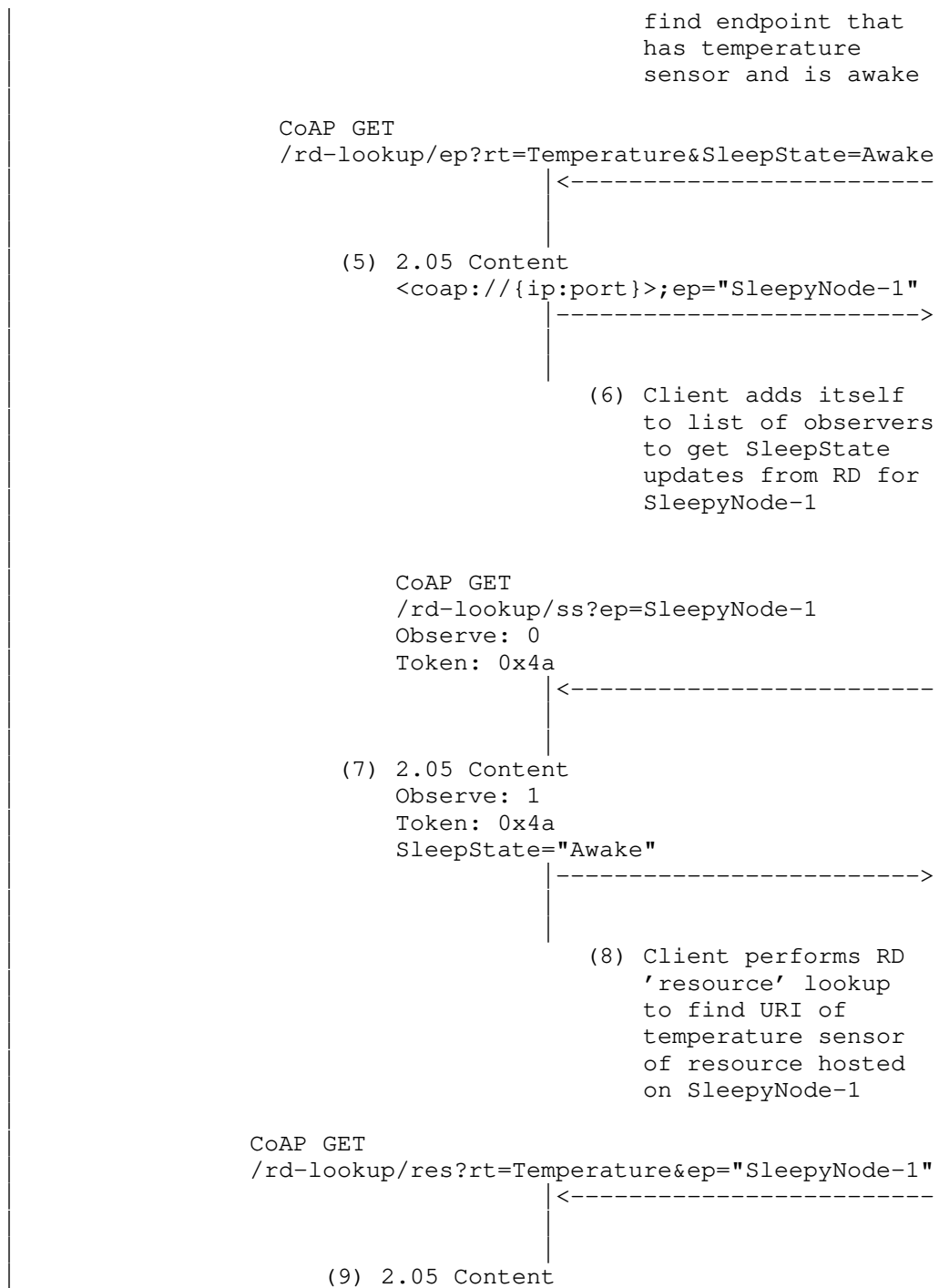
(15) Client application wants to get the temperature now but does not send the request as it knows SleepyNode-1 is currently sleeping.

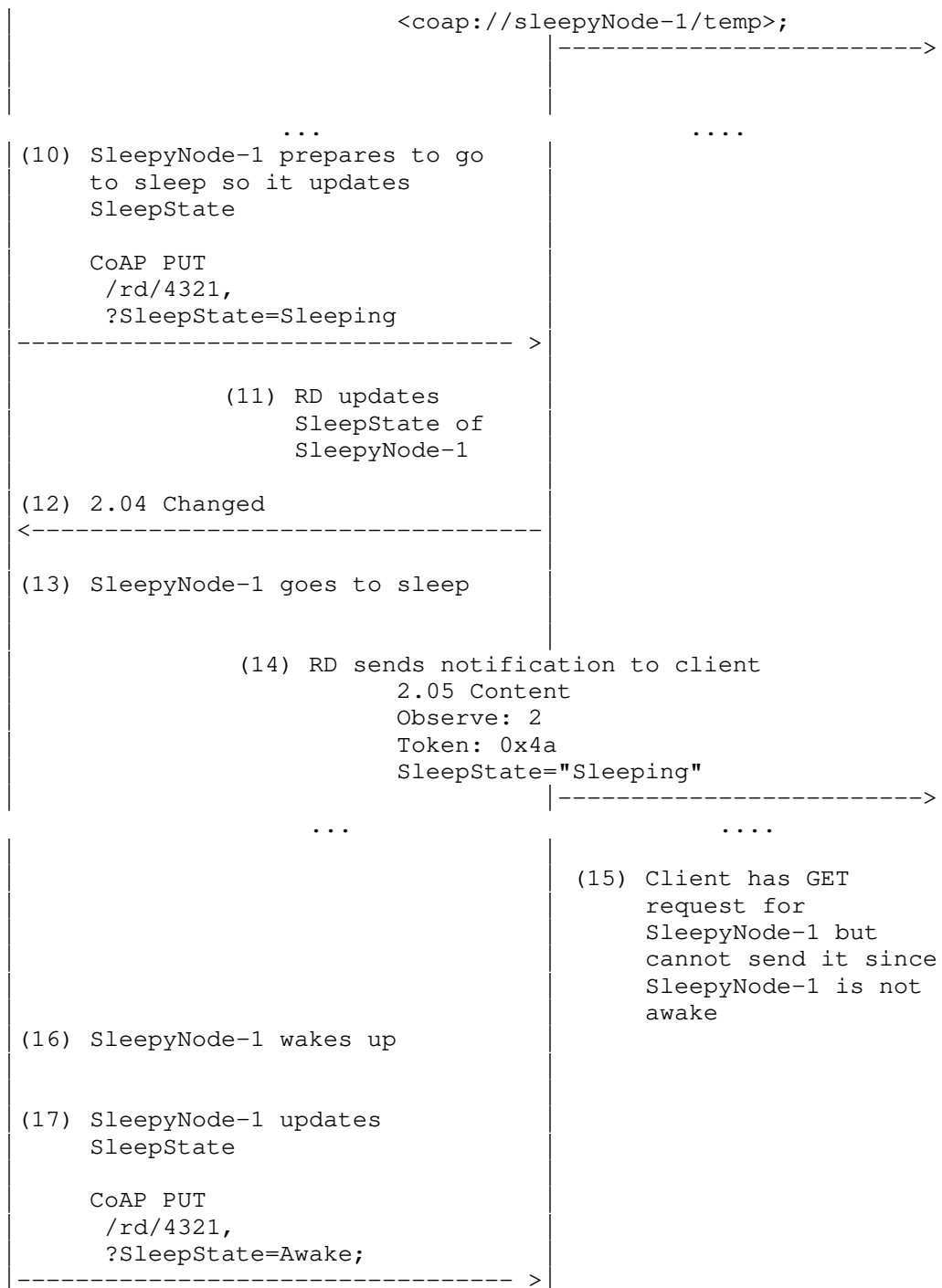
(16)-(19) SleepyNode-1 wakes up and updates the SleepState in the RD.

(20)-(21) RD notifies the client through previously established observe relationship.

(22)-(23) Client sends GET request for temperature sensors and successfully receives the content as SleepyNode-1 is awake.







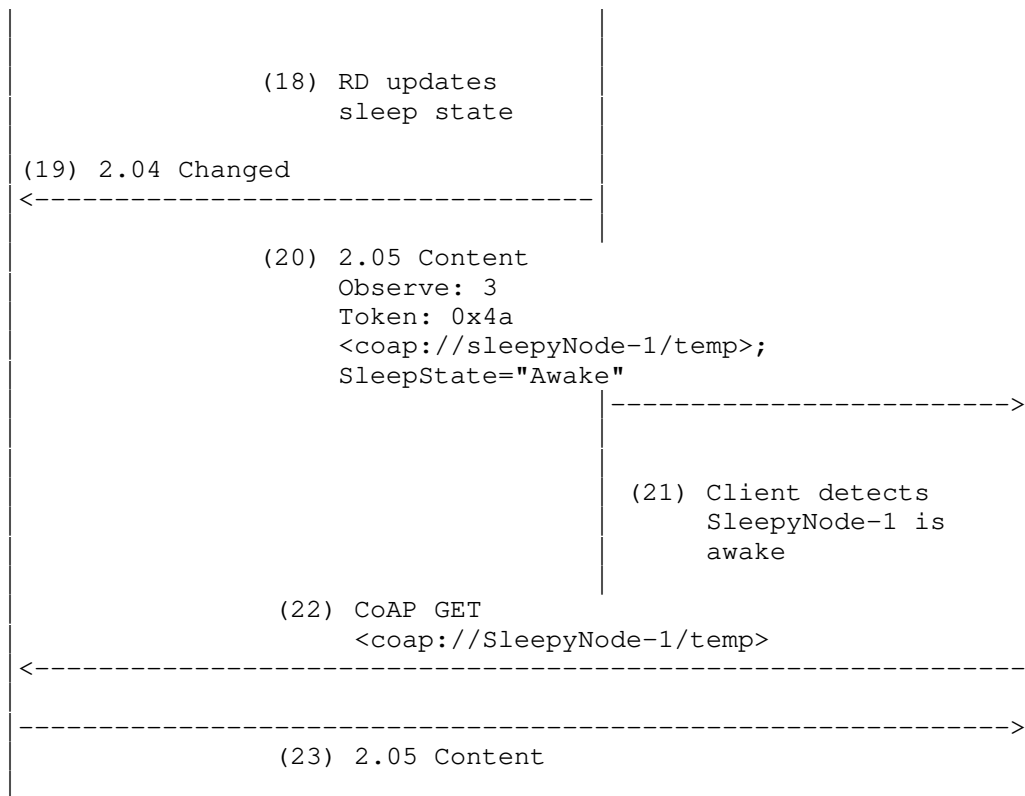


Figure 2: Asynchronous Resource Directory Based Sleep Tracking

3.4. RD Caching Proxy

It would be useful for an RD to be able to indicate which proxy performs caching for Sleepy CoAP nodes (see Section 2). This would be done through a new RD "CachingProxy" attribute for each device (similar to the attributes defined in Section 3.1):

- o An RD may be co-located with a proxy that performs caching for CoAP nodes. In this case, the RD automatically adds itself to each CachingProxy entry.
- o The sleepy node itself could suggest the CachingProxy if it is peered to a specific proxy.

This parameter would be added to the RD URI Template Variables defined in [I-D.shelby-core-resource-directory].

4. Acknowledgements

Thanks to Thomas Fossati, Salvatore Loreto, and Zach Shelby for valuable discussions and feedback on this document.

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

TBD. (All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.)

7. References

7.1. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-12 (work in progress), October 2012.

[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-06 (work in progress),
September 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

7.2. Informative References

[I-D.shelby-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-shelby-core-resource-directory-04 (work in progress), July 2012.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552,

July 2003.

Author's Address

Akbar Rahman
InterDigital Communications, LLC
Montreal, Quebec H3A 3G4
Canada

Phone: +1-514-585-0761
Email: akbar.rahman@interdigital.com

CORE WG
Internet-Draft
Intended status: Informational
Expires: April 24, 2013

A. Rahman
InterDigital Communications, LLC
T. Fossati
KoanLogic
S. Loreto
Ericsson
M. Vial
Schneider-Electric
October 21, 2012

Sleepy Devices in CoAP - Problem Statement
draft-rahman-core-sleepy-problem-statement-01

Abstract

This document analyzes the COAP protocol issues related to sleeping devices. The only goal of this document is to trigger discussions in the CORE WG so that all relevant considerations for sleeping devices are taken into account when designing CoAP.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 24, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Terminology and Conventions	3
2. What is a Sleeping Device?	4
2.1. Sleeping behaviors.	4
2.2. Different Sleep Modes	5
2.2.1. Always-On	5
2.2.2. Intermittent Presence	5
3. Assumptions	5
4. Objectives	6
5. Acknowledgements	6
6. IANA Considerations	6
7. Security Considerations	7
8. References	7
8.1. Normative References	7
8.2. Informative References	7
Authors' Addresses	7

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], [I-D.ietf-core-link-format], [I-D.ietf-core-observe], [I-D.shelby-core-resource-directory].

In addition, this document defines the following terminology:

Sleeping End Point (SEP): is a special kind of CoAP enabled device that spends a large amount of its lifetime disconnected from the network, mainly to save power, or just because it is downright unable to store the energy required for its functioning. It nonetheless owns and hosts a set of resources, and needs to make them available to the other participants in the same constrained RESTful environment. In this respect it has to devise and implement the mechanisms that allows to work around its limitation, and let its resources be accessible as if it were an usual, always connected, CoAP server.

Resource Delegation: is the transfer of control over the handling of a resource from an endpoint (the owner) to another (the deputy), without the actual ownership being relinquished.

The retention of ownership implies two things: first, that a genuine resource delegation cannot be recursive, and second, that it must always be entirely reversible, at any time the owning endpoint deems appropriate.

A Resource Delegation mechanism may comprise the transfer of the following information from the owner to the deputy endpoint:

- (a) complete or partial namespace,
- (b) one or more representations of the resource,
- (c) associated metadata,
- (d) allowed methods,
- (e) access control information,
- (f) temporal bounds of the delegation.

Said mechanism may also provide authentication to the parties involved in the delegation process.

2. What is a Sleeping Device?

A Sleeping device is a device able to cut power to unneeded subsystems and so significantly reduce battery consumption. Some Sleeping devices only cut power to the radio system while continuing to run normally the other ones. Other Sleeping devices are even more energy efficient being able to save the machine state in the RAM memory, putting the RAM into a minimum power state, and cutting power to all the other subsystems. Finally other Sleeping device are able to save the machine state on an hard disk and completely switching off themselves.

2.1. Sleeping behaviors.

In this section we discuss different behaviors and scenarios of sleeping nodes. Such behaviors can affect the design of applications (such as CoAP) and network topologies (such as proxies and caching).

Sleeping nodes can have various sleeping patterns. Sleep patterns can be predictable or totally unpredictable. For example, some nodes sleep at a fixed interval or upon certain triggers. Some nodes may sleep at irregular time intervals, or switch to sleep mode spontaneously. Some nodes stay in sleep mode and only wake up upon certain event triggers. A network may thus not be well aware of the sleeping state of a node at a given time.

The duration of sleeping mode also varies largely, possibly from a few seconds to days. From the perspective of applications, it may not be affected by the sleeping period if it is very short. For example, a HTTP/TCP connection may still work (even if sub-optimally) if the sleep cycle is much shorter than the TCP retransmission timer. In contrast, if the sleeping period of a node exceeds a certain threshold it can impact an application. This threshold however, can be difficult to predict and often can vary from device to device and network to network. For example, this threshold can be very dependent on the topology of a constrained network especially for the case where a multi-hop path consists of multiple sleeping nodes. For this case, the cumulative effect of multiple sleeping nodes must be considered.

The network topology also affects how to handle sleeping nodes. For example, in a star shaped network, a proxy node (assuming to be not a sleeping node) can cache for the sleeping nodes within the network in a centralized manner. However, in a P2P or mesh network, especially when multi-hops are involved, caching can be difficult and delivering of messages can be largely delayed due to nodes' sleeping cycles. In this case distributed proxying and caching at intermediate nodes within the network (rather than just a single node such as the border

node or sink) may make sense, if intermediate nodes are not sleeping nodes and have adequate resources to support caching.

2.2. Different Sleep Modes

2.2.1. Always-On

Any sleep is so short that it is invisible to L3 and upper which gives the illusion of the sleepy node of being always on => usual Server Model can be efficiently used.

2.2.2. Intermittent Presence

Long and possibly non pre-determined sleep periods (more than 1 sec, but typically in the order of minutes or hours) => Server Model not working anymore. SEP state must be handled by other mechanisms.

3. Assumptions

The characteristics of SEPs varies widely. Some may be cheap, rudimentary widgets with very limited computational and storage capabilities; other can be more functional devices yet in need to save energy since they have to be in operation for a long period while battery powered.

This great variance implies that a fair number of often contradictory assumptions must be taken into consideration, and carefully weighted, when designing a comprehensive solution for the problem. For example:

- o Is SEP able to maintain soft state ?
- o Is SEP sleep/awake scheduling predictable ?
- o Is SEP able to handle bidirectional communication ?

Luckily, and by definition, it can be assumed that all the SEPs participating in a CoRE domain share a (realistically limited) subset of the REST principles. At the very least we will assume a SEP understands and implements:

- o the concept of information resource and its representational state;
- o the semantics and syntax of CoAP URIs;

- o the semantics associated with the methods PUT or POST, and DELETE; in a way that is conformant with the CoAP protocol. This will provide the common ground on which to build their integration into the hosting CoRE domain.

4. Objectives

The CORE WG aim is to design a solution that, leveraging on the existing CoAP features and its REST architecture, allows SEP devices to be easily and smoothly integrated within any CoRE domain together with the all the other CoAP enabled devices.

The ideal solution should:

- o Make the set of resource owned and hosted by any SEP available to all the other participants, in the same constrained RESTful environment, without making any assumption on the presence of specific or special entities neither on the network topology.
- o Provide the possibility to use Client or Observer Model to access resources owned and hosted by a SEP.
- o Allow the (Secure) delegation of resource handling while retaining ownership.
- o Minimize the configuration needs to bootstrap a SEP within an existing CoRE domain.
- o Maximize the integration with base CoRE Features (i.e. Resource Discovery, Multicast, Observer, Block).
- o Reuse already available CoAP mechanisms as much as possible.

5. Acknowledgements

TBD.

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., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-12 (work in progress), October 2012.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format",
draft-ietf-core-link-format-14 (work in progress),
June 2012.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP",
draft-ietf-core-observe-06 (work in progress),
September 2012.

[I-D.shelby-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource
Directory", draft-shelby-core-resource-directory-04 (work
in progress), July 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.

8.2. Informative References

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC
Text on Security Considerations", BCP 72, RFC 3552,
July 2003.

Authors' Addresses

Akbar Rahman
InterDigital Communications, LLC
Montreal, Quebec H3A 3G4
Canada

Phone: +1-514-585-0761
Email: akbar.rahman@interdigital.com

Thomas Fossati
KoanLogic

Email: tho@koanlogic.com

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Salvatore.Loreto@ericsson.com

Matthieu Vial
Schneider-Electric

Email: matthieu.vial@schneider-electric.com

Core
Internet-Draft
Intended status: Standards Track
Expires: January 11, 2013

B. Sarikaya
Huawei USA
Y. Ohba
Toshiba
R. Moskowitz
Verizon Business Systems
Z. Cao
China Mobile
R. Cragie
Pacific Gas and Electric
July 10, 2012

Security Bootstrapping Solution for Resource-Constrained Devices
draft-sarikaya-core-sbootstrapping-05

Abstract

This document describes how to initially configure the network of resource constrained nodes securely, a.k.a., security bootstrapping. Bootstrapping architecture, communication channel and bootstrap security methods are described. System level objectives for security bootstrapping are stated. Bootstrapping solution is based on EAP-TLS authentication with the use of raw public keys as certificates.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 11, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents
(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
2. Bootstrapping Architecture	4
2.1. Areas of Booststrapping	4
2.2. Architecture	6
3. Bootstrap Security Method	7
3.1. None	7
3.2. Asymmetric with User Authentication, Followed by Symmetric	7
3.3. Asymmetric with Certificate Authority, Followed by Symmetric	7
3.4. Cryptographically Generated Address Based Address Ownership Verification	7
4. Secure Bootstrapping System Level Objectives	8
5. Secure Bootstrapping Solution using Raw Public Keys	9
6. Security Considerations	10
7. IANA Considerations	11
8. Contributors	12
9. Acknowledgements	12
10. References	12
10.1. Normative References	12
10.2. Informative References	13
Appendix A. Examples of Node Configuration	15
A.1. Smart Energy	15
A.1.1. Initial Meter Installation	15
A.1.2. Home Expansions	15
A.2. Consumer Products	15
A.2.1. Connecting DVD Remote to DVD Player	16
A.2.2. Adding a TV to a network with a DVD player and remote	16
A.2.3. Providing GPS Location Data	16
A.3. Commercial Building Automation	16
A.3.1. Light Installation	16
Appendix B. Example Exchanges	16
B.1. Smart Energy: Meter Manufacture	16
B.2. Smart Energy: Meter Installation	16
B.3. Smart Energy: Home Expansion	16
B.4. Consumer: Connecting DVD Remote to DVD Player	17
B.5. Consumer: Adding a TV to a network with a DVD player and remote	18
Appendix C. EAP, PANA, HIP-DEX and 802.1X	20
C.1. EAP Authentication Framework	20
C.2. PANA	22
C.3. HIP-DEX	23
C.4. 802.1X	24
Authors' Addresses	26

1. Introduction

Bootstrapping is any processing required before the network can operate. Typically this will require a number of settings to be transferred between nodes at all layers. This could include anything from link-layer information (i.e., wireless channels, link-layer encryption keys) to application-layer information (i.e., network names, application encryption keys).

Bootstrapping is complete when settings have been securely transferred prior to normal operation in the network.

The bootstrapping problem is not specific to any MAC or PHY. This problem exists across any two nodes which have no previous knowledge of each other. In particular, this problem is complicated when the nodes are resource-constrained and may not have an advanced user interface. The IETF is instrumental in defining standards which will be used by The Internet of Things (IOT). Ensuring these standards can be used across nodes and networks requires some form of bootstrapping which the sensor nodes can use [ROMER04].

Existing standards will be used as much as possible in this document. The method proposed here should work across many different underlying layers. It could be used to allow two nodes on the same physical network to join at the physical layer, or allow two nodes on an incompatible physical network to join at the IPv6 layer.

The document continues in Section 2 on bootstrapping architecture, in Section 3 on allowable security methods in bootstrapping, in Section 4 on system level objectives of secure bootstrapping, and Section 5 on secure bootstrapping solution. Appendix A is on examples of node configuration, Appendix B is on samples of exchanges during bootstrapping and Appendix C is on the protocols used to carry EAP in link layer/IP layer including HIP-DEX.

2. Bootstrapping Architecture

2.1. Areas of Bootstrapping

In order to provide a flexible architecture, the bootstrapping method is split into five distinct areas and two distinct phases. The five areas are a 'user interface', 'bootstrap profile', 'security method', 'bootstrap protocol', and the 'communications channel'.

The phases are provisioning phase and bootstrapping phase. In the provisioning phase, statically configured parameters (e.g., certificates) needed for the bootstrapping phase is provisioned. In

the bootstrapping phase, dynamically configured information is set up using the statically configured information provided in the provisioning phase.

The user interface provides both user input and user output. Simple nodes may only have a push-button and LED, more complex nodes may have a graphical display and keyboard. The user interface (which could be implemented as network management software graphical user interface running at the remote end) provides interaction between the user and bootstrapping methods. The user interface would be used during bootstrapping as an OOB channel. It may also be used to specify bootstrapping policies.

The user interface provides the interaction between the user and the bootstrap protocol. The user interface will vary depending on the capabilities of the node. Examples might include a push-button and LED on simple nodes, to full-blown graphical user interfaces. Note that a 'bootstrapping tool' used to initially deploy a network is just a special user interface. This allows a very uniform protocol in deployment and use of networks.

User interface is out-of-scope and will not be further discussed.

Two nodes communicate through some channel. For our purposes this is split into the 'control channel' and 'data channel'. The control channel is used for the bootstrap protocol, and the data channel is used during normal network operation. A node may support multiple control or data channels. When the control and data channels are the same, the bootstrapping is done In Band (IB). When the control and data channels are different, the bootstrapping is performed Out Of Band (OOB). An 802.15.4 network for instance would use an 802.15.4 control channel for IB bootstrapping, but a control channel of perhaps IrDA or USB for OOB bootstrapping.

The 'bootstrap profile', i.e. statically configured parameters during the provisioning phase, defines what information should be exchanged during the process. A single node may run the protocol multiple times with different profiles. If the user wishes to associate a new lightswitch, the protocol is first run with the '802.15.4 Wireless Profile', through which it learns the channel and PAN-ID. The node then runs a 'Security Exchange Profile' to learn the needed encryption keys. Finally it runs a 'Lightswitch Association Profile' through which it learns which light to associate with.

An example of the 'bootstrap profile' attribute is the 'administrative domain name'. 'Bootstrap profiles' are required to be modified when the corresponding administrative domains are changed, a.k.a. recommissioning. In recommissioning, the domains are

administratively repartitioned and nodes are therefore recommissioned.

The 'security method' defines supported security methods for bootstrapping. The supported security methods will depend on the control channel and bootstrap profile. In one node if the control channel is secure, then a simple clear-text security method is supported. For example when a physical connection between two nodes is used, the control channel is considered secure. However when the control channel is not secure, this clear-text security method is not supported. The 'bootstrap profile' additionally defines allowed security methods. Higher security nodes may outlaw ever performing a clear-text exchange, even if the control channel is deemed secure.

The 'bootstrap protocol' defines the actual messages exchanged during bootstrapping. The messages are used to transfer between nodes data, node information, and network state. The selected security method runs on top of the control channel, such as EAP-GPSK etc.

2.2. Architecture

Security bootstrapping architecture is structured in a hierarchy of nodes going from the least resource constraint to the most resource constraint. At the top there is a root node. The root node is called Coordinator or Trust Center in Zigbee and 6LoWPAN Border Router (6LBR) in 6LoWPAN ND.

At the next level there are interior Routers. Routers are able to run a routing protocol between other routers and the root. Routers are called 6LoWPAN Routers (6BR) in 6LoWPAN ND.

At the lowest level there are the nodes. The nodes do not run a routing protocol. They can connect to the nearest router over a single radio link. The nodes are called End Devices in Zigbee and hosts in 6LoWPAN ND.

Routers first join the network as a node and go through security bootstrapping operations in order to create a Master Session Key (MSK). Next, routers execute routing protocol, e.g. [RFC6550] specific steps to create session keys with their neighbors and to establish upstream and downstream next hop parents.

At each node hierarchy level described above, there are lower-layer and higher-layer protocols to bootstrap their ciphering keys, where the lower-layer refers to layers below IP layer including IEEE 802.15.4 MAC layer and LoWPAN adaptation layer and the higher-layer refers to IP layer and the above. In general, required bootstrapping procedures depend on the bootstrapping protocols to use. Section

Section 5 describes the bootstrapping procedures where EAP (Extensible Authentication Protocol) [RFC3748] and other protocols are used as the bootstrapping protocols.

3. Bootstrap Security Method

The bootstrap security method defines allowable security methods. A node may choose to support or use a subset of these methods. This is NOT the security architecture used for the application, but only the security used during bootstrapping. Typically some high-security method is used to generate a shared secret, which then switches to simpler symmetric encryption to secure the actual bootstrapping channel. The techniques negotiated should take advantage of hardware resources available, such as hardware encryption accelerators on an end node.

3.1. None

This is the simplest security method. No encryption or authentication is provided, messages are exchanged completely in clear-text. It is assumed some other layer provides security, such as a physical connection between devices.

3.2. Asymmetric with User Authentication, Followed by Symmetric

A Diffie-Hellman style key exchange is used to generate a shared secret. The authentication will be provided by the user, by confirming cryptographic signatures between two devices. With the shared secret generated through the DH, some symmetric encryption is used to secure the actual bootstrapping channel.

3.3. Asymmetric with Certificate Authority, Followed by Symmetric

Public key exchanges are used (aka: DH again), but with a Certificate Authority. Once a shared secret exists, symmetric encryption is used to secure the actual bootstrapping channel.

3.4. Cryptographically Generated Address Based Address Ownership Verification

A node may generate the global unique address using different techniques other than the stateless address autoconfiguration. For example, Cryptographically Generated Addresses (CGA) [RFC3972] is a type of global unique address that can be used to verify the address ownership. When the node uses CGA, it MUST execute SeND protocol [RFC3971]. In a 6LOWPAN network, a modified 6LOWPAN ND Protocol [I-D.ietf-6lowpan-nd] must be executed between the node and the edge

router.

4. Secure Bootstrapping System Level Objectives

Authentication/ reauthentication: nodes joining the network MUST at the first place authenticate to the trust center. In order to achieve secure multi-hop routing, the node MUST authenticate to its upstream and downstream neighbors. A bootstrapping solution MUST support re-authentication of resource-constrained devices and re-keying of dynamically generated keys.

Data Confidentiality: the data communication between two endpoints MAY be encrypted using the derived key, avoiding being eavesdropped by a non-trusted third part.

Data Integrity: the data communication between two endpoints MUST NOT be altered by some intermediate nodes. The nodes should be able to detect the non-integral data.

Keys and key freshness: the keys used for data communication MUST have a lifetime, in order to keep their freshness. A bootstrapping solution MUST support both symmetric and asymmetric key authentication. If distribution of a key to be used for a resource-constrained device is required, a bootstrapping solution MUST support secure key distribution to prevent the key from eavesdropping, alternation and replay attacks.

Multi domain support: A bootstrapping solution MUST be able to allow resource-constrained devices that may be subscribed to different administrative domains to be connected to the same access network at the same time.

Multi domain support: A bootstrapping solution MUST be able to allow resource-constrained devices to be recommissioned. Recommissioning a device is defined to be (1) an resource-constrained device is administratively switched to a different domain, or (2) acting a new role with a different function set, or (3) both administrative domain and function set are modified.

Identities: A bootstrapping solution MUST be able to allow a resource-constrained device to use various types of identities used for authentication, including device identities, user identities or combinations of different types of identities. Also a bootstrapping solution MUST be able to allow a resource-constrained device to change its identities used for authentication over time.

Authentication infrastructure: A bootstrapping solution MUST be able

to operate with or without an authentication infrastructure.

5. Secure Bootstrapping Solution using Raw Public Keys

When a new resource-constrained device is deployed, it configures its global unique IPv6 address first. This is done by 6LoWPAN Neighbor Discovery (6LoWPAN-ND)'s Router Solicitation/Router Advertisement message exchange [I-D.ietf-6lowpan-nd]. The newly generated IPv6 address can not be used until the joining device is authenticated and securely joins the network. After the authentication, the joining device receives the current group key of the network, so that the IPv6 registration and further communication can be protected by the link layer ciphering e.g. 802.15.4, then it can start using its global unique IPv6 address for communication.

For authentication, Extensible Authentication Protocol (EAP) MUST be used. EAP authentication framework is explained in Appendix C.1.

The EAP method EAP-TLS [RFC5216] can be used for the resource-constrained device authentication. Instead of X.509 certificates, raw public key of the device MUST be used. EAP-TLS is executed between the joining device and the AAA server which acts as the Authentication Server (AS). After a successful authentication, the device and the AAA server establish a Master Session Key (MSK), and then the AAA server exports the MSK to the authenticator. Upon receipt of the MSK, the authenticator distributes the group key to the joining device within the authentication success message. The group key is encrypted by a Key Encryption Key derived from the MSK.

The resource-constrained device initiates the EAP authentication process by sending a message of initiation to the authenticator, i.e. the root node or 6LBR. The root node requests the identity from the device by sending an EAP-Request/Identity packet. The device replies with an EAP-Response/Identity containing the device's ID. The identity information includes the device's network access ID (NAI). When the root node receives NAI of the device, it sends the identity information to the AS.

The AS starts the EAP-TLS authentication process by sending a EAP-TLS/Start packet which is an EAP-Request packet with EAP-Type=EAP-TLS to the device. The device generates a client random number and responds with an EAP-Response/TLS-Client-Hello message which contains the TLS version, a client random number, a set of cipher suites. Only one cipher suite MUST be offered in Client-Hello message with RC4-SHA1. EAP-Response packet MUST have the EAP- Type value set at EAP-TLS.

The device MUST add an extension of type `cert_type` defined in [I-D.ietf-tls-oob-pubkey] to Client-Hello message. This type MUST be set to `RawPublicKey`.

Upon receipt of Client Hello, if the AS supports raw public key extension, it generates a server random number, a new session ID, and includes only the `SubjectPublicKeyInfo` part of the certificate, rather than the whole certificate in the Certificate message and then sends them to the device with an EAP-Request/TLS-Server-Hello message.

With the client and server random number, the device generates a `pre_master_secret`, then sends it in Client-Key-Exchange field of EAP-Response/TLS-Client-Finished message to the AS.

The AS derives the Master Session Key (MSK) and replies with EAP-Request/TLS-Server-Finished message. The SE device also derives the MSK after receiving the Server Finished and acknowledges with EAP-Response/EAP-TLS message.

The AS then exports the MSK to the authenticator in RADIUS Access-Accept message, the authenticator subsequently sends the EAP-Success message to the SE device. The AS MUST send the group key in this message and the EAP-TLS ends.

When a device is not a direct neighbor of the authenticator, its parent node MUST act as relay. Different EAP encapsulation protocols have different mechanisms for the relay function, for details, see Appendix C.

6. Security Considerations

When security bootstrapping resource constraint nodes is undertaken, several attacks are possible and security bootstrapping methods described in this document do not protect the nodes against such attacks. These attacks are similar to the ones described in [RFC3971] and mainly stem from unsecured link layer. Link layer must be secured on each node before the node can begin security bootstrapping.

If a bootstrapping protocol does not rely on a pre-shared key for peer authentication, it must rely on an online or offline third-party (e.g., an authentication server, a key distribution center in Kerberos, a certification authority in PKI, a private key generator in ID-based cryptography and so on) to prevent man-in-the-middle attacks during peer authentication. Depending on use cases, a resource-constrained device may not always have access to an online

third-party for peer authentication.

Depending on use cases, a bootstrapping protocol may deal with authorization separately from authentication in terms of timing and signaling path. For example, two resource-constrained devices A and B may perform mutual authentication using authentication credentials provided by an offline third-party X whereas resource-constrained device A obtains authorization for running a particular application with resource-constrained device B from an online third-party Y before or after the authentication. In some use cases, authentication and authorization are tightly coupled, e.g., successful authentication also means successful authorization. A bootstrapping protocol supports various types of authentication and authorization or different bootstrapping protocols may be used for different types of authentication and authorization.

If authorization information includes cryptographic keys, a special care must be taken for dealing with the keys, e.g., guidelines for AAA-based key management are described in [RFC4962]. A recommissioning use case may require revocation and re-installation of authentication credentials (i.e., a certificate or a shared secret and identity information, etc.) to a large number of resource-constrained devices that are already deployed. Re-installation of authentication credentials must be as secure as the initial installation regardless of whether the re-installation is done manually or automatically.

If resource-constrained devices use a multicast group key for peer authentication or message authentication or encryption, the group key must be securely distributed to the current members of the group for both initial key distribution and key update. Protocols designed for group key management such as GSAKMP [RFC4535], GDOI [RFC3547] and MIKEY [RFC3830] may be used for group key distribution. Alternatively, key wrap attributes for securely encapsulating group key may be defined in network access authentication protocols such as PANA [RFC5191] and EAP-TTLSv0 [RFC5281]. Those protocols use an end-to-end, point-to-point communication channel with a pair-wise security association between a key distribution center and each key recipient. Further considerations may be needed for more efficient group key management to support a large number of resource-constrained devices.

7. IANA Considerations

This memo includes no request to IANA.

8. Contributors

The people listed below have made significant text contributions to this document.

Colin O'Flynn

colin.oflynn@atmel.com

9. Acknowledgements

Special thanks also to Rene Struik, Carsten Borman, Gary Yang, Alper Yegin and Tero Kivinen for their comments that helped us improve the writing. Discussions with Hannes Tschofenig have been very inspiring.

10. References

10.1. Normative References

- [802.15.4] IEEE Std 802.15.4-2006, "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)", September 2006.
- [I-D.ietf-tls-oob-pubkey] Wouters, P., Gilmore, J., Weiler, S., Kivinen, T., and H. Tschofenig, "TLS Out-of-Band Public Key Validation", draft-ietf-tls-oob-pubkey-03 (work in progress), April 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowitz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.
- [RFC5191] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.

- [RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, March 2008.
- [RFC5548] Dohler, M., Watteyne, T., Winter, T., and D. Barthel, "Routing Requirements for Urban Low-Power and Lossy Networks", RFC 5548, May 2009.
- [RFC5673] Pister, K., Thubert, P., Dwars, S., and T. Phinney, "Industrial Routing Requirements in Low-Power and Lossy Networks", RFC 5673, October 2009.

10.2. Informative References

- [802.1x] IEEE Std 802.1X-2010, "IEEE 802.1X Port-Based Network Access Control", February 2010.
- [C1222] American National Standard, "Protocol Specification For Interfacing to Data Communication Networks", ANSI C12.22-2008, 2008.
- [I-D.ietf-6lowpan-nd] Shelby, Z., Chakrabarti, S., and E. Nordmark, "Neighbor Discovery Optimization for Low Power and Lossy Networks (6LoWPAN)", draft-ietf-6lowpan-nd-18 (work in progress), October 2011.
- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-10 (work in progress), June 2012.
- [I-D.moskowitz-hip-rg-dex] Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-06 (work in progress), May 2012.
- [I-D.ohba-pana-relay] Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA) Relay Element", draft-ohba-pana-relay-03 (work in progress), February 2011.
- [NISTIR7628VOL1] The Smart Grid Interoperability Panel - Cyber Security Working Group, "Guidelines for Smart Grid Cyber Security: Vol. 1, Smart Grid Cyber Security Strategy, Architecture, and High-Level Requirements", NISTIR 7628, vol. 1, 2010.

- [RFC3547] Baugher, M., Weis, B., Hardjono, T., and H. Harney, "The Group Domain of Interpretation", RFC 3547, July 2003.
- [RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.
- [RFC3971] Arkko, J., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, March 2005.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.
- [RFC4423] Moskowitz, R. and P. Nikander, "Host Identity Protocol (HIP) Architecture", RFC 4423, May 2006.
- [RFC4535] Harney, H., Meth, U., Colegrove, A., and G. Gross, "GSAKMP: Group Secure Association Key Management Protocol", RFC 4535, June 2006.
- [RFC4962] Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, July 2007.
- [RFC5204] Laganier, J. and L. Eggert, "Host Identity Protocol (HIP) Rendezvous Extension", RFC 5204, April 2008.
- [RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.
- [RFC5281] Funk, P. and S. Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, August 2008.
- [RFC5295] Salowey, J., Dondeti, L., Narayanan, V., and M. Nakhjiri, "Specification for the Derivation of Root Keys from an Extended Master Session Key (EMSK)", RFC 5295, August 2008.
- [RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen,

"Internet Key Exchange Protocol Version 2 (IKEv2)",
RFC 5996, September 2010.

- [RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R.,
Levis, P., Pister, K., Struik, R., Vasseur, JP., and R.
Alexander, "RPL: IPv6 Routing Protocol for Low-Power and
Lossy Networks", RFC 6550, March 2012.
- [ROMER04] Romer, K. and F. Mattern, "The design space of wireless
sensor networks", IEEE Wireless Communications, vol. 11,
no. 6, pp. 54-61, December 2004.
- [SE2.0] ZigBee Alliance, "Smart Energy Profile 2.0 Technical
Requirements Document", April 2010.

Appendix A. Examples of Node Configuration

Before any detail on methods is explored, the following section will provide various examples this document could cover. Exact requirements will be brought forward in subsequent sections. For the reader's general understanding this section is placed to give an idea of an acceptable usage scenario.

A.1. Smart Energy

A.1.1. Initial Meter Installation

The meter is initially loaded with code and network keys through a physical interface at the factory. The meter is installed at a customers home, and configured by the installer through the backbone link (via GSM modem, etc). Both operations can be performed through methods defined herein.

A.1.2. Home Expansions

The user wishes to join a thermostat onto the network. They press a button on the thermostat, which enters join mode. They press a button on the smart meter, which allows nodes to join the network. The devices both have displays, so they display a certain number which the user verifies match on both devices. The thermostat has now securely joined the network.

A.2. Consumer Products

A.2.1. Connecting DVD Remote to DVD Player

The user pushes a join button on the DVD remote and DVD player. The devices find each other, and blink in unison to indicate to the user which two devices will join. The user presses the button to confirm this, and the two devices are now joined together.

A.2.2. Adding a TV to a network with a DVD player and remote

The user then presses the join button on the DVD player and TV. The devices again find each other and blink in unison, with the addition that the remote control also blinks to indicate it is present in the network.

A.2.3. Providing GPS Location Data

A user has a simple GPS receiver (that has no user interface) they wish to broadcast location data with. The user switches on their camera, and enters a PIN from the base of the GPS. The user can now view GPS information such as satellite health from their camera. In addition photos are automatically tagged with location information.

A.3. Commercial Building Automation

A.3.1. Light Installation

The electrician installs the light fixture. Each light has a barcode printed on it. They use a handheld barcode scanner tool, which acts as the commissioning tool. When they scan a barcode with the tool, the tool asks the electrician to enter some additional information such as light fixture location. The tool securely registers the light fixture on the network, along with setting parameters inside the light fixture.

Appendix B. Example Exchanges

The following details how the protocol handles certain conditions and situations through examples. Note that each example is a more detailed description of the examples in Appendix A.

B.1. Smart Energy: Meter Manufacture

B.2. Smart Energy: Meter Installation

B.3. Smart Energy: Home Expansion

B.4. Consumer: Connecting DVD Remote to DVD Player

Supported User Interface Profiles

Profile	DVD Player	Remote Control
none	Y	Y
simple	Y	Y
numerical	Y	N
alphanumeric	Y	N
Graphical	Y	N

Supported Bootstrap Transport Layers

Layer	DVD Player	Remote Control
Physical	Y	Y
802.15.4	Y	Y
IrDA	Y	N

Supported Security Methods

Method	DVD Player	Remote Control
None	Y	Y
EAP	Y	N
Asymmetric, User	Y	Y
Asymmetric, CA	Y	N

The DVD player and remote control have a number of ways in which they could be joined together. The remote control does not have any unique identifier printed on it, thus no pre-shared key can be identified. This leaves either an unsecure joining method, or some asymmetric security method.

The remote control has a button on it for 'join', as does the DVD player. The user pushes the button on the DVD player, and then pushes the button on the remote control. Based on the UI profile, this causes the following to occur:

- o DVD Player scans for existing network in advertise mode. Finding none, it starts a new network and that network enters advertise

mode.

- o The DVD remote scans for a network, and then finds the DVD player's network.
- o The devices generate a shared secret (ie: Diffie-Hellman), and both blink their LED in a unique pattern based on this shared secret.
- o The user confirms both devices are blinking the same pattern, as both LEDs are blinking in unison.
- o The DVD player displays 'JOIN OK' on it's LCD panel.

B.5. Consumer: Adding a TV to a network with a DVD player and remote

This network will have three devices: a TV, a DVD Player, and a Remote Control. The user will run the bootstrap protocol between the TV and Remote Control in this example, although it could also be run between the TV and DVD player.

Supported User Interface Profiles

Profile	TV	Remote Control
none	Y	Y
simple	Y	Y
numerical	Y	N
alphanumeric	Y	N
Graphical	Y	N

Supported Bootstrap Transport Layers

Layer	TV	Remote Control
Physical	Y	Y
802.15.4	Y	Y
IrDA	Y	N

Supported Security Methods

Method	TV	Remote Control
None	Y	Y
EAP-GPSK	Y	N
Asymmetric, User	Y	Y
Asymmetric, CA	Y	N

The TV and remote control have a number of ways in which they could be joined together. The remote control does not have any unique identifier printed on it, thus no pre-shared key can be identified. This leaves either an unsecure joining method, or some asymmetric security method.

The remote control has a button on it for 'join', as does the TV. In this example two sequence will be considered: where the TV button is pressed first, and where the remote control button is pressed first.

If the TV join button is pressed first:

- o TV scans for existing networks in advertise mode. Finding none, it starts a new network and that network enters advertise mode.
- o The remote scans for a network, and then finds the TV's network.
- o The remote informs the TV it is on an existing network, and thus will require the TV to join this network.
- o The devices generate a shared secret, and both blink their LED in a unique pattern.
- o The DVD player in addition blinks, so the user is informed that if they confirm the join action the resulting network will have all three devices in it.
- o The user confirms both devices are blinking the same pattern, as both LEDs are blinking in unison.
- o The TV displays 'JOIN OK' onscreen, along with any information about the network it just joined.

If the remote control join button is pressed first:

- o Remote control scans for existing networks in advertise mode. Finding none, it advertises it's network.

- o The TV scans for a network, and then finds the remote control's network.
- o The devices generate a shared secret, and both blink their LED in a unique pattern.
- o The DVD player in addition blinks, so the user is informed that if they confirm the join action the resulting network will have all three devices in it.
- o The user confirms both devices are blinking the same pattern, as both LEDs are blinking in unison.
- o The TV displays 'JOIN OK' onscreen, along with any information about the network it just joined.

Appendix C. EAP, PANA, HIP-DEX and 802.1X

C.1. EAP Authentication Framework

EAP is an authentication protocol that supports a number of authentication algorithms called EAP methods [RFC5247]. EAP messages can be transported in different layers: using 802.1X, EAP messages are carried in Layer 2 and using PANA in IP or Layer 3 between EAP peer and the authenticator. EAP messages between the authenticator and authentication server are carried using AAA protocols (RADIUS or Diameter). The associated AAA server address of each resource-constrained device is assigned by the domain administrator. In the recommissioning case, another AAA server is reassigned to devices by the domain administrator if the device is switched to another domain.

At the end of a successful EAP method execution a master session key (MSK) is generated at both the EAP peer and EAP server. Authenticator receives MSK from EAP server at the end of EAP method execution using key transport. MSK is used in deriving a session key between the node and the authenticator using a protocol called secure association protocol (SAP). Derivation of the session key terminates bootstrapping of a node.

Additional keying material derived between EAP client and server that is exported by the EAP method is called Extended Master Session Key (EMSK). EMSK is not used in session key derivation but it could be used for the needs of other applications in higher layer protocols.

In the architecture introduced in Section 2.2 the node or router is the peer and the root is the authenticator. When the supplicant and authenticator are one hop away the authenticator can be reached

directly. However, this is rarely the case. In other cases the authenticator authenticates neighboring supplicants first. The router nodes that are authenticated become relay authenticators in the next phase and they relay authentication messages from the supplicants to the authenticator and vice versa. This continues until all nodes are authenticated.

EAP is a lock-step protocol, i.e. it executes in pairs of EAP-Request messages sent by the server and EAP-Response messages sent by the peer. At the end, the server indicates the status of authentication, usually by EAP-Success message which also carries the MSK. The first EAP-Request/Response pair is used for the server to request the identity and the peer to provide it. In the other pairs of EAP exchanges EAP method is executed.

Several EAP methods have been standardized each for different purposes. To authenticate devices with certificates, EAP Transport Layer Security (TLS) v1.2 specified in [RFC5216] which supports certificate-based mutual authentication is used.

Zigbee Alliance's Smart Energy Profile 2.0 Application Protocol Specification [SE2.0] mandates each device to be factory programmed with a certificate. The certificate is bound to a unique network ID, e.g. the device's MAC address or EUI-64 address. During EAP-Identity exchange the EAP peer provides its EUI-64 address as an identity to EAP server. This enables the server to validate the device certificate.

There are three bootstrapping scenarios using EAP.

1. Use of EAP for bootstrapping link-layer security.

In this case, EAP is used for network access authentication to bootstrap link-layer ciphering. Security for higher-layer (i.e., IP layer and above) protocols is bootstrapped from an IB or OOB mechanism other than EAP.

2. Use of EAP for bootstrapping higher-layer security.

In this case, EAP is used as an OOB mechanism for higher-layer authentication to bootstrap ciphering keys for one or more higher-layer protocols independently of network access authentication. When bootstrapping Constrained Application Protocol (CoAP) security with DTLS protection, a PSK (Pre-Shared Key) credential in the combined usage of DTLS (Datagram Transport Layer Security) [RFC4347] and PSK mode of TLS [RFC4279] is derived from EAP key material and DTLS ciphering keys are generated as a result of a successful DTLS handshake. Similarly,

when bootstrapping CoAP security with IPsec ESP protection, a PSK credential of IKEv2 [RFC5996] is derived from EAP key material and IPsec ESP ciphering keys are generated as a result of a successful IKEv2 handshake.

The ability to bootstrap multiple higher-layer protocols from a single execution of PANA authentication is important to save the computational resources for resource-constrained devices especially where public-key based authentication is used.

3. Use of EAP for bootstrapping both link-layer and higher-layer security.

This case is the combination of the other two cases, and the most optimized way for bootstrapping resource-constrained devices. This case is only applicable where both the network access authentication and the higher-layer authentication use the same authentication server with the same authentication credentials.

The second and third scenarios are generally referred to as Single Sign-On in Section 4.2.2.2 of [NISTIR7628VOL1], where the root keys for higher-layer protocols can be derived from EAP EMSK (Extended Master Session Key) as an USRK (Usage-Specific Root Key) [RFC5295].

C.2. PANA

PANA (Protocol for carrying Authentication for Network Access) [RFC5191] defines an EAP transport over UDP where a PANA Client (PaC) is an EAP peer and a PANA Authentication Agent (PAA) is an EAP authenticator.

PANA can achieve the authentication, key freshness and data confidentiality objectives of security bootstrapping.

Multi domain operation is intrinsically supported due to the use of EAP and AAA.

Even though PANA architecture consisting of PaC, PAA and AAA Server is generic enough to be used in security bootstrapping, the architecture introduced in Section 2.2 requires a new element called PANA Relay Element (PRE). PRE is needed to enable PANA messaging between a PaC and PAA because the two nodes cannot reach each other by means of regular IP routing since only a link-local IPv6 address can be used by PaC prior to the completion of a successful authentication.

PRE which is one hop away from PaC receives PANA messages and relays the message contents (payload) by encapsulating it in a message

parameter called Attribute Value Pair (AVP). PRE also needs to send header contents such as PaC's IP address and UDP port number in a different AVP. PRE has IP routing established with PAA which could be several hops away. PAA sends its reply messages in which the payload is encapsulated in an AVP. It also adds the AVP containing PaC's IP address and UDP port number. PRE creates a link local PDU using these AVPs and sends it to PaC [I-D.ohba-pana-relay].

The requirements for the use of PANA as a bootstrapping protocol can be stated as follows:

- o A new entity called PANA Relay Element may be added to the PANA architecture. Behaviour of PANA Relay Element needs to be defined. New AVPs needed for PANA Relay Element operation for properly relaying messages from the client to the authenticator and vice versa are required to be specified.
- o An extension to PANA to securely distribute keys from the PANA Authentication Agent to the PANA Client using AES Key Wrap with Padding algorithm needs to be defined. This is needed in order to use PANA for multicast group key distribution.

C.3. HIP-DEX

[RFC4423] introduces the Host Identity Protocol (HIP) where the Host Identity (HI) is a Cryptographic key (RSA, DSA, or ECC). A 128-bit length Host Identity Tag (HIT) is derived from the HI (hashed) and functions as an IPv6 address (/128 prefix) for applications. A four-packet Peer-to-Peer Host Identity Protocol Base EXchange (HIP BEX) establishes a security association (SA, similar to IKE), indexed by the HITs, but independent of the IP address. So HIP can be considered as a shim layer between the transport (TCP/UDP) and IP, providing authentication, data confidentiality, mobility in one basket.

As with IKE, HIP is typically used as a Key Management Protocol (KMP) for ESP. However, HIP is independent of IP and ESP and can be used for a KMP for any secure communication protocol at any level. Thus HIP can provide keying material for the MAC, IP, and Transport layers. HIP can be run directly over the MAC in an Ethernet Frame or within an Information Element in a MAC control frame. HIP can be run over UDP, traversing firewalls, and push keys over to Transport security protocols like SRTP or (D)TLS. Further, HIP could start on the MAC, then be enveloped over UDP after the first link.

The HIP-BEX involves many crypto primitives that are difficult to run on constrained nodes. HIP Diet Exchange (HIP-DEX) [I-D.moskowitz-hip-rg-dex] is a way to make HIP lightweight.

Basically, HIP-DEX a variant of the HIP-BEX specifically designed to use as few crypto primitives as possible yet still provide the same class of security features as HIP-BEX.

HIP-DEX can be used for mutual authentication between two endpoints. After mutual authentication, the two endpoints establish a shared secret, which is fresh and fed into the encryption algorithm for data confidentiality. So HIP-DEX can achieve the authentication, key freshness and data confidentiality objectives of security bootstrapping.

When a node wants to authenticate to the network using HIP and Diet-HIP, it should be able to complete the HIP-BEX or HIP-DEX with the trust anchor or some delegate. In HIP, it does not matter how many domains, and nodes can authenticate each other as long as they have the secret.

In the architecture introduced in Section 2.2 the node and router could be the HIP end-points. Or the router could be a HIP Rendezvous Server, with the node registering to the rendezvous server (RVS) [RFC5204] to be reachable by other nodes. Typically the initial interaction will have the node be the HIP DEX Initiator with the router being the Responder. Using the RVS function on a Router any node could be the Initiator with any other Responder node. As long as there is IP connectivity between the Initiator and Responder, they can be multiple hops away from each other. Alternatively if the first hop from the Initiator has knowledge of the IP address of the Responder, it can act as a MAC/IP gateway for HIP.

An important requirement for the HIP-DEX to work in the architecture, the initiator should be able to get the IP address of the responder or have a gateway function as a forwarder. The IP address can be obtained from a 3rd party source like DNS or Distributed Hash Table (DHT), from local configuration, or from an RVS.

C.4. 802.1X

IEEE 802.1X defines how EAP packets can be transported over in Layer 2, i.e. Ethernet frames [802.1x] by encapsulating EAP packets into EAP Over Lan (EAPOL) frames between EAP peer, called supplicant and the authenticator. EAPOL can also be used in 802.11 wireless links.

To enable IEEE 802.15.4 devices to use EAP authentication, EAP packets encapsulated in EAPOL frames can be carried as payload in 802.15.4 data frames [802.15.4]. EAPOL is well defined and widely used and it lends itself to be easily carried in 802.15.4 data frames. For this, Frame Type subfield of the Frame Control Field of IEEE 802.15.4 MAC header needs to be set to a special value to

indicate the type of the payload, i.e. 802.1X encapsulated EAP packets. EAPOL packets are encoded following common EAPOL PDU structure defined in [802.1x] into the data payload field of 802.15.4 data frames.

Authentication proceeds as follows: authenticator authenticates the supplicants that are on the next hop first. This enables a secure link between the authenticator and these first-hop nodes. The architecture introduced in Section 2.2 requires a new entity called Relay Authenticator. First-hop nodes or router become Relay Authenticators in the next phase of authentication. Relay Authenticators tunnel EAPOL frames to the authenticator in the secure link established. This way all the supplicants are gradually authenticated.

After the keys are established from a successful EAP method (such as PSK mode of TLS), the node runs neighbor discovery protocol to get an IPv6 address assigned [I-D.ietf-6lowpan-nd]. Data transfer can be secured using DTLS or IPSec. Keys derived from EAP TLS are used in either generating DTLS ciphering keys after a successful DTLS handshake or IPSec ESP ciphering keys after a successful IKEv2 handshake.

802.1X can achieve the authentication, key freshness and data confidentiality objectives of security bootstrapping.

Multi domain operation is intrinsically supported due to the use of EAP and AAA. In order to support a device recommissioning case whereby the device's administrative domain is modified, after a new AAA server address assigned and a successful 802.1X method execution, the old set of device 'name and password' MUST be overwritten into the device by a new set of 'name and password' that are assigned by the domain administrator. The device MUST be rebooted to execute EAP method again for authentication and a new master session key MUST be generated.

It should be noted that currently 802.15.4 does not allow multiplexing multiple protocols on the same link like ethernet does. However this might change in the future.

The requirements for the use of 802.1X defined EAPOL as a bootstrapping protocol can be stated as follows:

- o A special value in the Frame Type subfield of the Frame Control Field of IEEE 802.15.4 MAC header to indicate the type of the payload,

- o Link Layer Multicast Group addresses for 802.15.4 corresponding to EAPOL Group Address Assignments defined in Table 11.1 of [802.1x], especially to be used in EAPOL-Start packet.
- o Which MAC frames of beacon, data, acknowledgment and MAC command as defined in [802.15.4] with what security levels are mapped to controlled port, which MAC frames with what security levels are mapped to uncontrolled port and which MAC frames are never mapped to any of controlled/uncontrolled port (i.e., the payload of those frames are used by the MAC-layer itself and never used by upper layers).
- o A new entity called Relay Authenticator may be added to the 802.1x architecture. Behaviour of Relay Authenticator needs to be defined.

Authors' Addresses

Behcet Sarikaya
Huawei USA
5340 Legacy Dr.
Plano, TX 75024

Email: sarikaya@ieee.org

Yoshihiro Ohba
Toshiba
Tokyo, Japan

Email: yoshihiro.ohba@toshiba.co.jp

Robert Moskowitz
Verizon Business Systems
15210 Sutherland
Oak Park, MI 48237

Email: rgm@labs.htt-consult.com

Zhen Cao
China Mobile
Beijing, China

Email: caozhen@chinamobile.com

Robert Cragie
Pacific Gas and Electric
89 Greenfield Crescent
Wakefield, UK WF4 4WA

Email: robert.cragie@gridmerge.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: August 10, 2013

J. Schoenwaelder
A. Sehgal
Jacobs University
T. Tsou
Huawei Technologies (USA)
C. Zhou
Huawei Technologies
February 6, 2013

Definition of Managed Objects for IPv6 over Low-Power Wireless Personal
Area Networks (6LoWPANs)
draft-schoenw-6lowpan-mib-03

Abstract

This memo defines a portion of the Management Information Base (MIB) for use with network management protocols in the Internet community. In particular, it defines objects for managing IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs).

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 10, 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. The Internet-Standard Management Framework	3
3. Conventions	3
4. Overview	3
5. Relationship to Other MIB Modules	6
6. Definitions	6
7. Security Considerations	14
8. IANA Considerations	15
9. Acknowledgements	15
10. References	15
10.1. Normative References	15
10.2. Informative References	16
Appendix A. JSON Representation	16

1. Introduction

This memo defines a portion of the Management Information Base (MIB) for use with network management protocols. In particular it defines objects for managing IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) [RFC4944].

2. The Internet-Standard Management Framework

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of RFC 3410 [RFC3410].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI). This memo specifies a MIB module that is compliant to the SMIV2, which is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].

3. 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].

4. Overview

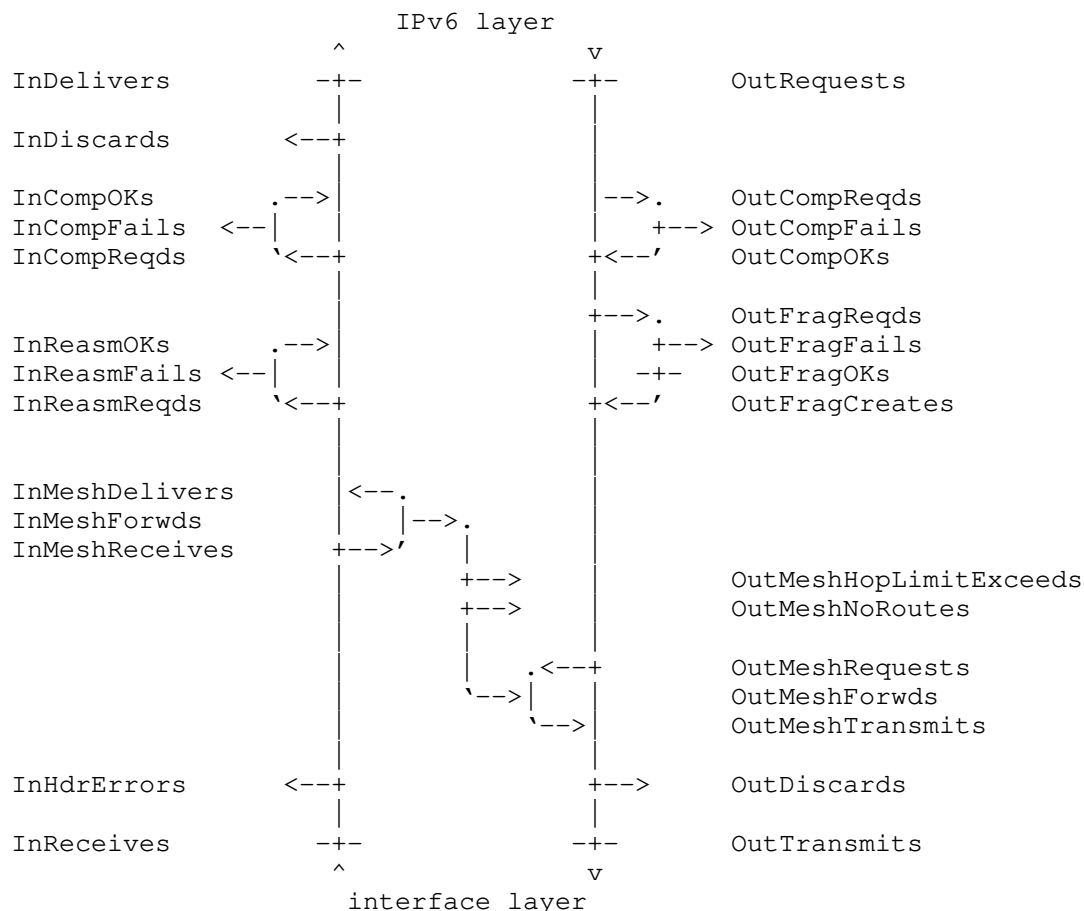
The MIB module is organized into groups of scalars and tables.

```
# LOWPAN-MIB registration tree (generated by smidump 0.4.8)
```

```
--lowpanMIB(1.3.6.1.2.1.XXXX)
+--lowpanNotifications(0)
+--lowpanObjects(1)
|   +-- r-n Unsigned32 lowpanReasmTimeout(1)
|   +-- r-n Counter32  lowpanInReceives(2)
|   +-- r-n Counter32  lowpanInHdrErrors(3)
|   +-- r-n Counter32  lowpanInMeshReceives(4)
|   +-- r-n Counter32  lowpanInMeshForwds(5)
|   +-- r-n Counter32  lowpanInMeshDelivers(6)
|   +-- r-n Counter32  lowpanInReasmReqds(7)
|   +-- r-n Counter32  lowpanInReasmFails(8)
|   +-- r-n Counter32  lowpanInReasmOKs(9)
|   +-- r-n Counter32  lowpanInCompReqds(10)
|   +-- r-n Counter32  lowpanInCompFails(11)
|   +-- r-n Counter32  lowpanInCompOKs(12)
|   +-- r-n Counter32  lowpanInDiscards(13)
|   +-- r-n Counter32  lowpanInDelivers(14)
|   +-- r-n Counter32  lowpanOutRequests(15)
|   +-- r-n Counter32  lowpanOutCompReqds(16)
|   +-- r-n Counter32  lowpanOutCompFails(17)
|   +-- r-n Counter32  lowpanOutCompOKs(18)
|   +-- r-n Counter32  lowpanOutFragReqds(19)
|   +-- r-n Counter32  lowpanOutFragFails(20)
|   +-- r-n Counter32  lowpanOutFragOKs(21)
|   +-- r-n Counter32  lowpanOutFragCreates(22)
|   +-- r-n Counter32  lowpanOutMeshHopLimitExceeds(23)
|   +-- r-n Counter32  lowpanOutMeshNoRoutes(24)
|   +-- r-n Counter32  lowpanOutMeshRequests(25)
|   +-- r-n Counter32  lowpanOutMeshForwds(26)
|   +-- r-n Counter32  lowpanOutMeshTransmits(27)
|   +-- r-n Counter32  lowpanOutDiscards(28)
|   +-- r-n Counter32  lowpanOutTransmits(29)
+--lowpanConformance(2)
|   +--lowpanGroups(1)
|   |   +--lowpanCoreGroup(1)
|   +--lowpanCompliances(2)
|       +--lowpanCompliance(1)
```

The counters defined in the MIB module provide information about the 6LoWPAN datagrams received and transmitted and how they are processed in the 6LoWPAN layer. For the purpose of this specification, a 6LoWPAN datagram is an IEEE 805.14.5 datagram with a dispatch byte matching the bit patterns 01xxxxxx, 10xxxxxx, or 11xxxxxx. The processing of IEEE 805.14.5 datagrams matching the bit pattern 00xxxxxx (NALP - not a LoWPAN frame) [RFC4944] is not considered by this specification. Other radio technologies may use different

mechanisms to identify 6LoWPAN datagrams (e.g., the BLUETOOTH Low Energy Logical Link Control and Adaptation Protocol uses Channel Identifiers [I-D.ietf-6lowpan-btle])). The following Case diagram illustrates the conceptual relationships of the counters.



The fragmentation related counters have been modeled after the fragmentation related counters of the IP-MIB [RFC4293]. The discard counters have been placed at the end of the input and output chains but they can be bumped any time if a datagram is discarded for a reason not covered by the other counters.

The compression related counters provide insights into compression requests and in particular also compression related failures. Note that the diagram is conceptual in the sense that compression happens after reassembly for incoming 6LoWPAN datagrams and compression happens before fragmentation for outgoing 6LoWPAN datagrams.

Implementations may choose to implement things slightly differently. For example, implementations may decompress FRAG1 fragments as soon as they are received, not waiting for reassembly to complete.

The mesh header processing related counters do not have an explicit discard counter. Implementations that do not support mesh forwarding MUST count the number of received 6LoWPAN datagrams with a MESH header (lowpanInMeshReceives) but they MUST NOT increment the lowpanInMeshReceives and lowpanInMeshDelivers counters if these 6LoWPAN datagrams are dropped.

5. Relationship to Other MIB Modules

The MIB module IMPORTS definitions from SNMPv2-SMI [RFC2578] and SNMPv2-CONF [RFC2580].

6. Definitions

LOWPAN-MIB DEFINITIONS ::= BEGIN

IMPORTS

```
MODULE-IDENTITY, OBJECT-TYPE, Unsigned32, Counter32, mib-2
    FROM SNMPv2-SMI                                -- RFC 2578
OBJECT-GROUP, MODULE-COMPLIANCE
    FROM SNMPv2-CONF;                               -- RFC 2580
```

lowpanMIB MODULE-IDENTITY

LAST-UPDATED "201301090000Z"

ORGANIZATION

"Jacobs University Bremen"

CONTACT-INFO

"Juergen Schoenwaelder

Jacobs University Bremen

Email: j.schoenwaelder@jacobs-university.de

Anuj Sehgal

Jacobs University Bremen

Email: s.anuj@jacobs-university.de

Tina Tsou

Huawei Technologies

Email: tina.tsou.zouting@huawei.com

Cathy Zhou

Huawei Technologies

Email: cathyzhou@huawei.com"

DESCRIPTION

"The MIB module for monitoring nodes implementing the IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) protocol.

Copyright (c) 2013 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>)."

REVISION "201301090000Z"

DESCRIPTION

"Initial version, published as RFC XXXX."

-- RFC Ed.: replace XXXX with actual RFC number and remove this note

::= { mib-2 XXXX }

-- object definitions

lowpanNotifications OBJECT IDENTIFIER ::= { lowpanMIB 0 }
lowpanObjects OBJECT IDENTIFIER ::= { lowpanMIB 1 }
lowpanConformance OBJECT IDENTIFIER ::= { lowpanMIB 2 }

lowpanReasmTimeout OBJECT-TYPE

SYNTAX Unsigned32

UNITS "seconds"

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The maximum number of seconds that received fragments are held while they are awaiting reassembly at this entity."

::= { lowpanObjects 1 }

lowpanInReceives OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The total number of 6LoWPAN datagrams received, including those received in error."

::= { lowpanObjects 2 }

lowpanInHdrErrors OBJECT-TYPE

SYNTAX Counter32

```
MAX-ACCESS read-only
STATUS      current
DESCRIPTION
    "The number of received 6LoWPAN datagrams discarded due to
    errors in their headers, including unknown dispatch values,
    errors discovered during any decompression attempts, etc."
 ::= { lowpanObjects 3 }

lowpanInMeshReceives OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received 6LoWPAN datagrams with a MESH header."
    ::= { lowpanObjects 4 }

lowpanInMeshForwds OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received 6LoWPAN datagrams requiring MESH
        forwarding."
    ::= { lowpanObjects 5 }

lowpanInMeshDelivers OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received 6LoWPAN datagrams with a MESH header
        delivered to the local system."
    ::= { lowpanObjects 6 }

lowpanInReasmReqds OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received 6LoWPAN fragments that needed to
        be reassembled. This includes both FRAG1 and FRAGN 6LoWPAN
        datagrams."
    ::= { lowpanObjects 7 }

lowpanInReasmFails OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
```

DESCRIPTION

"The number of failures detected by the re-assembly algorithm (e.g., timeouts). Note that this is not necessarily a count of discarded 6LoWPAN fragments since implementations can lose track of the number of fragments by combining them as received."

::= { lowpanObjects 8 }

lowpanInReasmOKs OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of IPv6 packets successfully reassembled."

::= { lowpanObjects 9 }

lowpanInCompReqds OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of 6LoWPAN datagrams requiring header decompression."

::= { lowpanObjects 10 }

lowpanInCompFails OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of 6LoWPAN datagrams where header decompression failed (e.g., because the necessary context information is not available)."

::= { lowpanObjects 11 }

lowpanInCompOKs OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of 6LoWPAN datagrams where header decompression was successful."

::= { lowpanObjects 12 }

lowpanInDiscards OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of received 6LoWPAN datagrams for which no problems were encountered to prevent their continued processing, but were discarded (e.g., for lack of buffer space). Note that this counter does not include any datagrams discarded due to a reassembly failure."

::= { lowpanObjects 13 }

lowpanInDelivers OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The total number of IPv6 packets successfully delivered to the IPv6 layer."

::= { lowpanObjects 14 }

lowpanOutRequests OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The total number of IPv6 packets supplied by the IPv6 layer."

::= { lowpanObjects 15 }

lowpanOutCompReqds OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The total number of IPv6 packets for which header compression was attempted."

::= { lowpanObjects 16 }

lowpanOutCompFails OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The total number of IPv6 packets for which header compression failed (e.g., because the UDP checksum check failed while performing UDP header compression)."

::= { lowpanObjects 17 }

lowpanOutCompOKs OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION
 "The total number of IPv6 packets for which header compression
 was successful."
 ::= { lowpanObjects 18 }

lowpanOutFragReqds OBJECT-TYPE
 SYNTAX Counter32
 MAX-ACCESS read-only
 STATUS current
 DESCRIPTION
 "The number of IPv6 packets that require fragmentation
 in order to be transmitted."
 ::= { lowpanObjects 19 }

lowpanOutFragFails OBJECT-TYPE
 SYNTAX Counter32
 MAX-ACCESS read-only
 STATUS current
 DESCRIPTION
 "The number of IPv6 packets that have been discarded because
 they needed to be fragmented but could not be."
 ::= { lowpanObjects 20 }

lowpanOutFragOKs OBJECT-TYPE
 SYNTAX Counter32
 MAX-ACCESS read-only
 STATUS current
 DESCRIPTION
 "The number of IPv6 packets that have been successfully
 fragmented."
 ::= { lowpanObjects 21 }

lowpanOutFragCreates OBJECT-TYPE
 SYNTAX Counter32
 MAX-ACCESS read-only
 STATUS current
 DESCRIPTION
 "The number of 6LoWPAN fragments that have been
 generated as a result of fragmentation. This includes
 both FRAG1 and FRAGN 6LoWPAN datagrams."
 ::= { lowpanObjects 22 }

lowpanOutMeshHopLimitExceeds OBJECT-TYPE
 SYNTAX Counter32
 MAX-ACCESS read-only
 STATUS current
 DESCRIPTION
 "The number of 6LoWPAN datagrams with a MESH header that

were dropped because the hop limit has been exceeded."
 ::= { lowpanObjects 23 }

lowpanOutMeshNoRoutes OBJECT-TYPE

SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "The number of 6LoWPAN datagrams with a MESH header that
 were dropped because there was no forwarding information
 available."
 ::= { lowpanObjects 24 }

lowpanOutMeshRequests OBJECT-TYPE

SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "The number of 6LoWPAN datagrams requiring MESH header
 encapsulation."
 ::= { lowpanObjects 25 }

lowpanOutMeshForwds OBJECT-TYPE

SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "The number of 6LoWPAN datagrams with a MESH header for
 which suitable forwarding information was available."
 ::= { lowpanObjects 26 }

lowpanOutMeshTransmits OBJECT-TYPE

SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "The number of 6LoWPAN datagrams with a MESH header
 created."
 ::= { lowpanObjects 27 }

lowpanOutDiscards OBJECT-TYPE

SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "The number of IPv6 packets for which no problem was
 encountered to prevent their transmission to their
 destination, but were discarded (e.g., for lack of

```
        buffer space)."  
 ::= { lowpanObjects 28 }  
  
lowpanOutTransmits OBJECT-TYPE  
    SYNTAX      Counter32  
    MAX-ACCESS  read-only  
    STATUS      current  
    DESCRIPTION  
        "The total number of 6LoWPAN datagram that this entity  
        supplied to the lower layers for transmission."  
 ::= { lowpanObjects 29 }  
  
-- conformance definitions  
  
lowpanGroups          OBJECT IDENTIFIER ::= { lowpanConformance 1 }  
lowpanCompliances     OBJECT IDENTIFIER ::= { lowpanConformance 2 }  
  
lowpanCompliance MODULE-COMPLIANCE  
    STATUS      current  
    DESCRIPTION  
        "Compliance statement for systems that implement 6LoWPAN."  
    MODULE      -- this module  
    MANDATORY-GROUPS {  
        lowpanCoreGroup  
    }  
 ::= { lowpanCompliances 1 }  
  
lowpanCoreGroup OBJECT-GROUP  
    OBJECTS {  
        lowpanReasmTimeout,  
        lowpanInReceives,  
        lowpanInHdrErrors,  
        lowpanInMeshReceives,  
        lowpanInMeshForwds,  
        lowpanInMeshDelivers,  
        lowpanInReasmReqds,  
        lowpanInReasmFails,  
        lowpanInReasmOKs,  
        lowpanInCompReqds,  
        lowpanInCompFails,  
        lowpanInCompOKs,  
        lowpanInDiscards,  
        lowpanInDelivers,  
        lowpanOutRequests,  
        lowpanOutCompReqds,  
        lowpanOutCompFails,  
        lowpanOutCompOKs,  
        lowpanOutFragReqds,
```



```
    lowpanOutFragFails,
    lowpanOutFragOKs,
    lowpanOutFragCreates,
    lowpanOutMeshHopLimitExceeds,
    lowpanOutMeshNoRoutes,
    lowpanOutMeshRequests,
    lowpanOutMeshForwds,
    lowpanOutMeshTransmits,
    lowpanOutDiscards,
    lowpanOutTransmits
}
STATUS      current
DESCRIPTION
    "A collection of objects providing information and
    statistics about the processing of 6LoWPAN datagrams."
 ::= { lowpanGroups 1 }
```

END

7. Security Considerations

Some of the readable objects in this MIB module (i.e., objects with a MAX-ACCESS other than not-accessible) may be considered sensitive or vulnerable in some network environments. It is thus important to control even GET and/or NOTIFY access to these objects and possibly to even encrypt the values of these objects when sending them over the network via SNMP. These are the tables and objects and their sensitivity/vulnerability:

The read-only counters provide insights into the amount of 6LoWPAN traffic a node is receiving or transmitting. This might provide information whether a device is regularly exchanging information with other devices or whether a device is mostly not participating in any communication (e.g., the device might be "easier" to take away unnoticed). The reassembly counters could be used to direct denial of service attacks on the reassembly mechanism.

SNMP versions prior to SNMPv3 did not include adequate security. Even if the network itself is secure (for example by using IPsec), even then, there is no control as to who on the secure network is allowed to access and GET/SET (read/change/create/delete) the objects in this MIB module.

It is RECOMMENDED that implementers consider the security features as provided by the SNMPv3 framework (see [RFC3410], section 8), including full support for the SNMPv3 cryptographic mechanisms (for authentication and privacy).

Further, deployment of SNMP versions prior to SNMPv3 is NOT RECOMMENDED. Instead, it is RECOMMENDED to deploy SNMPv3 and to enable cryptographic security. It is then a customer/operator responsibility to ensure that the SNMP entity giving access to an instance of this MIB module is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change/create/delete) them.

8. IANA Considerations

IANA is requested to assign a value for "XXXX" under the 'mib-2' subtree and to record the assignment in the SMI Numbers registry. When the assignment has been made, the RFC Editor is asked to replace "XXXX" (here and in the MIB module) with the assigned value and to remove this note.

9. Acknowledgements

This specification borrows heavily from the IP-MIB defined in [RFC4293].

Juergen Schoenwaelder and Anuj Sehgal were partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

10. References

10.1. Normative References

- | | |
|-----------|--|
| [RFC2119] | Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. |
| [RFC2578] | McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999. |
| [RFC2579] | McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999. |
| [RFC2580] | McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999. |

- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [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-11 (work in progress), October 2012.

10.2. Informative References

- [RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.
- [RFC4293] Routhier, S., "Management Information Base for the Internet Protocol (IP)", RFC 4293, April 2006.
- [RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, July 2012.
- [I-D.lhotka-netmod-yang-json] Lhotka, L., "Modeling JSON Text with YANG", draft-lhotka-netmod-yang-json-00 (work in progress), October 2012.

Appendix A. JSON Representation

Using the translation algorithm defined in [RFC6643], the SMIv2 module can be translated to YANG. Using the JSON representation of data modeled in YANG defined in [I-D.lhotka-netmod-yang-json], the objects defined in the MIB module can be represented in JSON as shown below. The compact representation without any white space uses 468 octets. (Of course, this number depends on the number of octets needed for the counter values.)

```
{
  "LOWPAN-MIB:LOWPAN-MIB": {
    "lowpanReasmTimeout": 20,
    "lowpanInReceives": 42,
    "lowpanInHdrErrors": 0,
    "lowpanInMeshReceives": 8,
    "lowpanInMeshForwds": 0,
    "lowpanInMeshDelivers": 0,
    "lowpanInReasmReqds": 22,
    "lowpanInReasmFails": 2,
    "lowpanInReasmOKs": 20,
    "lowpanInCompReqds": 16,
    "lowpanInCompFails": 2,
    "lowpanInCompOKs": 14,
    "lowpanInDiscards": 1,
    "lowpanInDelivers": 12,
    "lowpanOutRequests": 12,
    "lowpanOutCompReqds": 0,
    "lowpanOutCompFails": 0,
    "lowpanOutCompOKs": 0,
    "lowpanOutFragReqds": 5,
    "lowpanOutFragFails": 0,
    "lowpanOutFragOKs": 5,
    "lowpanOutFragCreates": 8,
    "lowpanOutMeshHopLimitExceeds": 0,
    "lowpanOutMeshNoRoutes": 0,
    "lowpanOutMeshRequests": 0,
    "lowpanOutMeshForwds": 0,
    "lowpanOutMeshTransmits": 0,
    "lowpanOutDiscards": 0,
    "lowpanOutTransmits": 15
  }
}
```

Authors' Addresses

Juergen Schoenwaelder
Jacobs University
Campus Ring 1
Bremen 28759
Germany

Email: j.schoenwaelder@jacobs-university.de

Anuj Sehgal
Jacobs University
Campus Ring 1
Bremen 28759
Germany

EMail: s.anuj@jacobs-university.de

Tina Tsou
Huawei Technologies (USA)
2330 Central Expressway
Santa Clara CA 95050
USA

EMail: tina.tsou.zouting@huawei.com

Cathy Zhou
Huawei Technologies
Bantian, Longgang District
Shenzhen 518129
P.R. China

EMail: cathyzhou@huawei.com

