CoRE K. Kuladinithi, Ed.
Internet-Draft ComNets, Hamburg University of Technology
Intended status: Standards Track M. Becker
Expires: August 24, 2017 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.

Table of Contents

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).

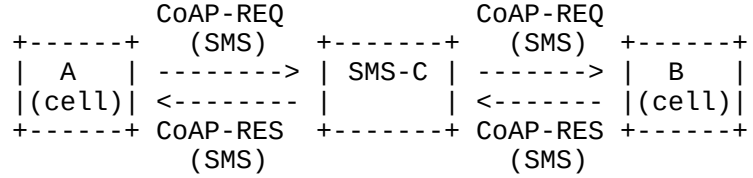


Figure 1: Cellular and Cellular Communication (only SMS-based)

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).

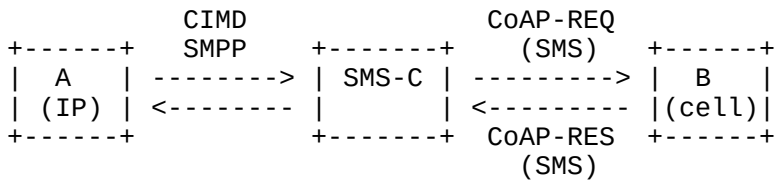


Figure 2: IP and Cellular Communication

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

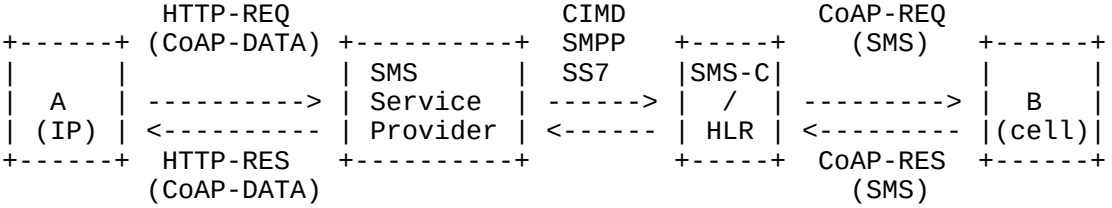


Figure 3: IP and Cellular Communication (using an SMS Service Provider)

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).

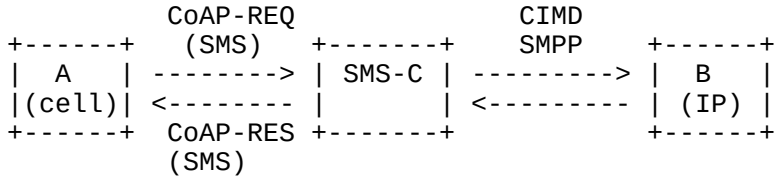


Figure 4: Cellular and IP Communication

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

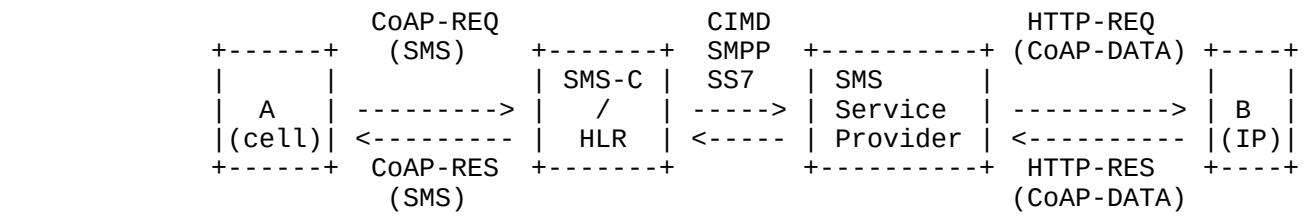


Figure 5: IP and Cellular Communication (using an SMS Service Provider)

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:

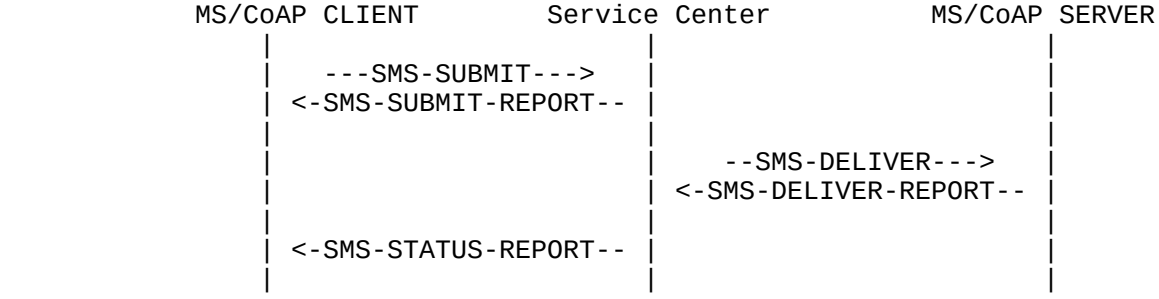


Figure 6: CoAP Messages over SMS

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 characters = (140 bytes * 8)/7.

Possible options for larger CoAP messages are:

Concatenated short messages
: Most MSs are able to send concatenation short messages in order to 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 it’s alternative address. However, that implies that the initiating CoAP Client is aware of the presence of the alternative interface. For this reason the new options Response-To-Uri-Host and Response-To-Uri-Port are proposed.

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

Table 1: New CoAP Option Numbers

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

The options SHOULD NOT be used in the response.

The options MUST NOT occur more than once.

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

Example of such URI :

- 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-transports] Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-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:

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

Changed from draft-04 to draft-05:

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

Changed from draft-03 to draft-04:

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

Changed from draft-02 to draft-03:

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

Changed from draft-01 to draft-02:

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

- Reply-To-* changed to Response-To-*. Section 12
- Added URI scheme.
- Added possible CON/NON/ACK interactions.
- Added possible M2M proxy scenarios.
- Added reference to bormann-coap-misc for other SMS encoding. Section 4
- Updated requirements on Uri-Host and Uri-Port for coap+tel://.
- Chose CoAP option numbers and updated the option number table to meet draft-ietf-core-coap-10. />
- Added an IANA registration for the URI scheme. Section 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 WG
INTERNET-DRAFT
Intended Status: Informational
Expires: April 26, 2016

Z. Cao
R. Jadhav
Huawei

October 27, 2016

CoAP Delegated Observe
draft-cao-core-delegated-observe-00

Abstract

This document discusses the scenarios for "delegated observe", in which a subscriber needs to register some resources on behalf of some other entities. This document also presents a CoAP protocol extension for the delegated observe operation.

Table of Contents

1 Introduction

CoAP [RFC7252] is a light-weight application protocol for constrained networks. To avoid keeping polling devices, CoAP supports the ’observe’ operation defined in [RFC7641], in which a subscriber can register its interest to certain resources and then be updated with their representation. However, in the current "observe" protocol, the subscriber can only register interests on behalf of itself, and therefore, updated information of the represented resources could not be notified to any parties other than the one who sends the observe request.

This document discusses the scenarios for "delegated observation", in which a subscriber needs to register some resources on behalf of other entities or a group of entities including itself. This document also presents a CoAP protocol extension for the delegated observe operation.

2 Scenarios for Delegated Observe

This section describes scenarios that needs delegated observe.

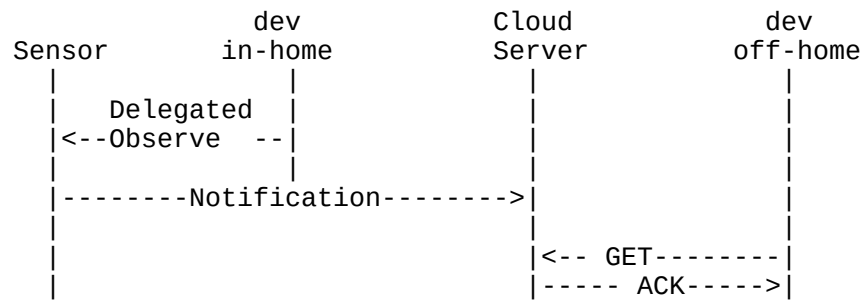
2.1 Multiple Devices

In a typical smart home network setup, a user with multiple devices wants to observe some sensor resource (e.g., thermometer, bulbs). Instead of sending CoAP Observe requests from every single device, one of the them can send an Observe Registration on behalf of that group of devices, so that all of them will be notified at once.

2.2 Delegation to Cloud

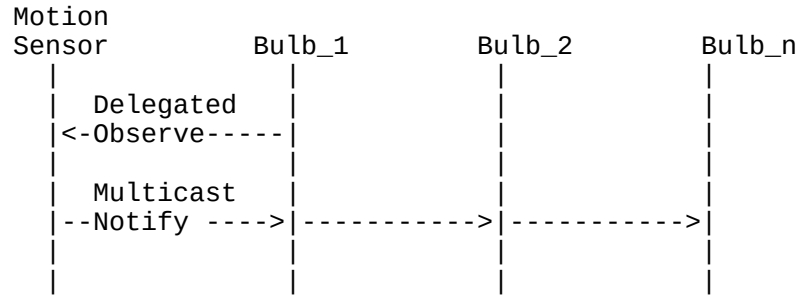
A user wants its mobile device to be notified of a certain sensor information both in-home and off-home. When the device moves out of its smart home network coverage, it is normally hidden behind NAT and FW that keep it being reached for such notification messages. In this case, the normal observe-notification scheme may fail. A walk-around for this case is to let the device send a delegated observe request while at home, asking the home sensors send notifications to the device’s representative cloud server, so that the device can always fetch the information from it cloud service while off-home.

This scenario is depicted in Fig. 1.



2.3 Multicast

A group of devices would like to observe the location information on a motion sensor. This is a useful case when a number of light bulbs need to adjust its lighting intensity based on the location of the observed motion object. Instead of let each device register an interest on the motion sensor, one of them could simply delegate the observe to this multicast group, so that the location update notifications will be send to the multicast address that they belong to. This scenario is visualized in Fig.2.



3 Overview of Delegated Observe

As show in Fig.3, we name the node that has the direct representation of the CoAP resource as the "Source node"; and the immediate node as the ’Delegate node’ that will send delegated Observe request to the Source node, on behalf of the "Delegated node" who will be notified by the Source Node.

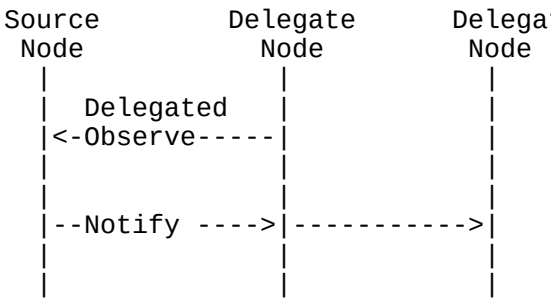


Figure 3: Overview of Delegated Observe

4 The Delegated Observe Option

The properties of the Delegated Observe Option are defined in Fig. 4.

In a GET request:

No.	C	U	N	R	Name	Format	Length	Default
TBD		x	-		Delegated Observe	string	0-256	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

In a Response:

No.	C	U	N	R	Name	Format	Length	Default
TBD		x	-		Delegated Observe	uint	0-3 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Figure 4: CoAP Delegated Observe Option

When included in a GET request, the Delegated Observe Option extends the GET method so it does not only retrieve a current representation of the target resource, but also requests the server to add an entry in the list of observers of the resource. This entry maps the target resource to an identifier of the observer contained in the value of this option.

When used to register the representation, the value of Delegated Observe in a GET request is the identifier of the nodes that will be notified, in the format of "IP:port" or "domain_name:port".

When used to de-register the representation, the value of Delegated

Observe in the GET is a special string, e.g., in the format of ":::", the specific format needs further discussion. [TBD]

When included in a response, the Delegated Observe Option identifies the message as a notification. The Option value is used as a sequence number used to infer the order of the notifications. The ordering inference is the same as what has been discussed in Section 3.4 of [RFC7641].

5 Handling Delegated Observe

Before sending the delegated observe, the "Delegate node" needs to know which address:port on the Delegated node will be open to receive the subsequent notification from the source node. This negotiation is out of scope of this document.

Upon receiving the delegated observe request, the "Source Node" will create an entry based on the identifier of the Delegated Node contained within the request. The Source Node can decide if it needs to verify the validity of the Delegated node, per its own policy. The validation way is also out of the scope of this document.

The Delegated Node, once being delegated and verified by Source Node, will be notified subsequently, although it does not send the request for that resource. The Delegated Node interprets the CoAP message, and will handle this message if the CoAP message contains a Delegated Observe option defined in Section 4.

6 Security Considerations

The security considerations in [RFC7252] and [RFC7641] apply.

Delegated observe may increase the risk of amplification attacks, given the source node will send notifications to delegated nodes who have not requested the resource directly. This negative effect can be controlled by several implementation considerations: a) the delegating node can negotiate with the delegated node before sending delegated observe, out of band; b) the source node will strictly control the rate of the notifications, so that flooding will be avoided; c) the delegated node can block any notifications beyond a certain data rate.

7 IANA Considerations

If approved, an Option Number for the defined Delegated Observe Option for CoAP will be needed.

Number	Name	Reference
TBD	Delegated Observe	[thisdoc]

7 References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

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

[RFC7641] K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, September 2015.

Appendix A. Examples

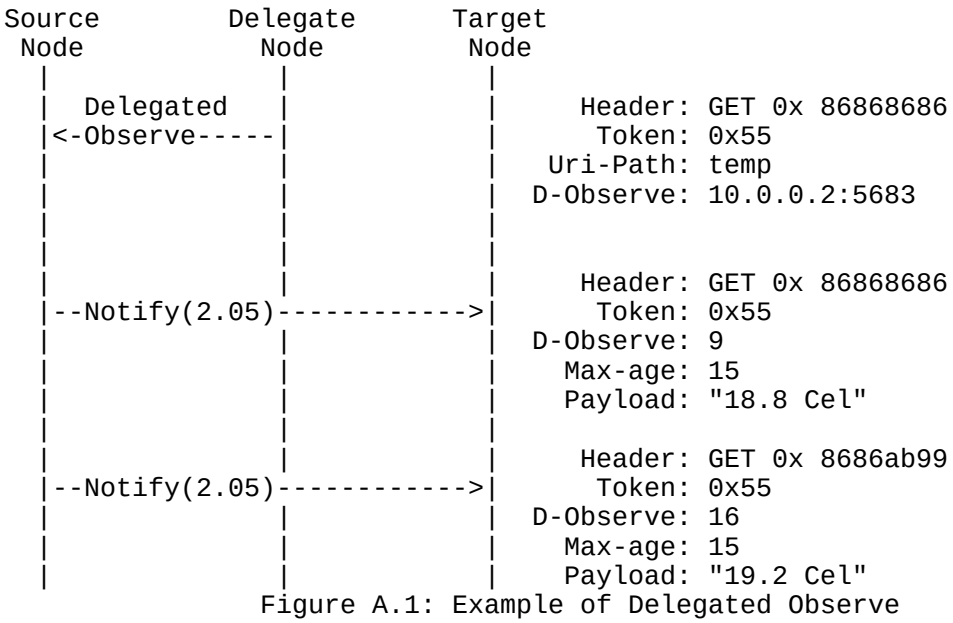


Figure A.1: Example of Delegated Observe

Authors' Addresses

Zhen Cao
Huawei Tech
Beijing, China

EMail: zhencao.ietf@gmail.com

Rahul Arvind Jadhav
Huawei Tech,
Kundalahalli Village,
Bangalore, India

EMail: rahul.jadhav@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Experimental
Expires: June 9, 2017

D. Garcia
S. Matheu
R. Marin
University of Murcia
December 6, 2016

Application Layer Security for CoAP using the (D)TLS Record Layer
draft-garcia-core-app-layer-sec-with-dtls-record-00

Abstract

This document briefly describes an idea to provide Application-Layer Security for CoAP using (D)TLS Record Layer, assuming it is operative in two CoAP endpoints.

Status of This Memo

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

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

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

This Internet-Draft will expire on June 9, 2017.

Table of Contents

1. Introduction

Secure communications in constrained scenarios is subject of current interest since the restrictions in those kinds of networks motivates rethinking the solutions that up to now have been used in networks that do not suffer from very stringent requirements. (D)TLS [RFC6347][RFC5246] is a standard proposed to secure the communications of CoAP and suitable for end-to-end communications unless a CoAP proxy participates in the communication. To overcome this problem [I-D.ietf-core-object-security] propose Object Security for CoAP (OSCOAP) to allow end-to-end security between two CoAP endpoints in case of a CoAP proxy intermediating between them.

In this document we explore that possibility of providing CoAP security at application layer, assuming a (D)TLS Record Layer is operative (i.e. have the required keys) in both CoAP endpoints. One possibility to "activate" the (D)TLS Record Layer is running (D)TLS handshake over CoAP, as mentioned in CoDTLS [I-D.schmertmann-dice-codtls]. Other (more challenging) options are discussed in [I-D.bhattacharyya-dice-less-on-coap]

1.1. 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. Application Layer Security for CoAP with (D)TLS Record Protocol

To achieve application layer security using the (D)TLS Record, we assume the (D)TLS [RFC6347] [RFC5246] Record layer is already activated, using a protocol such as CoDTLS [I-D.schmertmann-dice-codtls]. Once we have the (D)TLS Record Layer

active, the next step is to define how the CoAP message will be secured end-to-end using the (D)TLS Record Layer.

The entire CoAP message generated by a CoAP sender will need to arrive to the CoAP recipient, achieving integrity and confidentiality for certain parts of the CoAP message (specific CoAP options and payload) excluding the options CoAP intermediaries (proxies) will need to understand to process the CoAP message correctly. The CoAP header would need to arrive maintaining the semantics and version of the protocol.

2.1. CoAP Fields to protect

Here we discuss how the CoAP message is going to be processed to achieve application layer security. How each part of the CoAP message (Header, Options and Payload) is treated and which options are protected an which ones are left unprotected using the (D)TLS Record Layer. Following the procedure specified in OSCOAP [I-D.ietf-core-object-security], we protect all options that are intended to be read by the CoAP recipient.

- CoAP Header: version and code are protected.
- CoAP Options: All the options that can be modified by the proxy are left unprotected. All options that are intended for the the CoAP recipient are protected. There might be the case there an option is both left unprotected for the proxy to process and is also intended for the CoAP recipient to see.
- CoAP Payload: The payload is always protected.

Similarly to OSCON [I-D.ietf-core-object-security], it would be possible to only encrypt the payload of the original CoAP message.

3. Processing a CoAP message with the (D)TLS Record

In this section we analyze how the CoAP message is processed and protected using the (D)TLS Record. In Figure 1 we can see how from the Original CoAP Header we obtain the fields that have to be protected (Version and Code) in 2 bytes. We get the Version and the Code. We will have a padding of 6 bits, then the version and code all in 2 bytes.

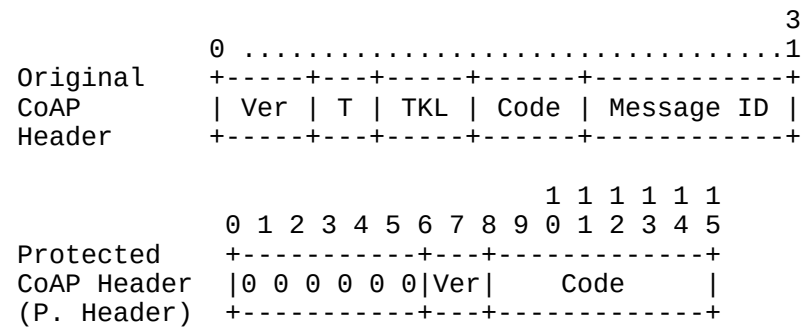


Figure 1: Protected Header Fields

This "denatured" CoAP header is concatenated with the CoAP options to be protected and the Payload (if any) of the Original CoAP message (see Figure 2). This array of bytes is sent to the (D)TLS Record Layer to be processed. The resulting information is a protected array of bytes with a (D)TLS Record Header which will process it generating the (D)TLS Record (adding the (D)TLS Record Header). After that, we add the (D)TLS record to the payload of the message to be sent, that will contain the original CoAP Header and only the options available for the proxies. The Protected CoAP message is formed by the Original CoAP Message Header, the Options that are not considered to be protected with this mechanism, then the marker 0xFF and finally the Payload that contains the (D)TLS Record.

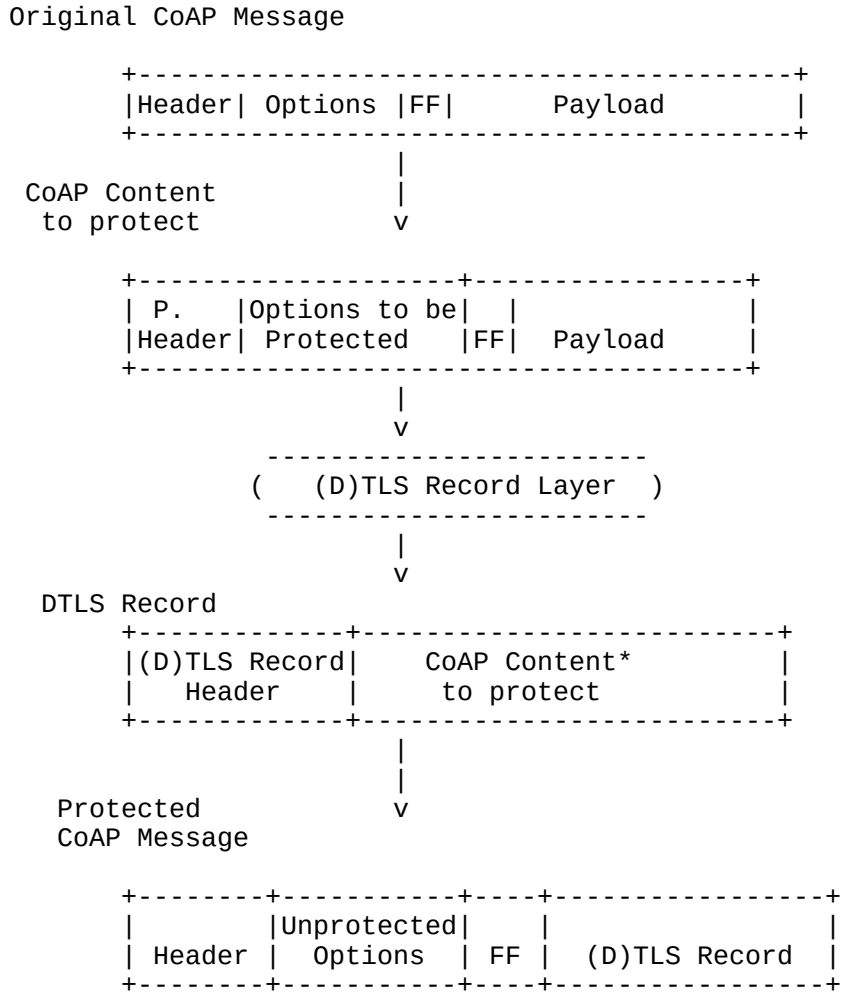


Figure 2: Processing CoAP message

Upon reception, the CoAP recipient will get the CoAP Payload of the Protected Message and send it to the (D)TLS Record Layer to obtain the Protected Header and the list of options within the (D)TLS Record. With this information, once it is verified correctly, the CoAP recipient constructs the Original CoAP Message.

4. Bootstrapping the (D)TLS Record Layer for Application Security

To enable this solution, the (D)TLS Record Layer in both CoAP endpoints must have a connection to process this information. One alternative is running (D)TLS over CoAP as specified in [I-D.schmertmann-dice-codtls]. However, we consider that it would be possible to define we define a separation between the (D)TLS Handshake and the (D)TLS Record Layer with an interface to be standarized. The (D)TLS Handshake is used to negotiate the parameters to establish a Security Association (SA) in (D)TLS Record Layer. With this interface, we argue that this SA can be set by the (D)TLS Handshake or any other Key Management Protocol (KMP), as we show in Figure 3. This would be similar to the separation in the IPsec architecture [RFC4301], where IKEv2 [RFC7296] is just one of the possible Key Management Protocol to establish IPsec SAs. In fact, IPsec defines a standard API (PFKEY_v2 [RFC2367])) for this purpose.

An example of this separation has been proposed in [I-D.bhattacharyya-dice-less-on-coap]. Another way to benefit from this separation could be that one of the CoAP endpoint has a token (e.g. Kerberos ticket, and ACE token, etc...), with the key material and information (cryptographic keys, algorithms) to start the (D)TLS Record Layer, just presenting the ticket, without the need of running the (D)TLS handshake.

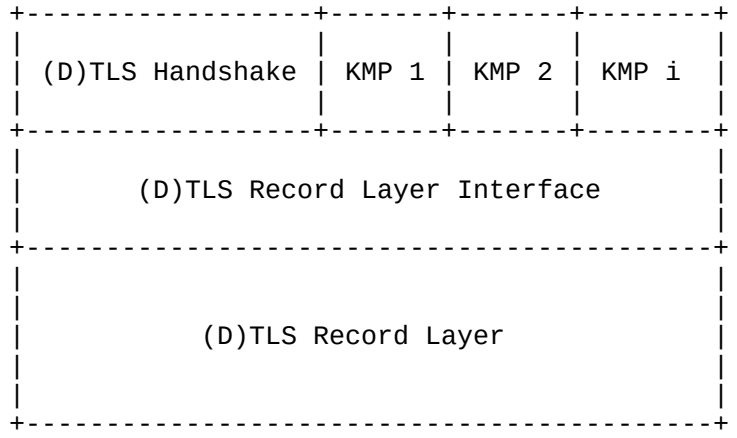


Figure 3: (D)TLS Record Layer Interface

5. Acknowledgments

This work has been possible partially by the ARMOUR project (FP7-ARMOUR-644852 EU Project) and the Spanish National Project CICYT EDISON (TIN2014-52099-R) granted by the Ministry of Economy and Competitiveness of Spain (including ERDF support).

6. Normative References

[I-D.bhattacharyya-dice-less-on-coap]
Bhattacharyya, A., Bandyopadhyay, S., Ukil, A., Bose, T., and A. Pal, "Lightweight Establishment of Secure Session (LESS) on CoAP", draft-bhattacharyya-dice-less-on-coap-00 (work in progress), April 2015.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-00 (work in progress), October 2016.

[I-D.schmertmann-dice-codtls]
Schmertmann, L., Hartke, K., and C. Bormann, "CoDTLS: DTLS handshakes over CoAP", draft-schmertmann-dice-codtls-01 (work in progress), August 2014.

[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>.

[RFC2367] McDonald, D., Metz, C., and B. Phan, "PF_KEY Key Management API, Version 2", RFC 2367, DOI 10.17487/RFC2367, July 1998, <http://www.rfc-editor.org/info/rfc2367>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <http://www.rfc-editor.org/info/rfc4301>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[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>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <http://www.rfc-editor.org/info/rfc7296>.

Authors' Addresses

Dan Garcia Carrillo
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia 30100
Spain

Phone: +34 868 88 78 82
Email: dan.garcia@um.es

Sara Nieves Matheu Garcia
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia 30100
Spain

Phone: +34 868 88 78 82
Email: saranieves.matheu@um.es

Rafa Marin-Lopez
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia 30100
Spain

Phone: +34 868 88 85 01
Email: rafa@um.es

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: October 21, 2017

C. Groves

W. Yang
Huawei

April 19, 2017

A WebRTC Data Channel Transport for the Constrained Application Protocol (CoAP)
draft-groves-coap-webrtcdc-02

Abstract

The WebRTC framework defines a generic transport service allowing WEB-browsers and other endpoints to exchange generic data from peer to peer utilizing a Stream Control Transmission Protocol (SCTP) transport. This service is known as Web Real Time Communication WebRTC data channels (WebRTC DC). The use of WebRTC DCs for the Constrained Application Protocol (CoAP) allows WebRTC enabled devices to exchange CoAP data between peers in a secure reliable manner.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 21, 2017.

Copyright Notice

Table of Contents

1. Introduction

Whilst the Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments in constrained network environments its ready adoption has seen the use of it in a multitude of different network environments. For example [I-D.silverajan-core-coap-alternative-transports] provides use cases for alternate CoAP transports.

[I-D.ietf-core-coap-tcp-tls] highlights a number of issues using the native User Datagram Transport (UDP) and envisages deployments more closely integrated with a Web environment. It also proposes the use of the WebSocket protocol [RFC6455]. The use of CoAP over WebRTC DCs has not yet been discussed.

WebRTC is a framework [I-D.ietf-rtcweb-overview] that defines real time protocols for browser-based applications. It allows communications between peer WebRTC endpoints (e.g. browsers) without the need to communicate through a web server.

In addition to protocols for the realtime transport of audio and video, the transport of generic peer-to-peer non-media data has been defined using WebRTC DCs. The non-media data is transported using the Stream Control Transmission Protocol (SCTP) [RFC4960] encapsulated in the Datagram Transport Layer Security (DTLS) [RFC6347]. It allows both reliable and partially reliable transport and provides confidentiality, source authenticated and integrity protected transfers. The use of Interactive Connectivity Establishment (ICE) [RFC5245] allows network address translator (NAT) traversal. The SCTP/DTLS association may be shared with existing audio and video streams enabling multiplexing of several data streams over a single port further facilitating NAT traversal.

Use cases for WebRTC DCs (section 3.1/[I-D.ietf-rtcweb-data-channel] envisage scenarios where the real-time gaming experience is enhanced by additional object state information. Additional scenarios are considered where information such as heart rate sensor or oxygen saturation sensors could augment audio and video in remote medicine scenarios. The transport of such sensor information is what CoAP has been designed for.

This is illustrated in Figure 1 showing the WebRTC Trapeziod with added sensor/CoAP information. The left hand side WebRTC endpoint acts as a CoAP to CoAP proxy.

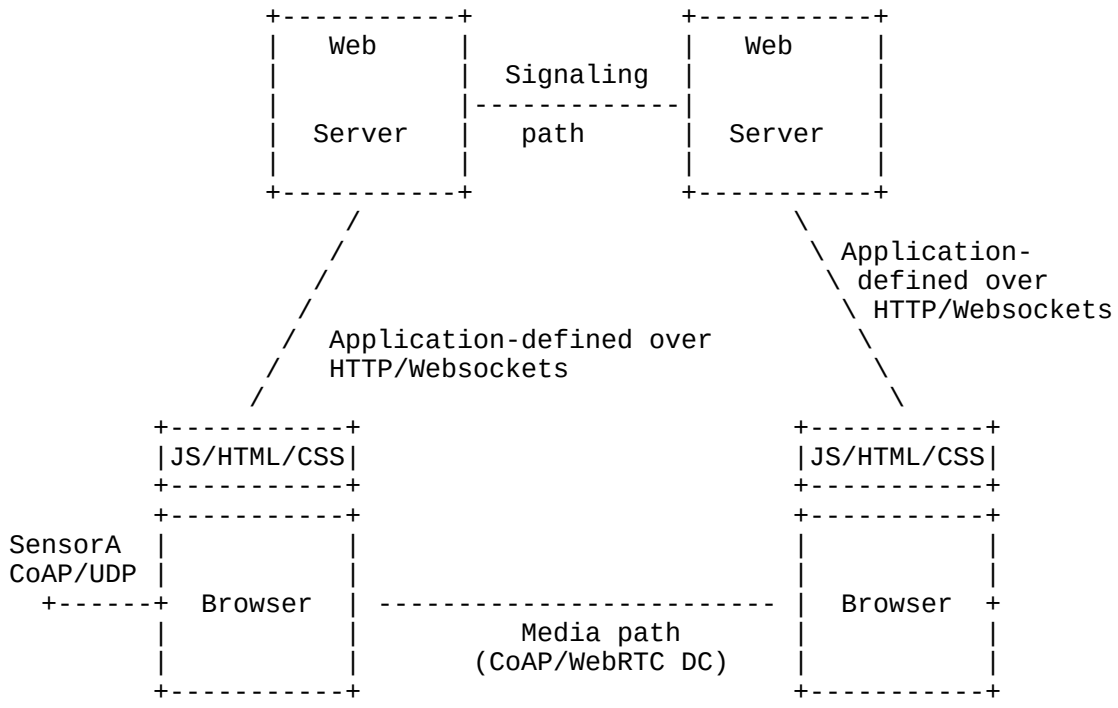


Figure 1: CoAP and WebRTC Trapeziod

By utilizing the WebRTC DC (SCTP over DTLS over ICE/UDP (or ICE/TCP)) transport for CoAP a number of important features are inherited including: congestion control, order and unordered messages delivery, large message transmission by providing segmentation and reassembly and multiple unidirectional streams. A more detailed analysis of the benefits of WebRTC DCs can be found in section 5/[I-D.ietf-rtcweb-data-channel]. [I-D.ietf-tsvwg-sctp-dtls-encaps] describes the usage of SCTP over DTLS.

WebRTC defines in-band and out-of-band methods for establishing a data channel and indicating its characteristics. The Data Channel Establishment Protocol (DCEP) [I-D.ietf-rtcweb-data-protocol] provides an in band means of establishing individual data channels. [I-D.ietf-mmusic-data-channel-sdpneg] uses the Session Description Protocol (SDP) [RFC4566] to provide an out-of-band means to establish data channels.

By defining the use of CoAP over WebRTC DC it negates the need for the WebRTC endpoint to interwork between any CoAP messages received from local devices to a proprietary WebRTC DC format when signalling a remote WebRTC endpoint.

The SCTP Payload Protocol Identifier (PPID) allows the identification of whether a UTF-8 or Binary encoding is being used and thus facilitates the use of text or binary CoAP protocol serializations.

2. Requirements Language

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

3. Constrained Application Protocol

This section describes the use of CoAP over WebRTC DC as a delta to the information contained in section 2/[RFC7252].

Figure 2 shows the CoAP abstract layering as applied to the WebRTC framework.

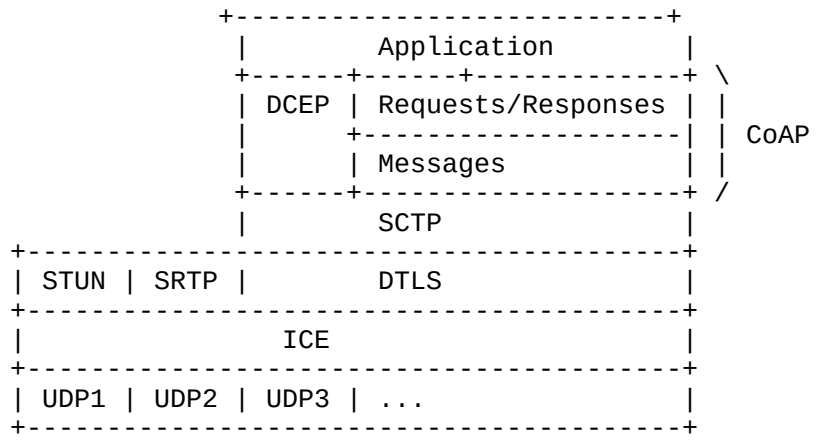


Figure 2: WebRTC protocol layers including CoAP

WebRTC DC mandates the use of SCTP over DTLS. Whilst the above diagram indicates the use of ICE over UDP the use of TCP is also possible in fall back scenarios.

3.1. Message Model

WebRTC DC allows application protocol messages to be exchanged by peers. WebRTC supports both a reliable and partially reliable methods of transmitting user messages.

CoAP [RFC7252] supports four message types "Confirmable, Non-Confirmable, Acknowledge and Reset". As SCTP provides the reliability mechanism the CoAP message types are not needed for CoAP over WebRTC DC.

WebRTC DC does not support multicast usage.

3.2. Request Response Model

WebRTC DCs are realized as a pair of one incoming and one outgoing SCTP stream (with the same identifier) allowing bi-directional communication. Each channel has properties (see section 6.4/[I-D.ietf-rtcweb-data-channel] as discussed below:

- reliable or unreliable message transmission: WebRTC DCs support the per message indication whether user messages are reliable or partially reliable. Partial reliability indicates that message retransmission is limited to a certain number of retransmissions or lifetime. This loosely parallels to the CoAP usage of Confirmable (CON) or Non-confirmable (NON) messages.

- in-order or out-of-order message delivery: WebRTC DCs support the per message indication whether user messages are delivered in or out of order. CoAP has been designed for unreliable transports and therefore assumes that messages may arrive out-of-order. CoAP implements a lightweight reliability mechanism to deal with this issue.

- priority: WebRTC DCs allows a priority to specified for stream scheduling. The usage of this is application specific. Usage of CoAP has no impact on this parameter. It’s up to the application using CoAP to set this indication.

- an optional label: This is an application/implementation specific label. Uniqueness is not guaranteed. Usage of CoAP has no impact on this parameter.

- an optional protocol: This is used to indicate the application protocol in use. A value is required to identify the usage of CoAP.

As discussed above WebRTC DC supports an unreliable / un-ordered delivery of messages. Implementations utilizing these data channel characteristics may use CoAP messages and request/response model largely unchanged. In this case the CoAP reliability mechanisms would be used. However as WebRTC DC’s usage of SCTP is reliable or partially reliable there is some redundancy between the functionality that WebRTC DCs and CoAP provides.

The redundancies are identified and discussed in section 2/[I-D.ietf-core-coap-tcp-tls]. Namely:

1. There is no need to carry acknowledgement semantics at a CoAP level.

2. There is no need for duplicate delivery detection. This is part of the SCTP layer.

3.3. Intermediaries and Caching

As CoAP over WebRTC DC is peer to peer no intermediares or caching is expected.

3.4. Resource Discovery

The usage of CoAP over WebRTC DC has no foreseeable impacts on resource discovery.

3.5. Opening Handshake

Prior to the establishment of a CoAP over WebRTC DC the characteristics of the SCTP association and data channel may be negotiated by signalling. See Section 4 for further details. For example when using SDP [I-D.ietf-mmusic-sctp-sdp] the use of the "SDP max-message-size" attribute indicates the maximum received SCTP message size.

Further characteristics (such as those described in Section 3.2) are negotiated at the establishment of the WebRTC DC.

On establishment of the CoAP over WebRTC DC the client and server MAY send a CoAP Capability and Settings message (CSM see Section 4.3/[I-D.ietf-core-coap-tcp-tls]) as its first message on the connection to establish CoAP specific capabilities. Any capabilities signalled SHALL not contradict previously negotiated chracteristics. Consideration for the individual options are below:

- Server-Name Setting: CoAP over WebRTC DC clients MAY use the server-name setting option. The initial value is derived based on the signalling method used to establish the WebRTC peer to peer communications. WebRTC does not mandate a signalling method. For example if Websockets is used then the value may be taken from the HTTP host header field.

- Max-message size Capability: The CoAP Max-Message-Size shall not exceed the SCTP message size.

- Block-wise Transfer Capability: CoAP over WebRTC DC client and server MAY support the use of BERT (Section 5/[I-D.ietf-core-coap-tcp-tls]). See Section 4.6 for message size considerations.

- Ping and Pong Messages: Ping and Pong messages MAY be sent by CoAP over WebRTC DC clients and servers. However its use as a basic keepalive is not required as WebRTC defines a method to determine liveness (see Section 4.1).

- Release Messages: CoAP over WebRTC DC clients and servers may support the CoAP Release message. On receipt of a release message the CoAP over WebRTC DC SHALL be closed as per Section 4.

- Abort Messages: CoAP over WebRTC DC clients and servers may support the CoAP Abort message. Senders SHALL then close the CoAP over WebRTC DC as per Section 4.

3.6. Message Format

As discussed in [I-D.ietf-core-coap-tcp-tls] the use of a reliable underlying transport allows the use of a modified CoAP header format. The modified format removes the "Type (T)" and "Message ID" fields and introduces a "length" as illustrated below in Figure 3.

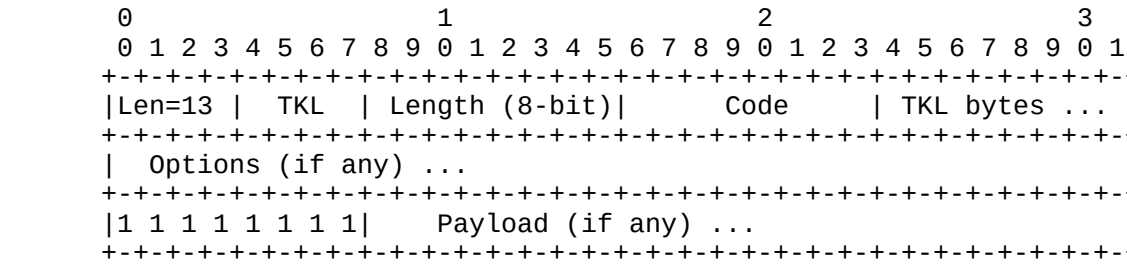


Figure 3: CoAP Header with TCP with 8-bit Length in Header

CoAP over WebRTC DC implementations shall also use the message format in Figure 3 with the following consideration:

- The length field was added for message delimitation to keep messages separate in TCP. WebRTC DC uses the message orientation of SCTP to preserve message boundaries thus the use of single application message per SCTP user message is mandated by the WebRTC framework. The length field shall be set to 0.

CoAP [RFC7252] supports the use of different content-formats. WebRTC DC defines the use of PPIDs per SCTP user message as follows:

- WebRTC String: to identify a non-empty JavaScript string encoded in UTF-8.

- WebRTC Binary: to identify a non-empty JavaScript binary data (ArrayBuffer, ArrayBufferView or Blob).

Depending on the content-format (see section 12.3/[RFC7252]) an appropriate PPID to the encoding type SHOULD be used to minimise the need for translating between encodings. For example content type of "text/plain" would result in the use of PPID "WebRTC String".

Author's note: Specific mappings for each content-format could be provided however given that the formats may change in the future it may be sufficient to offer broad guidance instead.

3.7. Option Format and Value

There are no impacts to option formats or values due to the use of CoAP over WebRTC DCs.

Author's note: Given that the host is determined by the usage of WebRTC are the Uri-Host and Uri-Port relevant? It would seem that this may be valuable to establish a resource tree independent of WebRTC.

4. Message Transmission

In order to use a WebRTC DC, a SCTP over DTLS over ICE/UDP (or ICE/TCP) association must be established. A DTLS connection is established followed by an SCTP association. The out-of-band establishment method through the use of SDP-based Data Channel Negotiation [I-D.ietf-mmusic-data-channel-sdpneg] allows the negotiation of SCTP over DTLS over ICE/UDP as well as the negotiation and establishment of the characteristics of an individual WebRTC DC.

The in-band establishment method through the use of the Data Channel Establishment Protocol (DCEP) [I-D.ietf-rtcweb-data-protocol] only allows for the establishment of a WebRTC DC once the SCTP over DTLS is established. It relies on DATA_CHANNEL_OPEN and DATA_CHANNEL_ACK messages on the relevant SCTP stream to negotiate the properties of the channel. A separate SCTP PPID (50) indicates that the SCTP user message is a WebRTC DCEP message to allow de-multiplexing by the endpoint.

WebRTC DCs are realized as a pair of one incoming and one outgoing SCTP stream (with the same identifier). Requests are sent on an outgoing SCTP stream and received on the peer incoming stream. The SCTP stream identifier is bound to the WebRTC DC instance at the

establishment of the data channel. The establishment protocol provides rules for determining the SCTP stream IDs.

WebRTC DC closure (Stream Reset) is supported through the use of the SCTP stream reconfiguration extension defined in [RFC6525]. The SCTP Stream Reconfiguration reset has the effect of setting the numbering sequence of the SCTP stream back to zero. This is separate function to the CoAP "Reset" message. There is no mapping between the SCTP Stream Reset and the CoAP "Reset" message.

4.1. Messages and Endpoints

As per section 2.5/[I-D.ietf-core-coap-tcp-tls] requests can be sent from both the connecting host and the endpoint that accepted the connection. Who initiated the SCTP/DTLS connection has no bearing on the meaning of the CoAP terms client and server.

WebRTC DC mandates the use of DTLS thus the endpoint is identified depending on the security mode.

WebRTC DCs allows the indication of whether a SCTP user message is empty through the use of PPIDs (WebRTC String Empty and WebRTC Binary Empty). CoAP defines the use of empty messages. However from the perspective of SCTP these CoAP messages would still contain header information thus PPIDs for empty data MUST not be used.

CoAP uses an Empty Confirmable message to provoke a Reset message to check the liveness of an endpoint (so called "CoAP" ping). In WebRTC liveness and the ability to send data is determined through the usage of Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness [RFC7675]. Therefore endpoints utilising CoAP over WebRTC DC MUST not use CoAP "reset" messages.

CoAP also uses Empty messages to acknowledge a request. This is not required due to the SCTP level acknowledgement. Therefore Empty messages MUST not be used with CoAP over WebRTC.

4.2. Messages Transmitted Reliably

For CoAP messages marked as confirmable the sender SHALL use a reliable SCTP user message.

A CoAP endpoint MUST use the ordered delivery SCTP service, as described in [RFC4960], for the CoAP protocol.

CoAP receivers MUST NOT generate CoAP "ACK" or "reset" messages. SCTP level acknowledgement mechanisms are used.

4.3. Messages Transmitted without Reliability

WebRTC DC makes use of the SCTP Partial Reliability (SCTP-PR) Extension [RFC3758]. This extension allows a user to indicate on a per message basis how persistent the transport service should be in attempting to send the message to the receiver. One of the benefits of using this extension identified by [RFC3758] is:

1. Some application layer protocols may benefit from being able to use a single SCTP association to carry both reliable content, - such as text pages, billing and accounting information, setup signaling - and unreliable content, e.g., state that is highly sensitive to timeliness, where generating a new packet is more advantageous than transmitting an old one.

This benefit is also one of the reasons the CoAP "Non-Confirmable" message was introduced. However the SCTP-PR and the CoAP "Non-Confirmable" message mechanisms differs in their approach. The SCTP-PR mechanism focuses on sender side behaviour (e.g. when to abandon retransmission). The CoAP "Non-Confirmable" message focuses on receiver side behaviour (e.g. must not send a CoAP ACK). Even with the use of SCTP-PR an SCTP receiver will send an SCTP level ACK for a successfully received SCTP CHUNK. The CoAP "Non-Confirmable" message has no effect on the SCTP level function.

Therefore the use of a CoAP "Non-Confirmable" message type is redundant as the CoAP receiver will never send a CoAP ACK message in response.

SCTP-PR provides a complimentary function and thus CoAP senders who send Non-confirmable messages SHALL also use SCTP-PR for that message.

4.4. Message Correlation

Due to reliability being handled at the SCTP layers the CoAP "Message ID" is not required.

4.5. Message Duplication

The SCTP layer provides message duplication protection. The CoAP application level procedure is not required.

4.6. Message Size

The considerations in section 4.1/[I-D.ietf-core-coap-tcp-tls] regarding message size limitations also apply to the use of WebRTC DCs. However [I-D.ietf-rtcweb-data-channel] indicates that senders

SHOULD limit the maximum message size to 16KB to avoid monopolization of the SCTP association. Section 5/[I-D.ietf-tsvwg-sctp-dtls-encaps] provides further details regarding segmentation and reassembly and path maximum transmission unit (MTU) discovery.

Interleaving of large user messages is supported by an SCTP protocol extension defined in [I-D.ietf-tsvwg-sctp-ndata].

4.7. Congestion Control

SCTP provides congestion control on a per-association basis (see section 5/[I-D.ietf-rtcweb-data-channel].

4.8. Transmission Parameters

The application level parameters defined in section 4.8/[RFC7252] are not relevant to SCTP.

5. Request/Response Semantics

Request and response semantics for CoAP over WebRTC DC is as per section 5/[RFC7252] with the following exceptions:

- section 5.2/[RFC7252]: separate responses MUST be used. Given that WebRTC DC provides an SCTP level acknowledgement it is not possible to piggy back CoAP responses.
- section 5.3.1/[RFC7252]: due to the use of DTLS the advice regarding token use without using TLS is invalid.
- section 5.3.2/[RFC7252]: In addition CoAP request/response matching is unique to a particular WebRTC DC (SCTP StreamID pair).
- section 5.8/[RFC7252]: It is not possible to use a 4.05 piggybacked response.

6. CoAP URI

CoAP [RFC7252] defines the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. [RFC7252] defines these resources for use with CoAP over UDP.

Section 8/[RFC7252] (Multicast CoAP), does not apply to the URI schemes defined in the present specification.

Resources made available via the "coaps+wr" schemes have no shared identity with the other scheme or with the "coap" or "coaps" scheme,

even if their resource identifiers indicate the same authority (the same host listening to the same port). The schemes constitute distinct namespaces and, in combination with the authority, are considered to be distinct origin servers.

6.1. coaps+wr URI scheme

```
coaps-wr-URI = "coaps+wr:" "//" host [ ":" port ] path-abempty
                  [ "?" query ]
```

The semantics defined in section 6.3/[RFC7252], apply to this URI scheme, with the following changes:

- The port SHALL be omitted. The underlying UDP or TCP port and SCTP port is negotiated prior to the establishment of the CoAP over WebRTC DC.

7. Discovery

7.1. Service Discovery

WebRTC does not define peer discovery mechanisms. Peers discover each other through the use of the ICE protocol. ICE candidates need to be sent from peer to peer via signalling. The Javascript Session Establishment Protocol (JSEP) [I-D.ietf-rtcweb-jsep] details the generic SDP media descriptions for peer endpoints to determine the characteristics of a session. The actual signalling protocol between application servers is unspecified. WebRTC endpoints MUST implement the network functions detailed by JSEP including ICE functionality.

Whilst the inter-application server signalling protocol is unspecified, the Session Initiation Protocol (SIP) is able to carry SDP for the purposes of establishing a CoAP over WebRTC DC session. SIP allows the use of media feature tags to indicate user agent capabilities [RFC3840]. In order to indicate that a SIP user agent supports the use of CoAP a new "sip.coap" media feature tag is proposed. A CoAP-capable endpoint SHOULD include this media feature tag in its REGISTER requests and OPTION responses. It SHOULD also include the media feature tag in INVITE and UPDATE [RFC3311] requests and responses. Presence of the media feature tag in the contact field of a requestor response can be used to determine that the far end supports CLUE.

The exchange of SDP results in: the underlying transport address (e.g. IPv4 or IPv6), the underlying transport port (e.g. UDP port) the SCTP port and the SCTP StreamID used for the CoAP WebRTC DC being exchanged between the peer endpoints.

7.2. Resource Discovery

On establishment of a CoAP WebRTC DC endpoints are able to use the resource discovery mechanism defined in [RFC6690] for CoAP resources.

8. Multicast CoAP

WebRTC DCs do not support multicast.

9. Securing CoAP

This document defines how to convey CoAP over WebRTC DCs. The WebRTC security architecture [I-D.ietf-rtcweb-security-arch] mandates the use of DTLS for data channels. The use of DTLS 1.2 is compatible with CoAP [RFC7252] which allows makes use of DTLS 1.2.

The use of DTLS for WebRTC is detailed in [I-D.ietf-rtcweb-security-arch].

10. Interworking

An WebRTC endpoint supporting CoAP may in affect act as a gateway between local sensor devices and a remote peer endpoint. The local sensors may utilise CoAP over an alternate signalling transport such as UDP to the local WebRTC endpoint. The WebRTC endpoint may then utilise CoAP over WebRTC to signal to the remote peer.

A CoAP gateway when converting to and from a WebRTC transport will in general perform the following functions:

- Map received Empty CoAP message to SCTP level operations and discard the empty message.
- Map received ACK message to SCTP level operations and discard the ACK message.
- Separate piggy-backed messages.
- Provide a mapping between received and sent Tokens in order to match requests and responses.

Other behaviour depends on the type of proxy behaviour the gateway is performing. See section 5.7/[RFC7252] for more details.

11. Security Considerations

Security considerations for WebRTC are discussed in [I-D.ietf-rtcweb-security].

The use of CoAP over WebRTC can potentially negate the risks mentioned in:

- section 11.3/[RFC7252] on insecure UDP and multicast being used to aid an amplification attack.
- section 11.4/[RFC7252] on IP address spoofing and section 11.5/[RFC7252] on Cross-Protocol attacks.
- section 11.6/[RFC7252] may also not be relevant as WebRTC endpoints are not expected to be severely constrained.

Of particular relevance to the support of CoAP over WebRTC DC is access to local devices. Devices generating CoAP data are essentially the same as cameras and microphones in that they may expose sensitive data about the user or the location of the device. Thus the guidance of section 4.1/[I-D.ietf-rtcweb-security] applies to devices generating CoAP data. Whilst CoAP has been designed for constrained devices where there is no user interface to inform/ request consent, it is assumed that device utilising WebRTC DC for CoAP is more likely at minimum a Class 2 [RFC7228] device that could facilitate consent.

The CoAP media feature tag defined by this document tag may be present in sessions not utilising CoAP, which increases the metadata available about the sending device, which can help an attacker differentiate between multiple devices and help them identify otherwise anonymised users via the fingerprint of features their device supports. To prevent this, SIP signalling SHOULD always be encrypted using TLS [RFC5630].

12. IANA Considerations

12.1. New WebRTC DC Protocol Value

NOTE: This registration is exactly the same as the registration in [I-D.savolainen-core-coap-websockets].

This document requests the registration of the subprotocol name "coap.v1" in the WebSocket Subprotocol Name Registry.

- Subprotocol Identifier: coap.v1

- Subprotocol Common Name: Constrained Application Protocol (CoAP)

- Subprotocol Definition: This document

12.2. Secure Service Name and Port Number Registration

No need has been identified to register a new service name and port number for CoAP over WebRTC. Port number allocation is dynamic. The use of the SCTP over DTLS over UDP/TCP results in a layering of services.

12.3. ALPN Protocol ID

[I-D.ietf-core-coap-tcp-tls] defines a new "coap" application protocol negotiation protocol identity. However as the DTLS connection is used to establish a WebRTC application the protocol identifiers defined in [I-D.ietf-rtcweb-alpn] MUST be used. Note: that confidentiality protection does not extend to WebRTC DCs.

12.4. URI Schemes

This document registers a new URI scheme "coaps+wr" for the use of CoAP over WebRTC DCs. The "coaps+wr" URI schemes can be compared to the "https" URI scheme.

The IANA is requested to add this new URI schemes to the registry established with [RFC7595].

12.5. New SIP Media Feature Tag

This specification registers a new media feature tag in the SIP [RFC3264] tree per the procedures defined in [RFC2506] and [RFC3840].

Media feature tag name: sip.coap

ASN.1 Identifier: 1.3.6.1.8.4.30

Summary of the media feature indicated by this tag: This feature tag indicates that the device supports the Constrained Application Protocol (CoAP).

Values appropriate for use with this feature tag: Boolean.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is useful to indicate the support of CoAP.

Related standards or documents: This document

Security Considerations: Security considerations for this media feature tag are discussed in Section 11.

Name(s) and email address(es) of person(s) to contact for further information:

- CORE workgroup: core@ietf.org
- CORE chairs: core-chairs@ietf.org

Intended usage: COMMON

13. Examples

The example SDP Offer shows a CoAP over WebRTC DC utilising out-of-band negotiation [I-D.ietf-mmusic-data-channel-sdpneg]. It is based on the example in section 7.2/[I-D.ietf-rtcweb-jsep]. Modified lines are indicated with ">>>" at the start of the line. These indicators are NOT part of the SDP syntax. Note: some lines have been broken into two lines for formatting reasons.

```
   v=0
      o=- 4962303333179871723 1 IN IP4 0.0.0.0
      s=-
      t=0 0
      a=group:BUNDLE a1 d1
      a=ice-options:trickle
      m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
      c=IN IP4 0.0.0.0
      a=rtcp:9 IN IP4 0.0.0.0
      a=mid:a1
      a=msid:57017fee-b6c1-4162-929c-a25110252400
             e83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
      a=sendrecv
      a=rtpmap:96 opus/48000/2
      a=rtpmap:0 PCMU/8000
      a=rtpmap:8 PCMA/8000
      a=rtpmap:97 telephone-event/8000
      a=rtpmap:98 telephone-event/48000
      a=maxptime:120
      a=ice-ufrag:ATEn1v9DoTMB9J4r
      a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
      a=fingerprint:sha-256
                    19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04
                   :BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
      a=setup:actpass
      a=rtcp-mux
      a=rtcp-rsize
      a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
      a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
      a=ssrc:1732846380 cname:FocUG1f0fcg/yvY7

      m=application 0 UDP/DTLS/SCTP webrtc-datachannel
      c=IN IP4 0.0.0.0
      a=bundle-only
      a=mid:d1
      a=fmtp:webrtc-datachannel max-message-size=65536
      a=sctp-port 5000
      a=fingerprint:sha-256
                    19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04
                    :BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
      a=setup:actpass
   >>>a=dcmap:0 subprotocol="coap.v1";"label="coap"
```

Figure 4: Example SDP Offer

14. Acknowledgements

We would like to thank the authors of [I-D.ietf-core-coap-tcp-tls] and [I-D.savolainen-core-coap-websockets] for providing a framework for this document. In addition we would like to thank Carsten Bormann for his feedback on message format.

15. Changelog

draft-groves-coap-webrtcdc-02:

- Keep alive update. No changes.

draft-groves-coap-webrtcdc-01:

- Updated message format to align with draft-core-coap-tcp-tls-04

- Updates to align with draft-core-coap-tcp-tls-04 as a result of the merger with websockets. Added section on opening handshake. Added support of CoAP capability messages and BERT.

16. References

16.1. Normative References

[I-D.ietf-mmusic-data-channel-sdpneg]
Drage, K., Makaraju, M., Stoetzer-Bradler, J., Ejzak, R., and J. Marcon, "SDP-based Data Channel Negotiation", draft-ietf-mmusic-data-channel-sdpneg-12 (work in progress), March 2017.

[I-D.ietf-mmusic-sctp-sdp]
Holmberg, C., Shpount, R., Loreto, S., and G. Camarillo, "Session Description Protocol (SDP) Offer/Answer Procedures For Stream Control Transmission Protocol (SCTP) over Datagram Transport Layer Security (DTLS) Transport.", draft-ietf-mmusic-sctp-sdp-25 (work in progress), March 2017.

[I-D.ietf-rtcweb-alpn]
Thomson, M., "Application Layer Protocol Negotiation for Web Real-Time Communications (WebRTC)", draft-ietf-rtcweb-alpn-04 (work in progress), May 2016.

[I-D.ietf-rtcweb-data-channel]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-13 (work in progress), January 2015.

[I-D.ietf-rtcweb-data-protocol]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Establishment Protocol", draft-ietf-rtcweb-data-protocol-09 (work in progress), January 2015.

[I-D.ietf-rtcweb-jsep]
Uberti, J., Jennings, C., and E. Rescorla, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-20 (work in progress), March 2017.

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-18 (work in progress), March 2017.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-08 (work in progress), February 2015.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-12 (work in progress), June 2016.

[I-D.ietf-tsvwg-sctp-dtls-encaps]
Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "DTLS Encapsulation of SCTP Packets", draft-ietf-tsvwg-sctp-dtls-encaps-09 (work in progress), January 2015.

[I-D.ietf-tsvwg-sctp-ndata]
Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-ndata-09 (work in progress), March 2017.

[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>.

[RFC2506] Holtman, K., Mutz, A., and T. Hardie, "Media Feature Tag Registration Procedure", BCP 31, RFC 2506, DOI 10.17487/RFC2506, March 1999, <http://www.rfc-editor.org/info/rfc2506>.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <http://www.rfc-editor.org/info/rfc3264>.

[RFC3311] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, DOI 10.17487/RFC3311, October 2002, <http://www.rfc-editor.org/info/rfc3311>.

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, DOI 10.17487/RFC3758, May 2004, <http://www.rfc-editor.org/info/rfc3758>.

[RFC3840] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, DOI 10.17487/RFC3840, August 2004, <http://www.rfc-editor.org/info/rfc3840>.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <http://www.rfc-editor.org/info/rfc4566>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://www.rfc-editor.org/info/rfc4960>.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <http://www.rfc-editor.org/info/rfc5245>.

[RFC5630] Audet, F., "The Use of the SIPS URI Scheme in the Session Initiation Protocol (SIP)", RFC 5630, DOI 10.17487/RFC5630, October 2009, <http://www.rfc-editor.org/info/rfc5630>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, DOI 10.17487/RFC6525, February 2012, <http://www.rfc-editor.org/info/rfc6525>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[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>.

[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>.

[RFC7675] Perumal, M., Wing, D., Ravindranath, R., Reddy, T., and M. Thomson, "Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness", RFC 7675, DOI 10.17487/RFC7675, October 2015, <http://www.rfc-editor.org/info/rfc7675>.

16.2. Informative References

[I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-07 (work in progress), March 2017.

[I-D.savolainen-core-coap-websockets]
Savolainen, T., Hartke, K., and B. Silverajan, "CoAP over WebSockets", draft-savolainen-core-coap-websockets-07 (work in progress), June 2016.

[I-D.silverajan-core-coap-alternative-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-09 (work in progress), December 2015.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <http://www.rfc-editor.org/info/rfc6455>.

Authors' Addresses

Christian Groves

Email: cngroves.std@gmail.com

Weiwei Yang
Huawei

Email: tommy@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

C. Groves
W. Yang
Huawei

March 13, 2017

Binding Attribute Scope
draft-groves-core-bas-01

Abstract

This document specifies an additional CoAP binding attribute that allows binding attributes to be scoped to an item (sub-resource) in a collection resource. This allows synchronisation of multiple resources in a collection based on the value of one resource.

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 14, 2017.

Table of Contents

1. Requirements Language

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].

2. Introduction

In some cases a CoAP client may require a notification of the value of a resource based on some condition associated with another resource. For example: a client wants to be notified of the machine state once a temperature sensor threshold is crossed. Currently the only way to implement this is for a client to be notified of the temperature resource change and then for the client to do a seperate read of the required resources. It would be more efficient to provide the information in a single response.

This document defines a new "Binding Attribute Scope" binding attribute to allow a client to request collection resource state synchronisation based on a conditional value of an item in the collection resource.

The "Binding Attribute Scope" (bas) attribute is set on a collection and is used to indicate which item in a collection an Observe and associated binding attributes apply to. A linked batch (or batch) interface (sect.4.3/[I-D.ietf-core-interfaces]) is used on the collection. The client constructs a collection resource using the

linked batch interface or uses a pre-provisioned collection via the batch interface to set the information that it requires. By placing the required information in a collection it allows a GET on the collection to return all the information based on the conditional value of an item.

Section Section 4 below indicates the text that would need to be added to [I-D.ietf-core-dynlink] to add the Binding Attribute Scope attribute.

2.1. Usage of a collection

[I-D.ietf-core-interfaces] indicates that a collection may be observed. An observation returns a representation of the entire collection. The "bas" binding attribute is only used on collection (e.g. batch, linked-batch) resources which enables the use of the binding attributes from [I-D.ietf-core-dynlink] on a collection. The "bas" attribute points to one item in the collection.

This approach does have the downside that it requires the use of a collection interface even if a single resource is required. The scope of the synchronization is limited to the current collection. However the use of a collection to return the information does seem to fit with the principle that a GET of a resource should return the same representation as a notification.

2.2. Multiple Conditions

Given that the "bas" binding attribute only applies to one item/sub-resource in a collection it means that multiple resources cannot be used as conditions for a binding synchronization. To allow the use of conditions on multiple resources to determine resource synchronization a different approach would be required.

One approach would be to use the FETCH method [I-D.ietf-core-etch] with a set of request parameters. A content type could be defined that allowed the binding attributes to be specified on multiple resources.

For example:

```
FETCH /s/?pmin=1&pmax=100 content-type=application/conditionals+json

[
  {
    "n": "/s/light",
    "st": 5
  },
  {
    "n": "/s/temp",
    "st": 1
  },
  {
    "n": "/s/humidity",
    "lt": 40,
    "gt" 70
  }
]
```

This approach would add complexity over the use of the bas binding attribute however such complexity may be warranted in certain use cases. A new content-format would need to be defined. This approach is for further study.

3. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690].

4. Binding Attributes

This document introduces one new attributes:

Attribute Name	Parameter	Data Format
Binding Attribute Scope	bas	xsd:string

Table 1: Binding Attribute Summary

The xsd:string contains a Uri-Path.

4.1. Binding Attribute Scope

The binding attribute scope attribute allows a client to indicate that the binding attributes contained in the query apply to a certain item in a collection resource.

The collection containing the Observed resource and the additional resources to be returned is first created through the use of the Batch or Linked-Batch [I-D.ietf-core-interfaces] interface. The Linked-Batch allows a client to dynamically associate resources.

The client then creates a binding (e.g. an Observe) with the collection indicating the required binding attributes (e.g. "gt", "lt" etc.) and to which sub-resource these apply through the use of the "bas" binding attribute containing a URI-path.

A client may create multiple bindings for the collection. When using Observe a client may use mutliple GET Observes with different queury parameters. Each observation is identified through the use of different Tokens.

If the server determines that the "bas" attribute contains an unknown URI-path then it responds with response code 4.04 "Not Found".

State sychronisation occurs when the sub-resource (item) value meets the conditions indicated by the binding attributes. When state synchronisation occurs the collection resource (including sub-resources) will be synchronised.

4.2. Interactions

The "bas" parameter modifies the scope of other binding attributes in a query to apply to the resource specified in the "bas" URI-path.

Editor’s note: the interaction with sample time window/sample number window needs to be added if accepted.

4.3. Examples

Given the resource links:

```
Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s/>;rt="simple.sen";if="core.b",
</s/light>;rt="simple.sen.light";if="core.s",
</s/temp>;rt="simple.sen.tmp";if="core.s";obs,
</s/humidity>;rt="simple.sen.hum";if="core.s"
```

4.3.1. Example 1 - Item Binding Attribute

```
A Req: GET /s?bas="temp"&gt=37
          Token: 0x4a
          Observe: 0
would produce the following when temp exceeds 37:

  Res: 2.05 Content (application/senml+json)
  Token: 0x4a
  {"e":[
     { "n": "/s/light", "v": 123, "u": "lx" },
     { "n": "/s/temp", "v": 38, "u": "degC" },
     { "n": "/s/humidity", "v": 80, "u": "%RH" }],
  }
```

4.3.2. Example 2 - Mutliple Observes

In addition to the GET in example 1 the client could also request a notification when the humidity raises above 90%.

```
A Req: GET /s?bas="humidity"&gt=90
          Token: 0x4b
          Observe: 0
would produce the following when temp exceeds 37:

  Res: 2.05 Content (application/senml+json)
  Token: 0x4b
  {"e":[
     { "n": "/s/light", "v": 123, "u": "lx" },
     { "n": "/s/temp", "v": 16, "u": "degC" },
     { "n": "/s/humidity", "v": 92, "u": "%RH" }],
  }
```

Editor’s note: Need to add example for pmin

5. Security Considerations

As per 5/[I-D.ietf-core-dynlink].

6. IANA Considerations

None.

7. Acknowledgements

Michael Koster for his comments and suggestion of the FETCH approach.
Friedhelm Rodermund for stimulating discussion of the use case.

Mojan Mohajer for raising the multiple observes on a collection use case.

8. Changelog

draft-groves-core-bas-01

- Section 2: Removed editor's note on alternate solution
- Section 2.1: Changed language around observe on collections.
- Section 4.1: Added text regarding multiple observations.

draft-groves-core-bas-00

- Initial version

9. Normative References

[I-D.ietf-core-dynlink]
Shelby, Z., Vial, M., Koster, M., and C. Groves, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-02 (work in progress), February 2017.

[I-D.ietf-core-etch]
Stok, P., Bormann, C., and A. Sehgal, "Patch and Fetch Methods for Constrained Application Protocol (CoAP)", draft-ietf-core-etch-04 (work in progress), November 2016.

[I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., Koster, M., and C. Groves, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-08 (work in progress), February 2017.

[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>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

Authors' Addresses

Christian Groves
Huawei
Australia

Email: cngroves.std@gmail.com

Weiwei Yang
Huawei
P.R.China

Email: tommy@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 25, 2017

C. Groves
W. Yang
Huawei

February 21, 2017

Additional CoAP Binding and Observe Attributes
draft-groves-core-obsattr-00

Abstract

[I-D.ietf-core-dynlink] defines five CoAP Observaton attributes (minimum period, maximum period, band step, less than and greater than) to control when notifications are sent. These attributes are insufficient for some use cases. This document specifies additional attributes allowing for notification bands, initialization values, band step, sample number window and sample time window to allow for a wider range of use cases.

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 25, 2017.

Copyright Notice

Table of Contents

1. Requirements Language

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].

2. Introduction

[I-D.ietf-core-dynlink] defines five CoAP Binding and Observaton attributes (minimum period, maximum period, change step, less than and greater than) to control when notifications are sent. The currently defined attributes have characteristics that means some use cases cannot be supported. These are described below:

- A transition across a less than (lt), or greater than (gt) limit or a change step (st) only generates one notification. This means that it is not possible to describe a case where multiple notifications are sent so long as the limit is exceeded.

- The change step (st) value is not deterministic when setting the attribute. A client cannot set the initial value that the change step applies to. The change step is based on the initial value sent by the server. This means that a client cannot indicate that it wants the change step (st=10) to apply to a certain increment (e.g. 10, 20, 30) instead it relies on the initial value from the client. Thus a change step of 10 could result in reporting (11, 21, 31) or equally (15, 27, 53).

- SenML allows for multiple values (records) to be reported for a resource. The current attributes do not allow a method for a client to request a particular number of records or sample time window.

In order to allow a more complete set of use cases to be supported this specification introduces several new attributes.

The notification band attributes "Notification Band Minimum" (bmn) and "Notification Band Maximum" (bmx) attributes allow a bounded or unbounded (based on a minimum or maximum) value range that may trigger multiple state synchronizations. This enables use cases where different ranges results in differing behaviour. For example: monitoring the temperature of machinery. Whilst the temperature is in the normal operating range only periodic observations are needed. However as the temperature moves to more abnormal ranges more frequent synchronization/reporting may be needed.

An "Initialization Value" (iv) attribute allows a seed value for the calculation of the change step to be specified. This allows use cases where synchronization occurs around a known value. For example: synchronization will occur based on the operating temperature set point of a machine. Without the initialization synchronization will occur around the first measured value.

A "Band Step" (bst) attribute defines a series of bands that will trigger state synchronization. This allows use cases where state synchronization is required against known levels. Rather than synchronizing based on a difference to a previous synchronization value, synchronization occurs against a fixed known level. For example: it allows state sychronisation for a sensor when it’s value is between (5,10],(10,15],(15,20] etc.

The "Sample Number Window" (snw) attribute allows a number of state synchronizations to be queued before the actual queue synchronization occurs. Once the number of queued state synchronizations has reached a certain level then a single queue synchronization occurs with the multiple resource values related to individual state synchronizations included. This allows use cases where multiple resource values are required but frequent synchronization is not required as there is a need to minimise resource usage. For example: a meter may need to be recorded once an hour but the values only need to be synchronized once a day.

The "Sample Time Window" (stw) attribute as per the sample number window allows state synchronizations to be queued before the actual queue synchronization occurs. The queue synchronization occurs when the indicated period expires independent of the number of samples.

3. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. This specification makes use of the following additional terminology:

Notification Band: A resource value range that results in state sychronization. The value range may be bounded by a minimum and maximum value or may be unbounded having either a minimum or maximum value.

4. Binding and Resource Observation Attributes

The attributes defined in this section are additional attributes that may be used as binding attributes (3.3/[I-D.ietf-core-dynlink]) and resource observation attributes (4.2/[I-D.ietf-core-dynlink]).

This specification introduces several new attributes:

Attribute Name	Parameter	Data Format
Initialialization Value	/{resource}?iv	xsd:decimal
Band Minimum Notification	/{resource}?bmn	xsd:decimal
Band Maximum Notification	/{resource}?bmx	xsd:decimal
Band Step	/{resource}?bst	xsd:decimal (>0)
Sample Number Window	/{resource}?snw	xsd:integer (>0)
Sample Time Window	/{resource}?stw	xsd:integer (>0)

Table 1: Resource Observation Attribute Summary

The attributes may only be included at most once in a query.

4.1. Initialization Value (iv)

The attribute indicates the initialization value to be used to determine when a change step is notified. As such it MUST only be present in a query when the change step (st) or band step (bst) attribute is present (see [I-D.ietf-core-dynlink]). If st or bst is not present then the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

Without iv, on reception of a query the synchronization initiator uses the current value for the observed resource as the initial value to which the change step is applied. The use of iv overrides this behaviour and the iv value is used for the initial value (STinit or BSTinit). A state synchronization occurs once the resource value differs from the initial value by the change step value (i.e. CurrVal >= STinit + ST or CurrVal <= STint - ST). The initial value is then set to the state synchronization value.

A state synchronization due to Pmax (or Pmin) does not cause an update of the initial value. However once the initial value is updated by a state synchronization due to the other attributes in the query then the normal behaviour defined by 3.3.7/[I-D.ietf-core-dynlink] occurs.

4.2. Notification Band Minimum (bmn)

This attribute defines the lower bound for the notification band. State synchronization occurs when the resource value is equal to or above the notification band minimum. This attribute is optional. If not present there is no minimum value for the band. If present bmn must be less than bmx if it is also present otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

4.3. Notification Band Maximum (bmx)

This attribute defines the upper bound for the notification band. State synchronization occurs when the resource value is equal to or less than the notification band maximum. This attribute is optional. If not present there is no maximum value for the band. If present bmx must be more than bmn if it is also present, otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

4.4. Band Step (bst)

Like change step (st) this attribute indicates how much the value of a resource SHOULD change before triggering a state synchronization. The difference however is that the values used for the band step calculation are based on a constant step rather than being based on the synchronized value.

The current resource value or the initialization value (if provided) is used a seed to determine the band thresholds.

For example: Given a bst=10 and an initialization value=25. This defines a series of band step thresholds: i.e. ..., (5,15],(15,25],(25,35], ...

When the resource value enters a new band step by exceeding the minimum threshold value and being less than or equal to the maximum threshold value for a band step then synchronisation occurs.

A new synchronization occurs whenever the value enters a new band step. If the value jumps across band steps e.g. from 13 to 27 only one synchronisation occurs.

The band step MUST be greater than zero otherwise the server MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

Change step (st) and band step (bst) MUST NOT occur together in the same query.

4.5. Sample Number Window

If queuing of a number of state synchronizations are required then the sample number window attribute is set to the desired size of the window. The attribute may be set with valid combinations of other binding/resource observation attributes. When a state synchronization is triggered due to the other attributes the resource value is added to the list of samples instead of resulting in an update of the source and destination resource (state synchronization). Only when the number of samples in the window reaches the sample number window is a state sycnhronisation peformed for the resource. The samples are then flushed from the window and the process is repeated.

The use of the sample number window attribute may require the use of a suitable content-format (such as SenML [I-D.ietf-core-senml]) that allows multiple values/data points to be specified during the state synchronization.

Consideration should also be given to the resource capacity (i.e. memory) of the CoAP server for storing data associated with the sample window. The sample number window should not exceed its capabilities. Even if the sample number window has not been reached, if resource (memory) consumption is an issue then state synchronization for the stored resource values SHOULD occur enabling resources to be freed.

The pmin and pmax attributes have an indirect effect on the overall state sychronization. Whilst pmin and pmax do not directly specify the period for the overall state sychronization the setting of pmin and pmax may trigger samples entering the sample window as thus affect the frequency of state synchronization.

4.6. Sample Time Window

If state synchronizations are to be queued during a certain period of time (in seconds) then the sample time window attribute is used. The attribute may be set with valid combinations of other binding/ resource observation attributes. On reception of a query with the stw attribute a timer (T1=0) is started. Whilst T1<stw when a state synchronization is triggered due to the other attributes, the resource value is added to the sample window instead of resulting in a state synchronization. When the time expires e.g. T1=stw the state sychronization for the resource occurs. The window is then flushed, T1 is re-started and the process is repeated.

The use of the sample time window attribute may require the use of a suitable content-format (such as SenML [I-D.ietf-core-senml]) that

allows multiple values/data points to be specified during the synchronization.

Consideration should also be given to the resource capacity (i.e. memory) of the CoAP server for storing data associated with the time window. Consideration should be given to that the expected frequency of adding resource values and length of the time doesn't exceed the memory capacity of the server. Even if the sample time window has not been reached, if resource (memory) consumption is an issue then state synchronization for the stored resource values SHOULD occur enabling resources to be freed.

4.7. Interactions

To enable an notification band at least bmn or bmx MUST be set. If both bmn and bmx are set then a finite band is specified. State synchronization occurs whenever the resource value is between bmn and bmx or is equal to bmn or bmx. If only one attribute bmx or bmn is set then the band has an open bound. That is all values above bmn or all values below bmx will be synchronized.

When using multiple resource bindings (e.g. multiple Observations of resource) with different bands, consideration should be given to the resolution of the resource value when setting sequential bands. For example: Given BandA (Abmn=10, Bbmx=20) and BandB (Bbmn=21, Bbmx=30). If the resource value returns an integer then notifications for values between and inclusive of 10 and 30 will be triggered. Whereas if the resolution is to one decimal point (0.1) then notifications for values 20.1 to 20.9 will not be triggered.

Note: The use of bmn and bmx allow for a synchronization whenever a change in the resource value occurs. Theoretically this could occur in-line with the server internal sample period for the determining the resource value. Implementors SHOULD consider the resolution needed before updating the resource, e.g. updating the resource when a temperature sensor value changes by 0.001 degree versus 1 degree.

If pmin and pmax are present in a query then they take precedence over the other parameters. Thus even if bmn and bmx are met if pmin has not been exceeded then no state synchronization occurs. Likewise if bmn and bmx have not been met and pmax time has expired then state synchronization occurs. The current value of the resource is used for the synchronization. If pmin time is exceeded and bmn and bmx are met then the current value of the resource is synchronized. If st is also included, a state synchronization resulting from pmin or pmax updates STinit with the synchronized value. If bst is included, a state synchronization resulting from pmin or pmax updates bstinit with the closest bst delta value as per Section 4.4.

It is an error to include a greater than (gt) or less than (lt) attribute in a query containing bmn or bmx.

If change step (st) is included in a query with bmn or bmx then state synchronization will occur whilst the resource value is in the notification band AND the resource value differs from STinit by the change step.

If band step (bst) is included in a query with bmn or bmx then state synchronization will occur whilst the resource value is in the notification band defined by bmn or bmx AND has entered a new bandstep band.

If bst is included in a query with a gt or lt attribute then state synchronizations occur only when the conditions described by bst AND gt or bst AND lt are met.

If bst is included in a query with a iv attribute then iv is used to calculate the band thresholds. Subsequent state synchronizations are as per Section 4.4.

If iv is included with the bmn and bmx or gt and lt attributes it has no affect on the synchronisation. If bst is also used then it used to calculate band step thresholds. If st is instead used then it is used to calculate the initial value.

The snw and stw attributes SHALL not be used in the same query together. Synchronizations based on pmin and pmax are added to the snw/stw sample window. In effect this overrides the pmin and pmax mechanism because resource state synchronizations will not occur between the source and destination resources based on these parameters. For snw the minimum period will be snw * pmin. The maximum period will be snw * pmax. For stw the state synchronization will occur after time stw and is not dependent on pmin and pmax (unless a state synchronization occurs due to memory constraints). The stw must be more than pmax if it is present, otherwise the pmax attribute becomes invalid.

4.8. Examples

4.8.1. Example 1 - Band Minimum and Maximum

Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto";anchor="/a/temperature";bind="obs";pmin="10";pmax="60";bmn="20",bmx="40"

Res: 2.04 Changed

The above will result in a state synchronization through an Observe:

- Every 60 seconds if the value is not between 20 and 40.
- When the temperature is equal to or between 20 and 40 at least every 10 seconds.

4.8.2. Example 2 - Band Minimum and Maximum and Step

Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto";anchor="/a/temperature";bind="obs";pmin="10";pmax="60";bmn="20",bmx="40",st="5"

Res: 2.04 Changed

The above will result in:

- STinit being set to the temperature value at the time of the POST.
- A state synchronization through an Observe:
  - Every 60 seconds if the value is not between 20 and 40 and if the value has not changed by 5
  - When the temperature is equal to or between 20 and 40 and the value has changed by 5 from STinit at least every 10 seconds.

4.8.3. Example 3 - Band Minimum and Maximum, Step and Initialization Value

Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto";anchor="/a/temperature";bind="obs";pmin="10";pmax="60";bmn="20",bmx="40",st="5",iv="20"

Res: 2.04 Changed

The above will result in:

- STinit being set to 20 due to iv.
- A state synchronization through an Observe:
  - Every 60 seconds if the value is not between 20 and 40 and if the value has not changed by 5 from 20
  - When the temperature is equal to or between 20 and 40 and the value has changed by 5 from STinit at least every 10 seconds.

4.8.4. Example 4 - Step and Initialization Value

Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto";anchor="/a/temperature";bind="obs";pmin="10";pmax="60";st="5",iv="20"

Res: 2.04 Changed

The above will result in:

- STinit being set to 20 due to iv.
- A state synchronization through an Observe:
  - Every 60 seconds if the temperature does not differ from STinit by 5.
  - When the temperature differs from STinit by 5 at least every 10 seconds.

4.8.5. Example 5 - Band Minimum and Maximum, Band Step and Initial Value

Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto";anchor="/a/temperature";bind="obs";pmin="10";pmax="60";bmn="20",bmx="40",bst="5",iv="15"

Res: 2.04 Changed

The above will result in:

- A series of bands being created of a width of 5 with the seed value 15. Given bmn="20" and bmx="40" this effectively means that bands with the following thresholds(15,20],(20,25],(25,30],(30,35],(35,40] are created.
- A state synchronization through an Observe:
  - Every 60 seconds if the value is not between 20 and 40 (inclusive) and if the value has not entered into a new band.
  - When the temperature is equal to or between 20 and 40 and the value changes between the bands at least every 10 seconds.

4.8.6. Example 6 - Band Minimum and Sample Number Window

Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto";anchor="/a/temperature";bind="obs";pmin="10";pmax="60";bmn="50";snw="5"

Res: 2.04 Changed

The above will result in:

- A state sychronization added to the queue at pmax or whenever the value changes and is equal to or above 50.
- A state sychronization through an Observe occuring once 5 synchronizations have been added to the queue resulting in multiple values being synchronized between the source and destination resources.

5. Security Considerations

As per 5/[I-D.ietf-core-dynlink].

6. IANA Considerations

None.

7. Acknowledgements

Michael Koster for discussions leading to the creation of these notification band and initialization attributes.

8. Changelog

draft-groves-core-intparam-00

- Initial version

9. References

9.1. Normative References

[I-D.ietf-core-dynlink]
Shelby, Z., Vial, M., Koster, M., and C. Groves, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-01 (work in progress), October 2016.

[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>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

9.2. Informative References

[I-D.ietf-core-senml] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-04 (work in progress), October 2016.

Authors' Addresses

Christian Groves
Huawei
Australia

Email: Christian.Groves@mail01.huawei.com

Weiwei Yang
Huawei
P.R.China

Email: tommy@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 21, 2017

C. Groves

W. Yang
Huawei

April 19, 2017

Addition of organisation prefix to RFC6690 IANA CoRE parameters registration
draft-groves-core-rfc6690up-01

Abstract

[RFC6690] defines the resource type 'rt' and interface description 'if' link attributes and defines procedures for registering values. Currently each 'rt' and 'if' attribute value must be registered with IANA. This specification updates the process to enable organisation prefixes to be registered allowing organisations to manage their own namespace within a certain set of rules.

Table of Contents

1. Requirements Language

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].

2. Introduction

[RFC6690] "Constrained RESTful Environments (CoRE) Link Format" defines the Resource Type 'rt' and Interface Description 'if' link attributes. In order to co-ordinate the use of these attributes, sections 7.4 and 7.5/[RFC6690] establish IANA registries to register link attribute values for 'rt' and 'if'.

In order to register a new 'rt' and 'if' link attribute value the [RFC5226] "Specification Required" process is followed for each value. As part of the process a designated expert will examine the specification to enforce a number of requirements including:

- Registration values MUST be related to the intended purpose of these attributes as described in Section 3/[RFC6690].

- Registered values MUST conform to the ABNF reg-rel-type definition of Section 2/[RFC6690], meaning that the value starts with a lowercase alphabetic character, followed by a sequence of

lowercase alphabetic, numeric, ".", or "-" characters, and contains no white space.

- It is recommended that the period "." character be used for dividing name segments and that the dash "-" character be used for making a segment more readable. Example Interface Description values might be "core.batch" and "core.link-batch".

- URIs are reserved for free use as extension values for these attributes and MUST NOT be registered.

The IANA CoRE resource type and interface description registry can be found at: IANA CoRE Registry [1].

Given the scope of the Internet of Things (IoT) the potential number of resource types (and to a lesser extent interface types) is potentially quite large. This would lead to a large number of requests for designated expert review. It would also mean additional work for the IANA to process each request.

The current trend for the definition of resource types and interface descriptions is that a few standards organisations have defined a large number of values.

For example the "OIC Resource Type Specification v1.1.0" [OICResSpec] contains 64 resource types.

ETSI oneM2M also defines a large number of resource types. For example is "Home Appliances Information Model and Mapping" [oneM2MTS0023].

The Open Mobile Alliance (OMA) also make use of resources types.

The above three organisations also have their own registry and procedures for adding resource types. Trying to keep the IANA registry aligned with the individual organisation registries would also add additional burden.

A significant amount of work could be saved by allowing organisations to register a prefix under which they can administer their own resources negating the need for the IANA and the designated IANA expert to be involved for each resource registration.

There were discussions ath the IETF#97 meeting in Seoul about how to tackle this issue. There were broadly two camps in the discussion:

1 - Prefixes are a bad idea as they discourage people from resource re-use and create little "kingdoms".

2 - Prefixes are OK. They've been used before and people will coalesce on a common set eventually.

Clearly some sort of middle ground is needed to move forward.

Clearly resource re-use is a valid goal. It order for this to occur organisations/people requesting a new resource type would need to consider the existing resource types and see if it is applicable to them. Presumably if they can't use an existing type then they must have a reason why? One approach could be to introduce a new step into the registration process where the requester must specify any similar resources and why they cannot be used. This does of course add extra burden on the requester to document it and extra burden on the expert to evaluate it. Then there is the issue of what suffices for the analysis and what are the criteria for the expert to accept it? This may not result in reduced registrations but instead create more workload for registrations.

Currently all the rt registrations have been from standards organisations not individuals. The process for registration needs to be simple enough that people/companies have a incentive to register than rather than simply use and squat on a name without registering it.

It does have to be noted that today there is nothing stopping people/ organisations from duplicating resources in their registration. An organisational prefix would not make this worse.

Editor's note: Interface descriptions should be considered

This specification updates the [RFC6690] IANA registration procedures to allow the possibility to register a pre-fix.

3. Organisation Prefix

As indicated by [RFC6690] registered values MUST conform to the ABNF reg-rel-type definition, meaning that the value starts with a lowercase alphabetic character. Therefore an organisation registering a prefix MUST register a lowercase alphabetic sequence of characters. It MUST be followed by a ".".

For example: "foo."

This will allocate the namespace "foo." to the organisation. The organisation will then be responsible for maintaining resources within this name space.

E.g. "foo.sensor", "foo.actuator" could be allocated without requiring registration with IANA.

4. Security Considerations

This specification updates the [RFC6690] IANA Considerations. No additional protocol security impacts to what is already described in [RFC6690] are foreseen.

The use of organisational prefixes introducing the possibility that people request prefixes for an organisation that they do not represent. The IANA considerations in this specification require that the designated expert determine if the person requesting a prefix represents the organisation related to the prefix.

5. IANA Considerations

5.1. Constrained RESTful Environments (CoRE) Parameters Registry Update

This specification updates the Constrained RESTful Environments (CoRE) Parameter Registry by allowing the registration of an organisation prefix for Resource Type (rt=) and Interface Description (if=) Link Target Attribute values.

Organisation prefixes are registered by using the Specification Required policy (see [RFC5226], which requires review by a designated expert appointed by the IESG or their delegate.

The designated expert will enforce the following requirements:

o The registered prefix MUST conform to the ABNF reg-rel-type definition of Section 2/[RFC6690], meaning that the value starts with a lowercase alphabetic character followed by a period ".".

o The registered prefixes are assigned on a first come first served basis.

o Prefixes must be requested by a representative of the organisation applying for the prefix and must be representative of the organisation. E.g. organisation "foo" trying to register "ietf." would not be representative.

The specification MUST:

- Specify the procedures for registering values within the prefixed namespace. It ideally SHOULD provide a link where current and future registered values may be found.

- Indicate that registered values within the prefixed namespace MUST conform to the ABNF reg-rel-type definition of Section 2/[RFC6690]. This means that the prefix MUST be followed by a sequence of lowercase alphabetic, numeric, ".", or "-" characters, and contains no white space. Note: It is not recommended to immediately follow the prefix with an additional period ".", e.g. "foo..".

- Use the recommendation that the period "." character be used for dividing name segments and that the dash "-" character be used for making a segment more readable. Example Interface Description values might be "core.batch" and "core.link-batch".

Registration requests consist of the completed registration template below, with the reference pointing to the required specification. To allow for the allocation of values prior to publication, the designated expert may approve registration once they are satisfied that a specification will be published.

The registration template for both sub-registries is:

- Prefix Value:

- Description:

- Reference:

- Notes: [optional]

Registration requests should be sent to the core-parameters@ietf.org mailing list, marked clearly in the subject line (e.g., "NEW RESOURCE TYPE PREFIX - example" to register an "example" relation type or "NEW INTERFACE DESCRIPTION PREFIX - example" to register an "example" Interface Description).

Handling and the decision process is as per section 7.4/[RFC6690].

6. Acknowledgements

TBD

7. Changelog

draft-groves-core-rfc6690up-01:

- Keepalive update. No changes.

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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

8.2. Informative References

[OICResSpec] "OIC Resource Type Specification v1.1.0", 2016, <https://openconnectivity.org/resources/specifications/draft-candidate-specifications>.

[oneM2MTS0023] "TS 0023 v2.0.0 Home Appliances Information Model and Mapping", 2015, <http://www.onem2m.org/technical/published-documents>.

8.3. URIs

[1] http://www.iana.org/assignments/core-parameters/core-parameters.xhtml

Authors' Addresses

Christian Groves
Australia

Email: cngroves.std@gmail.com

Weiwei Yang
Huawei
P.R.China

Email: tommy@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 21, 2017

C. Groves

W. Yang
Huawei

April 19, 2017

SenML Base Time Offset Attribute
draft-groves-core-senml-bto-01

Abstract

SenML [I-D.ietf-core-senml] defines a base time attribute and time value which is used to determine the time when a value is recorded. In some applications a SenML pack will contain a series of records related to a constant sample time interval, e.g. once every 60 seconds. This means that the time attribute will be required for each record. This document defines a new "time offset" base attribute that allows a sender to include the time for the sample interval between records. If the "time offset" base attribute is used the sender will not send the time attribute for each record, minimising message and storage size.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 21, 2017.

Copyright Notice

Table of Contents

1. Introduction

SenML currently defines the base time "bt" and time "t" attributes to indicate the time that each value in a SenML record is recorded. This means that for each record (value "v") that a "t" attribute is required. The example from section 5.1.2/[I-D.ietf-core-senml] is copied below to illustrate a SenML pack with multiple records.

```
[ {"bn": "urn:dev:ow:10e2073a01080063",
      "bt": 1320067464,
      "bu": "%RH",
      "v": 21.2, "t": 0 },
      { "v": 21.3, "t": 10 },
      { "v": 21.4, "t": 20 },
      { "v": 21.4, "t": 30 },
      { "v": 21.5, "t": 40 },
      { "v": 21.5, "t": 50 },
      { "v": 21.5, "t": 60 },
      { "v": 21.6, "t": 70 },
      { "v": 21.7, "t": 80 },
      { "v": 21.5, "t": 90 },
   ...
```

Figure 1: SenML pack with multiple records

As can be seen in the example above there is a fixed offset between the data points of 10 seconds.

This document proposes a base time offset "bto" attribute that is used to indicate the offset between datapoints when a fixed offset is used. This negates the need to include the "t" attribute for each data point. The example below takes the above example and applies the base time offset to it.

```
[ {"bn": "urn:dev:ow:10e2073a01080063",
      "bt": 1320067464,
      "bto": 10,
      "bu": "%RH",
      "v": 21.2},
      { "v": 21.3},
      { "v": 21.4},
      { "v": 21.4},
      { "v": 21.5},
      { "v": 21.5},
      { "v": 21.5},
      { "v": 21.6},
      { "v": 21.7},
      { "v": 21.5},
   ...
```

Figure 2: SenML pack with multiple records using base time offset

It can be seen that it results in a record and overall pack size reduction. The receiver of the record would use the base time "bt" for the initial data point, e.g. "v": 21.2 would have a timestamp of

1320067464. Each subsequent record would have a time stamp equal to the previous record plus the base time offset "bto", e.g. "v": 21.3 would have a timestamp of 1320067474, "v": 21.4 would have a timestamp of 1320067484, and so on.

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].

See [I-D.ietf-core-senml] for further definitions.

3. SenML Structure and Semantics

3.1. Base Attributes

This document adds an additional "base time offset" attribute.

Base Time Offset: The base time offset represents a fixed time offset between each record in a SenML pack. The first record represents time zero (or the base time if provided) and the offset is then applied to each subsequent record. Either a positive or negative base time offset is allowed.

3.2. Regular Attributes

This document modifies the behaviour of the time attribute.

Time: If the base time offset attribute is used in a SenML pack then the time attribute shall not be used in the individual records.

Editor’s note: One theoretical use case may be to include the time attribute if the entry’s time deviates from the regular offset. It is not seen that such a corner case warrants the extra complexity.

3.3. Considerations

To simplify implementation, the use of base time offset is limited to:

- Fixed time increasing or decreasing records from the time stamp associated with the first record in the SenML pack. This means that usages such as described by the 2nd example in 5.1.2/[I-D.ietf-core-senml] where negative time offset from time zero being provided by the last record in a SenML pack are not possible.

Editor's note: It could be possible to facilitate this use case by allowing a t=0 to be set on the last record with the use of a negative base time offset value. However again it's not seen that such a corner case warrants the extra complexity in having to store values and calculate offsets at the end of transmission/reception.

- It is not possible to for two records in the SenML pack to have the same time. For example the inclusion of a record for a voltage measurement followed by a current measurement would result in the records having times with a difference of the time offset.

4. JSON Representation (application/senml+json)

This document defines an additional SenML label (JSON object member name) as shown in Table 1 below.

Name	label	Type
Base Time Offset	bto	Numer

Table 1: JSON SenML Labels

5. CBOR Representation (application/senml+cbor)

As per section 6/[I-D.ietf-core-senml], for CBOR the string map key "bto" shall be used to indicate the use of the base time offset.

6. XML representation (application/senml+xml)

This document defines an addition XML attribute as shown in Table 2 below.

Name	XML	Type
Base Time Offset	bto	double

Table 2: XML SenML Labels

The RelaxNG schema for the XML is:

```
     senml = element senml {
        attribute bn { xsd:string }?,
        attribute bt { xsd:double }?,
        attribute bv { xsd:double }?,
        attribute bs { xsd:double }?,
        attribute bu { xsd:string }?,
        attribute bver { xsd:int }?,
        attribute bto { xsd:double }?,

        attribute l { xsd:string }?,

        attribute n { xsd:string }?,
        attribute s { xsd:double }?,
        attribute t { xsd:double }?,
        attribute u { xsd:string }?,
        attribute ut { xsd:double }?,

        attribute v { xsd:double }?,
        attribute vb { xsd:boolean }?,
        attribute vs { xsd:string }?,
        attribute vd { xsd:string }?
      }

      sensml =
        element sensml {
          senml+
      }

      start = sensml
```

7. EXI Representation (application/senml-exi)

As per clause 8/[I-D.ietf-core-senml] extensions are indicated through the use of an EXI schemaID options. The EXI schemaID options MUST be set to the value of "b" indicating the scheme provided in this specification.

The following is the XSD Schema to be used for strict schema guided EXI processing.

```
<?xml version="1.0" encoding="utf-8"?>
   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   targetNamespace="urn:ietf:params:xml:ns:senml"
   xmlns:ns1="urn:ietf:params:xml:ns:senml">
     <xs:element name="senml">
       <xs:complexType>
         <xs:attribute name="bn" type="xs:string" />
         <xs:attribute name="bt" type="xs:double" />
         <xs:attribute name="bv" type="xs:double" />
         <xs:attribute name="bu" type="xs:string" />
         <xs:attribute name="bver" type="xs:int" />
         <xs:attribute name="bto" type="xs:double" />
         <xs:attribute name="l" type="xs:string" />
         <xs:attribute name="n" type="xs:string" />
         <xs:attribute name="s" type="xs:double" />
         <xs:attribute name="t" type="xs:double" />
         <xs:attribute name="u" type="xs:string" />
         <xs:attribute name="ut" type="xs:double" />
         <xs:attribute name="v" type="xs:double" />
         <xs:attribute name="vb" type="xs:boolean" />
         <xs:attribute name="vs" type="xs:string" />
         <xs:attribute name="vd" type="xs:string" />
       </xs:complexType>
     </xs:element>
     <xs:element name="sensml">
       <xs:complexType>
         <xs:sequence>
           <xs:element maxOccurs="unbounded" ref="ns1:senml" />
         </xs:sequence>
       </xs:complexType>
     </xs:element>
   </xs:schema>
```

8. Security Considerations

No extra security issues are seen beyond those described in [I-D.ietf-core-senml].

9. IANA Considerations

This document proposes one new entry in the IANA SenML label reegistry.

Label Name: Base Time Offset

Label: bto

CBOR: -

XML Type: Double

ID: b

Reference: This specification.

10. Acknowledgements

TBD

11. Changelog

draft-groves-core-senml-bto-01

- General: Changed "SenML Package" to "SenML Pack"

12. Normative References

[I-D.ietf-core-senml] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-05 (work in progress), March 2017.

[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>.

Authors' Addresses

Christian Groves
Australia

Email: cngroves.std@gmail.com

Weiwei Yang
Huawei
P.R.China

Email: tommy@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 11, 2017

C. Groves
W. Yang
Huawei

March 10, 2017

SenML Options
draft-groves-core-senml-options-00

Abstract

SenML [I-D.ietf-core-senml] defines an initial set of base and regular attributes which are tied to a particular version of SenML. SenML also allows the definition of additional attributes by extending the syntax with a new label. Allowing the extension of attributes brings the problem of how do endpoints negotiate whether the new attribute can be used or not? This document discusses the issue and proposes some potential solutions to this issue.

Table of Contents

1. Introduction

SenML [I-D.ietf-core-senml] defines an initial set of base and regular attributes which are tied to a particular version of SenML. SenML also allows the definition of additional attributes by extending the defined syntax with a new label. Allowing the extension of attributes brings the problem of how do endpoints negotiate whether the new attribute can be used or not?

For example: A CoAP client issues a GET that indicates support of SenML through the use of an CoAP Accept option. A CoAP server supports the SenML attributes defined in [I-D.ietf-core-senml] and in addition supports the Base Time Offset (BTO) [I-D.groves-core-senml-bto] attribute. The server responds using the BTO attribute.

```
[ {"bn": "urn:dev:ow:10e2073a01080063",
      "bt": 1320067464,
      "bto": 10,
      "bu": "%RH",
      "v": 21.2},
      { "v": 21.3},
      { "v": 21.4},
      { "v": 21.4},
      { "v": 21.5},
      { "v": 21.5},
      { "v": 21.5},
      { "v": 21.6},
      { "v": 21.7},
      { "v": 21.5},
   ...
```

Figure 1: Response with SenML using base time offset

As the CoAP client does not understand the "bto" attribute it will ignore the attribute. This means that the time information is lost for each of the SenML records. Whereas if the Server had not used "bto" the client would have been able to understand the information.

This is mainly a problem when the server provides a response to a message (i.e. GET) rather than when a client uses the SenML media type in a message (i.e. PUT). In this later case the client can modify its behaviour and not use an attribute based on an error response from the server.

A solution is needed to prevent incompatible attributes from being used.

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].

See [I-D.ietf-core-senml] for further definitions.

3. Solution space analysis

The extension of protocols in a compatible manner is not a new problem. Some approaches to handle the are discussed below. The discussion below is not meant to be treatise on protocol extension but to highlight potential solutions.

3.1. Version Numbering

A version number is used to indicate when changes have been made to a syntax. In some protocols a major version is used to indicate non-backward compatible changes and a minor version is used to indicate backwards compatible changes. SenML does use a version number however it is an integer (no major and minor). It is used to indicate the version number of the media type format. The version does not appear in the media type format name. However presumably the proposed registrations (e.g. senml+json) imply the use of version number 10 (see sect.4.1/[I-D.ietf-core-senml]). If in the future there is a new version how does the endpoint negotiate which SenML version to use? A simple solution would be to create a new media type name incorporating the new version e.g. "senml_v11+json". A CoAP endpoint could then use an Accept option to indicate support for "senml+json" and/or "senml_v11+json".

This method does generally mean that for each new attribute and version the endpoints must at least understand all previously defined attributes. Although major versions in some protocols do not maintain backwards compatibility.

SenML does indicate that for certain representations i.e. EXI representation (application/senml+exi) that the schemaID number must be updated when the syntax is updated for a new attribute. This is effectively a version mechanism but the other representation formats do not follow this approach.

However the current SenML draft does allow the definition of additional attributes without increasing the SenML version number. Indeed there is a trend at the IETF that protocol versions change very rarely. Instead updates are incorporated via option or extension mechanisms.

Therefore it seems that an approach of utilising a version number for each additional new attribute does not seem appropriate for SenML.

3.2. Mandatory / Optional Indication

Some protocols use a mechanism to indicate whether a parameter is mandatory or optional to understand. This is based on the assumption that a sending endpoint knows the functions that a parameter/s relate to and can indicate whether the receiving endpoint must understand the parameter/s to implement the function. A receiving endpoint will anaylse the received parameters and if it does not understand the parameter it will check the mandatory/option tag to see what it should do. If the parameter is marked as mandatory then the receiving endpoint will generate an error. If the parameter is

marked as optional then the receiving endpoint will continue processing in knowledge that it doesn't need the parameter.

CoAP uses a mechanism similar to this for Options. By looking at the option number an endpoint can determine whether it is critical or not.

SenML does indicate that the version indicates the mandatory to understand attributes (sect. 4.3/[I-D.ietf-core-senml]). SenML also indicates that some attributes (i.e. base attributes) are optional but this is in context of "optional to be used" rather than "optional to be understood".

Whilst a method could be defined to indicate in SenML whether an attribute is mandatory or optional, its not clear that it would be useful. Given the number of use cases where CoAP can be used a server may not know which information in a SenML pack is relevant for a client. E.g. Whilst a server may return time information associated with a record it doesn't actually know whether it is useful to the client. The usefulness is application specific to the client.

Therefore it seems this approach is also not appropriate for SenML.

3.3. Options Mechanism

Some protocols allow the optional parts of the protocol to be negotiated during the initial protocol negotiation. For example the SIP protocol has the OPTIONS method [RFC3261]. The CLUE protocol [I-D.ietf-clue-protocol] also defines an extension method where an OPTIONS message is used to negotiate the protocol extensions. The benefits of such an approach is that two endpoints can negotiate which extensions they will use in a session ensuring compatible communications.

However these approaches assume an application level session where there are establishment, communication and release phases. SenML is a media type format primarly defined to be used with the HTTP and CoAP protocols. These protocols don't follow a session based approach.

HTTP/1.1 does have the OPTIONS method [RFC2616] however the use is largely undocumented. CoAP does not have an equivalent method.

CoAP does have a method for negotiating signalling through "Signaling Option Numbers" (sect.4.2/[I-D.ietf-core-coap-tcp-tls]). This however is more used to negotiate the properties of the signalling

connection than any elements of the CoAP payload (not withstanding the Blockwise transfer).

CoAP does have the OPTIONs mechanism allowing for the definition of optionality functionality associated with a CoAP message. It also defines the concept of critical and elective options. Two options related to Content-type/Content-format are "Content-Format" and "Accept". These options allow endpoints to indicate the media types are using or wish to use. HTTP also uses these options.

CoAP also allows a per message exchange of what is supported for a particular resource. This seems more appropriate than a protocol level negotiation of the support of SenML attributes.

This functionality is very close to what needs to be acheived for negotiating SenML attributes.

3.4. Media Type Definition and Parameters

Potentially the media type name could be used to indicate versions or extensions. This may be appropriate where there are seldom changes that affect the whole media type. However it may become unwieldy if the media type name is used to define combinations of SenML attributes, e.g. given 3 extensions a, b and c you could end up with media names / content formats for a, b, c, a+b, a+c, b+c, a+b+c. The problem gets worse each time an extension is added. It is made even worse because Table 7/[I-D.ietf-core-senml] defines 8 different content formats for SenML that would need to be updated. To allow combinations of these parameters on the media types defined in SenML it would need 56 Content-format code points. The content format range 0..255 for IETF specifications isn’t particularly large.

[RFC6838] allows for the registration of media type parameters. This allows further companion information to be included along with the media type. Charset is a common parameter (See [RFC3023]). This information could be used to provide version or option information associated with a media type.

This appears to be a good solution to indicate if additional SenML attributes are supported in a media type. Whilst HTTP supports media type parameters, CoAP does not support media type parameters or extensions (i.e. see sect.10.2.2/[RFC7252]). Meaning that parameters cannot be used as a common approach for HTTP and CoAP. However this solution is used in section 16.9.1/[I-D.ietf-cose-msg] which takes the approach of defining an optional parameter for the "application/cose". It then assigns multiple CoAP Content-Formats for the values associated with the optional parameter (see sect.16.10/[I-D.ietf-cose-msg]).

4. Solution Proposal

There doesn't appear to be one outstanding approach for negotiating SenML attributes common to both HTTP and CoAP. The authors believe that a hybrid approach as per [I-D.ietf-cose-msg] is needed in order to be able to negotiate which SenML extension attributes are used.

The first part would be the definition of a optional media-type parameter that allows an endpoint utilising HTTP to indicate the SenML extension attributes that it is using or accepts. This could be in the form of a comma separated string list of SenML labels from those registered in the SenML label registry. Only attributes NOT defined in [I-D.ietf-core-senml] would be allowed.

e.g. Content-type: application/senml+json; ext=abc,xyz

In order to allow this functionality in the base version of SenML an optional parameter would be needed in the media type registrations in sect.11.3/[I-D.ietf-core-senml].

i.e. o Optional parameter: SenML extensions

This parameter indicates which SenML extensions are associated with the media type. The parameter is defined by the following ABNF:

```
SenML-ext = "ext" "=" <"> senml-label *("," senml-label) <">
; Note: this follows the {{RFC2616}} quoted-string form.
; senml-label is the label string from the list of IANA
; registered SenML labels.
; Only non-{{I-D.ietf-core-senml}} labels are allowed.
```

For example: ext="a,b,c";

If the group decides that there will only ever be a small number of SenML extensions then the simplest approach would be to follow [I-D.ietf-cose-msg] and define multiple CoAP content formats associated with potential extensions. This would be done in which ever document defines the SenML extension. For example [I-D.groves-core-senml-bto] would add the following to the IANA considerations section:

Media Type	Encoding	ID	Reference
application/senml+json; ext="bto"		TBD	TBD
application/sensml+json; ext="bto"		TBD	TBD
application/senml+cbor; ext="bto"		TBD	TBD
application/sensml+cbor; ext="bto"		TBD	TBD
application/senml+xml; ext="bto"		TBD	TBD
application/sensml+xml; ext="bto"		TBD	TBD
application/senml+exi; ext="bto"		TBD	TBD
application/sensml+exi; ext="bto"		TBD	TBD

Table 1: New CoAP content formats

Text would also need to be added to [I-D.ietf-core-senml] to describe the procedure for support of the media type parameter in CoAP.

If issuing potentially a large number of content-format numbers is problematic a separate approach could be taken. A new CoAP option could be defined to allow media-type parameters to be carried in CoAP messages when then Content-Format or Accept options are used. As the Content-Format and Accept options may be used in the same request (with two different media types) two new options would be required, one for the media-type parameters associated with the Content-Format Option and one for the media-type parameters associated with the Accept option.

Editor’s note: Alternatives could include defining the option for both CoAP and HTTP. However this would likely mean that the option would become specific to SenML extensions rather than a general mechanism for carrying media type parameters.

As CoAP only allows a single Content-Format to be carried in the Content-Format and Accept options it would be straight forward to define an option that allows media-type parameters to be carried. One complication is that the encoding and syntax of the media-type parameters is up to the media-type definition. It could be a string, integer, binary, etc. Therefore the option value would need to be an opaque sequence of bytes. If the scope was limited to SenML then the option format would be a narrowed to a string of labels.

As multiple parameters could be defined for a media-type the mechanism must allow multiple media type parameter to be signalled in a CLUE message. One possible method is to define the option value syntax to allow multiple parameter to be specified as a single parameter value. Alternatively multiple instances of the option could be used in the CoAP message. This method is indicated in the IANA registration by allowing the option multiple times.

If a new generic option is defined it's not clear that [I-D.ietf-core-senml] would be the best place to define the option. If the scope is limited then [I-D.ietf-core-senml] would be appropriate. Whichever draft defines the options it would need to define them for registration with IANA along the lines of:

Number	Name	Reference
TBD	AMTP	TBD
TBD	CFMTP	TBD

Table 2: New CoAP Option Numbers

4.1. Accept Media Type Parameter (AMTP) Option

o The meaning of the option in a request: It indicates the media-type parameters associated with the content-format (media-type) specified in the Accept option.

Note: Some content-formats contain media-type parameters as part of the content-format ID registrations. The AMTP option SHALL not be used with these CoAP content-formats.

o The meaning of the option in a response: Not used.

o Whether the option is critical or elective: Critical as per the Accept option.

o Whether the option is Safe-to-Forward: Safe as per the Accept option.

Note: Potentially it could be unsafe to forward an opaque byte sequence that the proxy does not understand. However processing this option should only be done within the context of the media-type specified by the Accept option.

o The format and length of the option’s value: A variable length opaque sequence of bytes. The encoding of the bytes is defined as per the syntax for the parameters in the media-type definition document.

o Whether the option must occur at most once or whether it can occur multiple times: Multiple times. Each instance containing a seperate media-type parameter. Whether the option can be included multiple times for the one media type parameter is dependent on whether the media-type definition allows for multiple instances of the one media type parameter.

o Default value: None unless the media-type indicated in the accept option defines a default parameter/s value.

4.2. Content-Format Media-Type Parameter Option

o The meaning of the option in a request: When used together with the Content-Format option it indicates the media-type parameters associated with the content-format (media-type) specified in the content-format option.

Note: Some content-formats contain media-type parameters as part of the content-format ID registrations. The CFMTP option SHALL not be used with these CoAP content-formats.

o The meaning of the option in a response: When used together with the Content-Format option it indicates the media-type parameters associated with the content-format (media-type) specified in the content-format option.

o Whether the option is critical or elective: As per the Content-Format option it is elective.

o Whether the option is Safe-to-Forward: Safe as per the Content-format option.

Note: Potentially it could be unsafe to forward an opaque byte sequence that the proxy does not understand. However processing this option should only be done within the context of the media-type specified by the Content-Format options.

o The format and length of the option’s value: A variable length opaque sequence of bytes. The encoding of the bytes is defined as per the syntax for the parameters in the media-type definition document.

o Whether the option must occur at most once or whether it can occur multiple times: Multiple times. Each instance containing a seperate media-type parameter. Whether the option can be included multiple times for the one media type parameter is dependent on whether the media-type definition allows for multiple instances of the one media type parameter.

o Default value: None unless the media-type indicated in the content-format option defines a default parameter/s value.

5. Security Considerations

SenML security issues are described in [I-D.ietf-core-senml]. Some extra considerations are indicated above.

6. IANA Considerations

Section 4 discusses potential IANA registrations.

7. Acknowledgements

TBD

8. Changelog

TBD

9. References

9.1. Normative References

[I-D.groves-core-senml-bto]
Groves, C. and W. Yang, "SenML Base Time Offset Attribute", draft-groves-core-senml-bto-00 (work in progress), October 2016.

[I-D.ietf-core-senml]
Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-04 (work in progress), October 2016.

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

[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>.

[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, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <http://www.rfc-editor.org/info/rfc3261>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[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>.

9.2. Informative References

[I-D.ietf-clue-protocol] Presta, R. and S. Romano, "CLUE protocol", draft-ietf-clue-protocol-13 (work in progress), February 2017.

[I-D.ietf-core-coap-tcp-tls] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-07 (work in progress), March 2017.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, DOI 10.17487/RFC3023, January 2001, <http://www.rfc-editor.org/info/rfc3023>.

Authors' Addresses

Christian Groves
Huawei
Australia

Email: Christian.Groves@mail01.huawei.com

Weiwei Yang
Huawei
P.R.China

Email: tommy@huawei.com

Thing-to-Thing Research Group
Internet-Draft
Intended status: Informational
Expires: August 16, 2017

K. Hartke
Universitaet Bremen TZI

February 12, 2017

CoRE Application Descriptions
draft-hartke-core-apps-07

Abstract

The interfaces of RESTful, hypermedia-driven Web applications consist of reusable components such as Internet media types and link relation types. This document proposes CoRE Application Descriptions, a convention for application designers to describe the programmable interfaces of their applications in a structured way so that other parties can easily develop interoperable clients and servers or reuse the components in their own applications.

Note to Readers

This Internet-Draft should be discussed on the Thing-to-Thing Research Group (T2TRG) mailing list <t2trg@irtf.org> <https://www.irtf.org/mailman/listinfo/t2trg>.

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 16, 2017.

Copyright Notice

Table of Contents

1. Introduction

Representational State Transfer (REST) [16] is an architectural style for distributed hypermedia systems. Over the years, REST has gained popularity not only as an approach for large-scale information dissemination, but also as the basic principle for designing and building Internet-based applications in general.

In the coming years, the size and scope of the Internet is expected to greatly increase as physical-world objects become smart enough to communicate over the Internet -- a phenomenon known as the Internet of Things (IoT). As things learn to speak the languages of the net, the idea of applying REST principles to the design of IoT application architectures suggests itself. To this end, the Constrained Application Protocol (CoAP) [24] was created, an application-layer protocol that enables RESTful applications in constrained-node networks [10], thus giving rise to a new setting for Internet-based applications: the Constrained RESTful Environment (CoRE).

To realize the full benefits and advantages of the REST style, a set of constraints needs to be maintained when designing new applications and their application programming interfaces (APIs). One of the fundamental principles is that "REST APIs must be hypertext-driven" [17]. This principle is often ignored by application designers, who instead specify their APIs out-of-band in terms of fixed URI patterns, e.g., in the API documentation or in a machine-readable format that facilitates code generation. Although this approach may appear easy for clients to use, the fixed resource names and data formats lead to a tight coupling between client and server implementations and make the system less flexible. Violations of REST design principles like this result in APIs that may not be as scalable, extensible, and interoperable as promised by REST [20].

REST is intended for long-lived network-based applications that span multiple organizations [17]. Principled REST APIs require some design effort, since application designers do not only have to take current requirements into consideration, but also have to anticipate changes that may be required in the future -- years or even decades after the application has been deployed for the first time. The reward is long-term stability and evolvability, both of which are very desirable features in the Internet of Things.

To aid application designers in the design process, this document proposes CoRE Application Descriptions, a convention for describing the APIs of RESTful, hypermedia-driven Web applications. CoRE Application Descriptions help application designers avoid common mistakes by focusing almost all of the descriptive effort on defining the Internet media type(s) that are used for representing resources and driving application state.

A template provides a consistent format for the description of APIs so that implementers can easily build interoperable clients and servers and other application designers can reuse application components in their own applications.

2. Application Descriptions

A CoRE Application Description is a named set of reusable components. It describes a contract between a server hosting an instance of the described application and clients that wish to interface with that instance.

A CoRE Application Description is comprised of:

- URI schemes that identify communication protocols,
- Internet media types that identify representation formats,
- link relation types that identify link semantics,
- form relation types that identify form semantics,
- variable names that identify the semantics of variables in templated links,
- form field names that identify the semantics of form fields in forms, and
- optionally, well-known locations.

Together, these components provide the specific, in-band instructions for interfacing with a given application.

2.1. URI Schemes

The foundation of a hypermedia-driven REST API are the communication protocol(s) spoken between a client and a server. Although HTTP/1.1 [14] is by far the most common communication protocol for REST APIs, a REST API should typically not be dependent on any specific communication protocol.

The use of a particular protocol by a client is guided by URI schemes [7]. URI schemes specify the syntax and semantics of URI references [1] that the server includes in links (Section 2.2.2) and forms (Section 2.2.3).

A URI scheme refers to a family of protocols, typically distinguished by a version number. For example, the "http" URI scheme refers to the two members of the HTTP family of protocols: HTTP/1.1 [14] and HTTP/2 [8] (as well as some predecessors). The specific HTTP version used is negotiated between a client and a server in-band using the version indicator in the HTTP request-line or the TLS Application-Layer Protocol Negotiation (ALPN) extension [18].

IANA maintains a list of registered URI schemes at <http://www.iana.org/assignments/uri-schemes>.

2.2. Internet Media Types

One of the most important aspect of hypermedia-driven communications is the concept of Internet media types [2]. Media types are used to label representations so that it is known how the representation should be interpreted and how it is encoded. The centerpiece of a CoRE Application Description should be one or more media types.

A media type identifies a versioned series of representation formats (Section 2.2.1): a media type does not identify a particular version of a representation format; rather, the media type identifies the family, and includes provisions for version indicator(s) embedded in the representations themselves to determine more precisely the nature of how the data is to be interpreted [21]. A new media type is only needed to designate a completely incompatible format [21].

Note: The terms media type and representation format are often used interchangeably. In this document, the term "media type" refers specifically to a string of characters such as "application/xml" that is used to label representations; the term "representation format" refers to the definition of the syntax and semantics of representations, such as XML 1.0 [12] or XML 1.1 [13].

Media types consist of a top-level type and a subtype, structured into trees [2]. Optionally, media types can have parameters. For example, the media type "text/plain; charset=utf-8" is a subtype for plain text under the "text" top-level type in the standards tree and has a parameter "charset" with the value "utf-8".

Media types can be further refined by

- structured type name suffixes (e.g., "+xml" appended to the base subtype name; see Section 4.2.8 of RFC 6838 [2]),
- a "profile" parameter (see Section 3.1 of RFC 6906 [25]),
- subtype information embedded in the representations themselves (e.g., "xmlns" declarations in XML documents [11]),

or a similar annotation. An annotation directly in the media type is generally preferable, since subtype information embedded in representations can typically not be negotiated during content negotiation (e.g., using the CoAP Accept option).

In CoAP, media types are combined with content coding information [15] to indicate the "content format" [24] of a representation. Each content format is assigned a numeric identifier that can be used instead of the (typically much longer) textual name of the media type in representation formats with space constraints. The flat number space loses the structural information that the textual names have, however.

The media type of a representation must be determined from in-band information (e.g., from the CoAP Content-Format option). Clients must not assume a structure from the application context or other out-of-band information.

> IANA maintains a list of registered Internet media types at <http://www.iana.org/assignments/media-types>.
>
> IANA maintains a list of registered structured suffixes at <http://www.iana.org/assignments/media-type-structured-suffix>.
>
> IANA maintains a list of registered CoAP content formats at <http://www.iana.org/assignments/core-parameters>.

2.2.1. Representation Formats

In RESTful applications, clients and servers exchange representations that capture the current or intended state of a resource and that are labeled with a media type. A representation is a sequence of bytes whose structure and semantics are specified by a representation format: a set of rules for encoding information.

Representation formats should generally allow clients with different goals, so they can do different things with the same data. The specification of a representation format "describes a problem space, not a prescribed relationship between client and server. Client and server must share an understanding of the representations they’re passing back and forth, but they don’t need to have the same idea of what the problem is that needs to be solved." [22]

Representation formats and their specifications frequently evolve over time. It is part of the responsibility of the designer of a new version to insure both forward and backward compatibility: new representations should work reasonably (with some fallback) with old processors and old representations should work reasonably with new processors [21].

Representation formats enable hypermedia-driven applications when they support the expression of hypermedia controls, i.e., links (Section 2.2.2) and forms (Section 2.2.3).

2.2.2. Links

As described in RFC 5988 [5], a link is a typed connection between two resources. Additionally, a link is the primary means for a client to navigate from one resource to another.

A link is comprised of:

- a link context (usually the "current" resource),
- a link relation type that identifies the semantics of the link (see Section 2.3),
- a link target, identified by a URI, and
- optionally, target attributes that further describe the link or the link target.

A link can be viewed as a statement of the form "{link context} has a {link relation type} resource at {link target}, which has {target attributes}" [5]. For example, the resource <http://example.com/> could have a "terms-of-service" resource at <http://example.com/tos>, which has a representation with the media type "text/html".

There are two special kinds of links:

- An embedding link is a link with an additional hint: when the link is processed, it should be substituted with the representation of the referenced resource rather than cause the client to navigate away from the current resource. Thus, traversing an embedding link adds to the current state rather than replacing it.

  The most well known example for an embedding link is the HTML <img> element. When a Web browser processes this element, it automatically dereferences the "src" and renders the resulting image in place of the <img> element.

- A templated link is a link where the client constructs the link target URI from provided in-band instructions. The specific rules for such instructions are described by the representation format. URI Templates [3] provide a generic way to construct URIs through variable expansion.

  Templated links allow a client to construct resource URIs without being coupled to the resource structure at the server, provided that the client learns the template from a representation sent by the server and does not have the template hard-coded.

2.2.3. Forms

A form is the primary means for a client to submit information to a server, typically in order to change resource state.

A form is comprised of:

- a form context (usually the "current" resource),
- a form relation type that identifies the semantics of the form (see Section 2.4),
- a form target, identified by a URI,
- a submission method (PUT, POST, PATCH, FETCH, or DELETE),
- a description of a representation that the server expects as part of the form submission, and
- optionally, target attributes that further describe the form or the form target.

A form can be viewed as an instruction of the form "To {form relation type} the {form context}, make a {method} request to {form target}, which has {target attributes}". For example, to "update" the resource <http://example.com/config>, a client could be required to make a PUT request to <http://example.com/config>. (In many cases, the target of a form is the same resource as the context, but this is not required.)

The description of the expected representation can be a set of form fields (see Section 2.4.1) or simply a list of acceptable media types.

Note: A form with a submission method of GET is, strictly speaking, a templated link, since it provides a way to construct a URI and does not submit a representation to the server.

2.3. Link Relation Types

A link relation type identifies the semantics of a link [5]. For example, a link with the relation type "copyright" indicates that the resource identified by the target URI is a statement of the copyright terms applying to the link context.

Relation types are not to be confused with media types; they do not identify the format of the representation that results when the link

is dereferenced [5]. Rather, they only describe how the link context is related to another resource [5].

IANA maintains a list of registered link relation types at <http://www.iana.org/assignments/link-relations>.

Applications that don't wish to register a link relation type can use an extension link relation type [5], which is a URI that uniquely identifies the link relation type. For example, an application can use the string "http://example.com/foo" as link relation type without having to register it. Using a URI to identify an extension link relation type, rather than a simple string, reduces the probability of different link relation types using the same identifiers.

In order to minimize the overhead of link relation types in representation formats with space constraints, IANA-registered link relation types are assigned a numeric identifier that can be used in place of the textual name (see also Section 6.2). For example, the link relation type "copyright" has the number 12. A representation format may additionally provide numeric identifiers for extension link relation types.

2.3.1. Template Variable Names

A templated link enables clients to construct the target URI of a link, for example, when the link refers to a space of resources rather than a single resource. The most prominent mechanisms for this are URI Templates [3] and the HTML <form> element with a submission method of GET.

To enable an automated client to construct an URI reference from a URI Template, the name of the variable in the template can be used to identify the semantics of the variable. For example, when retrieving the representation of a collection of temperature readings, a variable named "threshold" could indicate the variable for setting a threshold of the readings to retrieve.

Template variable names are scoped to link relation types, i.e., two variables with the same name can have different semantics if they appear in links with different link relation types.

2.4. Form Relation Types

A form relation type identifies the semantics of a form. For example, a form with the form relation type "create-item" indicates that a new item can be created within the form context by making a request to the resource identified by the target URI.

IANA maintains a list of registered form relation types at <TBD>.

Similar to extension link relation types, applications can use extension form relation types when they don't wish to register a form relation type.

IANA-registered form relation types are assigned a numeric identifier that can be used in place of the textual name. For example, the form relation type "update" has the number 3. A representation format may additionally provide numeric identifiers for extension form relation types.

2.4.1. Form Field Names

Forms can have a detailed description of the representation expected by the server as part of form submission. This description typically consists of a set of form fields where each form field is comprised of a field name, a field type, and optionally a number of attributes such as a default value, a validation rule or a human-readable label.

To enable an automated client to fill out a form, the field name can be used to identify the semantics of the form field. For example, when controlling a smart light bulb, the field name "brightness" could indicate the field for setting the desired brightness of the light bulb.

Field names are scoped to form relation types, i.e., two form fields with the same name can have different semantics if they appear in forms with different form relation types.

The type of a form field is a data type such as "an integer between 1 and 100" or "a RGB color". The type is orthogonal to the field name, i.e., the type should not be determined from the field name even though the client can identify the semantics of the field from the name. This separation makes it easy to change the set of acceptable values in the future.

2.5. Well-Known Locations

Some applications may require the discovery of information about a host, known as "site-wide metadata" in RFC 5785 [4]. For example, RFC 6415 [19] defines a metadata document format for describing a host; similarly, RFC 6690 [23] defines a link format for the discovery of resources hosted by a server.

Applications that need to define a resource for this kind of metadata can register new "well-known locations". RFC 5785 [4] defines the

path prefix "/.well-known/" in "http" and "https" URIs for this purpose. RFC 7252 [24] extends this convention to "coap" and "coaps" URIs.

IANA maintains a list of registered well-known URIs at <http://www.iana.org/assignments/well-known-uris>.

3. Application Description Template

As applications are implemented and deployed, it becomes important to describe them in some structured way. This section provides a simple template for CoRE Application Descriptions. A uniform structure allows implementers to easily determine the components that make up the interface of an application.

The template below lists all components of applications that both the client and the server implementation of the application need to understand in order to interoperate. Crucially, items not listed in the template are not part of the contract between clients and servers -- they are implementation details. This includes in particular the URIs of resources (see Section 4).

CoRE Application Descriptions are intended to be published in human-readable format by designers of applications and by operators of deployed application instances. Application designers may publish an application description as a general specification of all application instances, so that implementers can create interoperable clients and servers. Operators of application instances may publish an application description as part of the API documentation of the service, which should also include instructions how the service can be located and which communication protocols and security modes are used.

The fields of the template are as follows:

Application name:
: Name of the application. The name is not used to negotiate capabilities; it is purely informational. A name may include a version number or, for example, refer to a living standard that is updated continuously.

URI schemes:
: URI schemes identifying the communication protocols that need to be understood by clients and servers. This information is mostly relevant for deploymed instances of the application rather than for the general specification of the application.

Media types:

Internet media types that identify the representation formats that need to be understood by clients and servers. An application description must comprise at least one media type. Additional media types may be required or optional.

Link relation types:
Link relation types that identify the semantics of links. An application description may comprise IANA-registered link relation types and extension link relation types. Both may be required or optional.

Template variable names:
For each link relation type, variable names that identify the semantics of variables in templated links with that link relation type. Whether a template variable is required or optional is indicated in-band inside the templated link.

Form relation types:
Form relation types that identify the semantics of forms and, for each form relation type, the submission method(s) to be used. An application description may comprise IANA-registered form relation types and extension form relation types. Both may be required or optional.

Form field names:
For each form relation type, form field names that identify the semantics of form fields in forms with that form relation type. Whether a form field is required or optional is indicated in-band inside the form.

Well-known locations:
Well-known locations in the resource identifier space of servers that clients can use to discover information given the DNS name or IP address of a server.

Interoperability considerations:
Any issues regarding the interoperable use of the components of the application should be given here.

Security considerations:
Security considerations for the security of the application must be specified here.

Contact:
Person (including contact information) to contact for further information.

Author/Change controller:

Person (including contact information) authorized to change this application description.

Each field should include full citations for all specifications necessary to understand the application components.

4. URI Design Considerations

URIs [1] are a cornerstone of RESTful applications. They enable uniform identification of resources via URI schemes [7] and are used every time a client interacts with a particular resource or when a resource representation references another resource.

URIs often include structured application data in the path and query components, such as paths in a filesystem or keys in a database. It is common for many RESTful applications to use these structures not only as an implementation detail but also make them part of the public REST API, prescribing a fixed format for this data. However, there are a number of problems with this practice [6], in particular if the application designer and the server owner are not the same entity.

In hypermedia-driven applications, URIs are therefore not included in the application interface. A CoRE Application Description must not mandate any particular form of URI substructure.

RFC 7320 [6] describes the problematic practice of fixed URI structures in detail and provides some acceptable alternatives.

Nevertheless, the design of the URI structure on a server is an essential part of implementing a RESTful application, even though it is not part of the application interface. The server implementer is responsible for binding the resources identified by the application designer to URIs.

A good RESTful URI is:

- Short. Short URIs are easier to remember and cause less overhead in requests and representations.
- Meaningful. A URI should describe the resource in a way that is meaningful and useful to humans.
- Consistent. URIs should follow a consistent pattern to make it easy to reason about the application.
- Bookmarkable. Cool URIs don’t change [9]. However, in practice, application resource structures do change. That should cause URIs

to change as well so they better reflect reality. Implementations should not depend on unchanging URIs.

- Shareable. A URI should not be context sensitive, e.g., to the currently logged-in user. It should be possible to share a URI with third parties so they can access the same resource.

- Extension-less. Some applications return different data for different extensions, e.g., for "contacts.xml" or "contacts.json". But different URIs imply different resources. RESTful URIs should identify a single resource. Different representations of the resource can be negotiated (e.g., using the CoAP Accept option).

5. Security Considerations

The security considerations of RFC 3986 [1], RFC 5785 [4], RFC 5988 [5], RFC 6570 [3], RFC 6838 [2], RFC 7320 [6], and RFC 7595 [7] are inherited.

All components of an application description are expected to contain clear security considerations. CoRE Application Descriptions should furthermore contain security considerations that need to be taken into account for the security of the overall application.

6. IANA Considerations

[Note to RFC Editor: Please replace XXXX in this section with the RFC number of this specification.]

6.1. Content-Format Registry

RFC 6838 [2] establishes a IANA registry for media types. Many of these media types are also useful in constrained environments as CoAP content formats. RFC 7252 [24] establishes a IANA registry for these content formats. This specification tasks IANA with the allocation of a content format for any existing or new media type registration that does not define any parameters (required or optional). The content formats shall be allocated in the range 1000-9999.

6.2. Link Relation Type Registry

RFC 5988 [5] establishes a IANA registry for link relation types. This specification extends the registration template with a "Relation ID": a numeric identifier that can be used instead of the "Relation Name" to identify a link relation type. IANA is tasked with the assignment of an ID to any existing or new link relation type. The IDs shall be assigned in the range 1-9999.

6.3. Form Relation Type Registry

This specification establishes a IANA registry for form relation types.

6.3.1. Registering New Form Relation Types

Form relation types are registered in the same way as link relation types [5], i.e., they are registered on the advice of a Designated Expert with a Specification Required.

The requirements for registered relation types are adopted from Section 4.1 of RFC 5988 [5].

The registration template is:

- Relation Name:
- Relation ID:
- Description:
- Reference:
- Notes: [optional]

The IDs shall be assigned in the range 1-9999.

6.3.2. Initial Registry Contents

The Form Relation Type registry’s initial contents are:

- Relation Name: create-item
  Relation ID: 1
  Description: Refers to a resource that can be used to create a resource in a collection of resources.
  Reference: [RFCXXXX]

- Relation Name: delete
  Relation ID: 2
  Description: Refers to a resource that can be used to delete a resource in a collection of resources.
  Reference: [RFCXXXX]

- Relation Name: update
  Relation ID: 3
  Description: Refers to a resource that can be used to update the state of the form context.

Reference: [RFCXXXX]

- Relation Name: search
  Relation ID: 4
  Description: Refers to a resource that can be used to search the form context.
  Reference: [RFCXXXX]

7. References

7.1. Normative References

[1] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[2] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[3] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <http://www.rfc-editor.org/info/rfc6570>.

[4] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <http://www.rfc-editor.org/info/rfc5785>.

[5] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[6] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <http://www.rfc-editor.org/info/rfc7320>.

[7] 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>.

7.2. Informative References

[8] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

[9] Berners-Lee, T., "Cool URIs don't change", 1998, <http://www.w3.org/Provider/Style/URI.html>.

[10] 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>.

[11] Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208>.

[12] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

[13] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., Yergeau, F., and J. Cowan, "Extensible Markup Language (XML) 1.1 (Second Edition)", World Wide Web Consortium Recommendation REC-xml11-20060816, August 2006, <http://www.w3.org/TR/2006/REC-xml11-20060816>.

[14] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[15] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[16] 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>.

[17] Fielding, R., "REST APIs must be hypertext-driven", October 2008, <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>.

[18] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

[19] Hammer-Lahav, E., Ed. and B. Cook, "Web Host Metadata", RFC 6415, DOI 10.17487/RFC6415, October 2011, <http://www.rfc-editor.org/info/rfc6415>.

[20] Li, L., Wei, Z., Luo, M., and W. Chou, "Requirements and Design Patterns for REST Northbound API in SDN", draft-li-sdnrg-design-restapi-02 (work in progress), July 2016.

[21] Masinter, L., "MIME and the Web", draft-masinter-mime-web-info-02 (work in progress), January 2011.

[22] Richardson, L. and M. Amundsen, "RESTful Web APIs", O'Reilly Media, ISBN 978-1-4493-5806-8, September 2013.

[23] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[24] 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>.

[25] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <http://www.rfc-editor.org/info/rfc6906>.

Acknowledgements

Jan Algermissen, Mike Amundsen, Mike Kelly, Julian Reschke, and Erik Wilde provided valuable input on link and form relation types.

Thanks to Olaf Bergmann, Carsten Bormann, Stefanie Gerdes, Ari Keranen, Michael Koster, Matthias Kovatsch, Teemu Savolainen, and Bilhanan Silverajan for helpful comments and discussions that have shaped the document.

Some of the text in this document has been borrowed from [5], [6], [17], [20], and [21]. All errors are my own.

This work was funded in part by Nokia.

Author's Address

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

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

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: July 10, 2017

G. Selander
F. Palombini
Ericsson AB
K. Hartke
Universitaet Bremen TZI
January 6, 2017

Requirements for CoAP End-To-End Security
draft-hartke-core-e2e-security-reqs-02

Abstract

This document analyses threats to CoAP message exchanges traversing proxies and derives the security requirements for mitigating those threats.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 10, 2017.

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a Web application protocol designed for constrained nodes and networks [RFC7228]. CoAP makes use of Datagram Transport Layer Security (DTLS) [RFC6347] for security. At the same time, CoAP relies on proxies for scalability and efficiency. Proxies reduce response time and network bandwidth use by serving responses from a shared cache or enable clients to make requests that these otherwise could not make.

CoAP proxies need to perform a number of operations on requests and responses to fulfill their purpose, which requires the DTLS security associations to be terminated at each proxy. The proxies therefore do not only have access to the data required for performing the desired functionality, but are also able to eavesdrop on or manipulate any part of the CoAP payload and metadata exchanged between client and server, or inject new CoAP messages without being protected or detected by DTLS.

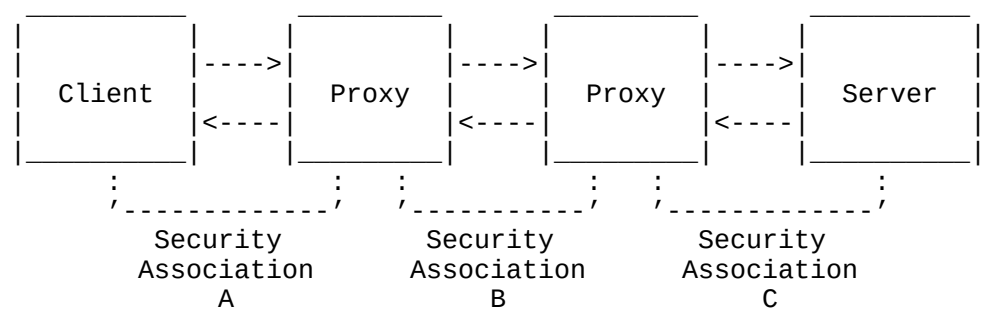


Figure 1: Hop-by-Hop Security

One way to mitigate this threat is to secure CoAP communication at the application layer using an object-based security mechanism such as CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg] instead of or in addition to the security mechanisms at the network layer or transport layer. Such a mechanism can provide "end-to-end

security" at the application layer (Figure 2) in contrast to the "hop-by-hop security" that DTLS provides (Figure 1).

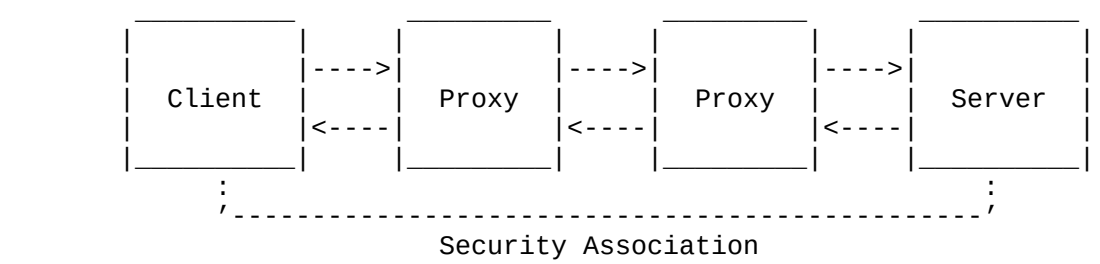


Figure 2: End-to-End Security

This document analyses security aspects of sensor and actuator communications over CoAP that involve proxies (Section 2) and publish-subscribe brokers (Section 3). The analysis is based on the identification of assets associated with these communications and considering the potential threats posed by proxies to these assets. The threat analysis provides the basis for deriving security requirements that a solution for CoAP end-to-end security should meet.

1.1. Assets and Scope

In general, there are the following assets that need to be protected:

- The devices at the two ends and their (often very constrained) system resources such as available memory, storage, processing power and energy.
- The physical environment of the devices fitted with sensors and actuators. Access to the physical environment is assumed to be provided through CoAP resources that allow a remote entity to retrieve information about the physical environment (such as the current temperature) or to produce an effect on the physical environment (such as the activation of a heater).
- The communication infrastructure linking the two devices, which often contains some very constrained networks.
- The data generated and stored in the involved devices.

An intermediary can directly interfere with the interactions between the two ends and thereby have an impact on all these assets. For example, flooding a device with messages has an impact on system resources, and the successful manipulation of an actuator command

(data generated by an involved device) can have a severe impact on the physical environment. An intermediary can also affect the communication infrastructure, e.g., by dropping messages.

Even if an intermediary is trustworthy, it may be an attractive target for an attack, since such nodes are aggregation points for message flows and may be an easier target from the Internet than the sensor and actuator nodes residing behind them. An intermediary may become subject to intrusion or be infected by malware and perform the attacks of a man-in-the-middle.

The focus of this document is on threats from intermediaries to interactions between two CoAP endpoints. Other types of threats, for example, attacks involving physical access to the CoAP-speaking devices, are out of scope of this document.

Since intermediaries may perform a service for the interacting endpoints, there is a trade-off between the intermediaries' desired functionality and the ability to mitigate threats to the endpoints executed through an intermediary.

1.2. Terminology

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641].

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

2. Proxying

To assess what impact various threats have to the assets, we need to specify and analyse how the proxies operate.

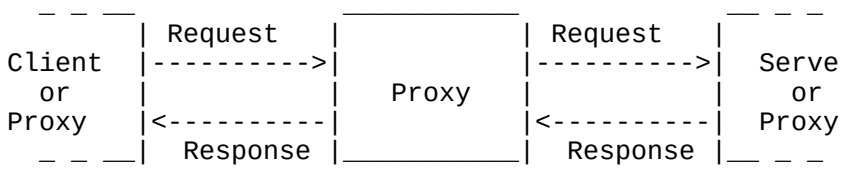


Figure 3: A Proxy

Generally speaking, the functionality of a proxy is to receive a request from a client and to send a response back to that client. There are two ways for the proxy to satisfy the request:

- The proxy constructs and sends a request to the server indicated in the client's request, receives a response from that server and uses the received data to construct the response to the client.
- The proxy uses cached data to construct the response to the client.

In both cases, the proxy needs to read some parts both of the request from the client and the response from the server to accomplish its task.

The following subsections analyse the threats posed by a proxy from the perspective of the client on the one hand (Section 2.1.1) and the perspective of the server on the other hand (Section 2.1.2). Section 2.2 then presents the design space for possible security solutions to mitigate the threats.

2.1. Threats and Security Requirements

2.1.1. Client-side

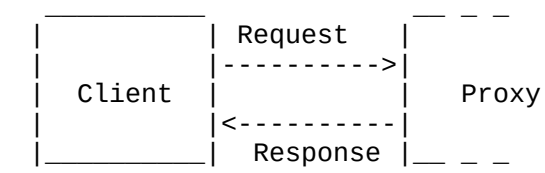


Figure 4: The Client End

The client sends a request to the proxy and waits for a response.

From the perspective of the client, there are three possible flows:

- The client receives a response.
  Reasons include:
  - The proxy duly processed the request and returns a response based on data it obtained from the origin server.
  - The proxy encountered an unexpected condition and returns an error response according to specification (e.g., 5.02 Bad Gateway or 5.04 Gateway Timeout).
  - (Threat 1:) The proxy spoofs a response. For example, the proxy could return a stale or outdated response based on data it previously obtained from the server or some fourth party, or could craft an illicit response itself.
  - (Threat 2:) The proxy duly processed the request but delays the return of the response.
- The client does not receive a response.
  Reasons include:
  - The client times out too early.
  - (Threat 3:) The proxy withholds the response.
- The client receives too many responses.
  Reasons include:
  - (Threat 4:) The proxy floods the client with responses.

Furthermore, there are threats related to privacy:

- (Threat 5:) The proxy eavesdrops on the data in the request from the client.

- (Threat 6:) The proxy measures the size, frequency or distribution of requests from the client.

Note that "cache poisoning" -- the case of caching injected incorrect responses -- is covered from the point of view of the client: it may result in the client receiving a spoofed message, or being flooded, or affect other nodes such that the client times out too early.

2.1.1.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the client MUST verify that the response is an "authentic response" before processing it.

The definition of an "authentic response" depends on the desired proxy functionality and protection level (see Section 2.2), but usually means that the client can obtain proof for some or all of the following things:

- that the requested action was executed by the origin server;

- that the data originates from the origin server and has not been altered on the way;

- that the data matches the specifications of the request (such as the target resource);

- that the data is fresh (when the data is cacheable);

- that the data is in sequence (when observing a resource).

The proof can, for example, include a message authentication code that the proxy obtains from the origin server and includes in the response or an additional challenge-response roundtrip.

Exception: A CoAP proxy is specified to return an error response (such as 5.02 Bad Gateway or 5.04 Gateway Timeout) when it encounters an error condition. Since the condition occurs at the proxy and not at the origin server, the response will not be an "authentic response" according to the above definition. (A proxy cannot obtain a proof that the server is unreachable from an unreachable server.) Thus, a client cannot tell if the proxy sends the response according to specification or if it spoofs the response. This threat is NOT REQUIRED to be mitigated by the security solution.

2.1.1.2. Threat 2: Delaying

This threat is REQUIRED to be mitigated by the security solution. Delay attacks are important to mitigate in certain applications, e.g., when using CoAP with actuators. A problem statement and candidate solution can be found in [I-D.mattsson-core-coap-actuators].

2.1.1.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a client cannot tell if the proxy does not send a response because it is hasn't received a response from the origin server yet or if it intentionally withholds the response.

2.1.1.4. Threat 4: Flooding

A CoAP client is specified to reject any response that it does not expect. This can happen before the client verifies whether the response is authentic. Therefore, a flood of responses is primarily a threat to the system resources of the client, in particular to its energy. This threat is NOT REQUIRED to be mitigated by the security solution, but a client SHOULD generally defend against flooding attacks.

2.1.1.5. Threat 5: Eavesdropping

This threat is REQUIRED to be mitigated by the security solution: clients MUST confidentiality protect the data in the requests they send.

Note that this requirement is in conflict with the requirement that the proxy needs to be able to read some parts of the requests in order to accomplish its task. Section 2.2 analyses which parts can be encrypted depending on the desired proxy functionality and protection level. In general, a security solution SHOULD confidentiality protect all data that is not needed by the proxy to accomplish its task.

The keys used for confidentiality protection MUST provide forward secrecy.

2.1.1.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the

feasibility to protect against various threats, e.g., by obfuscating parameters transported in plain text, aligning message flow and traffic between the different cases, adding padding so different messages become indistinguishable, etc.

2.1.2. Server-side

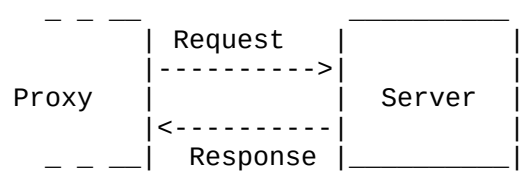


Figure 5: The Server End

A server listens for a request and returns a response.

From the perspective of the server, there are three possible flows:

- The server receives a request.
  Reasons include:
  - The proxy makes a request on behalf of a client according to specification.
  - The proxy makes a request (e.g., to validate cached data) on its own behalf.
  - (Threat 1:) The proxy spoofs a request.
  - (Threat 2:) The proxy sends a request with delay.
- The server does not receive a request.
  Reasons include:
  - The proxy does not need to send a request.
  - (Threat 3:) The proxy withholds a request.
- The server receives too many requests.
  Reasons include:
  - (Threat 4:) The proxy floods the server with requests.

A proxy eavesdropping or inferring information from messages it operates on has an impact on a server in the same way as on a client:

- (Threat 5:) The proxy eavesdrops on the data in the response from the server.
- (Threat 6:) The proxy measures the frequency and distribution of responses from the server.

2.1.2.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the server MUST verify that the request is an _authentic request_ before processing it.

The definition of an "authentic request" depends on the desired proxy functionality and protection level (Section 2.2), but usually means that the server can obtain proof for some or all of the following things:

- that the proxy acts on behalf of a client;
- that the data originates from the client and has not been altered on the way;
- that the request has not been received previously.

The proof can, for example, include a message authentication code that the proxy obtains from the client and includes in the request or a challenge-response roundtrip.

Exception: A CoAP proxy may make certain requests without acting on behalf of a client (e.g., to validate cached data). Since such a request does not originate from a client, the server cannot tell if the proxy sends the request according to specification or if it spoofs the request. It is up to the security solution how this issue is addressed.

2.1.2.2. Threat 2: Delaying

This threat is REQUIRED to be mitigated by the security solution; see Section 2.1.1.2.

2.1.2.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a server cannot tell if the proxy does not send a request because it has no work to do or if it intentionally withholds a request.

2.1.2.4. Threat 4: Flooding

This threat is NOT REQUIRED to be mitigated by the security solution in particular, but a server SHOULD generally defend against flooding attacks.

2.1.2.5. Threat 5: Eavesdropping

This threat is REQUIRED to be mitigated by the security solution; see Section 2.1.1.5.

2.1.2.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution; see Section 2.1.1.6.

2.2. Solutions

A security solution has to find a trade-off between desired proxy functionality (such as caching) and the provided level of protection. From this trade-off results the definition of what constitutes an "authentic request" or "authentic response" and when a request or response is considered confidentiality protected.

This section presents two exemplary choices of trade-offs:

- The first case focuses on a high protection level by tying requests and responses uniquely together and confidentiality protecting as much as possible, at the cost of reduced proxy functionality.

- The second case aims to preserve proxy functionality as much as possible, at the cost of reduced confidentiality protection.

For both cases, this section presents an overview of the functionality and processing rules of the proxy and analyses the required authenticity and confidentiality properties of requests and responses. Due to space constraints, the analysis is limited to the CoAP header fields, the payload and the request and response options shown in Table 1.

Requests	Responses
Accept	Content-Format
Content-Format	ETag
ETag	Location-Path
If-Match	Location-Query
If-None-Match	Max-Age
Observe	Observe
Proxy-Scheme	
Proxy-Uri	
Uri-Host	
Uri-Port	
Uri-Path	
Uri-Query	

Table 1: Analysed CoAP Options

Note that, since CoAP was not designed with end-to-end security in mind, a security solution extends the applicability of CoAP beyond its original scope.

2.2.1. Forwarding

In this case we study forwarding functionality of a CoAP forward proxy, and assume that caching is disabled. This is applicable to many security critical use cases where a response needs to be securely linked to a unique request from a client and cannot be re-used with another request.

There may be a unique response for each request (see Figure 6) or multiple responses for each request (see Figure 7).

2.2.1.1. Examples

Examples of the need for unique response for each request include alarm status retrieval and actuator command confirmation.

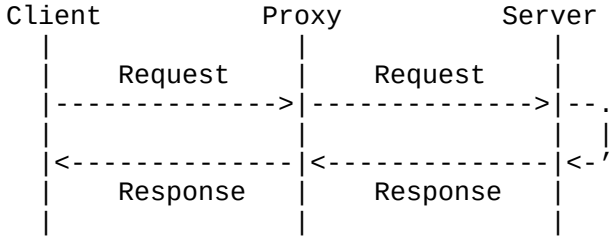


Figure 6: Message Flow with a Unique Response for Each Request

Example: Alarm status retrieval

Figure 6 can be seen as an illustration of a message exchange for a client requesting the alarm status (e.g., GET /alarm_status) from a server. Since the client wants to ensure that the alarm status received is reflecting the current alarm status and not a cached or spoofed response to the same resource, it must be able to verify that the received response is a response to this particular request made by the client. Therefore, the response must be securely linked to the request.

Example: Actuation confirmation

Another example for which Figure 6 serves as illustration is the confirmation of an actuator request. In this case a client, say in an industrial control system, requests a server that a valve should be turned to a certain level, e.g. PUT /valve_42/level with payload "3". In order for the client to correctly evaluate the result of a potential changed valve level, it is important that the client gets a confirmation how the server responded to the requested change, e.g., whether the request was performed or

not. Again, the client wants to ensure that the response is reflecting the result of this particular actuation request made by the client and not a cached or spoofed response. Therefore, the response must be securely linked to the request.

An example of the use of multiple responses for each request is in security critical monitoring scenarios where time synchronization cannot be guaranteed. By avoiding repeated requests from the same client to the same resource, constrained node resources and bandwidth is saved.

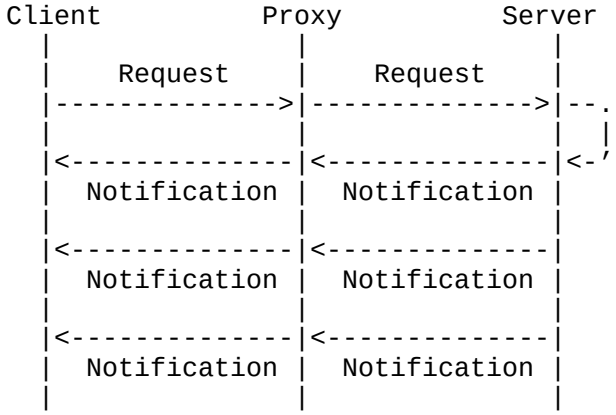


Figure 7: Message Flow of Notifications of Linked to a Unique Request

Example: Secure parameter monitoring

Figure 7 can be seen as an illustration of a message exchange for a client monitoring an important parameter measured by the server, e.g., in a medical or process industry application. The client makes a subscription request for a resource and the server responds with notifications, e.g. providing updates to the parameter on regular time intervals.

The client wants to ensure that the first received notification reflects the current parameter value and that subsequent notifications are timely updates of the initial request. Since notifications may be lost or reordered, the client needs to be able to verify the order of the messages, as sent by the server. By monitoring the received messages and the time they are received, the client can detect missing notifications and take appropriate action.

2.2.1.2. Functional Requirements

FR1.1 The caching functionality MUST be inhibited; the CoAP option Max-Age of the responses SHALL be 0 (see Section 5.7.1 of [RFC7252]).

FR1.2 To limit information leaking about the resource (see Section 2.2.1.5) the Proxy-Uri does not contain Uri-Path or Uri-Query.

2.2.1.3. Processing Rules

In this case, the desired proxy functionality is to forward a translated request to the determined destination. There are two modes of operation for requests: Either using the Proxy-Uri option (PR1.1) or using the Proxy-Scheme option together with the Uri-Host, Uri-Port, Uri-Path and Uri-Query options (PR1.2).

PR1.1 The Proxy-Uri option contains the request URI including request scheme (e.g. "coaps://"); the Proxy-Scheme and Uri-* options are not present.

If the proxy is configured to forward requests to another proxy, then it keeps the Proxy-Uri option; otherwise, it splits the option into its components, adds the corresponding Uri-* options and removes the Proxy-Uri option. Then it makes the request using the request scheme indicated in the Proxy-Uri.

PR1.2 The Proxy-Scheme option and the Uri-* options together contain the request URI; the Proxy-Uri option is not present.

If the proxy is configured to forward requests to another forwarding proxy, then it keeps the Proxy-Scheme and Uri-* options; otherwise, it removes the Proxy-Scheme option. Then it makes the request using the request scheme indicated in the removed Proxy-Scheme option.

PR1.3 Responses are forwarded by the proxy, without any modification.

2.2.1.4. Authenticity

A request is considered authentic by the server (Section 2.1.2.1) if the server can obtain proof for all of the following things:

A1.1 that the proxy acts on behalf of a client;

A1.2 that the following parts of the request originate from the client and have not been altered on the way:

* the CoAP version,
* the request method,
* all options except Proxy-Uri, Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path and Uri-Query, and
* the payload, if any.

A1.3 that the effective request URI originates from the client and has not been altered on the way;

A1.4 that the request has not been received previously;

A1.5 that the request from the client to the proxy was sent recently.

A response is considered authentic by the client (Section 2.1.1.1) if the client can obtain proof for all of the following things:

A1.6 that the following parts of the response originate from the server and have not been altered on the way:

* the CoAP version,
* the response code,
* all options, and
* the payload, if any.

A1.7 that the response corresponds uniquely to the request sent by the client.

A1.8 that the response has not been received previously;

A1.9 that the response from the server to the proxy was sent recently;

A1.10 that the response is in sequence if there are multiple responses.

2.2.1.5. Confidentiality

The following parts of the message are confidentiality protected (Section 2.1.1.5):

- all options except Proxy-Uri, Proxy-Scheme, Uri-Host and Uri-Port;
- the payload, if any.

2.2.2. Caching

In this case we study caching: how a proxy may serve the same cached response to multiple clients requesting the same resource.

The caching functionality protects communication-constrained servers from repeated requests for the same resources, possibly originating from different clients. This saves system resources, bandwidth, and round-trip time.

There may be one response for each request (see Figure 8) or multiple responses for each request (see Figure 9).

2.2.2.1. Examples

The first example is a simple case of caching.

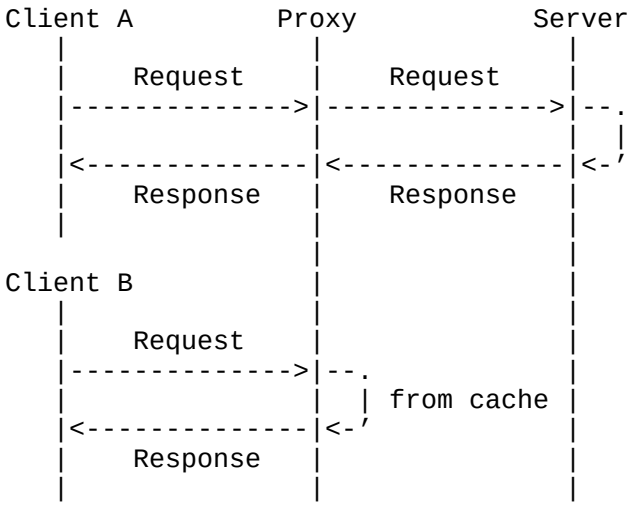


Figure 8: Message Flow for Cached Responses

Example: Caching

In Figure 8, client A requests the proxy to make a certain request to the server and to return the server's response. The proxy services the request by making a request message to the server according to the processing rules. If the server returns a cacheable response, then the proxy stores the response in its cache, performs any necessary translations, and forwards it to the client. Later, client B makes an equivalent request to the proxy that the proxy services by returning the response from its cache. Both client A and B want to verify that the response is valid.

In addition to multiple clients' requests being served by one response, each request may result in multiple responses. The difference compared to Section 2.2.1 is that in this example multiple clients may be served with the same response, further saving server resources.

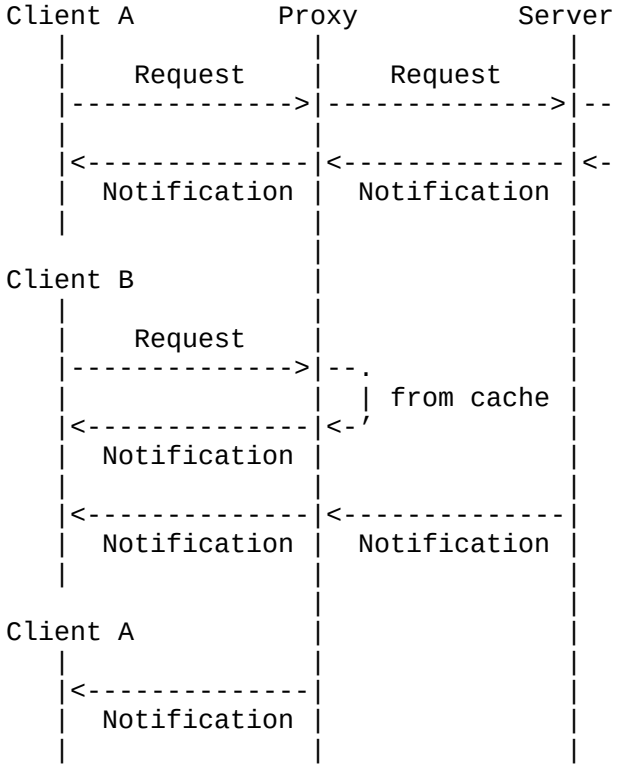


Figure 9: Message Flow for Observe with Multiple Observers

Example: Observe with caching

In Figure 9, the server exposes an observable resource (e.g., the current reading of a temperature sensor). Multiple clients are interested in the current state of the resource and observe it using the CoAP resource observation mechanism [RFC7641]. The goal is to keep the state observed by the clients closely in sync with the actual state of the resource at the server. Another goal is to minimize the burden on the server by moving the task to fan out notifications to multiple clients from the server to the proxy.

2.2.2.2. Functional Requirements

The security solution SHOULD protect requests and responses in a way that a proxy can perform the following tasks:

FR2.1 Storing a cacheable response in a cache. This requires that the proxy is able to calculate the cache-key of the request. Cacheable responses include 2.05 (Content) responses and all error responses.

FR2.2 Returning a fresh response from its cache without contacting the server.

FR2.3 Performing validation of a response cached by the proxy as well as validation of a response cached by the client.

FR2.4 Observing a resource on behalf of one or more clients.

2.2.2.3. Processing Rules

The proxy complies with the forwarding rules PR1.1 - 1.3 (Section 2.2.1.3) and the rules below. The rules below have priority.

PR2.1 If the proxy receives a request where the cache key matches that of a cached fresh response, then the proxy discards the request and replies with that response, else it makes a translated request.

PR2.2 The proxy caches and forwards cacheable responses. If there is already a response in the cache with the cache key of the corresponding request, then the old response in the cache is marked as stale.

PR2.3 If the proxy receives a request that contains an ETag option and the proxy has a fresh response with the same cache key and ETag, then the proxy replies to the request with a 2.03 (Valid) response without payload, else it forwards a translated request.

PR2.4 The proxy updates the Max-Age option according to the Max-Age associated with the resource representation it receives, decreasing its value to reflect the time spent in the cache.

PR2.5 If the request contains an Accept option and if there is a fresh response that matches the cache key for the corresponding request except for the Accept option, and if the Content-Format of the response matches that of the Accept option, then the proxy forwards the cached response to the requesting client.

2.2.2.4. Authenticity

A request is considered authentic by the server (Section 2.1.2.1) if the server can obtain proof for all of the following things:

A2.1 that the following parts of the request originate from the client and have not been altered on the way:

* the CoAP version,
* the request method,
* all options except ETag, Observe, Proxy-Uri, Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path and Uri-Query, and
* the payload, if any.

A2.2 that the effective request URI originates from the client and has not been altered on the way;

A response is considered authentic by the client (Section 2.1.1.1) if the client can obtain proof for all of the following things:

A2.3 that the following parts of the response originate from the server and have not been altered on the way:

* the CoAP version,
* the response code,
* all options except Max-Age and Observe, and
* the payload, if any.

A2.4 that the response matches the specifications of the request;

A2.5 that the data is fresh (when the response is cacheable);

A2.6 that the response is in sequence (when observing a resource).

2.2.2.5. Confidentiality

No parts of a request are confidentiality protected (Section 2.1.2.5).

A response is considered confidentiality protected (Section 2.1.2.5) if the payload of the response is confidentiality protected.

3. Publish-Subscribe

Much of the concerns about proxies as described previously in this document also applies to other kinds of intermediary nodes. In this section we study brokers in a publish-subscribe setting [I-D.ietf-core-coap-pubsub]. The case of combining brokers and proxies is out of scope for this version of the document.

There are different ways for a pub-sub broker to operate. We consider the following broker operations:

- The broker receives a request for a topic from a subscriber.
- The broker receives a request for a publication to a topic from a publisher and forwards the request to the subscribers of the topic.

We consider the setting where there is a security association between publisher and subscriber such that the publications can be protected during transfer, see Figure 10.

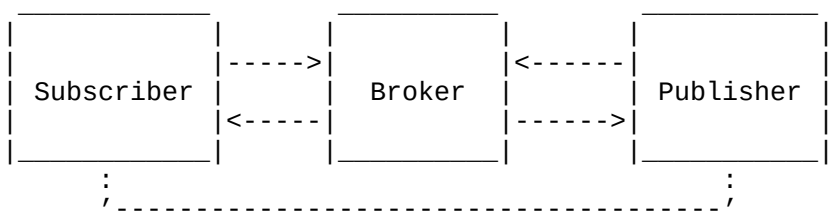


Figure 10: Publisher-to-Subscriber Security

Since there is no security association with the broker, we only consider the subscribe and publish functionality of the broker. Note that the broker needs to read the topic to accomplish this task.

3.1. Threats and Security Requirements

3.1.1. Subscriber-side

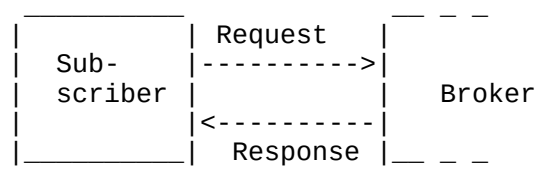


Figure 11: The Subscriber End

The subscriber sends a subscription request to the broker and waits for a response.

From the perspective of the subscriber, there are three possible flows:

- The subscriber receives a response.
  Reasons include:
  - The broker duly processed the request and returns a response based on data it obtained from a publisher.
  - The subscriber made a bad request and the broker returns an error response accordingly (e.g., 4.04 Not Found).
  - The broker encountered an unexpected condition and returns an error response accordingly (e.g., 5.03 Service Unavailable).
  - (Threat 1:) The broker spoofs a response.
  - (Threat 2:) The broker duly processed the request but delays the return of a response.
- The subscriber does not receive a response.
  Reasons include:
  - The subscriber times out too early.
  - (Threat 3:) The broker withholds a response.
- The subscriber receives too many responses.
  Reasons include:
  - (Threat 4:) The broker floods the subscriber with responses.

Furthermore, there are threats related to privacy:

- (Threat 5:) The broker eavesdrops on the data in the request from the subscriber.

- (Threat 6:) The broker measures the size, frequency or distribution of requests from the subscriber.

Note that "topic poisoning" -- the case of storing injected incorrect publications -- is covered from the point of view of the subscriber: it may result in the subscriber receiving a spoofed message, or being flooded, or affect other nodes such that the subscriber times out too early.

3.1.1.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the subscriber MUST verify that a response is an "authentic publication" before processing it.

The definition of an "authentic publication" depends on the setting (Section 3.2), but usually means that the subscriber can obtain proof for some or all of the following things:

- that the data matches the specifications of the request (such as the topic);
- that the data originates from a publisher that is authorized to publish to the topic;
- that the data has not been altered on the way between publisher and subscriber;
- that the data is fresh (when the data is cacheable);
- that the data is in sequence (when observing a topic).

The proof can, for example, include a message authentication code that the proxy obtains from the origin server and includes in the response or an additional challenge-response roundtrip.

Exception: A CoAP server like the broker is specified to return an error response (such as 4.04 Not Found or 5.03 Service Unavailable) when it encounters an error condition. Since the condition occurs at the broker and not at the publisher, the response will not be an "authentic response" according to the above definition. Thus, a subscriber cannot tell if the broker sends the error response according to specification or if it spoofs the response. This threat is NOT REQUIRED to be mitigated by the security solution.

3.1.1.2. Threat 2: Delaying

This threat is NOT REQUIRED to be mitigated by the security solution.

3.1.1.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a subscriber cannot tell if the broker does not send a response because it is hasn’t received a publication from the publisher yet or if it intentionally withholds the response.

3.1.1.4. Threat 4: Flooding

A CoAP client like the subscriber is specified to reject any response that it does not expect. This can happen before the subscriber verifies if the response is authentic. Therefore, a flood of responses is primarily a threat to the system resources of the client, in particular to its energy. This threat is NOT REQUIRED to be mitigated by the security solution, but a subscriber SHOULD generally defend against flooding attacks.

3.1.1.5. Threat 5: Eavesdropping

This threat is NOT REQUIRED to be mitigated: The broker needs to read all parts of the request from the subscriber to accomplish its task.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the feasibility to protect against various threats, e.g., by obfuscating topic content.

3.1.1.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the feasibility to protect against various threats, e.g., by obfuscating parameters transported in plain text, aligning message flow and traffic between the different cases, adding padding so different messages become indistinguishable, etc.

3.1.2. Publisher-side

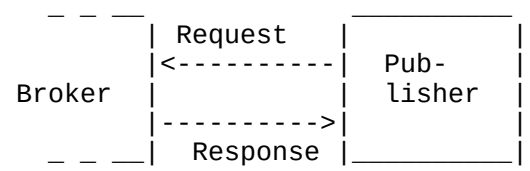


Figure 12: The Publisher End

The publisher sends a publication request to the broker and waits for a response.

The threat of the broker eavesdropping on the data in the publication request is REQUIRED to be mitigated by the security solution: publishers MUST confidentiality protect the data in the requests they send. This excludes parts that the broker needs to read to perform its job, e.g., the topic.

The threat of the broker measuring the size, frequency or distribution of publication requests is NOT REQUIRED to be mitigated by the security solution; see Section 3.1.1.6.

The broker is in full control of the response and may therefore arbitrarily spoof, delay, or withhold it. This threat is NOT REQUIRED to be mitigated. For example, a proof that the broker has notified all subscribers is NOT REQUIRED.

3.2. Solutions

3.2.1. Brokering

In this case we study brokering: how a broker may serve the same publication to multiple subscribers observing the same topic.

The brokering functionality protects communication-constrained publishers from repeated requests for the same resources, possibly originating from different subscribers. This saves system resources, bandwidth, and round-trip time.

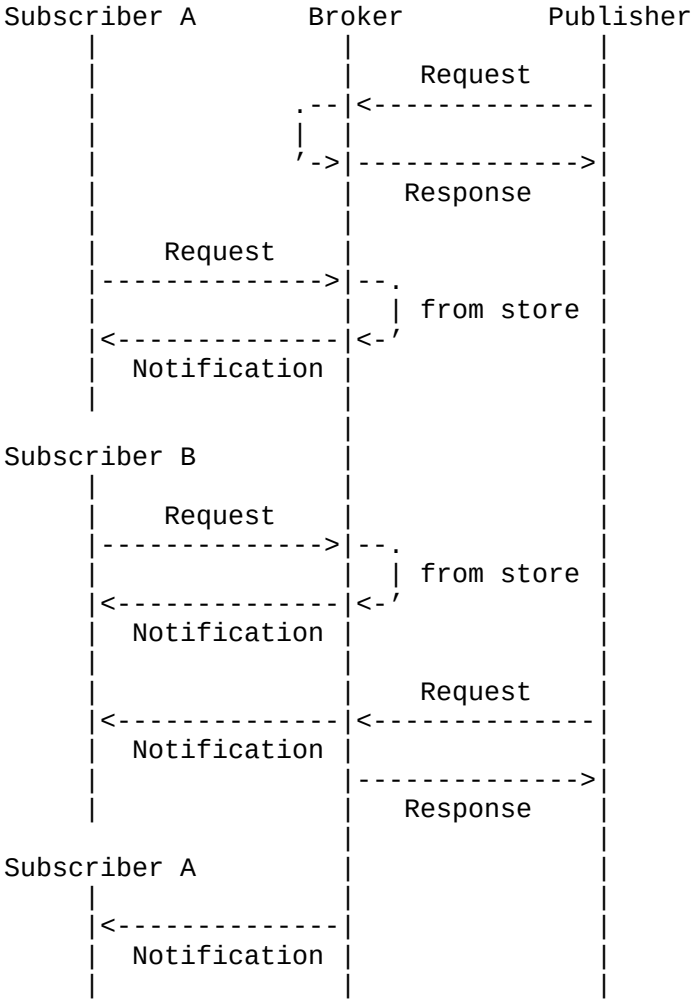


Figure 13: Message Flow for Publish Subscribe

Example

In Figure 13, the publisher publishes to a topic (e.g., the current reading of a temperature sensor). Multiple subscribers are interested in the current state of the topic and observe the topic as specified in [I-D.ietf-core-coap-pubsub]. The goal is to keep the state observed by the subscribers closely in sync with the actual state of the resource at the publisher. Another goal is to minimize the burden on the publisher by moving the task to fan out notifications to multiple subscribers from the publisher to the broker.

3.2.1.1. Functional Requirements

The security solution SHOULD protect subscription and publication requests in a way that a broker can perform the following tasks:

FR3.1 Storing publications. This requires that the broker is able to read the topic of the request.

FR3.2 Returning a stored publication without contacting the publisher.

3.2.1.2. Processing Rules

The broker complies with the following rules:

PR3.1 If the broker receives a request where the topic matches that of a cached publication, then the broker responds with that publication.

PR3.2 The broker caches and forwards publication notifications.

3.2.1.3. Authenticity

A publication is considered authentic by the subscriber if the subscriber can obtain proof for all all of the following things:

A3.1 that the payload is associated to the topic;

A3.2 that the payload has not been altered since published;

A3.3 that the publication is in sequence.

3.2.1.4. Confidentiality

The payload of a publication request is confidentiality protected.

4. Security Considerations

This document is about security; as such, there are no additional security considerations.

5. IANA Considerations

This document includes no request to IANA.

6. References

6.1. 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>.

[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>.

[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>.

6.2. Informative References

[I-D.ietf-core-coap-pubsub] Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-00 (work in progress), October 2016.

[I-D.ietf-cose-msg] Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

[I-D.mattsson-core-coap-actuators] Mattsson, J., Fornehed, J., Selander, G., and F. Palombini, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-02 (work in progress), November 2016.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[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>.

Acknowledgments

Thanks to Ari Keranen, John Mattsson, Jim Schaad and Ludwig Seitz for helpful comments and discussions that have shaped the document.

Authors' Addresses

Goeran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen 28359
Germany

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

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: August 31, 2017

P. van der Stok
consultant
K. Hartke
Universitaet Bremen TZI
February 27, 2017

The 'Pending' Response Code for the Constrained Application Protocol (CoAP)
draft-hartke-core-pending-00

Abstract

This document proposes a new CoAP response code, 2.06 Pending. A CoAP server can use this response code to signal that it has accepted the request but has not yet started processing it or that processing the request will take longer than a client is typically willing to wait for a response. A 2.06 response can include status information and indicate a location where the result will become available.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a request/response protocol not unlike HTTP. CoAP defines no upper bound for the time between a request and the resulting response. E.g., in CoAP over UDP, a server is expected to return an empty Acknowledgement to the client if it cannot provide a response right away, but there is no limit on when the server should return the Separate Response.

In particular in the case of requests with long processing times, a CoAP client faces the problem that it cannot easily determine how long it should wait for the response and whether the CoAP server is even still processing the request. Long processing times occur, for example, when requests need manual intervention to authorize their processing, or when they perform a long sequence of remote actions. An example is provided by the "possibly long" authorization request specified in EST-coaps [I-D.vanderstok-ace-coap-est].

This document proposes a new CoAP response code, 2.06 Pending. The semantics of this response code are modelled after the HTTP [RFC7231] 202 (Accepted) status code:

> The 202 (Accepted) status code indicates that the request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. [...] The representation sent with this response ought to describe the request's current status and point to (or embed) a status monitor that can provide the user with an estimate of when the request will be fulfilled.

The 2.06 (Pending) response code is not meant for overload cases, which are better handled by the 5.03 (Service Unavailable) response code.

1.1. Terminology

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641].

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].

2. 2.06 Pending

A 2.06 (Pending) response in reply to a GET request indicates that the target resource exists but no representation of the resource is available yet. The Max-Age Option indicates after what time a client should retry its GET request to retrieve the representation. The client MAY observe the resource (see Section 2.1) to be notified when the representation becomes available.

A 2.06 (Pending) response in reply to a POST request indicates that the result of processing the request is not available yet, for example, because the server needs more time to process the request than a client is typically willing to wait for a response. The server MAY specify a location using the Location-* options where the result will become available. If the server does not specify a location, the result will become available at the target resource of the POST request. To receive the result, the client MAY poll or observe the resource at the specified location using a GET request. The Max-Age Option indicates how long the client should wait before making the GET request.

A 2.06 (Pending) response MAY contain a payload that represents the progress of processing the original request or any other status information. The content format of this representation is specified by the Content-Format Option.

A 2.06 (Pending) response is cacheable, but cannot be validated. If it contains Location-* options, it invalidates any cached response for the resource at the specified location; otherwise, it invalidates any cached response for the target resource of the request.

As a consequence of being cacheable, a 2.06 (Pending) response in reply to a POST request makes the POST method temporarily idempotent: until Max-Age expires, any POST request with the same cache-key -- be

it from the same client or any another client -- can yield the same 2.06 (Pending) response. (This is the same behavior as for 4.xx and 5.xx error responses in reply to POST requests.)

2.1. Observing Resources

When a client registers to observe [RFC7641] a resource for which no representation is available yet, the server MAY send one or more 2.06 (Pending) notifications before sending the first 2.05 (Content) or 2.03 (Valid) notification. The possible resulting sequence of notifications is shown in Figure 1.

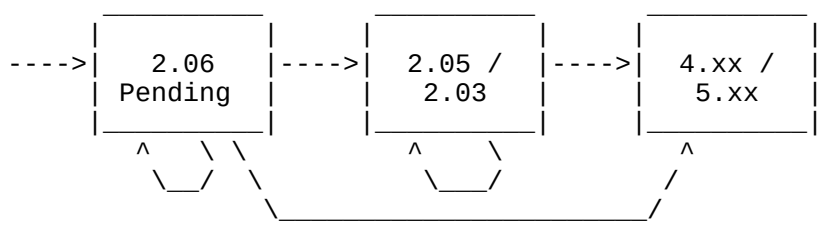


Figure 1: Sequence of Notifications

Unless the server is not willing to add the client to the list of observers, each 2.06 (Pending) notification MUST include an Observe Option with a sequence number as specified in [RFC7641]. Otherwise, the registration request falls back to a normal GET request.

3. Security Considerations

This section analyses the possible threats related to 2.06 (Pending) responses. It is meant to inform protocol and application developers about the security limitations of the response code as described in this document.

A 2.06 (Pending) response is subject to the same general security considerations as all CoAP responses as described in Section 11 of [RFC7252]. Specifically, the security considerations for the response code are closest to those of the Observe Option as stated in Section 7 of [RFC7641], because the server stores additional state over an extended period.

2.06 (Pending) responses are secured following the recommendations for the existing CoAP response codes as specified in Section 9 of [RFC7252]. When additional security techniques are standardized for CoAP (e.g., based on object security), these are then also available for securing the responses.

4. IANA Considerations

This document adds the 2.06 (Pending) response code to the "CoAP Response Codes" registry.

Code	Description	Reference
2.06	Pending	[RFCXXXX]

Table 1: New CoAP Response Codes

5. References

5.1. 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>.

[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>.

[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>.

5.2. Informative References

[I-D.vanderstok-ace-coap-est] Kumar, S. and P. Stok, "EST based on DTLS secured CoAP (EST-coaps)", draft-vanderstok-ace-coap-est-00 (work in progress), December 2016.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

Authors' Addresses

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

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: Standards Track
Expires: September 13, 2017

M. Koster
SmartThings
A. Keranen
J. Jimenez
Ericsson

March 12, 2017

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-ietf-core-coap-pubsub-01

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 13, 2017.

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports machine-to-machine communication across networks of constrained devices. CoAP uses a request/response model where clients make requests to servers in order to request actions on resources. Depending on the situation the same device may act either as a server or a client.

One important class of constrained devices includes devices that are intended to run for years from a small battery, or by scavenging

energy from their environment. These devices have limited reachability because they spend most of their time in a sleeping state with no network connectivity. Devices may also have limited reachability due to certain middle-boxes, such as Network Address Translators (NATs) or firewalls. Such middle-boxes often prevent connecting to a device from the Internet unless the connection was initiated by the device.

This document specifies the means for nodes with limited reachability to communicate using simple extensions to CoAP. The extensions enable publish-subscribe communication using a broker node that enables store-and-forward messaging between two or more nodes. Furthermore the extensions facilitate many-to-many communication using CoAP.

2. Terminology

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

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

This specification makes use of the following additional terminology:

Publish-Subscribe (pubsub): A messaging paradigm where messages are published to a broker and potential receivers can subscribe to the broker to receive messages. The publishers do not (need to) know where the message will be eventually sent: the publications and subscriptions are matched by a broker and publications are delivered by the broker to subscribed receivers.

CoAP pubsub service: A group of REST resources, as defined in this document, which together implement the CoAP pubsub service.

CoAP pubsub Broker: A server node capable of receiving messages (publications) from and sending messages to other nodes, and able to match subscriptions and publications in order to route messages to the right destinations. The broker can also temporarily store publications to satisfy future subscriptions.

CoAP pubsub Client: A CoAP client which is capable of publish or subscribe operations as defined in this specification.

Topic: A unique identifier for a particular item being published and/or subscribed to. A broker uses the topics to match subscriptions to publications. A topic is a valid CoAP URI as defined in [RFC7252]

3. Architecture

3.1. CoAP pubsub Architecture

Figure 1 shows the architecture of a CoAP pubsub service. CoAP pubsub Clients interact with a CoAP pubsub Broker through the CoAP pubsub API which is hosted by the Broker. State information is updated between the Clients and the Broker. The CoAP pubsub Broker performs a store-and-forward of state update representations between certain CoAP pubsub Clients. Clients Subscribe to topics upon which representations are Published by other Clients, which are forwarded by the Broker to the subscribing clients. A CoAP pubsub Broker may be used as a REST resource proxy, retaining the last published representation to supply in response to Read requests from Clients.

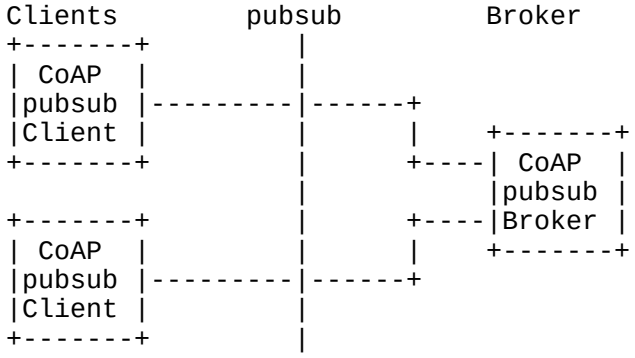


Figure 1: CoAP pubsub Architecture

3.2. CoAP pubsub Broker

A CoAP pubsub Broker is a CoAP Server that exposes an API for clients to use to initiate publish-subscribe interactions. Avoiding the need for direct reachability between clients, the broker only needs to be reachable from all clients. The broker also needs to have sufficient resources (storage, bandwidth, etc.) to host CoAP resource services, and potentially buffer messages, on behalf of the clients.

3.3. CoAP pubsub Client

A CoAP pubsub Client interacts with a CoAP pubsub Broker using the CoAP pubsub API. Clients initiate interactions with a CoAP pubsub broker. A data source (e.g., sensor clients) can publish state updates to the broker and data sinks (e.g., actuator clients) can read from or subscribe to state updates from the broker. Application clients can make use of both publish and subscribe in order to exchange state updates with data sources and sinks.

3.4. CoAP pubsub Topic

The clients and broker use topics to identify a particular resource or object in a publish-subscribe system. Topics are conventionally formed as a hierarchy, e.g. "/sensors/weather/barometer/pressure" or "EP-33543/sen/3303/0/5700". The topics are hosted at the broker and all the clients using the broker share the same namespace for topics. Every CoAP pubsub topic has a link, consisting of a reference path on the broker using URI path [RFC3986] construction and link attributes [RFC6690]. Every topic is associated with zero or more stored representations with a content-format specified in the link. A CoAP pubsub topic value may alternatively be a collection of one or more sub-topics, consisting of links to the sub-topic URIs and indicated by a link-format content-format.

3.5. Brokerless pubsub

Figure 2 shows an arrangement for using CoAP pubsub in a "brokerless" configuration between peer nodes. Nodes in a brokerless system may act as both broker and client. The Broker interface in a brokerless node may be pre-configured with topics that expose services and resources. Brokerless peer nodes can be mixed with client and broker nodes in a system with full interoperability.

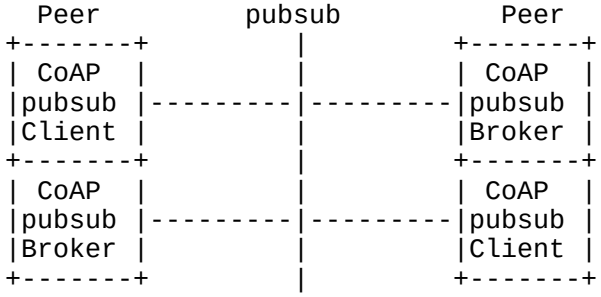


Figure 2: Brokerless pubsub

4. CoAP pubsub API

This section defines the API exposed by a CoAP pubsub Broker to pubsub Clients. The examples throughout this section assume the use of CoAP [RFC7252]. A CoAP pubsub Broker implementing this specification MUST support the DISCOVERY, CREATE, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, READ, and REMOVE operations defined in this section.

4.1. DISCOVERY

CoAP pubsub Clients discover CoAP pubsub Brokers by using CoAP Simple Discovery or through a Resource Directory (RD) [I-D.ietf-core-resource-directory]. A CoAP pubsub Broker SHOULD indicate its presence and availability on a network by exposing a link to its pubsub API at its .well-known/core location [RFC6690]. A CoAP pubsub broker MAY register its pubsub API location with a Resource Directory. Figure 3 shows an example of a client discovering a local pubsub API using CoAP Simple Discovery. A broker wishing to advertise the CoAP pubsub API for Simple Discovery or through a Resource Directory MUST use the link relation rt=core.ps. A broker MAY advertise its supported content formats and other attributes in the link to its pubsub API.

A CoAP pubsub Broker MAY offer a topic discovery API to enable Clients to find topics of interest, either by topic name or by link attributes which may be registered when the topic is created. Figure 4 shows an example of a client looking for a topic with a resource type (rt) of "temperature" using Discover. The client then receives the URI of the resource and its content-format. A pubsub broker wishing to advertize topic discovery MUST use the relation rt=core.ps.discover in the link.

A CoAP pubsub Broker MAY expose the Discover interface through the .well-known/core resource. Links to topics may be exposed at .well-known/core in addition to links to the pubsub API. Figure 5 shows an example of topic discovery through .well-known/core.

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: /.well-known/core

URI Template: /{+ps/}{topic}{/topic*}{?q*}

URI Template Variables:

/.well-known/core := for discovering the pubsub API (optional)

ps := pubsub API path (optional). The base URI path of the pubsub API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: application/link-format

The following response codes are defined for this interface:

Success: 2.05 "Content" with an application/link-format payload containing one or more matching entries for the broker resource. A pubsub broker SHOULD use the value "/ps/" for the base URI of the pubsub API wherever possible.

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

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

Failure: No error response to a multicast request.

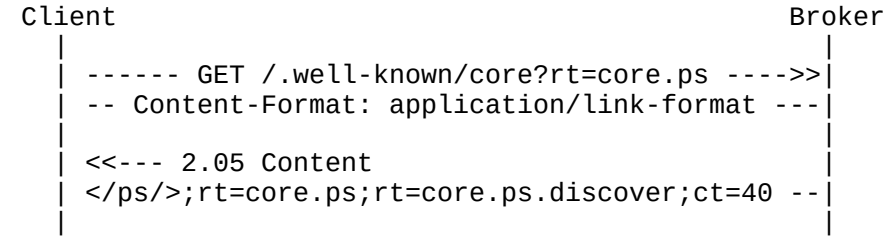


Figure 3: Example of DISCOVER pubsub function

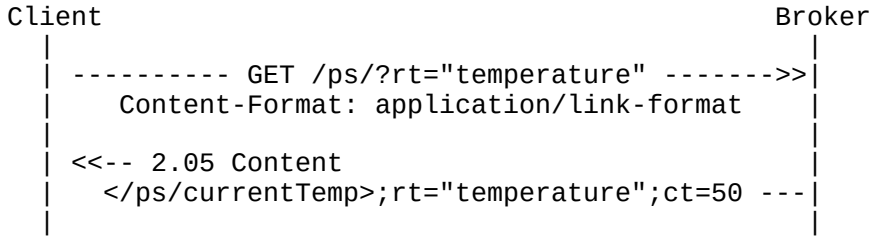


Figure 4: Example of DISCOVER topic

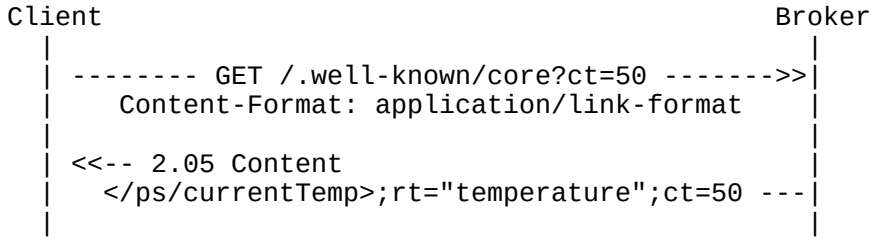


Figure 5: Example of DISCOVER topic

4.2. CREATE

Clients create new topics on the broker using CREATE. A client wishing to create a topic MUST use CoAP POST to the pubsub API with a payload indicating the desired topic. The topic specification sent in the payload MUST use a supported serialization of the CoRE link format [RFC6690]. The target of the link MUST be a URI formatted string. The client MUST indicate the desired content format for publishes to the topic by using the ct (Content Format) link attribute in the link-format payload. The client MAY indicate the lifetime of the topic by including the Max-Age option in the CREATE request.

A Broker MUST return a response code of "2.01 Created" if the topic is created and return the URI path of the created topic via Location-Path options. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if a new topic can not be created. Broker SHOULD remove topics if the Max-Age of the topic is exceeded without any publishes to the topic. Broker SHOULD retain a topic indefinitely if the Max-Age option is elided or is set to zero upon topic creation. The lifetime of a topic MUST be refreshed upon create operations with a target of an existing topic.

Topics may be created as sub-topics of other topics. A client MAY create a topic with a ct (Content Format) link attribute value which describes a supported serialization of the CoRE link format [RFC6690] such as application/link-format (ct=40) or its JSON or CBOR serializations. If a topic is created which describes a link serialization, that topic may then have sub-topics created under it as shown in Figure 7.

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

URI Template: /{+ps/}{topic}{/topic*}

URI Template Variables:

ps := pubsub API path (mandatory). The path of the pubsub API, as obtained from discovery. A pubsub broker SHOULD use the value "ps" for this variable whenever possible.

Content-Format: application/link-format

Payload: The desired topic to CREATE

The following response codes are defined for this interface:

Success: 2.01 "Created". Successful Creation of the topic

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.03 "Forbidden". Topic already exists.

Failure: 4.06 "Not Acceptable". Unsupported content format for topic.

Figure 6 shows an example of a topic called "topic1" being successfully created.

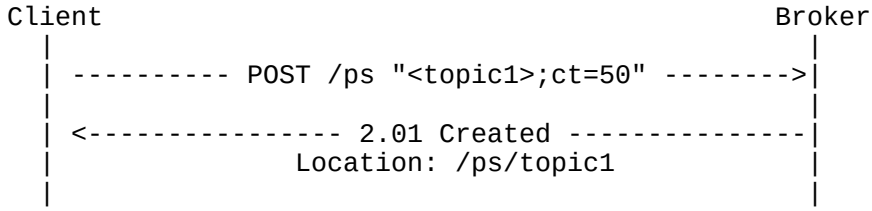


Figure 6: Example of CREATE topic

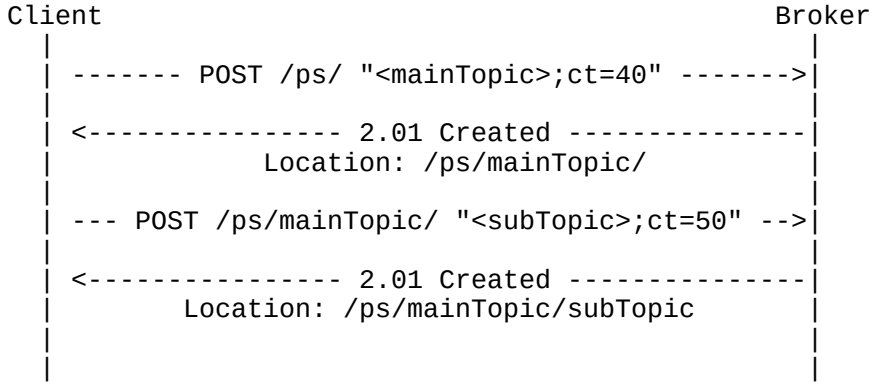


Figure 7: Example of CREATE sub-topic

4.3. PUBLISH

A CoAP pubsub Client uses the PUBLISH interface for updating topics on the broker. A client MAY use the PUT or the POST method to publish state updates to the CoAP pubsub Broker. A client MUST use the content format specified upon creation of a given topic to publish updates to that topic. The broker MUST reject publish operations which do not use the specified content format. A CoAP client publishing on a topic MAY indicate the maximum lifetime of the value by including the Max-Age option in the publish request. The broker MUST return a response code of "2.04 Changed" if the publish is accepted. A Broker MAY return a "4.04 Not Found" if the topic does not exist. A broker MAY return "4.29 Too Many Requests" if simple flow control as described in Section 7 is implemented.

A Broker MUST accept PUBLISH operations using the PUT method. PUBLISH operations using the PUT method replace any stored representation associated with the topic, with the supplied representation. A Broker MAY reject, or delay responses to, PUT

requests to a topic while pending resolution of notifications to subscribers from previous PUT requests.

Create on PUBLISH: A Broker MAY accept PUBLISH operations to new topics using the PUT method. If a Broker accepts a PUBLISH using PUT to a topic that does not exist, the Broker MUST create the topic using the information in the PUT operation. The Broker MUST create a topic with the URI-Path of the request, including all of the sub-topics necessary, and create a topic link with the ct attribute set to the content-format of the payload of the PUT request. If topic is created, the Broker MUST return the response "2.01 Created" with the URI of the created topic, including all of the created path segments, returned via the Location-Path option.

A Broker MAY accept PUBLISH operations using the POST method. If a broker accepts PUBLISH using POST it MAY store an ordered list of published representations, with an element of the list for each published representation. A Broker MAY reject, or delay responses to, POST requests if the internal capacity to store representations is exceeded.

A Broker MAY perform garbage collection of stored representations which have been delivered to all subscribers or which have timed out. A Broker MAY retain at least one most recently published representation to return in response to SUBSRCIBE and READ requests.

A Broker MUST make a best-effort attempt to notify all clients subscribed on a particular topic each time it receives a publish on that topic. An example is shown in Figure 10. If a client publishes to a broker with the Max-Age option, the broker MUST include the same value for the Max-Age option in all notifications. A broker MUST use CoAP Notification as described in [RFC7641] to notify subscribed clients.

The PUBLISH interface is specified as follows:

Interaction: Client -> Broker

Method: PUT, POST

URI Template: /{+ps/}{topic}{/topic*}

URI Template Variables:

ps := pubsub API path (mandatory). The path of the pubsub API, as obtained from discovery.

topic := The desired topic to publish on.

Content-Format: Any valid CoAP content format

Payload: Representation of the topic value (CoAP resource state representation) in the indicated content format

The following response codes are defined for this interface:

Success: 2.01 "Created". Successful publish, topic is created

Success: 2.04 "Changed". Successful publish, topic is updated

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Failure: 4.29 "Too Many Requests". The client should slow down the rate of publish messages for this topic (see Section 7).

Figure 8 shows an example of a new value being successfully published to the topic "topic1". See Figure 10 for an example of a broker forwarding a message from a publishing client to a subscribed client.

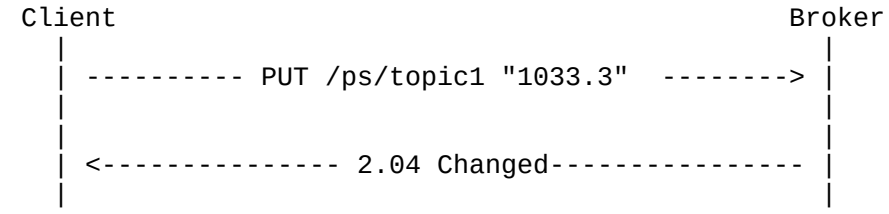


Figure 8: Example of PUBLISH

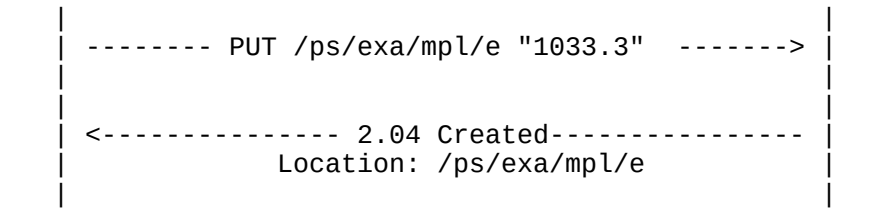


Figure 9: Example of CREATE on PUBLISH

4.4. SUBSCRIBE

CoAP pubsub Clients subscribe to topics on the Broker using CoAP Observe as described in [RFC7641]. A CoAP pubsub Client wishing to Subscribe to a topic on a broker MUST use a CoAP GET with Observe registration. The Broker MAY add the client to a list of observers. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the broker can supply the requested content format. The broker MUST reject Subscribe requests on a topic if the content format of the request is not supported by the content format the topic was created with. The broker MAY accept Subscribe requests which specify content formats that the broker can supply as alternate content formats to the content format the topic was registered with. If the topic was published with the Max-Age option, the broker MUST set the Max-Age option in the valid response to the amount of time remaining for the value to be valid since the last publish operation on that topic. The Broker MUST return a response code of "2.04 No Content" if the Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if it can not return the requested content format. If a Broker is unable to accept a new Subscription on a topic, it SHOULD return the appropriate response code without the Observe option as per as per [RFC7641] Section 4.1. There is no explicit maximum lifetime of a Subscription, thus a Broker may remove subscribers at any time. The Broker, upon removing a Subscriber, will transmit the appropriate response code without the Observe option, as per [RFC7641] Section 4.2, to the removed Subscriber.

The SUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

URI Template: /{+ps/}{topic}{/topic*}

URI Template Variables:

ps := pubsub API path (mandatory). The path of the pubsub API, as obtained from discovery.

topic := The desired topic to subscribe to.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful subscribe, current value included

Success: 2.04 "No Content". Successful subscribe, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Failure: 4.15 "Unsupported Content Format". Unsupported content format.

Figure 10 shows an example of Client2 subscribing to "topic1" and receiving a response from the broker, with a subsequent notification. The subscribe response from the broker uses the last stored value associated with the topic1. The notification from the broker is sent in response to the publish received from Client1.

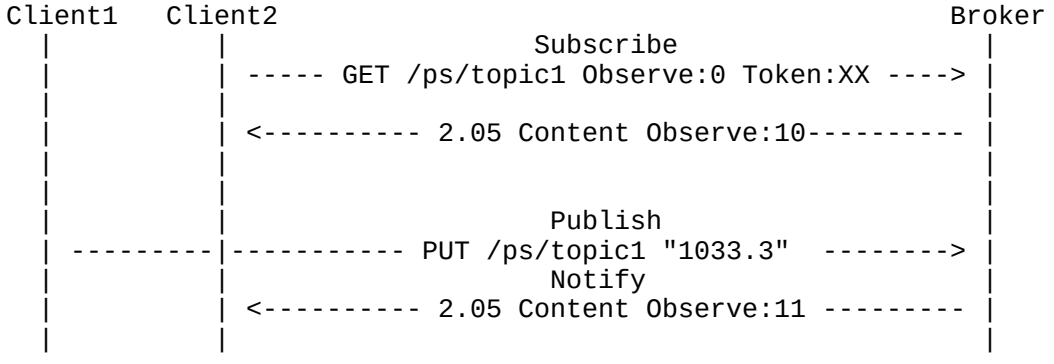


Figure 10: Example of SUBSCRIBE

4.5. UNSUBSCRIBE

CoAP pubsub Clients unsubscribe from topics on the Broker using the CoAP Cancel Observation operation. A CoAP pubsub Client wishing to unsubscribe to a topic on a Broker MUST either use CoAP GET with Observe using an Observe parameter of 1 or send a CoAP Reset message in response to a publish, as per [RFC7641].

The UNSUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1

URI Template: /{+ps/}{topic}{/topic*}

URI Template Variables:

ps := pubsub API path (mandatory). The path of the pubsub API, as obtained from discovery.

topic := The desired topic to unsubscribe from.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful unsubscribe, current value included

Success: 2.04 "No Content". Successful unsubscribe, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Figure 11 shows an example of a client unsubscribe using the Observe=1 cancellation method.

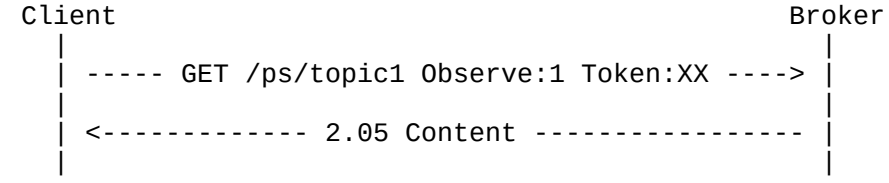


Figure 11: Example of UNSUBSCRIBE

4.6. READ

A CoAP pubsub client wishing to obtain only the most recent published value on a topic MAY use the READ interface. For reading, the client uses the CoAP GET method. The broker MUST accept Read requests on a topic if the content format of the request matches the content format the topic was created with. The broker MAY accept Read requests which specify content formats that the broker can supply as alternate content formats to the content format the topic was registered with. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the broker can supply the requested content format. If the topic was published with the Max-Age option, the broker MUST set the Max-Age option in the valid response to the amount of time remaining for the topic to be valid since the last publish. The Broker MUST return a response code of "2.04 No Content" if the Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if the broker can not return the requested content format.

The READ interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: /{+ps/}{topic}{/topic*}

URI Template Variables:

ps := pubsub API path (mandatory). The path of the pubsub API, as obtained from discovery.

topic := The desired topic to READ.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful READ, current value included

Success: 2.04 "No Content". Topic exists, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Failure: 4.15 "Unsupported Content Format". Unsupported content-format.

Figure 12 shows an example of a successful READ from topic1, followed by a Publish on the topic, followed at some time later by a read of the updated value from the recent Publish.

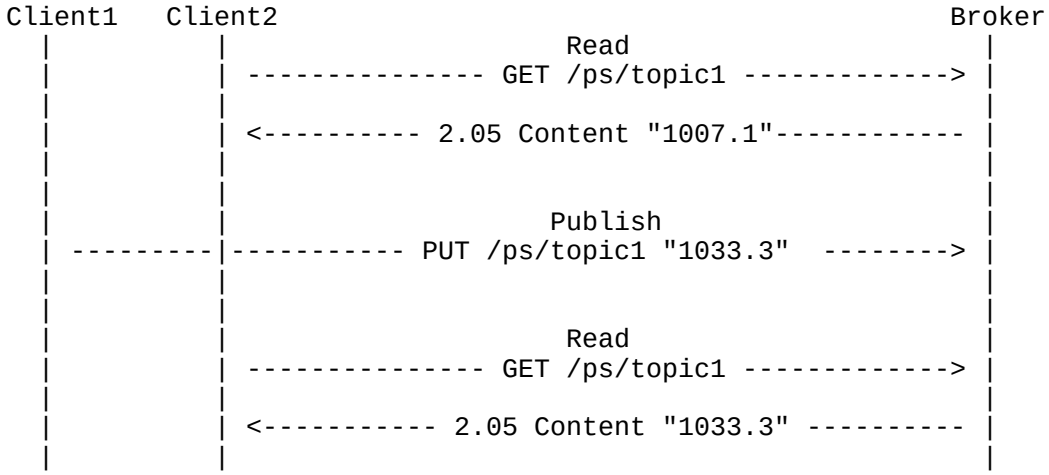


Figure 12: Example of READ

4.7. REMOVE

A CoAP pubsub Client wishing to remove a topic MAY use the CoAP Delete operation on the URI of the topic. The CoAP pubsub Broker MUST return "2.02 Deleted" if the remove operation is successful. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if the topic can not be removed. When a topic is removed for any reason, the Broker SHOULD return the response code 4.04 Not Found and remove all of the observers from the list of observers as per as per [RFC7641] Section 3.2. If a topic which has sub-topics is removed, then all of its sub-topics MUST be recursively removed.

The REMOVE interface is specified as follows:

Interaction: Client -> Broker

Method: DELETE

URI Template: /{+ps/}{topic}{/topic*}

URI Template Variables:

ps := pubsub API path (mandatory). The path of the pubsub API, as obtained from discovery.

topic := The desired topic to REMOVE.

Content-Format: None

Response Payload: None

The following response codes are defined for this interface:

Success: 2.02 "Deleted". Successful remove

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Figure 13 shows a successful remove of topic1.

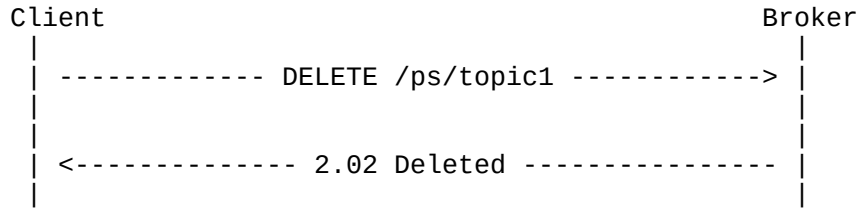


Figure 13: Example of REMOVE

5. CoAP pubsub Operation with Resource Directory

A CoAP pubsub Broker may register the base URI of a pubsub API with a Resource Directory. A pubsub Client may use an RD to discover a pubsub Broker.

A CoAP pubsub Client may register links [RFC6690] with a Resource Directory to enable discovery of created pubsub topics. A pubsub Client may use an RD to discover pubsub Topics. A client which registers pubsub Topics with an RD MUST use the context relation (con) [I-D.ietf-core-resource-directory] to indicate that the context of the registered links is the pubsub Broker.

A CoAP pubsub Broker may alternatively register links to its topics to a Resource Directory by triggering the RD to retrieve it's links from .well-known/core. In order to use this method, the links must first be exposed in the .well-known/core of the pubsub broker. See Section 4.1 in this document.

The pubsub broker triggers the RD to retrieve its links by sending a POST with an empty payload to the .well-known/core of the Resource Directory. The RD server will then retrieve the links from the .well-known/core of the pubsub broker and incorporate them into the Resource Directory. See [I-D.ietf-core-resource-directory] for further details.

6. Sleep-Wake Operation

CoAP pubsub provides a way for client nodes to sleep between operations, conserving energy during idle periods. This is made possible by shifting the server role to the broker, allowing the broker to be always-on and respond to requests from other clients while a particular client is sleeping.

For example, the broker will retain the last state update received from a sleeping client, in order to supply the most recent state update to other clients in response to read and subscribe operations.

Likewise, the broker will retain the last state update received on the topic such that a sleeping client, upon waking, can perform a read operation to the broker to update its own state from the most recent system state update.

7. Simple Flow Control

Since the broker node has to potentially send a large amount of notification messages for each publish message and it may be serving a large amount of subscribers and publishers simultaneously, the broker may become overwhelmed if it receives many publish messages to popular topics in a short period of time.

If the broker is unable to serve a certain client that is sending publish messages too fast, the broker MUST respond with Response Code 4.29, "Too Many Requests". This Response Code is like HTTP 429 "Too Many Requests" but uses the Max-Age Option in place of the "Retry-After" header field to indicate the number of seconds after which to retry. The broker MAY stop creating notifications from the publish messages from this client and to this topic for the indicated time.

If a client receives the 4.29 Response Code from the broker for a publish message to a topic, it MUST NOT send new publish messages to

the broker on the same topic before the time indicated in Max-Age has passed.

8. Security Considerations

CoAP pubsub re-uses CoAP [RFC7252], CoRE Resource Directory [I-D.ietf-core-resource-directory], and Web Linking [RFC5988] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP pubsub broker and the clients SHOULD authenticate each other and enforce access control policies. A malicious client could subscribe to data it is not authorized to or mount a denial of service attack against the broker by publishing a large number of resources. The authentication can be performed using the already standardized DTLS offered mechanisms, such as certificates. DTLS also allows communication security to be established to ensure integrity and confidentiality protection of the data exchanged between these relevant parties. Provisioning the necessary credentials, trust anchors and authorization policies is non-trivial and subject of ongoing work.

The use of a CoAP pubsub broker introduces challenges for the use of end-to-end security between for example a client device on a sensor network and a client application running in a cloud-based server infrastructure since brokers terminate the exchange. While running separate DTLS sessions from the client device to the broker and from broker to client application protects confidentially on those paths, the client device does not know whether the commands coming from the broker are actually coming from the client application. Similarly, a client application requesting data does not know whether the data originated on the client device. For scenarios where end-to-end security is desirable the use of application layer security is unavoidable. Application layer security would then provide a guarantee to the client device that any request originated at the client application. Similarly, integrity protected sensor data from a client device will also provide guarantee to the client application that the data originated on the client device itself. The protected data can also be verified by the intermediate broker ensuring that it stores/caches correct request/response and no malicious messages/ requests are accepted. The broker would still be able to perform aggregation of data/requests collected.

Depending on the level of trust users and system designers place in the CoAP pubsub broker, the use of end-to-end object security is RECOMMENDED [I-D.selander-ace-object-security].

When only end-to-end encryption is necessary and the CoAP Broker is trusted, Payload Only Protection (Mode:PAYL) could be used. The Publisher would wrap only the payload before sending it to the broker

and set the option Content-Format to application/smpayl. Upon receival, the Broker can read the unencrypted CoAP header to forward it to the subscribers.

9. IANA Considerations

This document registers one attribute value in the Resource Type (rt=) registry established with [RFC6690] and appends to the definition of one CoAP Response Code in the CoRE Parameters Registry.

9.1. Resource Type value 'core.ps'

- Attribute Value: core.ps
- Description: Section 4 of [[This document]]
- Reference: [[This document]]
- Notes: None

9.2. Resource Type value 'core.ps.discover'

- Attribute Value: core.ps.discover
- Description: Section 4 of [[This document]]
- Reference: [[This document]]
- Notes: None

9.3. Response Code value '2.04'

- Response Code: 2.04
- Description: Add No Content response to GET to the existing definition of the 2.04 response code.
- Reference: [[This document]]
- Notes: None

9.4. Response Code value '4.29'

- Response Code: 4.29
- Description: This error code is used by a server to indicate that a client is making too many requests on a resource.

- Reference: [[This document]]

- Notes: None

10. Acknowledgements

The authors would like to thank Hannes Tschofenig, Zach Shelby, Mohit Sethi, Peter van der Stok, Tim Kellogg, Anders Eriksson, Goran Selander, Mikko Majanen, and Olaf Bergmann for their contributions and reviews.

11. References

11.1. Normative References

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-09 (work in progress), October 2016.

[I-D.selander-ace-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-06 (work in progress), October 2016.

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <http://www.rfc-editor.org/info/rfc6570>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[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>.

[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>.

11.2. Informative References

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

Authors' Addresses

Michael Koster
SmartThings

Email: Michael.Koster@smartthings.com

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson

Email: jaime.jimenez@ericsson.com

CORE
Internet-Draft
Updates: 7252, 7641, 7959 (if approved)
Intended status: Standards Track
Expires: November 17, 2017

C. Bormann
Universitaet Bremen TZI
S. Lemay
Zebra Technologies
H. Tschofenig
ARM Ltd.
K. Hartke
Universitaet Bremen TZI
B. Silverajan
Tampere University of Technology
B. Raymor, Ed.
Microsoft
May 16, 2017

CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets
draft-ietf-core-coap-tcp-tls-09

Abstract

The Constrained Application Protocol (CoAP), although inspired by HTTP, was designed to use UDP instead of TCP. The message layer of the CoAP over UDP protocol includes support for reliable delivery, simple congestion control, and flow control.

Some environments benefit from the availability of CoAP carried over reliable transports such as TCP or TLS. This document outlines the changes required to use CoAP over TCP, TLS, and WebSockets transports. It also formally updates RFC 7252 fixing an erratum in the URI syntax, RFC 7641 for use with the new transports, and RFC 7959 to enable the use of larger messages over a reliable transport.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 17, 2017.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP [RFC0768] or Datagram Transport Layer Security (DTLS) [RFC6347] over UDP can be used unimpeded. UDP is a good choice for transferring small amounts of data across networks that follow the IP architecture.

Some CoAP deployments need to integrate well with existing enterprise infrastructures, where UDP-based protocols may not be well-received or may even be blocked by firewalls. Middleboxes that are unaware of

CoAP usage for IoT can make the use of UDP brittle, resulting in lost or malformed packets.

Emerging standards such as Lightweight Machine to Machine [LWM2M] currently use CoAP over UDP as a transport and require support for CoAP over TCP to address the issues above and to protect investments in existing CoAP implementations and deployments. Although HTTP/2 could also potentially address these requirements, there would be additional costs and delays introduced by such a transition. Currently, there are also fewer HTTP/2 implementations available for constrained devices in comparison to CoAP.

To address these requirements, this document defines how to transport CoAP over TCP, CoAP over TLS, and CoAP over WebSockets. For these cases, the reliability offered by the transport protocol subsumes the reliability functions of the message layer used for CoAP over UDP. (Note that both for a reliable transport and the CoAP over UDP message layer, the reliability offered is per transport hop: where proxies -- see Sections 5.7 and 10 of [RFC7252] -- are involved, that layer's reliability function does not extend end-to-end.) Figure 1 illustrates the layering:

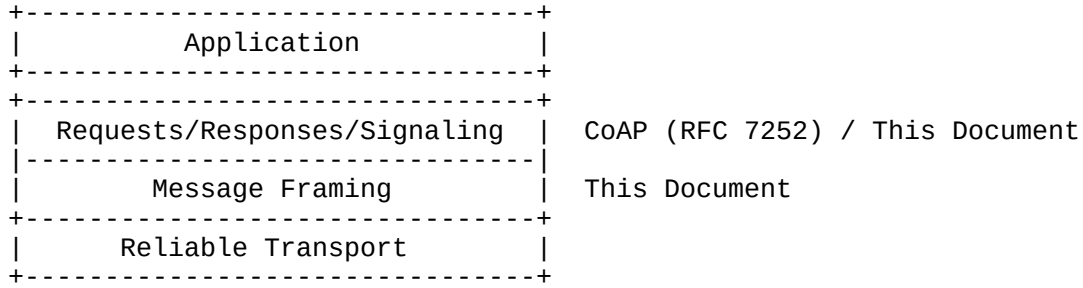


Figure 1: Layering of CoAP over Reliable Transports

Where NATs are present, CoAP over TCP can help with their traversal. NATs often calculate expiration timers based on the transport layer protocol being used by application protocols. Many NATs maintain TCP-based NAT bindings for longer periods based on the assumption that a transport layer protocol, such as TCP, offers additional information about the session life cycle. UDP, on the other hand, does not provide such information to a NAT and timeouts tend to be much shorter [HomeGateway].

Some environments may also benefit from the ability of TCP to exchange larger payloads, such as firmware images, without application layer segmentation and to utilize the more sophisticated congestion control capabilities provided by many TCP implementations.

Note that there is ongoing work to add more elaborate congestion control to CoAP (see [I-D.ietf-core-cocoa]).

CoAP may be integrated into a Web environment where the front-end uses CoAP over UDP from IoT devices to a cloud infrastructure and then CoAP over TCP between the back-end services. A TCP-to-UDP gateway can be used at the cloud boundary to communicate with the UDP-based IoT device.

To allow IoT devices to better communicate in these demanding environments, CoAP needs to support different transport protocols, namely TCP [RFC0793], in some situations secured by TLS [RFC5246].

CoAP applications running inside a web browser without access to connectivity other than HTTP and the WebSocket protocol [RFC6455] may cross-proxy their CoAP requests via HTTP to a HTTP-to-CoAP cross-proxy or transport them via the the WebSocket protocol, which provides two-way communication between a WebSocket client and a WebSocket server after upgrading an HTTP/1.1 [RFC7230] connection.

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

Appendix A updates the "Observing Resources in the Constrained Application Protocol" [RFC7641] specification for use with CoAP over reliable transports. [RFC7641] is an extension to the CoAP protocol that enables CoAP clients to "observe" a resource on a CoAP server. (The CoAP client retrieves a representation of a resource and registers to be notified by the CoAP server when the representation is updated.)

Section 7 fixes an erratum on the URI scheme syntax in [RFC7252]. Section 6 defines semantics for a value 7 for the field "SZX" in a Block1 or Block2 option, updating [RFC7959].

2. Conventions and Terminology

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

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455], [RFC7252], [RFC7641], and [RFC7959].

The term "reliable transport" is used only to refer to transport protocols, such as TCP, which provide reliable and ordered delivery of a byte-stream.

Block-wise Extension for Reliable Transport (BERT):
: BERT extends [RFC7959] to enable the use of larger messages over a reliable transport.

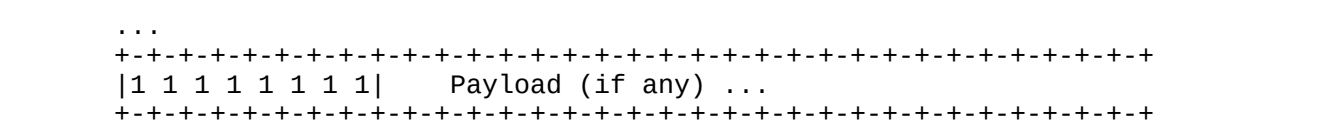
BERT Option:
: A Block1 or Block2 option that includes an SZX value of 7.

BERT Block:
: The payload of a CoAP message that is affected by a BERT Option in descriptive usage (see Section 2.1 of [RFC7959]).

Connection Initiator:
: The peer that opens a reliable byte stream connection, i.e., the TCP active opener, TLS client, or WebSocket client.

Connection Acceptor:
: The peer that accepts the reliable byte stream connection opened by the other peer, i.e., the TCP passive opener, TLS server, or WebSocket server.

For simplicity, a Payload Marker (0xFF) is shown in all examples for message formats:



The Payload Marker indicates the start of the optional payload and is absent for zero-length payloads (see Section 3 of [RFC7252]).

3. CoAP over TCP

The request/response interaction model of CoAP over TCP is the same as CoAP over UDP. The primary differences are in the message layer. The message layer of CoAP over UDP supports optional reliability by defining four types of messages: Confirmable, Non-confirmable, Acknowledgement, and Reset. In addition, messages include a Message ID to relate Acknowledgments to Confirmable messages and to detect duplicate messages.

3.1. Messaging Model

Conceptually, CoAP over TCP replaces most of the message layer of CoAP over UDP with a framing mechanism on top of the byte-stream provided by TCP/TLS, conveying the length information for each message that on datagram transports is provided by the UDP/DTLS datagram layer.

TCP ensures reliable message transmission, so the message layer of CoAP over TCP is not required to support acknowledgements or to detect duplicate messages. As a result, both the Type and Message ID fields are no longer required and are removed from the CoAP over TCP message format.

Figure 2 illustrates the difference between CoAP over UDP and CoAP over reliable transport. The removed Type and Message ID fields are indicated by dashes.

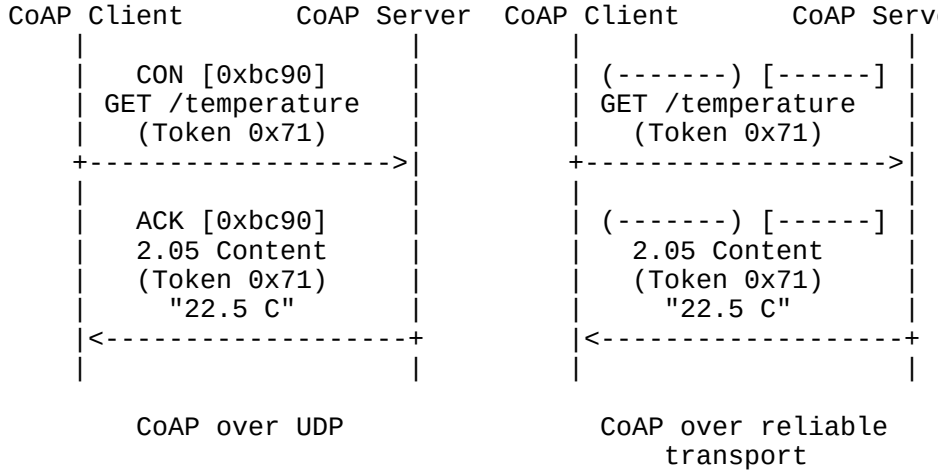


Figure 2: Comparison between CoAP over unreliable and reliable transport

3.2. Message Format

The CoAP message format defined in [RFC7252], as shown in Figure 3, relies on the datagram transport (UDP, or DTLS over UDP) for keeping the individual messages separate and for providing length information.

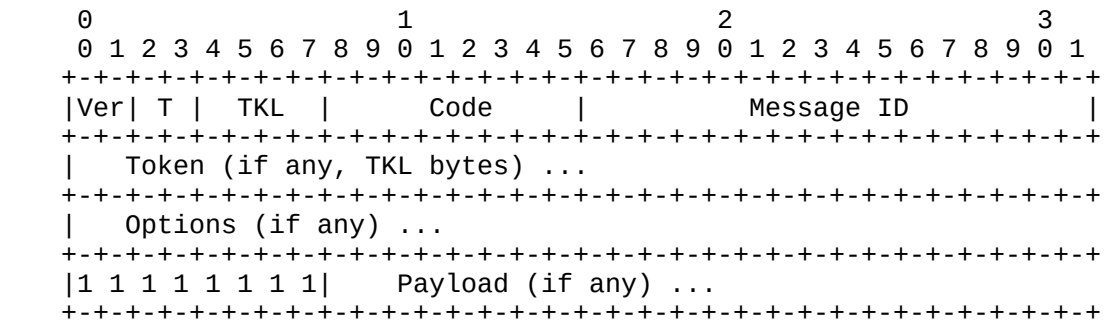


Figure 3: RFC 7252 defined CoAP Message Format

The CoAP over TCP message format is very similar to the format specified for CoAP over UDP. The differences are as follows:

- Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. The Type (T) and Message ID fields in the CoAP message header are elided.

- The Version (Vers) field is elided as well. In contrast to the message format of CoAP over UDP, the message format for CoAP over TCP does not include a version number. CoAP is defined in [RFC7252] with a version number of 1. At this time, there is no known reason to support version numbers different from 1. If version negotiation needs to be addressed in the future, then Capabilities and Settings Messages (CSM see Section 5.3) have been specifically designed to enable such a potential feature.

- In a stream oriented transport protocol such as TCP, a form of message delimitation is needed. For this purpose, CoAP over TCP introduces a length field with variable size. Figure 4 shows the adjusted CoAP message format with a modified structure for the fixed header (first 4 bytes of the CoAP over UDP header), which includes the length information of variable size, shown here as an 8-bit length.

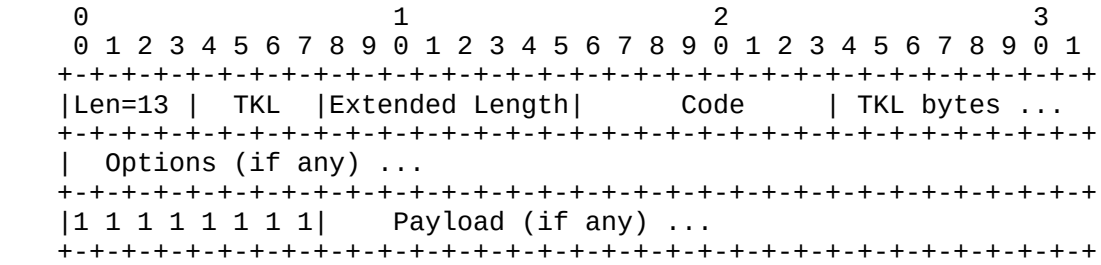


Figure 4: CoAP frame with 8-bit Extended Length field

Length (Len): 4-bit unsigned integer. A value between 0 and 12 directly indicates the length of the message in bytes starting with the first bit of the Options field. Three values are reserved for special constructs:

13: An 8-bit unsigned integer (Extended Length) follows the initial byte and indicates the length of options/payload minus 13.

14: A 16-bit unsigned integer (Extended Length) in network byte order follows the initial byte and indicates the length of options/payload minus 269.

15: A 32-bit unsigned integer (Extended Length) in network byte order follows the initial byte and indicates the length of options/payload minus 65805.

The encoding of the Length field is modeled after the Option Length field of the CoAP Options (see Section 3.1 of [RFC7252]).

The following figures show the message format for the 0-bit, 16-bit, and the 32-bit variable length cases.

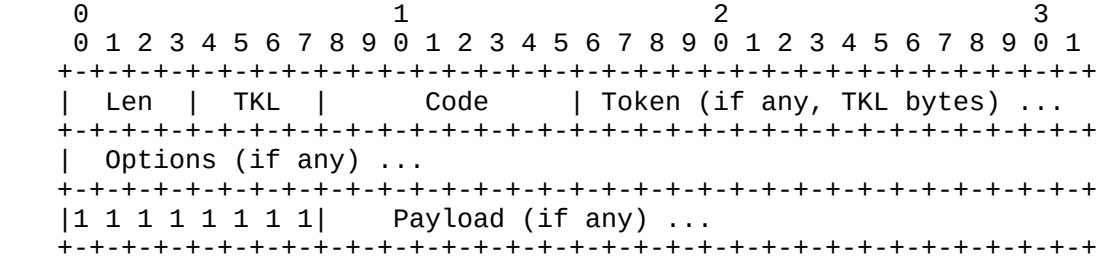


Figure 5: CoAP message format without an Extended Length field

For example: A CoAP message just containing a 2.03 code with the token 7f and no options or payload would be encoded as shown in Figure 6.

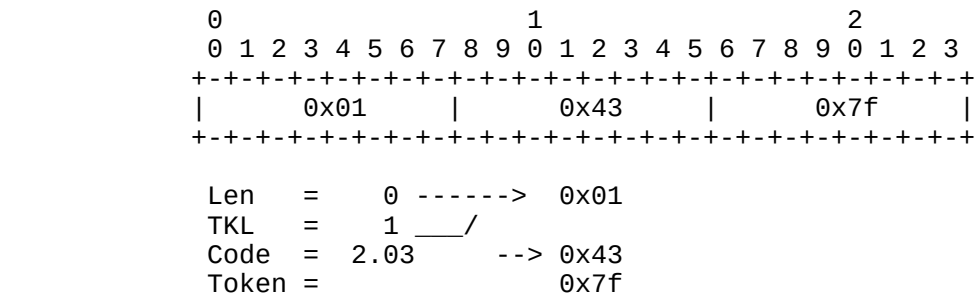


Figure 6: CoAP message with no options or payload

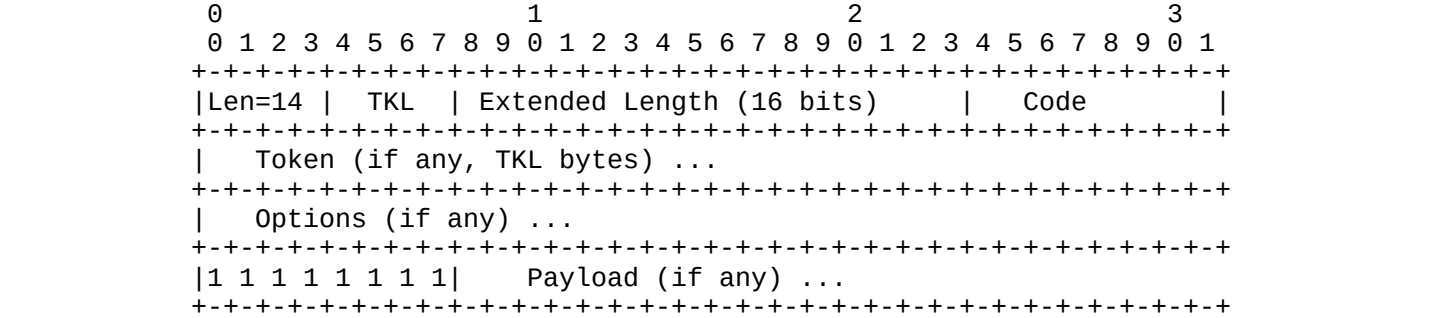


Figure 7: CoAP message format with 16-bit Extended Length field

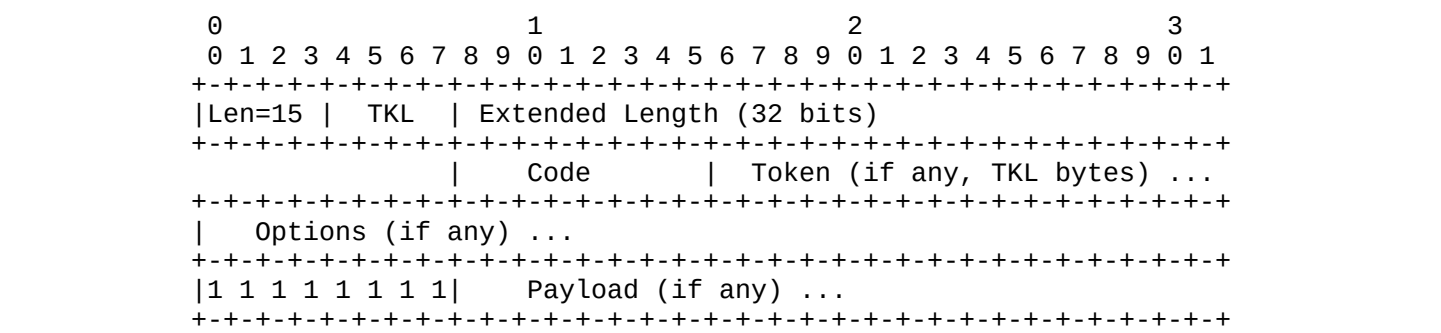


Figure 8: CoAP message format with 32-bit Extended Length field

The semantics of the other CoAP header fields are left unchanged.

3.3. Message Transmission

Once a connection is established, both endpoints MUST send a Capabilities and Settings message (CSM see Section 5.3) as their first message on the connection. This message establishes the initial settings and capabilities for the endpoint, such as maximum message size or support for block-wise transfers. The absence of options in the CSM indicates that base values are assumed.

To avoid a deadlock, the Connection Initiator MUST NOT wait for the Connection Acceptor to send its initial CSM message before sending its own initial CSM message. Conversely, the Connection Acceptor MAY wait for the Connection Initiator to send its initial CSM message before sending its own initial CSM message.

To avoid unnecessary latency, a Connection Initiator MAY send additional messages without waiting to receive the Connection Acceptor’s CSM; however, it is important to note that the Connection Acceptor’s CSM might advertise capabilities that impact how the initiator is expected to communicate with the acceptor. For example, the acceptor CSM could advertise a Max-Message-Size option (see Section 5.3.1) that is smaller than the base value (1152).

Endpoints MUST treat a missing or invalid CSM as a connection error and abort the connection (see Section 5.6).

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

The connection is bi-directional, so requests can be sent both by the entity that established the connection (Connection Initiator) and the remote host (Connection Acceptor). If one side does not implement a CoAP server, an error response MUST be returned for all CoAP requests from the other side. The simplest approach is to always return 5.01 (Not Implemented). A more elaborate mock server could also return 4.xx responses such as 4.04 (Not Found) or 4.02 (Bad Option) where appropriate.

Retransmission and deduplication of messages is provided by the TCP protocol.

3.4. Connection Health

Empty messages (Code 0.00) can always be sent and MUST be ignored by the recipient. This provides a basic keep-alive function that can refresh NAT bindings.

If a CoAP client does not receive any response for some time after sending a CoAP request (or, similarly, when a client observes a resource and it does not receive any notification for some time), it can send a CoAP Ping Signaling message (see Section 5.4) to test the connection and verify that the CoAP server is responsive.

When the underlying TCP connection is closed or reset, the signaling state and any observation state (see Appendix A.4) associated with the reliable connection are removed. In flight messages may or may not be lost.

4. CoAP over WebSockets

CoAP over WebSockets is intentionally similar to CoAP over TCP; therefore, this section only specifies the differences between the transports.

CoAP over WebSockets can be used in a number of configurations. The most basic configuration is a CoAP client retrieving or updating a CoAP resource located on a CoAP server that exposes a WebSocket endpoint (see Figure 9). The CoAP client acts as the WebSocket client, establishes a WebSocket connection, and sends a CoAP request, to which the CoAP server returns a CoAP response. The WebSocket connection can be used for any number of requests.

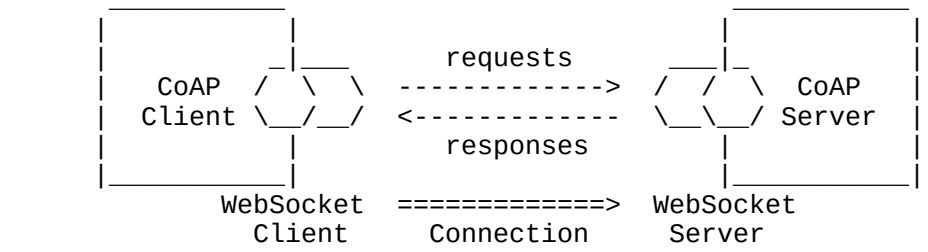


Figure 9: CoAP Client (WebSocket client) accesses CoAP Server (WebSocket server)

The challenge with this configuration is how to identify a resource in the namespace of the CoAP server. When the WebSocket protocol is used by a dedicated client directly (i.e., not from a web page through a web browser), the client can connect to any WebSocket

endpoint. Section 7.3 and Section 7.4 define how the "coap" and "coaps" URI schemes can be used to enable the client to identify both a WebSocket endpoint and the path and query of the CoAP resource within that endpoint.

Another possible configuration is to set up a CoAP forward proxy at the WebSocket endpoint. Depending on what transports are available to the proxy, it could forward the request to a CoAP server with a CoAP UDP endpoint (Figure 10), an SMS endpoint (a.k.a. mobile phone), or even another WebSocket endpoint. The CoAP client specifies the resource to be updated or retrieved in the Proxy-Uri Option.

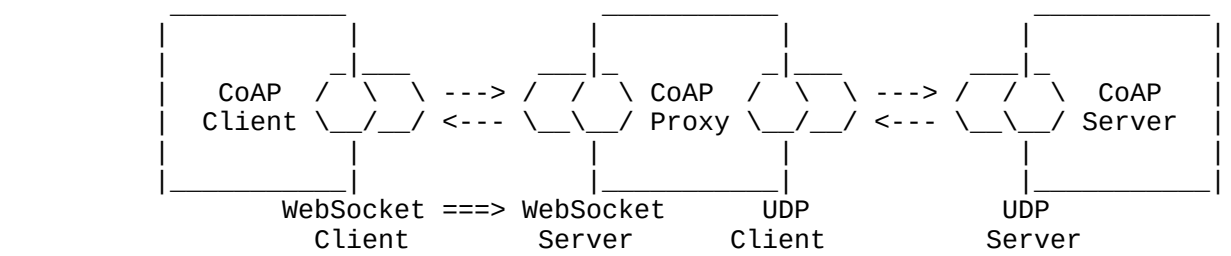


Figure 10: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

A third possible configuration is a CoAP server running inside a web browser (Figure 11). The web browser initially connects to a WebSocket endpoint and is then reachable through the WebSocket server. When no connection exists, the CoAP server is unreachable. Because the WebSocket server is the only way to reach the CoAP server, the CoAP proxy should be a reverse-proxy.

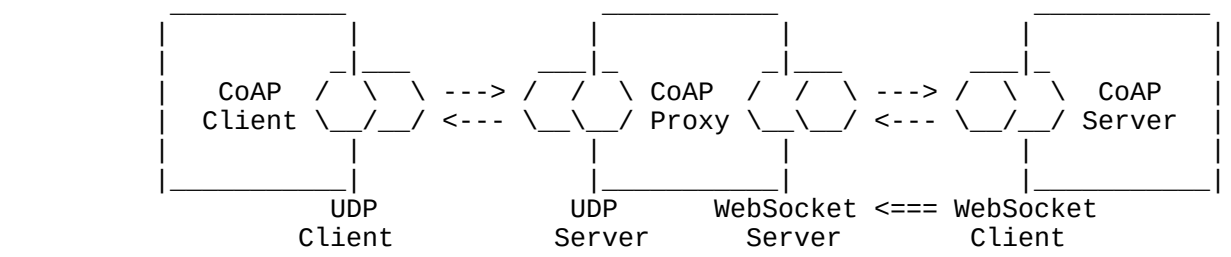


Figure 11: CoAP Client (UDP client) accesses CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

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

4.1. Opening Handshake

Before CoAP requests and responses are exchanged, a WebSocket connection is established as defined in Section 4 of [RFC6455]. Figure 12 shows an example.

The WebSocket client MUST include the subprotocol name "coap" in the list of protocols, which indicates support for the protocol defined in this document. Any later, incompatible versions of CoAP or CoAP over WebSockets will use a different subprotocol name.

The WebSocket client includes the hostname of the WebSocket server in the Host header field of its handshake as per [RFC6455]. The Host header field also indicates the default value of the Uri-Host Option in requests from the WebSocket client to the WebSocket server.

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

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

Figure 12: Example of an Opening Handshake

4.2. Message Format

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

The message format shown in Figure 13 is the same as the CoAP over TCP message format (see Section 3.2) with one change. The Length (Len) field MUST be set to zero because the WebSockets frame contains the length.

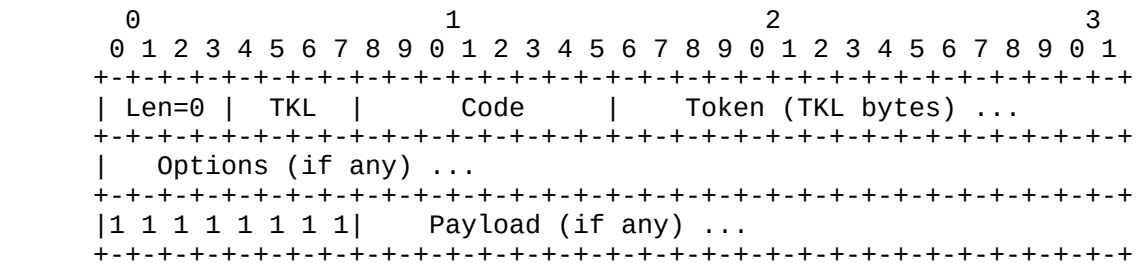


Figure 13: CoAP Message Format over WebSockets

As with CoAP over TCP, the message format for CoAP over WebSockets eliminates the Version field defined in CoAP over UDP. If CoAP version negotiation is required in the future, CoAP over WebSockets can address the requirement by the definition of a new subprotocol identifier that is negotiated during the opening handshake.

Requests and response messages can be fragmented as specified in Section 5.4 of [RFC6455], though typically they are sent unfragmented as they tend to be small and fully buffered before transmission. The WebSocket protocol does not provide means for multiplexing. If it is not desirable for a large message to monopolize the connection, requests and responses can be transferred in a block-wise fashion as defined in [RFC7959].

4.3. Message Transmission

As with CoAP over TCP, both endpoints MUST send a Capabilities and Settings message (CSM see Section 5.3) as their first message on the WebSocket connection.

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

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

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

4.4. Connection Health

As with CoAP over TCP, a CoAP client can test the health of the CoAP over WebSocket connection by sending a CoAP Ping Signaling message (Section 5.4). WebSocket Ping and unsolicited Pong frames (Section 5.5 of [RFC6455]) SHOULD NOT be used to ensure that redundant maintenance traffic is not transmitted.

5. Signaling

Signaling messages are introduced to allow peers to:

- Learn related characteristics, such as maximum message size for the connection
- Shut down the connection in an orderly fashion
- Provide diagnostic information when terminating a connection in response to a serious error condition

Signaling is a third basic kind of message in CoAP, after requests and responses. Signaling messages share a common structure with the existing CoAP messages. There is a code, a token, options, and an optional payload.

(See Section 3 of [RFC7252] for the overall structure of the message format, option format, and option value format.)

5.1. Signaling Codes

A code in the 7.00-7.31 range indicates a Signaling message. Values in this range are assigned by the "CoAP Signaling Codes" sub-registry (see Section 10.1).

For each message, there is a sender and a peer receiving the message.

Payloads in Signaling messages are diagnostic payloads as defined in Section 5.5.2 of [RFC7252]), unless otherwise defined by a Signaling message option.

5.2. Signaling Option Numbers

Option numbers for Signaling messages are specific to the message code. They do not share the number space with CoAP options for request/response messages or with Signaling messages using other codes.

Option numbers are assigned by the "CoAP Signaling Option Numbers" sub-registry (see Section 10.2).

Signaling options are elective or critical as defined in Section 5.4.1 of [RFC7252]. If a Signaling option is critical and not understood by the receiver, it MUST abort the connection (see Section 5.6). If the option is understood but cannot be processed, the option documents the behavior.

5.3. Capabilities and Settings Messages (CSM)

Capabilities and Settings messages (CSM) are used for two purposes:

- Each capability option advertises one capability of the sender to the recipient.
- Each setting option indicates a setting that will be applied by the sender.

One CSM MUST be sent by both endpoints at the start of the connection. Further CSM MAY be sent at any other time by either endpoint over the lifetime of the connection.

Both capability and setting options are cumulative. A CSM does not invalidate a previously sent capability indication or setting even if it is not repeated. A capability message without any option is a no-operation (and can be used as such). An option that is sent might override a previous value for the same option. The option defines how to handle this case if needed.

Base values are listed below for CSM Options. These are the values for the capability and setting before any Capabilities and Settings messages send a modified value.

These are not default values for the option, as defined in Section 5.4.4 in [RFC7252]. A default value would mean that an empty Capabilities and Settings message would result in the option being set to its default value.

Capabilities and Settings messages are indicated by the 7.01 code (CSM).

5.3.1. Max-Message-Size Capability Option

The sender can use the elective Max-Message-Size Option to indicate the maximum message size in bytes that it can receive.

#	C	R	Applies to	Name	Format	Length	Base Value
2			CSM	Max-Message-Size	uint	0-4	1152

C=Critical, R=Repeatable

As per Section 4.6 of [RFC7252], the base value (and the value used when this option is not implemented) is 1152.

The active value of the Max-Message-Size Option is replaced each time the option is sent with a modified value. Its starting value is its base value.

5.3.2. Block-wise Transfer Capability Option

#	C	R	Applies to	Name	Format	Length	Base Value
4			CSM	Block-wise Transfer	empty	0	(none)

C=Critical, R=Repeatable

A sender can use the elective Block-wise Transfer Option to indicate that it supports the block-wise transfer protocol [RFC7959].

If the option is not given, the peer has no information about whether block-wise transfers are supported by the sender or not. An implementation that supports block-wise transfers SHOULD indicate the Block-wise Transfer Option. If a Max-Message-Size Option is indicated with a value that is greater than 1152 (in the same or a different CSM message), the Block-wise Transfer Option also indicates support for BERT (see Section 6). Subsequently, if the Max-Message-Size Option is indicated with a value equal to or less than 1152, BERT support is no longer indicated.

5.4. Ping and Pong Messages

In CoAP over reliable transports, Empty messages (Code 0.00) can always be sent and MUST be ignored by the recipient. This provides a basic keep-alive function. In contrast, Ping and Pong messages are a bidirectional exchange.

Upon receipt of a Ping message, the receiver MUST return a Pong message with an identical token in response. Unless there is an option with delaying semantics such as the Custody Option, it SHOULD respond as soon as practical. As with all Signaling messages, the recipient of a Ping or Pong message MUST ignore elective options it does not understand.

Ping and Pong messages are indicated by the 7.02 code (Ping) and the 7.03 code (Pong).

5.4.1. Custody Option

#	C	R	Applies to	Name	Format	Length	Base Value
2			Ping, Pong	Custody	empty	0	(none)

C=Critical, R=Repeatable

When responding to a Ping message, the receiver can include an elective Custody Option in the Pong message. This option indicates that the application has processed all the request/response messages received prior to the Ping message on the current connection. (Note that there is no definition of specific application semantics for "processed", but there is an expectation that the receiver of a Pong Message with a Custody Option should be able to free buffers based on this indication.)

A sender can also include an elective Custody Option in a Ping message to explicitly request the inclusion of an elective Custody Option in the corresponding Pong message. In that case, the receiver SHOULD delay its Pong message until it finishes processing all the request/response messages received prior to the Ping message on the current connection.

5.5. Release Messages

A Release message indicates that the sender does not want to continue maintaining the connection and opts for an orderly shutdown. The details are in the options. A diagnostic payload (see Section 5.5.2 of [RFC7252]) MAY be included. A peer will normally respond to a Release message by closing the TCP/TLS connection. Messages may be in flight when the sender decides to send a Release message. The general expectation is that these will still be processed.

Release messages are indicated by the 7.04 code (Release).

Release messages can indicate one or more reasons using elective options. The following options are defined:

#	C	R	Applies to	Name	Format	Length	Base Value
2		x	Release	Alternative-Address	string	1-255	(none)

C=Critical, R=Repeatable

The elective Alternative-Address Option requests the peer to instead open a connection of the same scheme as the present connection to the alternative transport address given. Its value is in the form "authority" as defined in Section 3.2 of [RFC3986].

The Alternative-Address Option is a repeatable option as defined in Section 5.4.5 of [RFC7252]. When multiple occurrences of the option are included, the peer can choose any of the alternative transport addresses.

#	C	R	Applies to	Name	Format	Length	Base Value
4			Release	Hold-Off	uint	0-3	(none)

C=Critical, R=Repeatable

The elective Hold-Off Option indicates that the server is requesting that the peer not reconnect to it for the number of seconds given in the value.

5.6. Abort Messages

An Abort message indicates that the sender is unable to continue maintaining the connection and cannot even wait for an orderly release. The sender shuts down the connection immediately after the abort (and may or may not wait for a Release or Abort message or connection shutdown in the inverse direction). A diagnostic payload (see Section 5.5.2 of [RFC7252]) SHOULD be included in the Abort message. Messages may be in flight when the sender decides to send

an Abort message. The general expectation is that these will NOT be processed.

Abort messages are indicated by the 7.05 code (Abort).

Abort messages can indicate one or more reasons using elective options. The following option is defined:

#	C	R	Applies to	Name	Format	Length	Base Value
2			Abort	Bad-CSM-Option	uint	0-2	(none)

C=Critical, R=Repeatable

The elective Bad-CSM-Option Option indicates that the sender is unable to process the CSM option identified by its option number, e.g. when it is critical and the option number is unknown by the sender, or when there is parameter problem with the value of an elective option. More detailed information SHOULD be included as a diagnostic payload.

For CoAP over UDP, messages which contain syntax violations are processed as message format errors. As described in Sections 4.2 and 4.3 of [RFC7252], such messages are rejected by sending a matching Reset message and otherwise ignoring the message.

For CoAP over reliable transports, the recipient rejects such messages by sending an Abort message and otherwise ignoring the message. No specific option has been defined for the Abort message in this case, as the details are best left to a diagnostic payload.

5.7. Signaling examples

An encoded example of a Ping message with a non-empty token is shown in Figure 14.

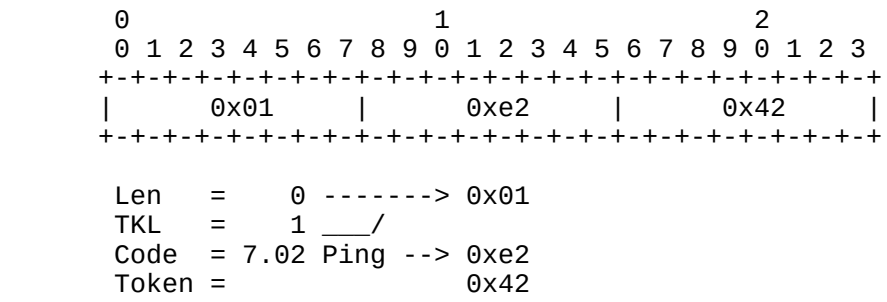


Figure 14: Ping Message Example

An encoded example of the corresponding Pong message is shown in Figure 15.

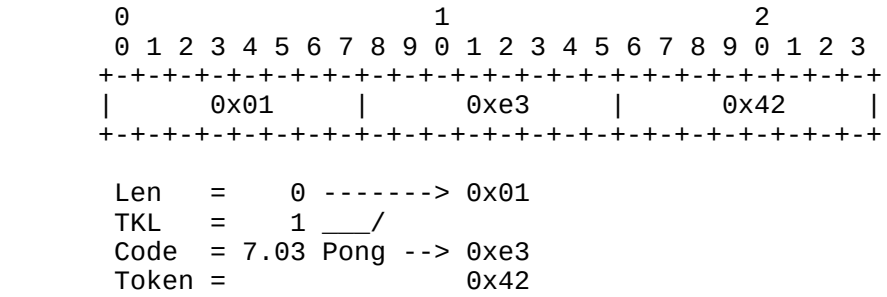


Figure 15: Pong Message Example

6. Block-wise Transfer and Reliable Transports

The message size restrictions defined in Section 4.6 of CoAP [RFC7252] to avoid IP fragmentation are not necessary when CoAP is used over a reliable transport. While this suggests that the Block-wise transfer protocol [RFC7959] is also no longer needed, it remains applicable for a number of cases:

- large messages, such as firmware downloads, may cause undesired head-of-line blocking when a single TCP connection is used
- a UDP-to-TCP gateway may simply not have the context to convert a message with a Block Option into the equivalent exchange without any use of a Block Option (it would need to convert the entire blockwise exchange from start to end into a single exchange)

The 'Block-wise Extension for Reliable Transport (BERT)' extends the Block protocol to enable the use of larger messages over a reliable transport.

The use of this new extension is signaled by sending Block1 or Block2 Options with SZX == 7 (a "BERT option"). SZX == 7 is a reserved value in [RFC7959].

In control usage, a BERT option is interpreted in the same way as the equivalent Option with SZX == 6, except that it also indicates the capability to process BERT blocks. As with the basic Block protocol, the recipient of a CoAP request with a BERT option in control usage is allowed to respond with a different SZX value, e.g. to send a non-BERT block instead.

In descriptive usage, a BERT Option is interpreted in the same way as the equivalent Option with SZX == 6, except that the payload is also allowed to contain a multiple of 1024 bytes (non-final BERT block) or more than 1024 bytes (final BERT block).

The recipient of a non-final BERT block (M=1) conceptually partitions the payload into a sequence of 1024-byte blocks and acts exactly as if it had received this sequence in conjunction with block numbers starting at, and sequentially increasing from, the block number given in the Block Option. In other words, the entire BERT block is positioned at the byte position that results from multiplying the block number with 1024. The position of further blocks to be transferred is indicated by incrementing the block number by the number of elements in this sequence (i.e., the size of the payload divided by 1024 bytes).

As with SZX == 6, the recipient of a final BERT block (M=0) simply appends the payload at the byte position that is indicated by the block number multiplied with 1024.

The following examples illustrate BERT options. A value of SZX == 7 is labeled as "BERT" or as "BERT(nnn)" to indicate a payload of size nnn.

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

6.1. Example: GET with BERT Blocks

Figure 16 shows a GET request with a response that is split into three BERT blocks. The first response contains 3072 bytes of payload; the second, 5120; and the third, 4711. Note how the block

number increments to move the position inside the response body forward.

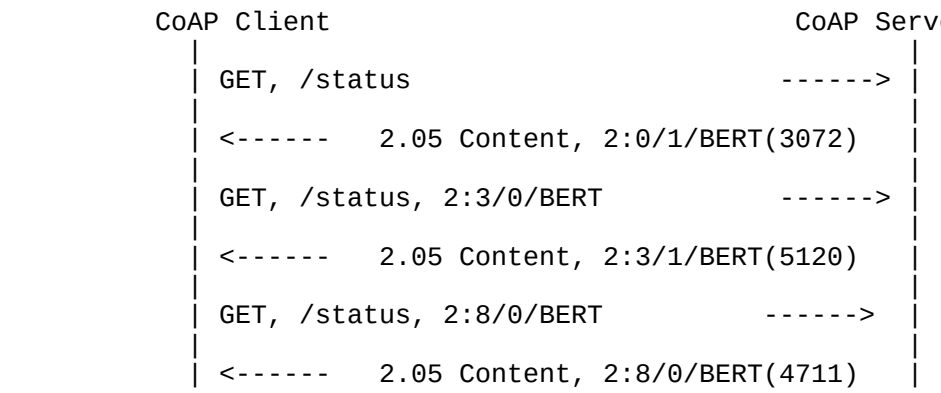


Figure 16: GET with BERT blocks

6.2. Example: PUT with BERT Blocks

Figure 17 demonstrates a PUT exchange with BERT blocks.

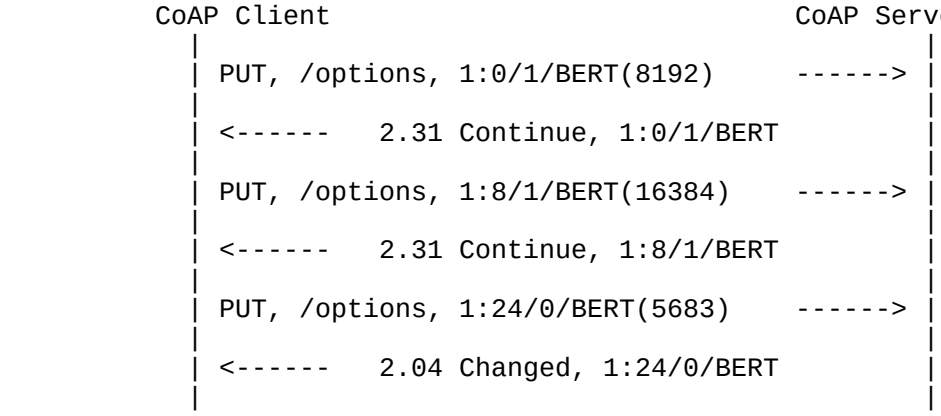


Figure 17: PUT with BERT blocks

7. CoAP over Reliable Transport URIs

CoAP over UDP [RFC7252] defines the "coap" and "coaps" URI schemes. This document corrects an erratum in Sections 6.1 and 6.2 of [RFC7252] and defines how to use the schemes with the new transports. Section 8 (Multicast CoAP) in [RFC7252] is not applicable to these new transports.

The syntax for the URI schemes in this section are specified using Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", "query", and "fragment" are adopted from [RFC3986].

The ABNF syntax defined in Sections 6.1 and 6.2 of [RFC7252] for "coap" and "coaps" schemes lacks the fragment identifer. This specification updates the two rules in those sections as follows:

```
coap-URI = "coap:" "//" host [ ":" port ]
  path-abempty [ "?" query ]  [ "#" fragment ]
coaps-URI = "coaps:" "//" host [ ":" port ]
  path-abempty [ "?" query ] [ "#" fragment ]
```

7.1. Use of the "coap" URI scheme with TCP

The "coap" URI scheme defined in Section 6.1 of [RFC7252] can also be used to identify CoAP resources that are intended to be accessible using CoAP over TCP.

The syntax defined in Section 6.1 of [RFC7252] applies to this transport, with the following change:

- The port subcomponent indicates the TCP port at which the CoAP server is located. (If it is empty or not given, then the default port 5683 is assumed, as with UDP.)

7.2. Use of the "coaps" URI scheme with TLS over TCP

The "coaps" URI scheme defined in Section 6.2 of [RFC7252] can also be used to identify CoAP resources that are intended to be accessible using CoAP over TCP secured with TLS.

The syntax defined in Section 6.2 of [RFC7252] applies to this transport, with the following changes:

- The port subcomponent indicates the TCP port at which the TLS server for the CoAP Connection Acceptor is located. If it is empty or not given, then the default port 5684 is assumed.

- If a TLS server does not support the Application-Layer Protocol Negotiation Extension (ALPN) [RFC7301] or wishes to accommodate TLS clients that do not support ALPN, it MAY offer a coaps endpoint on the default TCP port 5684. This endpoint MAY also be ALPN enabled. A TLS server MAY offer coaps endpoints on TCP ports other than 5684; these then MUST be ALPN enabled.

- For TCP ports other than port 5684, the TLS client MUST use the ALPN extension to advertise the "coap" protocol identifier (see Section 10.6) in the list of protocols in its ClientHello. If the TCP server selects and returns the "coap" protocol identifier using the ALPN extension in its ServerHello, then the connection succeeds. If the TLS server either does not negotiate the ALPN extension or returns a no_application_protocol alert, the TLS client MUST close the connection.

- For TCP port 5684, a TLS client MAY use the ALPN extension to advertise the "coap" protocol identifier in the list of protocols in its ClientHello. If the TLS server selects and returns the "coap" protocol identifier using the ALPN extension in its ServerHello, then the connection succeeds. If the TLS server returns a no_application_protocol alert, then the TLS client MUST close the connection. If the TLS server does not negotiate the ALPN extension, then coaps over TCP is implicitly selected.

- For TCP port 5684, if the TLS client does not use the ALPN extension to negotiate the protocol, then coaps over TCP is implicitly selected.

7.3. Use of the "coap" URI scheme with WebSockets

The "coap" URI scheme defined in Section 6.1 of [RFC7252] can also be used to identify CoAP resources that are intended to be accessible using CoAP over WebSockets.

The WebSocket endpoint is identified by a "ws" URI that is composed of the authority part of the "coap" URI and the well-known path "/.well-known/coap" [RFC5785] [I-D.bormann-hybi-ws-wk]. The path and query parts of the "coap" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP:

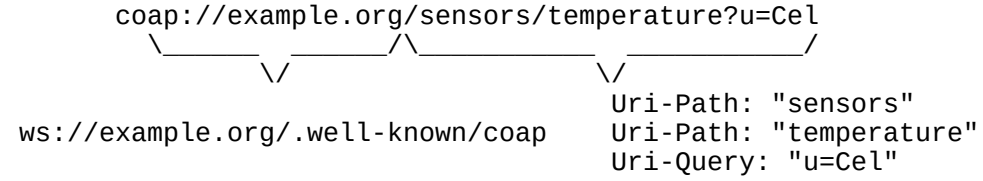


Figure 18: Building ws URIs and Uri options from coap URIs

Note that the default port for "coap" is 5683, while the default port for "ws" is 80. Therefore, if the port given for "coap" is 80, the default port for "ws" can be used. If the port is not given for

"coap", then an explicit port number of 5683 needs to be given for "ws".

7.4. Use of the "coaps" URI scheme with WebSockets

The "coaps" URI scheme defined in Section 6.2 of [RFC7252] can also be used to identify CoAP resources that are intended to be accessible using CoAP over WebSockets secured by TLS.

The WebSocket endpoint is identified by a "wss" URI that is composed of the authority part of the "coaps" URI and the well-known path "/.well-known/coap" [RFC5785] [I-D.bormann-hybi-ws-wk]. The path and query parts of the "coaps" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP.

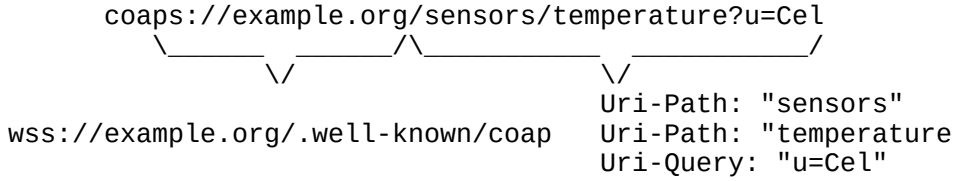


Figure 19: Building wss URIs and Uri options from coaps URIs

Note that the default port for "coaps" is 5684, while the default port for "wss" is 443. If the port given for "coap" is 443, the default port for "wss" can be used. If the port is not given for "coaps", then an explicit port number of 5684 needs to be given for "wss".

7.5. Uri-Host and Uri-Port Options

Except for the transports over WebSockets, CoAP over reliable transports maintains the property from Section 5.10.1 of [RFC7252]:

> The default values for the Uri-Host and Uri-Port Options are sufficient for requests to most servers.

Unless otherwise noted, the default value of the Uri-Host Option is the IP literal representing the destination IP address of the request message. The default value of the Uri-Port Option is the destination TCP port.

For CoAP over TLS, these default values are the same unless Server Name Indication (SNI) [RFC6066] is negotiated. In this case, the default value of the Uri-Host Option in requests from the TLS client to the TLS server is the SNI host.

For CoAP over WebSockets, the default value of the Uri-Host Option in requests from the WebSocket client to the WebSocket server is indicated by the Host header field from the WebSocket handshake.

7.6. Decomposing URIs into Options

The steps are the same as specified in Section 6.4 of [RFC7252] with minor changes.

This step from [RFC7252]:

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|.

is updated to:

7. If |port| does not equal the request's destination UDP port or TCP port, include a Uri-Port Option and let that option's value be |port|.

7.7. Composing URIs from Options

The steps are the same as specified in Section 6.5 of [RFC7252] with minor changes.

This step from [RFC7252]:

1. If the request is secured using DTLS, let |url| be the string "coaps://". Otherwise, let |url| be the string "coap://".

is updated to:

1. If the request is secured using DTLS or TLS, let |url| be the string "coaps://". Otherwise, let |url| be the string "coap://".

This step from [RFC7252]:

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.

is updated to:

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 or TCP port.

7.8. Trying out multiple transports at once

As in the "Happy Eyeballs" approach to using IPv6 and IPv4 [RFC6555], an application may want to try out multiple transports for a given URI at the same time, e.g., DTLS over UDP and TLS over TCP. However, two important caveats need to be considered:

- Initiating multiple instances of the same exchange with the intention of using only one of the successful results is only safe for idempotent exchanges (see Section 5.1 of [RFC7252]).

- An important setback in using the UDP or DTLS over UDP transport through NATs and other middleboxes can be the quick loss of NAT bindings during idling periods [HomeGateway]. This will not be evident right on the initial exchange.

After the initial exchange, or whenever important information is learned about which selection to prefer, an endpoint may want to cache this information; however, the information may become stale after the endpoint moves or the network changes. A cache timeout (possibly enhanced by movement detection) is advisable.

Alternatively, or additionally, the choice of transport may be aided by configuration and resource directory information; the self-description of a node may also include target attributes for links given to resources there. Details of such attributes are out of scope for the present document; see for instance [I-D.ietf-core-resource-directory].

8. Securing CoAP

Security Challenges for the Internet of Things [SecurityChallenges] recommends:

> ... it is essential that IoT protocol suites specify a mandatory to implement but optional to use security solution. This will ensure security is available in all implementations, but configurable to use when not necessary (e.g., in closed environment). ... even if those features stretch the capabilities of such devices.

A security solution MUST be implemented to protect CoAP over reliable transports and MUST be enabled by default. This document defines the TLS binding, but alternative solutions at different layers in the protocol stack MAY be used to protect CoAP over reliable transports when appropriate. Note that there is ongoing work to support a data object-based security model for CoAP that is independent of transport (see [I-D.ietf-core-object-security]).

8.1. TLS binding for CoAP over TCP

The TLS usage guidance in [RFC7925] applies, including the guidance about cipher suites in that document that are derived from the mandatory to implement (MTI) cipher suites defined in [RFC7252]. (Note that this selection caters for the device-to-cloud use case of CoAP over TLS more than for any use within a back-end environment, where the standard TLS 1.2 cipher suites or the more recent ones defined in [RFC7525] are more appropriate.)

During the provisioning phase, a CoAP device is provided with the security information that it needs, including keying materials, access control lists, and authorization servers. At the end of the provisioning phase, the device will be in one of four security modes:

NoSec: TLS is disabled.

PreSharedKey: TLS is enabled. The guidance in Section 4.2 of [RFC7925] applies.

RawPublicKey: TLS is enabled. The guidance in Section 4.3 of [RFC7925] applies.

Certificate: TLS is enabled. The guidance in Section 4.4 of [RFC7925] applies.

The "NoSec" mode is optional-to-implement. The system simply sends the packets over normal TCP which is indicated by the "coap" scheme and the TCP 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.

"PreSharedKey", "RawPublicKey", or "Certificate" is mandatory-to-implement for the TLS binding depending on the credential type used with the device. These security modes are achieved using TLS and are indicated by the "coaps" scheme and TLS-secured CoAP default port.

8.2. TLS usage for CoAP over WebSockets

A CoAP client requesting a resource identified by a "coaps" URI negotiates a secure WebSocket connection to a WebSocket server endpoint with a "wss" URI. This is described in Section 7.4.

The client MUST perform a TLS handshake after opening the connection to the server. The guidance in Section 4.1 of [RFC6455] applies. When a CoAP server exposes resources identified by a "coaps" URI, the guidance in Section 4.4 of [RFC7925] applies towards mandatory-to-implement TLS functionality for certificates. For the server-side

requirements in accepting incoming connections over a HTTPS (HTTP-over-TLS) port, the guidance in Section 4.2 of [RFC6455] applies.

Note that this formally inherits the mandatory to implement cipher suites defined in [RFC5246]. However, modern usually browsers implement more recent cipher suites that then are automatically picked up via the JavaScript WebSocket API. WebSocket Servers that provide Secure CoAP over WebSockets for the browser use case will need to follow the browser preferences and MUST follow [RFC7525].

9. Security Considerations

The security considerations of [RFC7252] apply. For CoAP over WebSockets and CoAP over TLS-secured WebSockets, the security considerations of [RFC6455] also apply.

9.1. Signaling Messages

The guidance given by an Alternative-Address Option cannot be followed blindly. In particular, a peer MUST NOT assume that a successful connection to the Alternative-Address inherits all the security properties of the current connection.

10. IANA Considerations

10.1. Signaling Codes

IANA is requested to create a third sub-registry for values of the Code field in the CoAP header (Section 12.1 of [RFC7252]). The name of this sub-registry is "CoAP Signaling Codes".

Each entry in the sub-registry must include the Signaling Code in the range 7.00-7.31, its name, and a reference to its documentation.

Initial entries in this sub-registry are as follows:

Code	Name	Reference
7.01	CSM	[RFCthis]
7.02	Ping	[RFCthis]
7.03	Pong	[RFCthis]
7.04	Release	[RFCthis]
7.05	Abort	[RFCthis]

Table 1: CoAP Signal Codes

All other Signaling Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG Approval" as described in [RFC5226].

10.2. CoAP Signaling Option Numbers Registry

IANA is requested to create a sub-registry for Options Numbers used in CoAP signaling options within the "CoRE Parameters" registry. The name of this sub-registry is "CoAP Signaling Option Numbers".

Each entry in the sub-registry must include one or more of the codes in the Signaling Codes subregistry (Section 10.1), the option number, the name of the option, and a reference to the option’s documentation.

Initial entries in this sub-registry are as follows:

Applies to	Number	Name	Reference
7.01	2	Max-Message-Size	[RFCthis]
7.01	4	Block-wise-Transfer	[RFCthis]
7.02, 7.03	2	Custody	[RFCthis]
7.04	2	Alternative-Address	[RFCthis]
7.04	4	Hold-Off	[RFCthis]
7.05	2	Bad-CSM-Option	[RFCthis]

Table 2: CoAP Signal Option Codes

The IANA policy for future additions to this sub-registry is based on number ranges for the option numbers, analogous to the policy defined in Section 12.2 of [RFC7252].

The documentation for a Signaling Option Number should specify the semantics of an option with that number, including the following properties:

- Whether the option is critical or elective, as determined by the Option Number.
- Whether the option is repeatable.
- The format and length of the option’s value.
- The base value for the option, if any.

10.3. Service Name and Port Number Registration

IANA is requested to assign the port number 5683 and the service name "coap", in accordance with [RFC6335].

Service Name.
coap

Transport Protocol.
tcp

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFCthis]

Port Number.
5683

10.4. Secure Service Name and Port Number Registration

IANA is requested to assign the port number 5684 and the service name "coaps+tcp", in accordance with [RFC6335]. The port number is requested also to address the exceptional case of TLS implementations that do not support the "Application-Layer Protocol Negotiation Extension" [RFC7301].

Service Name.
coaps

Transport Protocol.
tcp

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFC7301], [RFCthis]

Port Number.
5684

10.5. Well-Known URI Suffix Registration

IANA is requested to register the ’coap’ well-known URI in the "Well-Known URIs" registry. This registration request complies with [RFC5785]:

URI Suffix.

coap

Change controller.
IETF

Specification document(s).
[RFCthis]

Related information.
None.

10.6. ALPN Protocol Identifier

IANA is requested to assign the following value in the registry "Application Layer Protocol Negotiation (ALPN) Protocol IDs" created by [RFC7301]. The "coap" string identifies CoAP when used over TLS.

Protocol.
CoAP

Identification Sequence.
0x63 0x6f 0x61 0x70 ("coap")

Reference.
[RFCthis]

10.7. WebSocket Subprotocol Registration

IANA is requested to register the WebSocket CoAP subprotocol under the "WebSocket Subprotocol Name Registry":

Subprotocol Identifier.
coap

Subprotocol Common Name.
Constrained Application Protocol (CoAP)

Subprotocol Definition.
[RFCthis]

10.8. CoAP Option Numbers Registry

IANA is requested to add [RFCthis] to the references for the following entries registered by [RFC7959] in the "CoAP Option Numbers" sub-registry defined by [RFC7252]:

Number	Name	Reference
23	Block2	RFC 7959, [RFCthis]
27	Block1	RFC 7959, [RFCthis]

Table 3: CoAP Option Numbers

11. References

11.1. Normative References

[I-D.bormann-hybi-ws-wk]
Bormann, C., "Well-known URIs for the WebSocket Protocol", draft-bormann-hybi-ws-wk-00 (work in progress), May 2017.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <http://www.rfc-editor.org/info/rfc5785>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <http://www.rfc-editor.org/info/rfc6066>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <http://www.rfc-editor.org/info/rfc6455>.

[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>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[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>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <http://www.rfc-editor.org/info/rfc7925>.

[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>.

11.2. Informative References

[HomeGateway] Eggert, L., "An experimental study of home gateway characteristics", Proceedings of the 10th annual conference on Internet measurement , 2010.

[I-D.ietf-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-ietf-core-cocoa-01 (work in progress), March 2017.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-03 (work in progress), May 2017.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-10 (work in progress), March 2017.

[LWM2M] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification Version 1.0", February 2017, <http://www.openmobilealliance.org/release/LightweightM2M/V1_0-20170208-A/OMA-TS-LightweightM2M-V1_0-20170208-A.pdf>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <http://www.rfc-editor.org/info/rfc768>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[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, DOI 10.17487/RFC6335, August 2011, <http://www.rfc-editor.org/info/rfc6335>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, DOI 10.17487/RFC6555, April 2012, <http://www.rfc-editor.org/info/rfc6555>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[SecurityChallenges] Polk, T. and S. Turner, "Security Challenges for the Internet of Things", Interconnecting Smart Objects with the Internet / IAB Workshop , February 2011, <http://www.iab.org/wp-content/IAB-uploads/2011/03/Turner.pdf>.

Appendix A. Updates to RFC 7641 Observing Resources in the Constrained Application Protocol (CoAP)

In this appendix, "client" and "server" refer to the CoAP client and CoAP server.

A.1. Notifications and Reordering

When using the Observe Option with CoAP over UDP, notifications from the server set the option value to an increasing sequence number for reordering detection on the client since messages can arrive in a different order than they were sent. This sequence number is not required for CoAP over reliable transports since the TCP protocol ensures reliable and ordered delivery of messages. The value of the Observe Option in 2.xx notifications MAY be empty on transmission and MUST be ignored on reception.

A.2. Transmission and Acknowledgements

For CoAP over UDP, server notifications to the client can be confirmable or non-confirmable. A confirmable message requires the client to either respond with an acknowledgement message or a reset message. An acknowledgement message indicates that the client is alive and wishes to receive further notifications. A reset message indicates that the client does not recognize the token which causes the server to remove the associated entry from the list of observers.

Since TCP eliminates the need for the message layer to support reliability, CoAP over reliable transports does not support confirmable or non-confirmable message types. All notifications are delivered reliably to the client with positive acknowledgement of receipt occurring at the TCP level. If the client does not recognize the token in a notification, it MAY immediately abort the connection (see Section 5.6).

A.3. Freshness

For CoAP over UDP, if a client does not receive a notification for some time, it MAY send a new GET request with the same token as the original request to re-register its interest in a resource and verify that the server is still responsive. For CoAP over reliable transports, it is more efficient to check the health of the connection (and all its active observations) by sending a CoAP Ping Signaling message (Section 5.4) rather than individual requests to confirm active observations.

A.4. Cancellation

For CoAP over UDP, a client that is no longer interested in receiving notifications can "forget" the observation and respond to the next notification from the server with a reset message to cancel the observation.

For CoAP over reliable transports, a client MUST explicitly deregister by issuing a GET request that has the Token field set to the token of the observation to be cancelled and includes an Observe Option with the value set to 1 (deregister).

If the client observes one or more resources over a reliable transport, then the CoAP server (or intermediary in the role of the CoAP server) MUST remove all entries associated with the client endpoint from the lists of observers when the connection is either closed or times out.

Appendix B. CoAP over WebSocket Examples

This section gives examples for the first two configurations discussed in Section 4.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap" URI might be as follows. Figure 20 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI <coap://example.org/sensors/temperature?u=Cel>, for example, from a resource representation that it retrieved previously.

2. It establishes a WebSocket connection to the endpoint URI composed of the authority "example.org" and the well-known path "/.well-known/coap", <ws://example.org/.well-known/coap>.

3. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") options.

4. It waits for the server to return a response.

5. The CoAP client uses the connection for further requests, or the connection is closed.

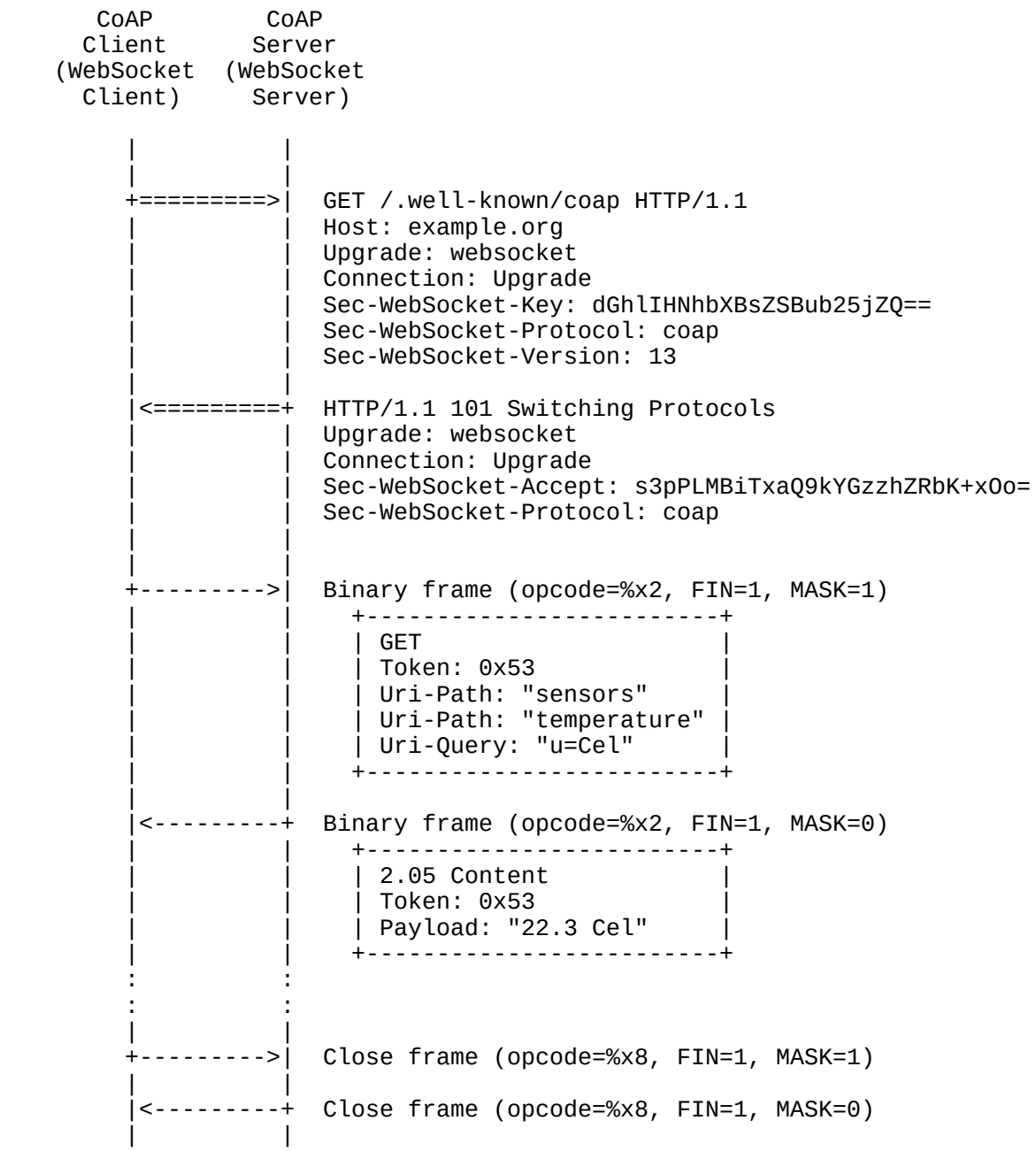


Figure 20: A CoAP client retrieves the representation of a resource identified by a "coap" URI over the WebSocket protocol

Figure 21 shows how a CoAP client uses a CoAP forward proxy with a WebSocket endpoint to retrieve the representation of the resource "coap://[2001:db8::1]/". The use of the forward proxy and the address of the WebSocket endpoint are determined by the client from local configuration rules. The request URI is specified in the Proxy-Uri Option. Since the request URI uses the "coap" URI scheme, the proxy fulfills the request by issuing a Confirmable GET request over UDP to the CoAP server and returning the response over the WebSocket connection to the client.

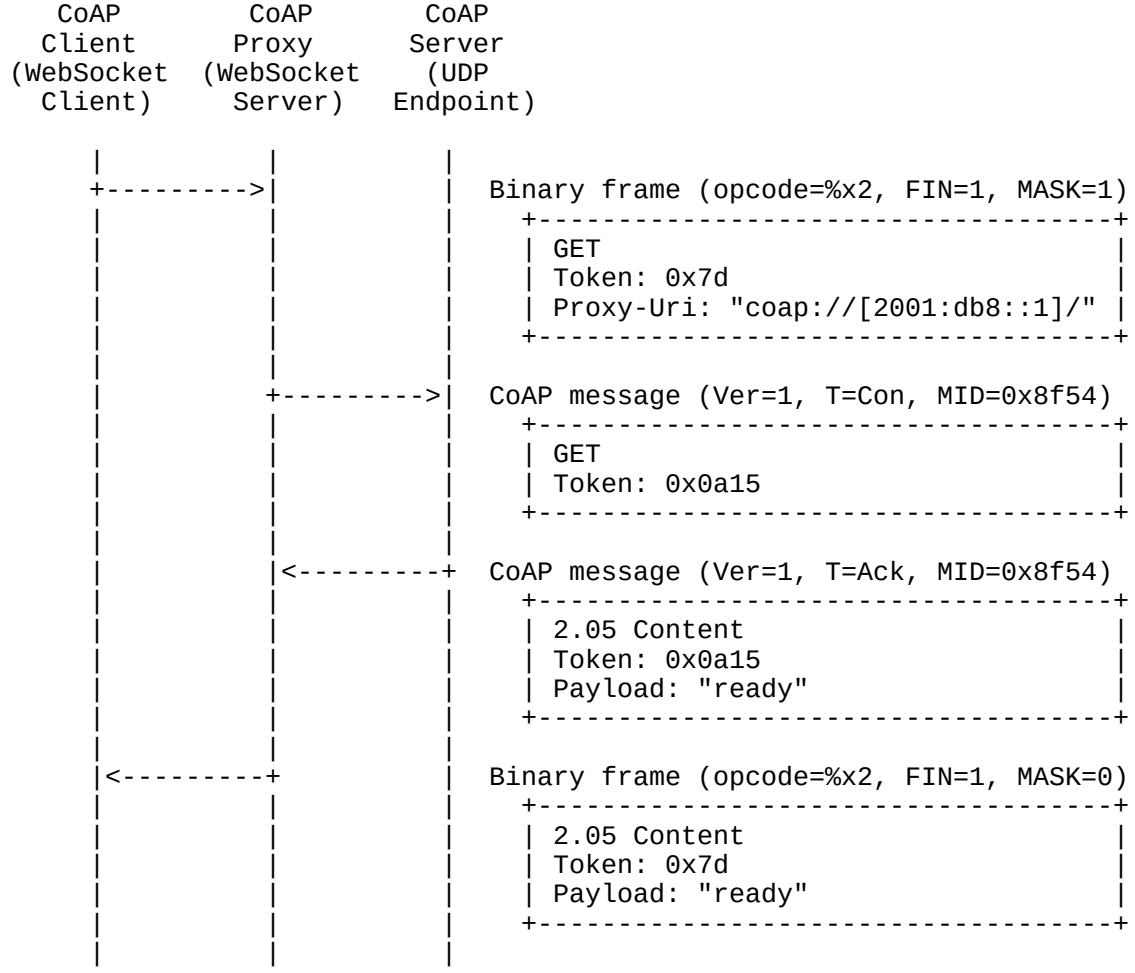


Figure 21: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSocket-enabled CoAP proxy

Appendix C. Change Log

The RFC Editor is requested to remove this section at publication.

C.1. Since draft-ietf-core-coap-tcp-tls-02

Merged draft-savolainen-core-coap-websockets-07 Merged draft-bormann-core-block-bert-01 Merged draft-bormann-core-coap-sig-02

C.2. Since draft-ietf-core-coap-tcp-tls-03

Editorial updates

Added mandatory exchange of Capabilities and Settings messages after connecting

Added support for coaps+tcp port 5684 and more details on Application-Layer Protocol Negotiation (ALPN)

Added guidance on CoAP Signaling Ping-Pong versus WebSocket Ping-Pong

Updated references and requirements for TLS security considerations

C.3. Since draft-ietf-core-coap-tcp-tls-04

Updated references

Added Appendix: Updates to RFC7641 Observing Resources in the Constrained Application Protocol (CoAP)

Updated Capability and Settings Message (CSM) exchange in the Opening Handshake to allow initiator to send messages before receiving acceptor CSM

C.4. Since draft-ietf-core-coap-tcp-tls-05

Addressed feedback from Working Group Last Call

Added Securing CoAP section and informative reference to OSCOAP

Removed the Server-Name and Bad-Server-Name Options

Clarified the Capability and Settings Message (CSM) exchange

Updated Pong response requirements

Added Connection Initiator and Connection Acceptor terminology where appropriate

Updated LWM2M 1.0 informative reference

C.5. Since draft-ietf-core-coap-tcp-tls-06

Addressed feedback from second Working Group Last Call

C.6. Since draft-ietf-core-coap-tcp-tls-07

Addressed feedback from IETF Last Call

Addressed feedback from ARTART review

Addressed feedback from GENART review

Addressed feedback from TSVART review

Added fragment identifiers to URI schemes

Added "Updates RFC7959" for BERT

Added "Updates RFC6455" to extend well-known URI mechanism to ws and wss

Clarified well-known URI mechanism use for all URI schemes

Changed NoSec to optional-to-implement

C.7. Since draft-ietf-core-coap-tcp-tls-08

Reverted "Updates RFC6455" to extend well-known URI mechanism to ws and wss; point to [I-D.bormann-hybi-ws-wk] instead

Don’t use port 443 as the default port for coaps+tcp

Remove coap+tt and coaps+tt URI schemes (where tt is tcp or ws); map everything to coap/coaps

Acknowledgements

We would like to thank Stephen Berard, Geoffrey Cristallo, Olivier Delaby, Esko Dijk, Christian Groves, Nadir Javed, Michael Koster, Matthias Kovatsch, Achim Kraus, David Navarro, Szymon Sasin, Goran Selander, Zach Shelby, Andrew Summers, Julien Vermillard, and Gengyu Wei for their feedback. Last-call reviews from Mark Nottingham and Yoshifumi Nishida as well as several IESG reviewers provided extensive comments; from the IESG, we would like to specifically call out Adam Roach, Ben Campbell, Eric Rescorla, Mirja Kuehlewind, and the responsible AD Alexey Melnikov.

Contributors

Matthias Kovatsch
Siemens AG
Otto-Hahn-Ring 6
Munich D-81739

Phone: +49-173-5288856
EMail: matthias.kovatsch@siemens.com

Teemu Savolainen
Nokia Technologies
Hatanpaan valtatie 30
Tampere FI-33100
Finland

Email: teemu.savolainen@nokia.com

Valik Solorzano Barboza
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: vsolorzanobarboza@zebra.com

Authors' Addresses

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

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

Simon Lemay
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: slemay@zebra.com

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

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

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

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

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

Brian Raymor (editor)
Microsoft
One Microsoft Way
Redmond 98052
United States of America

Email: brian.raymor@microsoft.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

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

March 13, 2017

CoAP Simple Congestion Control/Advanced
draft-ietf-core-cocoa-01

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.

Table of Contents

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 [RFC8085] 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:

RTO := 0.25 * E_weak_ + 0.75 * RTO (1)

RTO := 0.5 * E_strong_ + 0.5 * RTO (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 x RTO and ACK_RANDOM_FACTOR x 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. This approach appears to contravene the mandate in Section 3.1.1 of [RFC8085] that "latency samples MUST NOT be derived from ambiguous transactions". However, those samples are not simply combined into the strong estimator, but are used to correct the limited knowledge that can be gained from the strong RTT measurements by employing an additional weak estimator. Evidence that has been collected from experiments appears to support that the overall effect of using this data in the way described is beneficial (Appendix B).

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

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

```
RTO := 1 s + (0.5 * 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/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/RTO.

[RFC7641], Section 4.5.1, specifies that the rate of NONs SHOULD NOT exceed 1/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/RTO. Assuming that the RTO estimation in CoCoA works as expected, RTO[k] should be slightly greater than the 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 [RFC8085] 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>.

[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>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <http://www.rfc-editor.org/info/rfc8085>.

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).

PLIMIT = LAMBDA (3)

PLIMIT = max(LAMBDA, LAMBDA*ACK_TIMEOUT/mean(RTO)) (4)

LAMBDA = max(4, KNOWN_DEST_ENDPOINTS/4) (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

LAMBDA = max(4, 3/4) = 4.

Then PLIMIT is calculated:

PLIMIT = max(4, (4*2 s)/mean(0.5 s, 1.5 s)) = max (4, 8 s/1 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

LAMBDA = max(4, 101/4) = 26

Based on this, PLIMIT is calculated as follows:

PLIMIT= max(26, (26*2 s)/1 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.

Appendix B. Supporting evidence

(Editor’s note: The WG needs to decide wbether this appendix or something like it should be present in the published version of this specification. If that is deemed desirable, the references local to

this appendix need to be merged with those from the specification proper.)

CoCoA has been evaluated by means of simulation and experimentation in diverse scenarios comprising different link layer technologies, network topologies, traffic patterns and device classes. The main overall evaluation result is that CoCoA consistently delivers a performance which is better than, or at least similar to, that of default CoAP congestion control. While the latter is insensitive to network conditions, CoCoA is adaptive and makes good use of RTT samples.

It has been shown over real GPRS and IEEE 802.15.4 mesh network testebds that in these settings, in comparison to default CoAP, CoCoA increases throughput and reduces the time it takes for a network to process traffic bursts, while not sacrificing fairness. In contrast, other RTT-sensitive approaches such as Linux-RTO or Peak-Hopper-RTO may be too simple or do not adapt well to IoT scenarios, underperforming default CoAP under certain conditions [1]. On the other hand, CoCoA has been found to reduce latency in GPRS and Wi-Fi setups, compared with default CoAP [2].

CoCoA performance has also been evaluated for non-confirmable traffic over emulated GPRS/UMTS links and over a real IEEE 802.15.4 mesh testbed. Results show that since CoCoA is adaptive, it yields better packet delivery ratio than default CoAP (which does not apply congestion control to non-confirmable messages) or Observe (which introduces congestion control that is not adaptive to network conditions) [3, 4].

B.1. Older versions of the draft and improvement

CoCoA has evolved since its initial draft version. Its core has remained mostly stable since draft-bormann-core-cocoa-02. The evolution of CoCoA has been driven by research work. This process, including evaluations of early versions of CoCoA, as well as improvement proposals that were finally incorporated in CoCoA, is reflected in published works [5-10].

B.2. References

[1] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoAP congestion control for the Internet of Things", IEEE Communications Magazine, July 2016.

[2] F. Zheng, B. Fu, Z. Cao, "CoAP Latency Evaluation", draft-zheng-core-coap-lantency-evaluation-00, 2016 (work in progress).

[3] A. Betzler, C. Gomez, I. Demirkol, "Evaluation of Advanced Congestion Control Mechanisms for Unreliable CoAP Communications", PE-WASUN, Cancun, Mexico, 2015.

[4] A. Betzler, J. Isern, C. Gomez, I. Demirkol, J. Paradells, "Experimental Evaluation of Congestion Control for CoAP Communications without End-to-End Reliability", Ad Hoc Networks, Volume 52, 1 December 2016, Pages 183-194.

[5] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "Congestion Control in Reliable CoAP Communication", 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM'13), Barcelona, Spain, Nov. 2013.

[6] A. Betzler, C. Gomez, I. Demirkol, M. Kovatsch, "Congestion Control for CoAP cloud services", 8th International Workshop on Service-Oriented Cyber-Physical Systems in Converging Networked Environments (SOCNE) 2014, Barcelona, Spain, Sept. 2014.

[7] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoCoA+: an advanced congestion control mechanism for CoAP", Ad Hoc Networks journal, 2015.

[8] Bhalerao, Rahul, Sridhar Srinivasa Subramanian, and Joseph Pasquale. "An analysis and improvement of congestion control in the CoAP Internet-of-Things protocol." 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC). IEEE, 2016.

[9] I Jaervinen, L Daniel, M Kojo, "Experimental evaluation of alternative congestion control algorithms for Constrained Application Protocol (CoAP)", IEEE 2nd World Forum on Internet of Things (WF-IoT), 2015.

[10] Balandina, Ekaterina, Yevgeni Koucheryavy, and Andrei Gurtov. "Computing the retransmission timeout in coap." Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg, 2013. 352-362.

Acknowledgements

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

Michael Scharf did a review of CoAP congestion control issues that asked a lot of good questions. Several Transport Area representatives made further significant inputs this discussion during IETF84, including Lars Eggert, Michael Scharf, and David

Black. Andrew McGregor, Eric Rescorla, Richard Kelsey, Ed Beroset, Jari Arkko, Zach Shelby, Matthias Kovatsch and many others provided very useful additions.

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

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
Internet-Draft
Intended status: Standards Track
Expires: July 30, 2017

P. van der Stok
consultant
A. Bierman
YumaWorks
M. Veillette
Trilliant Networks Inc.
A. Pelov
Acklio

January 26, 2017

CoAP Management Interface
draft-ietf-core-comi-00

Abstract

This document describes a network management interface for constrained devices and networks, called CoAP Management Interface (CoMI). The Constrained Application Protocol (CoAP) is used to access data resources specified in YANG, or SMIv2 converted to YANG. CoMI uses the YANG to CBOR mapping and converts YANG identifier strings to numeric identifiers for payload size reduction. CoMI extends the set of YANG based protocols, NETCONF and RESTCONF, with the capability to manage constrained devices and networks.

Note

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

Status of This Memo

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

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

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

This Internet-Draft will expire on July 30, 2017.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. The messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

This draft describes the CoAP Management Interface which uses CoAP methods to access structured data defined in YANG [RFC7950]. This draft is complementary to the draft [I-D.ietf-netconf-restconf] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data sets, specified in a standardized language such as YANG, promotes interoperability between devices and applications from different manufacturers. A large amount of Management Information Base (MIB) [mibreg] specifications already exists for monitoring purposes. This data can be accessed in RESTCONF or CoMI if the server converts the SMIv2 modules to YANG, using the mapping rules defined in [RFC6643].

CoMI and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs up to 10 round trips.

To promote small packets, CoMI uses a YANG to CBOR mapping [I-D.ietf-core-yang-cbor] and numeric identifiers [I-D.ietf-core-sid] to minimize CBOR payloads and URI length.

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].

Readers of this specification should be familiar with all the terms and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

The following terms are defined in the NETCONF protocol [RFC6241]: client, configuration data, datastore, and server.

The following terms are defined in the YANG data modelling language [RFC7950]: anydata, anyxml, container, data node, key, key leaf, leaf, leaf-list, and list.

The following terms are defined in RESTCONF protocol [I-D.ietf-netconf-restconf]: data resource, datastore resource, edit operation, query parameter, and target resource.

The following terms are defined in this document:

data node instance: An instance of a data node specified in a YANG module present in the server. The instance is stored in the memory of the server.

Notification instance: An instance of a schema node of type notification, specified in a YANG module present in the server. The instance is generated in the server at the occurrence of the corresponding event and appended to a stream.

YANG schema item identifier: Numeric identifier which replaces the name identifying a YANG item (see section 6.2 of [RFC7950]) (anydata, anyxml, data node, RPC, Action, Notification, Identity, Module name, Submodule name, Feature).

list instance identifier: Handle used to identify a YANG data node that is an instance of a YANG "list" specified with the values of the key leaves of the list.

single instance identifier: Handle used to identify a specific data node which can be instantiated only once. This includes data nodes defined at the root of a YANG module or submodule and data

nodes defined within a container. This excludes data nodes defined within a list or any children of these data nodes.

instance identifier: List instance identifier or single instance identifier.

data node value: Value assigned to a data node instance. Data node values are encoded based on the rules defined in section 4 of [I-D.ietf-core-yang-cbor].

set of data node instances: Represents the payload of CoAP methods when a collection is sent or returned. There are two possibilities, dependent on Request context:

1. CBOR array of pair(s) <instance identifier, data node value >

2. CBOR map of pair(s) <instance identifier, data node value >

TODO: Reduce to one, if possible

The following list contains the abbreviations used in this document.

SID: YANG Schema Item iDentifier.

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for the reading and modifying the content of a datastore used for the management of the instrumented node.

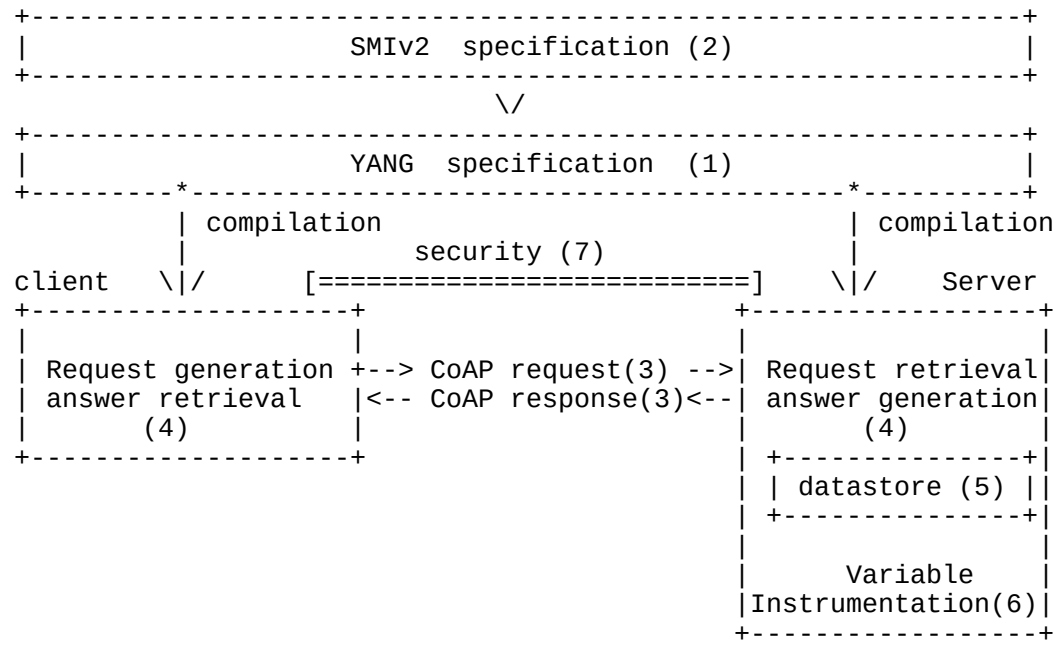


Figure 1: Abstract CoMI architecture

Figure 1 is a high level representation of the main elements of the CoAP management architecture. A client sends requests as payload in packets over the network to a managed constrained node.

The different numbered components of Figure 1 are discussed according to component number.

(1) YANG specification: contains a set of named and versioned modules.

(2) SMIv2 specification: A named module specifies a set of variables and "conceptual tables". There is an algorithm to translate SMIv2 specifications to YANG specifications.

(3) CoMI messages: The CoMI client sends request messages to and receives response messages from the CoMI server.

(4) Retrieval, generation: The server and client parse the CoMI request/response and identify the corresponding instances in the datastore based on YANG specification.

(5) Datastore: The store is composed of two parts: Operational state and Configuration datastore. Datastore also supports RPCs and event streams.

(6) Variable instrumentation: This code depends on implementation of drivers and other node specific aspects.

(7) Security: The server MUST prevent unauthorized users from reading or writing any data resources. CoMI relies on security protocols such as DTLS [RFC6347] to secure CoAP communication.

2.1. Major differences between RESTCONF and CoMI

CoMI uses CoAP/UDP as transport protocol and CBOR as payload format [I-D.ietf-core-yang-cbor]. RESTCONF uses HTTP/TCP as transport protocol and JSON [RFC7159] or XML [XML] as payload formats. CoMI encodes YANG identifier strings as numbers, where RESTCONF does not.

CoMI uses the methods FETCH and iPATCH, not used by RESTCONF. RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not used by CoMI.

CoMI servers cannot change the order of user-ordered data. CoMI does not support insert-mode (first, last, before, after) and insertion-point (before, after) which are supported by RESTCONF. Many CoAP servers will not support date and time functions. For that reason CoMI does not support the start, stop options for events.

CoMI servers only implement the efficient "trim" mode for default values.

The CoMI servers do not support the following RESTCONF functionality:

- The "fields" query parameter to query multiple instances.
- The ’filter’ query that involves XML parsing, ’content’, and ’depth’, query parameters.

2.2. Compression of YANG identifiers

In the YANG specification items are identified with a name string. In order to significantly reduce the size of identifiers used in CoMI, numeric object identifiers are used instead of these strings. The specific encoding of the object identifiers is not hard-wired in the protocol.

Examples of object identifier encoding formats are described in [I-D.ietf-core-sid].

3. Example syntax

This section presents the notation used for the examples. The YANG specifications that are used throughout this document are shown in Appendix A. The example specifications are taken over from existing modules and annotated with SIDs. The values of the SIDs are taken over from [yang-cbor].

CBOR is used to encode CoMI request- and response- payloads. The CBOR syntax of the YANG payloads is specified in [RFC7049]. The payload examples are notated in Diagnostic notation (defined in section 6 of [RFC7049]) that can be automatically converted to CBOR.

A YANG (item identifier, item value) pair is mapped to a CBOR (key, value) pair. The YANG item value is encoded as specified in [I-D.ietf-core-yang-cbor]. The YANG item identifier can be a SID (single node identifier) or a CBOR array with the structure [SID, key1, key2] (list node identifier), where SID is a list identifier and the key values specify the list instance. The YANG item value can be any CBOR major type.

Delta encoding is used for the SIDs. The notation +n is used when the SID has the value PREC+n where PREC is the SID of the parent container, or PREC is the SID of the preceding entity in a CBOR array.

In all examples the resource path in the URI is expressed as a SID, represented as a base64 number. SIDs in the payload are represented as decimal numbers.

4. CoAP Interface

In CoAP a group of links can constitute a Function Set.

TODO: what will happen to term Function Set ?

The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Function Set. CoMI end-points that implement the CoMI management protocol support at least one discoverable management resource of resource type (rt): core.c, with path: /c, where c is short-hand for CoMI. The path root /c is recommended but not compulsory (see Section 8).

The path prefix /c has resources accessible with the following three paths:

/c: YANG-based data with path "/c" and using CBOR content encoding format. This path represents a datastore resource which contains

YANG data resources as its descendant nodes. The data nodes are identified with their SID with format /c/SID.

/c/mod.uri: URI identifying the location of the server module information, with path "/c/mod.uri" and CBOR content format. This YANG data is encoded with plain identifier strings, not YANG encoded values. An Entity Tag MUST be maintained for this resource by the server, which MUST be changed to a new value when the set of YANG modules in use by the server changes.

/c/s: String identifying the default stream resource to which YANG notification instances are appended. Notification support is optional, so this resource will not exist if the server does not support any notifications.

The mapping of YANG data node instances to CoMI resources is as follows: A YANG module describes a set of data trees composed of YANG data nodes. Every data node of the YANG modules loaded in the CoMI server represents a resource of the datastore container (e.g. /c/<sid>

When multiple instances of a list node exist, instance selection is possible as described in Section 5.2.4 and Section 5.2.3.1.

TODO; reference to fetch and patch content formats.

The profile of the management function set, with IF=core.c, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

name	path	rt	Data Type
Management	/c	core.c	n/a
Data	/c	core.c.data	application/cbor
Module Set URI	/c/mod.uri	core.c.moduri	application/cbor
Events	/c/s	core.c.stream	application/cbor

5. /c Function Set

The /c Function Set provides a CoAP interface to manage YANG servers.

The methods used by CoMI are:

Operation	Description
GET	Retrieve the datastore resource or a data resource
FETCH	Retrieve (partial) data resource(s)
POST	Create a data resource, invoke RPC
PUT	Create or replace a data resource
iPATCH	Idem-potently create, replace, and delete data resource(s) (partially)
DELETE	Delete a data resource

There is one query parameters for the GET, PUT, POST, and DELETE methods.

Query Parameter	Description
k	Select an instance of a list node

This parameter is not used for FETCH and iPATCH, because their request payloads support list instance selection.

5.1. Using the 'k' query parameter

The "k" (key) parameter specifies the instance of a list node. The SID in the URI is followed by the (?k=key1, key2,..). Where SID identifies a list node, and key1, key2 are the values of the key leaves that specify an instance of the list.

Key values are encoded using the rules defined in the following table:

YANG datatype	Binary representation	Text representation
uint8,uint16,unit32, uint64	CBOR unsigned integer	int_to_text(number)
int8, int16,int32, int64	CBOR negative integer	base64 (CBOR representation)
decimal64	CBOR decimal fractions	base64 (CBOR representation
string	CBOR text or string	text
boolean	CBOR false or true	"0" or "1"
enumeration	CBOR unsigned integer	int_to_text (number)
bits	CBOR byte string	base64 (CBOR representation)
binary	CBOR byte string	base64 (binary value)
identityref	CBOR unsigned integer	int_to_text (number)
union		base64 (CBOR representation)
List instance identifier	CBOR unsigned integer	base64 (CBOR representation)
List instance identifier	CBOR array	Base64 (CBOR representation)

5.2. Data Retrieval

One or more data node instances can be retrieved by the client. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252] and to the FETCH method defined in section 2 of [I-D.ietf-core-etch].

It is possible that the size of the payload is too large to fit in a single message. In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [RFC7959] is used, as explained in more detail in Section 7.

CoMI uses the FETCH payload for retrieving a subset of the datastore.

There are two additional query parameters for the GET and FETCH methods.

Query Parameter	Description
c	Control selection of configuration and non-configuration data nodes (GET and FETCH)
d	Control retrieval of default values.

5.2.1. Using the 'c' query parameter

The 'c' (content) parameter controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

Value	Description
c	Return only configuration descendant data nodes
n	Return only non-configuration descendant data nodes
a	Return all descendant data nodes

This parameter is only allowed for GET and FETCH methods on datastore and data resources. A 4.00 Bad Request error is returned if used for other methods or resource types.

If this query parameter is not present, the default value is "a".

5.2.2. Using the 'd' query parameter

The "d" (with-defaults) parameter controls how the default values of the descendant nodes of the requested data nodes will be processed.

The allowed values are:

Value	Description
a	All data nodes are reported| Defined as 'report-all' in section 3.1 of [RFC6243].
t	Data nodes set to the YANG default are not reported. Defined as 'trim' in section 3.2 of [RFC6243].

If the target of a GET or FETCH method is a data node that represents a leaf that has a default value, and the leaf has not been given a value yet, the server MUST return the leaf.

If the target of a GET method is a data node that represents a container or list that has any child resources with default values, for the child resources that have not been given value yet, the server MUST not return the child resource if this query parameter is set to 't' and MUST return the child resource if this query parameter is set to 'a'.

If this query parameter is not present, the default value is 't'.

5.2.3. GET

A request to read the values of a data node instance is sent with a confirmable CoAP GET message. A single instance identifier is specified in the URI path prefixed with /c.

FORMAT:

```
GET /c/<instance identifier>

2.05 Content (Content-Format: application/cbor)
<data node value>
```

The returned payload is composed of all the children associated with the specified data node instance.

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.2.3.1. GET Examples

Using for example the current-datetime leaf from Appendix A.1, a request is sent to retrieve the value of system-state/clock/current-datetime specified in container system-state. The ID of system-

state/clock/current-datetime is 1719, encoded in base64 this yields a3. The answer to the request returns a <value>, transported as a single CBOR string item.

```
REQ: GET example.com/c/a3

RES: 2.05 Content (Content-Format: application/cbor)
"2014-10-26T12:16:31Z"
```

For example, the GET of the clock node (ID = 1717; base64: a1), sent by the client, results in the following returned value sent by the server, transported as a CBOR map containing 2 pairs:

```
REQ: GET example.com/c/a1

RES: 2.05 Content (Content-Format: application/cbor)
{
      +2 : "2014-10-26T12:16:51Z",   / ID 1719 /
      +1 : "2014-10-21T03:00:00Z"    / ID 1718 /
}
```

A "list" node can have multiple instances. Accordingly, the returned payload of GET is composed of all the instances associated with the selected list node.

For example, look at the example in Appendix A.3. The GET of the /interfaces/interface/ (with identifier 1533, base64: X9) results in the following returned payload, transported as a CBOR array with 2 elements.

```
REQ: GET example.com/c/X9

RES: 2.05 Content (Content-Format: application/cbor)
[
 {+4 : "eth0",                / name  (ID 1537) /
  +1 : "Ethernet adaptor",    / description (ID 1534) /
  +5 : 1179,                  / type, (ID 1538) identity /
                              / ethernetCsmacd (ID 1179) /
  +2 : true                   / enabled ( ID 1535) /
 },
 {+4 : "eth1",                / name (ID 1537) /
  +1 : "Ethernet adaptor",    / description (ID 1534) /
  +5 : 1179,                  / type, (ID 1538) identity /
                              / ethernetCsmacd (ID 1179) /
  +2 : false                  / enabled /
  }
]
```

It is equally possible to select a leaf of one instance of a list or a complete instance container with GET. The instance identifier is the numeric identifier of the list followed by the specification of the values for the key leaves that uniquely identify the list instance. The instance identifier looks like: SID?k=key-value. The key of "interface" is the "name" leaf. The example below requests the description leaf of the instance with name="eth0" (ID=1534, base64: X-). The value of the description leaf is returned.

```
REQ: GET example.com/c/X-?k="eth0"

RES: 2.05 Content (Content-Format: application/cbor)
"Ethernet adaptor"
```

5.2.4. FETCH

The FETCH is used to retrieve a list of data node values. The FETCH Request payload contains a CBOR list of instance identifiers.

```
FORMAT:
    FETCH /c/ Content-Format (application/YANG-fetch+cbor)
    <CBOR array of instance identifiers>

    2.05 Content (Content-Format: application/YANG-patch+cbor)
    <CBOR array of data node values>
```

The instance identifier is a SID or a CBOR array containing the SID followed by key values that identify the list instance (sec 5.13.1 of [I-D.ietf-core-yang-cbor]. In the payload of the returned data node

values, delta encoding is used as described in [I-D.ietf-core-yang-cbor].

5.2.4.1. FETCH examples

The example uses the current-datetime leaf and the interface list from Appendix A.1. In the following example the value of current-datetime (ID 1719)and the interface list (ID 1533) instance identified with name="eth0" are queried.

```
REQ:  FETCH /c Content-Format (application/YANG-fetch+cbor)
      [ 1719,            / ID 1719 /
      [-186, "eth0"]   / ID 1533 with name = "eth0" /
      ]

RES:  2.05 Content Content-Format (application/YANG-patch+cbor)
[
  "2014-10-26T12:16:31Z",
 {
   +4 : "eth0",                / name (ID 1537) /
   +1 : "Ethernet adaptor",    / description (ID 1534) /
   +5 : 1179,                  / type (ID 1538), identity /
                               / ethernetCsmacd (ID 1179) /
   +2 : true                   / enabled (ID 1535) /
  }
]
```

TODO: align with future FETCH content format.

5.3. Data Editing

CoMI allows datastore contents to be created, modified and deleted using CoAP methods.

5.3.1. Data Ordering

A CoMI server SHOULD preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as arrays so messages will preserve their order.

5.3.2. POST

Data resources are created with the POST method. The CoAP POST operation is used in CoMI for creation of data resources and the invocation of "ACTION" and "RPC" resources. Refer to Section 5.6 for details on "ACTION" and "RPC" resources.

A request to create the values of an instance of a container or leaf is sent with a confirmable CoAP POST message. A single SID is specified in the URI path prefixed with /c.

FORMAT:

```
POST /c/<instance identifier> Content-Format(application/cbor)
<data node value>

2.01 Created (Content-Format: application/cbor)
```

If the data resource already exists, then the POST request MUST fail and a "4.09 Conflict" status-line MUST be returned

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.2.1. Post example

The example uses the interface list from Appendix A.1. Example is creating a new version of the container interface (ID = 1533):

```
REQ: POST /c/X9 Content-Format(application/cbor)
      {
        +4 : "eth0",              / name (ID 1537) /
        +1 : "Ethernet adaptor", / description (ID 1534) /
        +5 : 1179,                / type (ID 1538), identity /
                                  / ethernetCsmacd (ID 1179) /
        +2 : true                 / enabled (ID 1535) /
      }

RES: 2.01 Created (Content-Format: application/cbor)
```

5.3.3. PUT

Data resource instances are created or replaced with the PUT method. The PUT operation is supported in CoMI. A request to set the value of a data node instance is sent with a confirmable CoAP PUT message.

FORMAT:

```
PUT /c/<instance identifier> Content-Format(application/cbor)
<data node value>

2.01 Created
```

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.3.1. PUT example

The example uses the interface list from Appendix A.1. Example is renewing an instance of the list interface (ID = 1533) with key name="eth0":

```
REQ:  PUT /c/X9?k="eth0" Content-Format(application/cbor)
      {
        +4 : "eth0",             / name (ID 1537) /
        +1 : "Ethernet adaptor", / description (ID 1534) /
        +5 : 1179,               / type (ID 1538), identity /
                                 / ethernetCsmacd ( ID 1179) /
        +2 : true                / enabled (ID 1535) /
      }
RES:  2.04 Changed
```

5.3.4. iPATCH

One or multiple data resource instances are replaced with the idempotent iPATCH method [I-D.ietf-core-etch]. A request is sent with a confirmable CoAP iPATCH message.

There are no query parameters for the iPATCH method.

The processing of the iPATCH command is specified by the CBOR payload. The CBOR patch payload describes the changes to be made to target YANG data nodes [I-D.bormann-appsawg-cbor-merge-patch]. If the CBOR patch payload contains data node instances that are not present in the target, these instances are added or silently ignored dependent of the payload information. If the target contains the specified instance, the contents of the instances are replaced with the values of the payload. Null values indicate the removal of existing values.

FORMAT:

```
iPATCH /c Content-Format(application/YANG-patch+cbor)
<set of data node instances>

2.04 Changed
```

5.3.4.1. iPATCH example

The example uses the interface list from Appendix A.3, and the timezone-utc-offset leaf from Appendix A.1. In the example one leaf (timezone-utc-offset) and one container (interface) instance are changed.

```
REQ: iPATCH /c Content-Format(application/YANG-patch+cbor)
[
  [1533, "eth0"] ,                / interface (ID = 1533) /
    {
      +4 : "eth0",                / name (ID 1537) /
      +1 : "Ethernet adaptor",    / description (ID 1534) /
      +5 : 1179,                  / type (ID 1538), identity /
                                  / ethernetCsmacd (ID 1179) /
      +2 : true                   / enabled (ID 1535) /
    },
  +203 , 60          / timezone-utc-offset (delta = 1736-1533) /
]
```

RES: 2.04 Changed

TODO: Align with future cbor-merge-patch content format.

5.3.5. DELETE

Data resource instances are deleted with the DELETE method. The RESTCONF DELETE operation is supported in CoMI.

FORMAT:

```
    Delete /c/<instance identifier>

    2.02 Deleted
```

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.5.1. DELETE example

The example uses the interface list from Appendix A.3. Example is deleting an instance of the container interface (ID = 1533):

REQ: DELETE /c/X9?k="eth0"

RES: 2.02 Deleted

5.4. Full Data Store access

The methods GET, PUT, POST, and DELETE can be used to return, replace, create, and delete the whole data store respectively.

```
FORMAT:
    GET /c
    2.05 Content (Content-Format: application/cbor)
    <array of data node instances>

   PUT /c
   (Content-Format: application/cbor)
    <array of data node instances>
   2.04 Changed

   POST /c
   (Content-Format: application/cbor)
    <array of data node instances>
   2.01 Created

   DELETE /c
   2.02 Deleted
```

The array of data node instances represents an array of all root nodes in the data store after the PUT, POST and GET method invocations.

5.4.1. Full Data Store examples

The example uses the interface list and the clock container from Appendix A.3. Assume that the data store contains two root objects: the list interface (ID 1533) with one instance and the container Clock (ID 1717). After invocation of GET an array with these two objects is returned:

```
RQ:  GET /c
RES: 2.05 Content Content-Format (application/YANG-patch+cbor)
[
  {1717:
     { +2: "2016-10-26T12:16:31Z", / current-datetime (ID 1719) /
       +1: "2014-10-05T09:00:00Z"  / boot-datetime (ID 1718) /
     },
  -186:                           / clock (ID 1533) /
     {
       +4 : "eth0",               / name (ID 1537) /
       +1 : "Ethernet adaptor",   / description (ID 1534) /
       +5 : 1179,                 / type (ID 1538), identity: /
                                  / ethernetCsmacd (ID 1179) /
       +2 : true                  / enabled (ID 1535) /
      }
    }
]
```

5.5. Notify functions

Notification by the server to a selection of clients when an event occurs in the server is an essential function for the management of servers. CoMI allows events specified in YANG [RFC5277] to be notified to a selection of requesting clients. The server appends newly generated events to a stream. There is one, so-called "default", stream in a CoMI server. The /c/s resource identifies the default stream. The server MAY create additional stream resources. When a CoMI server generates an internal event, it is appended to the chosen stream, and the content of a notification instance is ready to be sent to all CoMI clients which observe the chosen stream resource.

Reception of generated notification instances is enabled with the CoAP Observe [RFC7641] function. The client subscribes to the notifications by sending a GET request with an "Observe" option, specifying the /c/s resource when the default stream is selected.

Every time an event is generated, the chosen stream is cleared, and the generated notification instance is appended to the chosen stream(s). After appending the instance, the contents of the instance is sent to all clients observing the modified stream.

```
FORMAT:
  Get /<stream-resource>
      Content-Format(application/YANG-patch+cbor) Observe(0)

2.05 Content Content-Format(application/YANG-patch+cbor)
<set of data node instances>
```

5.5.1. Notify Examples

Suppose the server generates the event specified in Appendix A.4. By executing a GET on the /c/s resource the client receives the following response:

```
REQ:  GET /c/s Observe(0) Token(0x93)

RES:  2.05 Content Content-Format(application/YANG-patch+cbor)
                        Observe(12) Token(0x93)
{
   60010 :                  / example-port-fault (ID 60010) /
    {
      +1 : "0/4/21",       / port-name (ID 60011) /
      +2 : "Open pin 2"    / port-fault (ID 60012) /
    },
   60010 :                  / example-port-fault (ID 60010) /
    {
      +1 : "1/4/21",       / port-name (ID 60011) /
      +2 : "Open pin 5"    / port-fault (ID 60012) /
    }
}
```

In the example, the request returns a success response with the contents of the last two generated events. Consecutively the server will regularly notify the client when a new event is generated.

To check that the client is still alive, the server MUST send confirmable notifications once in a while. When the client does not confirm the notification from the server, the server will remove the client from the list of observers [RFC7641].

5.6. RPC statements

The YANG "action" and "RPC" statements specify the execution of a Remote procedure Call (RPC) in the server. It is invoked using a POST method to an "Action" or "RPC" resource instance. The Request payload contains the values assigned to the input container when specified with the action station. The Response payload contains the values of the output container when specified.

The returned success response code is 2.05 Content.

```
FORMAT:
 POST /c/<instance identifier>
           Content-Format(application/YANG-patch+cbor)
<input node value>

2.05 Content Content-Format (application/YANG-patch+cbor)
<output node value>
```

There "k" query parameter is allowed for the POST method when used for an action invocation.

5.6.1. RPC Example

The example is based on the YANG action specification of Appendix A.2. A server list is specified and the action "reset" (ID 60002, base64: Opq), that is part of a "server instance" with key value "myserver", is invoked.

```
REQ:  POST /c/Opq?k="myserver"
              Content-Format(application/YANG-patch+cbor)
{
  +1 : "2016-02-08T14:10:08Z09:00" / reset-at (ID 60003) /
}

RES:  2.05 Content Content-Format(application/YANG-patch+cbor)
{
  +2 : "2016-02-08T14:10:08Z09:18" / reset-finished-at (ID 60004)/
}
```

6. Access to MIB Data

Appendix A.5 shows a YANG module mapped from the SMI specification "IP-MIB" [RFC4293]. The following example shows the "ipNetToPhysicalEntry" list with 2 instances, using diagnostic notation without delta encoding.

```
{
   60021 :                    / list ipNetToPhysicalEntry /
   [
     {
       60022 : 1,             / ipNetToPhysicalIfIndex /
       60023 : 1,             / ipNetToPhysicalNetAddressType: ipv4 /
       60024 : h'0A000033',   / ipNetToPhysicalNetAddress /
       60025 : h'00000A01172D',/ ipNetToPhysicalPhysAddress /
       60026 : 2333943,       / ipNetToPhysicalLastUpdated /
       60027 : 4,             / ipNetToPhysicalType: static /
       60028 : 1,             / ipNetToPhysicalState: reachable /
       60029 : 1              / ipNetToPhysicalRowStatus: active /
     },
     {
       60022 : 1,             / ipNetToPhysicalIfIndex /
       60023 : 1,             / ipNetToPhysicalNetAddressType: ipv4 /
       60024 : h'09020304',   / ipNetToPhysicalNetAddress  /
       60025 : h'00000A36200A',/ ipNetToPhysicalPhysAddress /
       60026 : 2329836,       / ipNetToPhysicalLastUpdated /
       60027 : 3,             / ipNetToPhysicalType: dynamic /
       60028 : 6,             / ipNetToPhysicalState: unknown /
       60029 : 1              / ipNetToPhysicalRowStatus: active /
     }
   ]
}
```

The IPv4 addresses A.0.0.33 and 9.2.3.4 are encoded in CBOR as h'0A000033' and h'09020304' respectively. In the following example exactly one instance is requested from the ipNetToPhysicalEntry (ID 60021, base64: Oz1). The h'09020304' value is encoded in base64 as AJAgME.

In this example one instance of /ip/ipNetToPhysicalEntry that matches the keys ipNetToPhysicalIfIndex = 1, ipNetToPhysicalNetAddressType = ipv4 and ipNetToPhysicalNetAddress = 9.2.3.4 (h'09020304', base64: AJAgME).

```
REQ: GET example.com/c/Oz1?k="1,1,AJAgME"

RES: 2.05 Content (Content-Format: application/YANG-patch+cbor)
{
   +1 : 1,                  / ( SID 60022 ) /
   +2 : 1,                  / ( SID 60023 ) /
   +3 : h'09020304',        / ( SID 60024 ) /
   +4 : h'00000A36200A',    / ( SID 60025 ) /
   +5 : 2329836,            / ( SID 60026 ) /
   +6 : 3,                  / ( SID 60027 ) /
   +7 : 6,                  / ( SID 60028 ) /
   +8 : 1                   / ( SID 60029 ) /
}
```

7. Use of Block

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of text need to be transported the datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block option [RFC7959] allows the transport of the total payload in individual blocks of which the size can be adapted to the underlying transport sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

Beware of race conditions. Blocks are filled one at a time and care should be taken that the whole data representation is sent in multiple blocks sequentially without interruption. In the server, values are changed, lists are re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the variables, the transported results do not correspond with a state having occurred in the server; or worse the returned values are inconsistent. For example: array length does not correspond with actual number of items. It may be advisable to use CBOR maps or CBOR arrays of undefined length which are foreseen for data streaming purposes.

8. Resource Discovery

The presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c" [RFC6690].

Upon success, the return payload will contain the root resource of the management data. It is up to the implementation to choose its root resource, but it is recommended that the value "/c" is used, where possible. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.c

RES: 2.05 Content </c>; rt="core.c"
```

Management objects MAY be discovered with the standard CoAP resource discovery. The implementation can add the encoded values of the object identifiers to /.well-known/core with rt="core.c.data". The available objects identified by the encoded values can be discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.data". Upon success, the return payload will contain the registered encoded values and their location. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.c.data

RES: 2.05 Content </c/BaAiN>; rt="core.c.data",
</c/CF_fA>; rt="core.c.data"
```

Lists of encoded values may become prohibitively long. It is discouraged to provide long lists of objects on discovery. Therefore, it is recommended that details about management objects are discovered by reading the YANG module information stored in for example the "ietf-comi-yang-library" module [I-D.veillette-core-cool-library]. The resource "/c/mod.uri" is used to retrieve the location of the YANG module library.

The module list can be stored locally on each server, or remotely on a different server. The latter is advised when the deployment of many servers are identical.

```
Local in example.com server:

REQ: GET example.com/c/mod.uri

RES: 2.05 Content (Content-Format: application/cbor)
{
  "mod.uri" : "example.com/c/modules"
}


Remote in example-remote-server:

REQ: GET example.com/c/mod.uri

RES: 2.05 Content (Content-Format: application/cbor)
{
  "moduri" : "example-remote-server.com/c/group17/modules"
}
```

Within the YANG module library all information about the module is stored such as: module identifier, identifier hierarchy, grouping, features and revision numbers.

9. Error Handling

In case a request is received which cannot be processed properly, the CoMI server MUST return an error message. This error message MUST contain a CoAP 4.xx or 5.xx response code, and SHOULD include additional information in the payload.

Such an error message payload is a text string, using the following structure:

```
CoMI error: xxxx "error text"
```

The characters xxxx represent one of the values from the table below, and the OPTIONAL "error text" field contains a human readable explanation of the error.

CoMI Error Code	CoAP Error Code	Description
0	4.xx	General error
1	4.13	Request too big
2	4.00	Response too big
3	4.00	Unknown identifier
4	4.00	Invalid value
5	4.05	Attempt to write read-only variable
6	5.01	No access
7	4.00	Wrong type
8	4.15	Unknown encoding
9	4.0	Wrong value
10	4.0	Not created
11	4.04	Resource unavailable
12	4.01	Authorization error
13	4.0	Bad attribute
14	4.0	Unknown attribute
15	4.0	Missing attribute

The CoMI error codes are motivated by the error-status values defined in [RFC3416], and the error tags defined in [I-D.ietf-netconf-restconf].

10. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. This requires integrity protection of both requests and responses, and depending on the application encryption.

CoMI re-uses the security mechanisms already available to CoAP as much as possible. This includes DTLS [RFC6347] for protected access to resources, as well suitable authentication and authorization mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorization may need to be selected.

CoMI avoids defining new security mechanisms as much as possible. However some adaptations may still be required, to cater for CoMI's specific requirements.

11. IANA Considerations

'rt="core.c"' needs registration with IANA.

'rt="core.c.data"' needs registration with IANA.

'rt="core.c.moduri"' needs registration with IANA.

'rt="core.c.stream"' needs registration with IANA.

Content types to be registered:

- application/YANG-patch+cbor
- application/YANG-fetch+cbor

12. Acknowledgements

We are very grateful to Bert Greevenbosch who was one of the original authors of the CoMI specification and specified CBOR encoding and use of hashes.

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR.

Timothy Carey has provided the text for Appendix B.

The draft has benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Juergen

Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

13. Changelog

Copy of vanderstok-core-comi-11.

14. References

14.1. 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>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <http://www.rfc-editor.org/info/rfc5277>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <http://www.rfc-editor.org/info/rfc6243>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[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>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

[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>.

[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>.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-18 (work in progress), October 2016.

[I-D.ietf-core-etch]
Stok, P., Bormann, C., and A. Sehgal, "Patch and Fetch Methods for Constrained Application Protocol (CoAP)", draft-ietf-core-etch-04 (work in progress), November 2016.

[I-D.bormann-appsawg-cbor-merge-patch]
Bormann, C. and P. Stok, "CBOR Merge Patch", draft-bormann-appsawg-cbor-merge-patch-00 (work in progress), March 2016.

[I-D.ietf-core-yang-cbor]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-03 (work in progress), October 2016.

14.2. Informative References

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <http://www.rfc-editor.org/info/rfc2578>.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <http://www.rfc-editor.org/info/rfc3410>.

[RFC3416] Presuhn, R., Ed., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, DOI 10.17487/RFC3416, December 2002, <http://www.rfc-editor.org/info/rfc3416>.

[RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <http://www.rfc-editor.org/info/rfc4293>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <http://www.rfc-editor.org/info/rfc6643>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

[I-D.ietf-core-interfaces] Shelby, Z., Vial, M., Koster, M., and C. Groves, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-07 (work in progress), December 2016.

[I-D.ietf-core-sid] Somaraju, A., Veillette, M., Pelov, A., Turner, R., and A. Minaburo, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-00 (work in progress), October 2016.

[I-D.veillette-core-cool] Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "Constrained Objects Language", draft-veillette-core-cool-02 (work in progress), July 2016.

[I-D.veillette-core-cool-library] Veillette, M., "Constrained YANG Module Library", draft-veillette-core-cool-library-00 (work in progress), August 2016.

[XML] "Extensible Markup Language (XML)", Web http://www.w3.org/xml.

[OMA] "OMA-TS-LightweightM2M-V1_0-20131210-C", Web http://technical.openmobilealliance.org/Technical/current_releases.aspx.

[OMNA] "Open Mobile Naming Authority (OMNA)", Web http://http://technical.openmobilealliance.org/Technical/technical-information/omna.

[netconfcentral] "NETCONF Central: library of YANG modules", Web http://www.netconfcentral.org/modulelist.

[mibreg] "Structure of Management Information (SMI) Numbers (MIB Module Registrations)", Web http://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml/.

[yang-cbor] "yang-cbor Registry", Web https://github.com/core-wg/yang-cbor/tree/master/registry/.

Appendix A. YANG example specifications

This appendix shows 5 YANG example specifications taken over from as many existing YANG modules. The YANG modules are available from [netconfcentral]. Each YANG item identifier is accompanied by its SID shown after the "//" comment sign, taken from [yang-cbor].

A.1. ietf-system

Excerpt of the YANG module ietf-system [RFC7317].

```
module ietf-system {
  container system {                    // SID 1715
    container clock {                   // SID 1734
      choice timezone {
        case timezone-name {
          leaf timezone-name {          // SID 1735
            type timezone-name;
          }
        }
        case timezone-utc-offset {
          leaf timezone-utc-offset {    // SID 1736
            type int16 {
            }
          }
        }
      }
```

```
      }
      container ntp {                    // SID 1750
        leaf enabled {                   // SID 1751
          type boolean;
          default true;
        }
        list server {                    // SID 1752
          key name;
          leaf name {                    // SID 1755
            type string;
          }
          choice transport {
            case udp {
              container udp {            // SID 1757
                leaf address {           // SID 1758
                  type inet:host;
                }
                leaf port {              // SID 1759
                  type inet:port-number;
                }
              }
            }
          }
          leaf association-type {        // SID 1753
            type enumeration {
              enum server {
              }
              enum peer {
              }
              enum pool {
              }
            }
          }
          leaf iburst {                  // SID 1754
            type boolean;
          }
          leaf prefer {                  // SID 1756
            type boolean;
            default false;
          }
        }
      }
    container system-state {             // SID 1716
      container clock {                  // SID 1717
        leaf current-datetime {          // SID 1719
          type yang:date-and-time;
        }
        leaf boot-datetime {             // SID 1718
```

```
          type yang:date-and-time;
        }
      }
    }
  }
```

A.2. server list

Taken over from [RFC7950] section 7.15.3.

```
  module example-server-farm {
    yang-version 1.1;
    namespace "urn:example:server-farm";
    prefix "sfarm";

    import ietf-yang-types {
      prefix "yang";
    }

    list server {                        // SID 60000
      key name;
      leaf name {                        // SID 60001
        type string;
      }
      action reset {                     // SID 60002
        input {
          leaf reset-at {                // SID 60003
            type yang:date-and-time;
            mandatory true;
          }
        }
        output {
          leaf reset-finished-at {       // SID 60004
            type yang:date-and-time;
            mandatory true;
          }
        }
      }
    }
  }
```

A.3. interfaces

Excerpt of the YANG module ietf-interfaces [RFC7223].

```
   module ietf-interfaces {
     container interfaces {                // SID 1505
       list interface {                    // SID 1533
         key "name";
         leaf name {                       // SID 1537
           type string;
         }
         leaf description {                // SID 1534
           type string;
         }
         leaf type {                       // SID 1538
           type identityref {
             base interface-type;
           }
           mandatory true;
         }

         leaf enabled {                    // SID 1535
           type boolean;
           default "true";
         }

         leaf link-up-down-trap-enable {   // SID 1536
           if-feature if-mib;
           type enumeration {
             enum enabled {
               value 1;
             }
             enum disabled {
               value 2;
             }
           }
         }
       }
     }
   }
```

A.4. Example-port

Notification example defined within this document.

```
   module example-port {
           ...
           notification example-port-fault {   // SID 60010
             description
               "Event generated if a hardware fault on a
                line card port is detected";
             leaf port-name {                  // SID 60011
               type string;
               description "Port name";
             }
             leaf port-fault {                 // SID 60012
               type string;
               description "Error condition detected";
             }
           }
         }
```

A.5. IP-MIB

The YANG translation of the SMI specifying the IP-MIB [RFC4293], extended with example SID numbers, yields:

```
module IP-MIB {
  import IF-MIB {
    prefix if-mib;
  }
  import INET-ADDRESS-MIB {
    prefix inet-address;
  }
  import SNMPv2-TC {
    prefix smiv2;
  }
  import ietf-inet-types {
    prefix inet;
  }
  import yang-smi {
    prefix smi;
  }
  import ietf-yang-types {
    prefix yang;
  }

  container ip {                              // SID 60020
    list ipNetToPhysicalEntry {               // SID 60021
      key "ipNetToPhysicalIfIndex ipNetToPhysicalNetAddressType ipNetToPhysicalN
etAddress";
      leaf ipNetToPhysicalIfIndex {           // SID 60022
        type if-mib:InterfaceIndex;
```

```
      }
      leaf ipNetToPhysicalNetAddressType {  // SID 60023
        type inet-address:InetAddressType;
      }
      leaf ipNetToPhysicalNetAddress {      // SID 60024
        type inet-address:InetAddress;
      }
      leaf ipNetToPhysicalPhysAddress {     // SID 60025
        type yang:phys-address {
          length "0..65535";
        }
      }
      leaf ipNetToPhysicalLastUpdated {     // SID 60026
        type yang:timestamp;
      }
      leaf ipNetToPhysicalType {            // SID 60027
        type enumeration {
          enum "other" {
            value 1;
          }
          enum "invalid" {
            value 2;
          }
          enum "dynamic" {
            value 3;
          }
          enum "static" {
            value 4;
          }
          enum "local" {
            value 5;
          }
        }
      }
      leaf ipNetToPhysicalState {           // SID 60028
        type enumeration {
          enum "reachable" {
            value 1;
          }
          enum "stale" {
            value 2;
          }
          enum "delay" {
            value 3;
          }
          enum "probe" {
            value 4;
          }
```

```
          enum "invalid" {
            value 5;
          }
          enum "unknown" {
            value 6;
          }
          enum "incomplete" {
            value 7;
          }
        }
      }
      leaf ipNetToPhysicalRowStatus {       // SID 60029
        type smiv2:RowStatus;
    }  // list ipNetToPhysicalEntry
  }  // container ip
}  // module IP-MIB
```

Appendix B. Comparison with LWM2M

B.1. Introduction

CoMI and LWM2M [OMA], both, provide RESTful device management services over CoAP. Differences between the designs are highlighted in this section.

The intent of the LWM2M protocol is to provide a single protocol to control and manage IoT devices. This means the IoT device implements and uses the same LWM2M agent function for the actuation and sensing features of the IoT device as well as for the management of the IoT device. The intent of CoMI Interface as described in the Abstract section of this document is to provide management of constrained devices and devices in constrained networks using RESTCONF and YANG. This implies that the device, although reusing the CoAP protocol, would need a separate CoAP based agent in the future to control the actuation and sensing features of the device and another CoMI agent that performs the management functions.

It should be noted that the mapping of a LWM2M server to YANG is specified in [YANGlwm2m]. The converted server can be invoked with CoMI as specified in this document.

For the purposes of managing IoT devices the following points related to the protocols compare how management resources are defined, identified, encoded and updated.

B.2. Defining Management Resources

Management resources in LWM2M (LWM2M objects) are defined using a standardized number. When a new management resource is defined, either by a standards organization or a private enterprise, the management resource is registered with the Open Mobile Naming Authority [OMNA] in order to ensure different resource definitions do not use the same identifier. CoMI, by virtue of using YANG as its data modeling language, allows enterprises and standards organizations to define new management resources (YANG nodes) within YANG modules without having to register each individual management resource. Instead YANG modules are scoped within a registered name space. As such, the CoMI approach provides additional flexibility in defining management resources. Likewise, since CoMI utilizes YANG, existing YANG modules can be reused. The flexibility and reuse capabilities afforded to CoMI can be useful in management of devices like routers and switches in constrained networks. However for management of IoT devices, the usefulness of this flexibility and applicability of reuse of existing YANG modules may not be warranted. The reason is that IoT devices typically do not require complex sets of configuration or monitoring operations required by devices like a router or a switch. To date, OMA has defined approximately 15 management resources for constrained and non-constrained mobile or fixed IoT devices while other 3rd Party SDOs have defined another 10 management resources for their use in non-constrained IoT devices. Likewise, the Constrained Object Language [I-D.veillette-core-cool] which is used by CoMI when managing constrained IoT devices uses YANG schema item identifiers, which are registered with IANA, in order to define management resources that are encoded using CBOR when targeting constrained IoT Devices.

B.3. Identifying Management Resources

As LWM2M and CoMI can similarly be used to manage IoT devices, comparison of the CoAP URIs used to identify resources is relevant as the size of the resource URI becomes applicable for IoT devices in constrained networks. LWM2M uses a flat identifier structure to identify management resources and are identified using the LWM2M object’s identifier, instance identifier and optionally resource identifier (for access to and object’s attributes). For example, identifier of a device object (object id = 3) would be "/3/0" and identification of the device object’s manufacturer attribute would be "/3/0/0". Effectively LWM2M identifiers for management resources are between 4 and 10 bytes in length.

CoMI is expected to be used to manage constrained IoT devices. CoMI utilizes the YANG schema item identifier[SID] that identify the resources. CoMI recommends that IoT device expose resources to

identify the data stores and event streams of the CoMI agent. Individual resources (e.g., device object) are not directly identified but are encoded within the payload. As such the identifier of the CoMI resource is smaller (4 to 7 bytes) but the overall payload size isn't smaller as resource identifiers are encoded on the payload.

B.4. Encoding of Management Resources

LWM2M provides a separation of the definition of the management resources from how the payloads are encoded. As of the writing of this document LWM2M encodes LWM2M encodes payload data in Type-length-value (TLV), JSON or plain text formats. JSON encoding is the most common encoding scheme with TLV encoding used on the simplest IoT devices. CoMI's use of CBOR provides a more efficient transfer mechanism [RFC7049] than the current LWM2M encoding formats.

In situations where resources need to be modified, CoMI uses the CoAP PATCH operation resources only require a partial update. LWM2M does not currently use the CoAP PATCH operation but instead uses the CoAP PUT and POST operations which are less efficient.

Authors' Addresses

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

Email: andy@yumaworks.com

Michel Veillette
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

Z. Shelby
ARM
Z. Vial
Schneider-Electric
M. Koster
SmartThings
C. Groves
Huawei

March 13, 2017

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-03

Abstract

For CoAP [RFC7252] Dynamic linking of state updates between resources, either on an endpoint or between endpoints, is defined with the concept of Link Bindings. This specification defines conditional observation attributes that work with Link Bindings or with CoAP Observe [RFC7641].

Editor’s note:

o The git repository for the draft is found at https://github.com/core-wg/dynlink

Table of Contents

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format is a standard for doing Web Linking [RFC5988] in constrained environments.

This specification introduces the concept of a Link Binding, which defines a new link relation type to create a dynamic link between resources over which to exchange state updates. Specifically, a Link Binding is a link for binding the state of 2 resources together such that updates to one are sent over the link to the other. CoRE Link Format representations are used to configure, inspect, and maintain Link Bindings. This specification additionally defines a set of conditional Observe Attributes for use with Link Bindings and with the standalone CoRE Observe [RFC7641] method.

2. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. This specification makes use of the following additional terminology:

Link Binding: A unidirectional logical link between a source resource and a destination resource, over which state information is synchronized.

State Synchronization: Depending on the binding method (Polling, Observe, Push) different REST methods may be used to synchronize the resource values between a source and a destination. The process of using a REST method to achieve this is defined as "State Synchronization". The endpoint triggering the state synchronization is the synchronization initiator.

3. Link Bindings

In a M2M RESTful environment, endpoints may directly exchange the content of their resources to operate the distributed system. For example, a light switch may supply on-off control information that may be sent directly to a light resource for on-off control.

Beforehand, a configuration phase is necessary to determine how the resources of the different endpoints are related to each other. This can be done either automatically using discovery mechanisms or by means of human intervention and a so-called commissioning tool. In this specification the abstract relationship between two resources is called a link Binding. The configuration phase necessitates the exchange of binding information so a format recognized by all CoRE endpoints is essential. This specification defines a format based on the CoRE Link-Format to represent binding information along with the rules to define a binding method which is a specialized relationship between two resources. The purpose of a binding is to synchronize the content between a source resource and a destination resource. The destination resource MAY be a group resource if the authority component of the destination URI contains a group address (either a multicast address or a name that resolves to a multicast address). Since a binding is unidirectional, the binding entry defining a relationship is present only on one endpoint. The binding entry may be located either on the source or the destination endpoint depending on the binding method.

3.1. Binding Methods

A binding method defines the rules to generate the web-transfer exchanges that synchronize state between source and destination resources. By using REST methods content is sent from the source resource to the destination resource.

The following table gives a summary of the binding methods defined in this specification.

Name	Identifier	Location	Method
Polling	poll	Destination	GET
Observe	obs	Destination	GET + Observe
Push	push	Source	PUT

Table 1: Binding Method Summary

The description of a binding method must define the following aspects:

Identifier: This is the value of the "bind" attribute used to identify the method.

Location: This information indicates whether the binding entry is stored on the source or on the destination endpoint.

REST Method: This is the REST method used in the Request/Response exchanges.

Conditions: A binding method definition must state how the condition attributes of the abstract binding definition are actually used in this specialized binding.

The binding methods are described in more detail below.

3.1.1. Polling

The Polling method consists of sending periodic GET requests from the destination endpoint to the source resource and copying the content to the destination resource. The binding entry for this method MUST be stored on the destination endpoint. The destination endpoint MUST ensure that the polling frequency does not exceed the limits defined by the pmin and pmax attributes of the binding entry. The copying process MAY filter out content from the GET requests using value-based conditions (e.g based on the Change Step, Less Than, Greater Than attributes).

3.1.2. Observe

The Observe method creates an observation relationship between the destination endpoint and the source resource. On each notification the content from the source resource is copied to the destination resource. The creation of the observation relationship requires the CoAP Observation mechanism [RFC7641] hence this method is only permitted when the resources are made available over CoAP. The binding entry for this method MUST be stored on the destination endpoint. The binding conditions are mapped as query string parameters (see Section 4.2).

3.1.3. Push

When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource when the binding condition attributes are satisfied for the source resource. The source endpoint MUST only send a notification request if the binding conditions are met. The binding entry for this method MUST be stored on the source endpoint.

3.2. Link Relation

Since Binding involves the creation of a link between two resources, Web Linking and the CoRE Link-Format are a natural way to represent binding information. This involves the creation of a new relation type, named "boundto". In a Web link with this relation type, the target URI contains the location of the source resource and the context URI points to the destination resource.

3.3. Binding Attributes

Web link attributes allow a fine-grained control of the type of state synchronization along with the conditions that trigger an update. This specification defines the attributes below:

Attribute	Parameter	Value
Binding method	bind	xsd:string
Minimum Period (s)	pmin	xsd:integer (>0)
Maximum Period (s)	pmax	xsd:integer (>0)
Change Step	st	xsd:decimal (>0)
Greater Than	gth	xsd:decimal
Less Than	lth	xsd:decimal

Table 2: Binding Attributes Summary

**Editor’s note: Naming of lth and gth to be confirmed at IETF98.

3.3.1. Bind Method (bind)

This is the identifier of a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

3.3.2. Minimum Period (pmin)

When present, the minimum period indicates the minimum time to wait (in seconds) before triggering a new state synchronization (even if it has changed). In the absence of this parameter, the minimum period is up to the synchronization initiator. The minimum period MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.3.3. Maximum Period (pmax)

When present, the maximum period indicates the maximum time in seconds between two consecutive state synchronizations (regardless if it has changed). In the absence of this parameter, the maximum period is up to the synchronization initiator. The maximum period MUST be greater than zero and MUST be greater than the minimum period parameter (if present) otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.3.4. Change Step (st)

When present, the change step indicates how much the value of a resource SHOULD change before triggering a new state synchronization (compared to the value of the previous synchronization). Upon reception of a query including the st attribute the current value (CurrVal) of the resource is set as the initial value (STinit). Once the resource value differs from the STinit value (i.e. CurrVal >= STinit + ST or CurrVal <= STint - ST) then a new state synchronization occurs. STinit is then set to the state synchronization value and new state synchronizations are based on a change step against this value. The change step MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

Note: Due to the state synchronization based update of STint it may result in that resource value received in two sequential state synchronizations differs by more than st.

3.3.5. Greater Than (gth)

When present, Greater Than indicates the upper limit value the resource value SHOULD cross before triggering a new state synchronization. State synchronization only occurs when the resource value exceeds the specified upper limit value. The actual resource value is used for the synchronization rather than the gth value. If the value continues to rise, no new state synchronizations are generated as a result of gth. If the value drops below the upper limit value and then exceeds the upper limit then a new state synchronization is generated.

3.3.6. Less Than (lth)

When present, Less Than indicates the lower limit value the resource value SHOULD cross before triggering a new state synchronization. State synchronization only occurs when the resource value is less than the specified lower limit value. The actual resource value is used for the synchronization rather than the lth value. If the value

continues to fall no new state synchronizations are generated as a result of lth. If the value rises above the lower limit value and then drops below the lower limit then a new state synchronization is generated.

3.3.7. Attribute Interactions

Pmin, pmax, st, gth and lth may be present in the same query.

If pmin and pmax are present in a query then they take precedence over the other parameters. Thus even if st, gth or lth are met, if pmin has not been exceeded then no state synchronization occurs. Likewise if st, gth or lth have not been met and pmax time has expired then state synchronization occurs. The current value of the resource is used for the synchronization. If pmin time is exceeded and st, gth or lth are met then the current value of the resource is synchronized. If st is also included, a state synchronization resulting from pmin or pmax updates STinit with the synchronized value.

If gth and lth are included gth MUST be greater than lth otherwise an error CoAP error code 4.00 "Bad Request" (or equivalent) MUST be returned.

If st is included in a query with a gth or lth attribute then state synchronizations occur only when the conditions described by st AND gth or st AND gl are met.

4. Binding Table

The binding table is a special resource that gives access to the bindings on a endpoint. A binding table resource MUST support the Binding interface defined below. A profile SHOULD allow only one resource table per endpoint.

4.1. Binding Interface Description

This section defines a REST interface for Binding table resources. The interface supports the link-format type.

The if= column defines the Interface Description (if=) attribute value to be used in the CoRE Link Format for a resource conforming to that interface. When this value appears in the if= attribute of a link, the resource MUST support the corresponding REST interface described in this section. The resource MAY support additional functionality, which is out of scope for this specification. Although this interface description is intended to be used with the

CoRE Link Format, it is applicable for use in any REST interface definition.

The Methods column defines the REST methods supported by the interface, which are described in more detail below.

Interface	if=	Methods	Content-Formats
Binding	core.bnd	GET, POST, DELETE	link-format

Table 3: Binding Interface Description

The Binding interface is used to manipulate a binding table. A request with a POST method and a content format of application/link-format simply appends new bindings to the table. All links in the payload MUST have a relation type "boundTo". A GET request simply returns the current state of a binding table whereas a DELETE request empties the table. Individual entries may be dreeleted from the table by specifying the resource path in a DELETE request.

The following example shows requests for adding, retrieving and deleting bindings in a binding table.

```
Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/switch>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"
Res: 2.04 Changed

Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/s/switch>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"

Req: DELETE /bnd/a/switch
Res: 2.04 Changed

Req: DELETE /bnd/
Res: 2.04 Changed
```

Figure 1: Binding Interface Example

4.2. Resource Observation Attributes

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [RFC7641] MAY be used to observe any changes in a resource, and receive asynchronous

notifications as a result. In addition, a set of query string parameters are defined here to allow a client to control how often a client is interested in receiving notifications and how much a resource value should change for the new representation to be interesting. These query parameters are described in the following table. A resource using an interface description defined in this specification and marked as Observable in its link description SHOULD support these observation parameters. The Change Step parameter can only be supported on resources with an atomic numeric value.

Editor's note: There is a proposal to use the query parameters on the GET Observe as the default pattern. This allows multiple observations of the same resource. The PUT behaviour below would be treated as a legacy option. This will be discussed at IETF98.

These query parameters MUST be treated as resources that are read using GET and updated using PUT, and MUST NOT be included in the Observe request. Multiple parameters MAY be updated at the same time by including the values in the query string of a PUT. Before being updated, these parameters have no default value.

Resource	Parameter	Data Format
Minimum Period	/{resource}?pmin	xsd:integer (>0)
Maximum Period	/{resource}?pmax	xsd:integer (>0)
Change Step	/{resource}?st	xsd:decimal (>0)
Less Than	/{resource}?lth	xsd:decimal
Greater Than	/{resource}?gth	xsd:decimal

Table 4: Resource Observation Attribute Summary

Minimum Period: As per Section 3.3.2

Maximum Period: As per Section 3.3.3

Change Step: As per Section 3.3.4

Greater Than: As per Section 3.3.5

Less Than: As per Section 3.3.6

5. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this specification. A server implementing what the client thinks is a resource with one of these interface descriptions could return malformed representations and response codes either by accident or maliciously. A server sending maliciously malformed responses could attempt to take advantage of a poorly implemented client for example to crash the node or perform denial of service.

6. IANA Considerations

6.1. Interface Description

The specification registers the "binding" CoRE interface description link target attribute value as per [RFC6690].

Attribute Value: core.binding

Description: The binding interface is used to manipulate a binding table which describes the link bindings between source and destination resources for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

6.2. Link Relations Type

This specification registers the new "bounto" link relation type as per [RFC5988].

Relation Name: boundto

Description: The purpose of a boundto relation type is to indicate that there is a binding between a source resource and a destination resource for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

Application Data: None

7. Acknowledgements

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface types have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this specification.

8. Changelog

draft-ietf-core-dynlink-03

- Section 4.2: Update the Href to use "switch" instead of "light".
- General: Added editor’s notes for issues to be resolved at IETF98.

draft-ietf-core-dynlink-02

- General: Changed the name of the greater than attribute "gt" to "gth" and the name of the less than attribute "lt" to "lth" due to conlict with the core resource directory draft lifetime "lt" attribute.
- Clause 6.1: Addressed the editor’s note by changing the link target attribute to "core.binding".
- Added Appendix A for examples.

draft-ietf-core-dynlink-01

- General: The term state synchronization has been introduced to describe the process of synchronization between destination and source resources.
- General: The document has been restructured the make the information flow better.
- Clause 3.1: The descriptions of the binding attributes have been updated to clarify their usage.
- Clause 3.1: A new clause has been added to discuss the interactions between the resources.
- Clause 3.4: Has been simplified to refer to the descriptions in 3.1. As the text was largely duplicated.

- Clause 4.1: Added a clarification that individual resources may be removed from the binding table.
- Clause 6: Formailised the IANA considerations.

draft-ietf-core-dynlink Initial Version 00:

- This is a copy of draft-groves-core-dynlink-00

draft-groves-core-dynlink Draft Initial Version 00:

- This initial version is based on the text regarding the dynamic linking functionality in I.D.ietf-core-interfaces-05.
- The WADL description has been dropped in favour of a thorough textual description of the REST API.

9. References

9.1. 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>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

9.2. Informative References

[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>.

[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. Examples

This appendix provides some examples of the use of binding attribute / observe attributes.

Note: For brevity the only the method or response code is shown in the header field.

A.1. Greater Than (gth) example

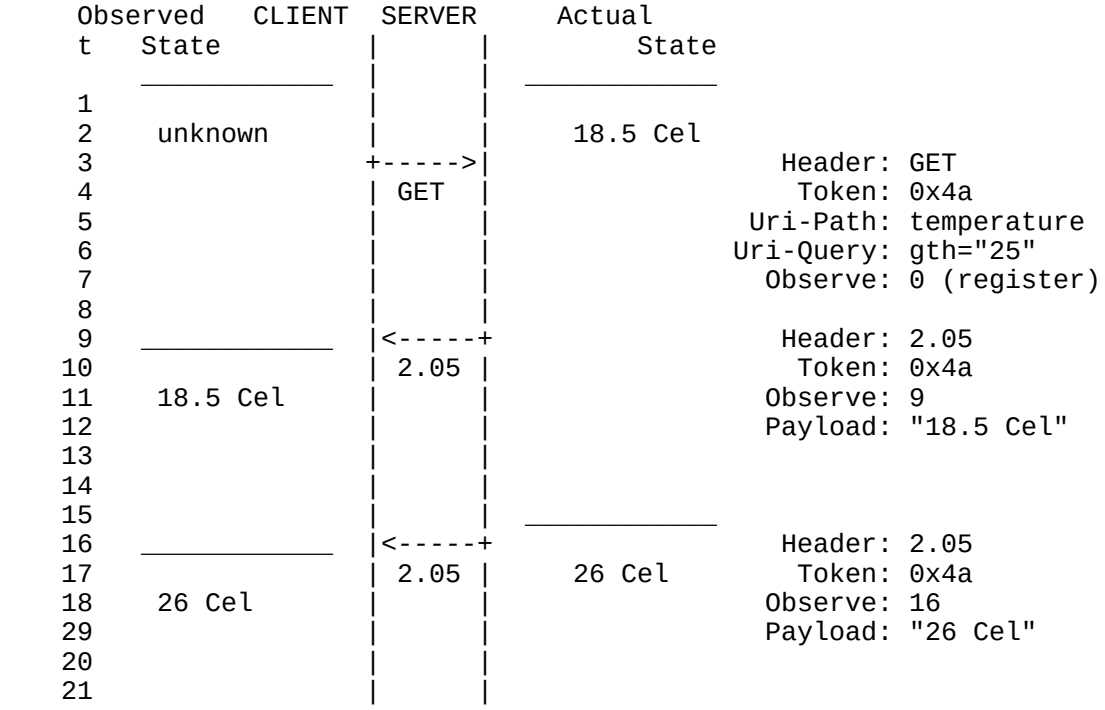
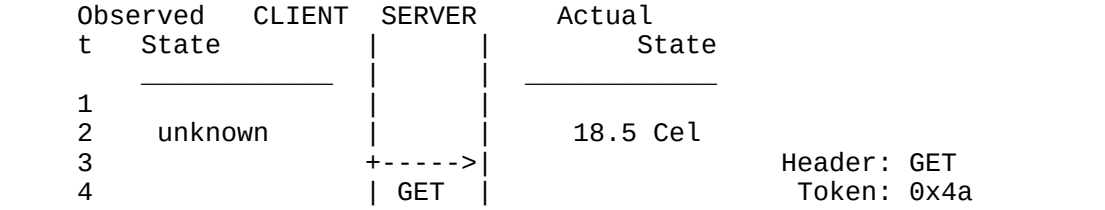


Figure 2: Client Registers and Receives one Notification of the Current State and One of a New State when it passes through the greather than threshold of 25.

A.2. Greater Than (gth) and Period Max (pmax) example



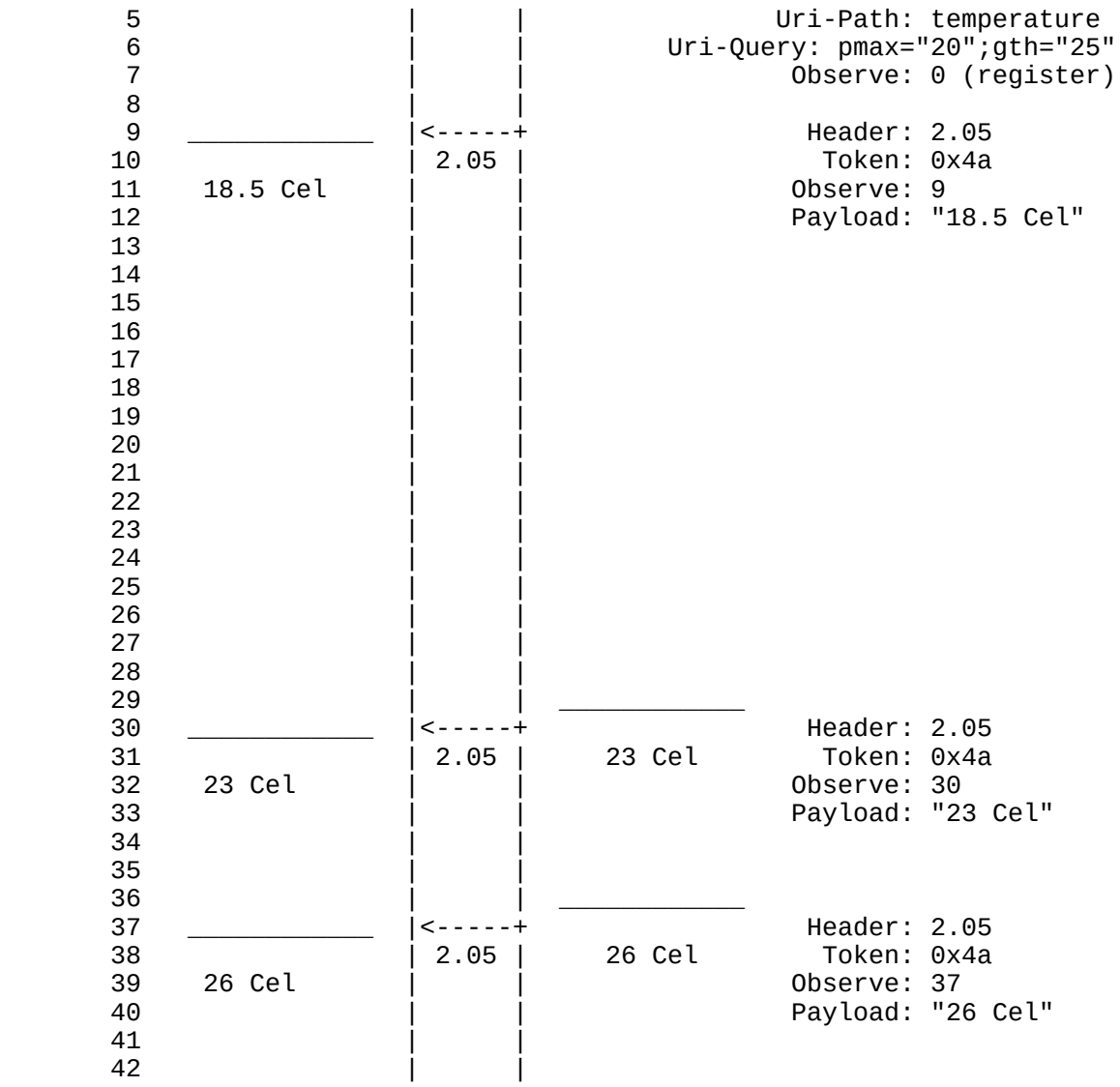


Figure 3: Client Registers and Receives one Notification of the Current State, one when pmax time expires and one of a new State when it passes through the greather than threshold of 25.

Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
FINLAND

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

Matthieu Vial
Schneider-Electric
Grenoble
FRANCE

Phone: +33 (0)47657 6522
Email: matthieu.vial@schneider-electric.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smartthings.com

Christian Groves
Huawei
Australia

Email: cngroves.std@gmail.com

core
Internet-Draft
Intended status: Standards Track
Expires: May 18, 2017

P. van der Stok
Consultant
C. Bormann
Universitaet Bremen TZI
A. Sehgal
Consultant

November 14, 2016

Patch and Fetch Methods for Constrained Application Protocol (CoAP)
draft-ietf-core-etch-04

Abstract

The methods defined in RFC 7252 for the Constrained Application Protocol (CoAP) only allow access to a complete resource, not to parts of a resource. In case of resources with larger or complex data, or in situations where a resource continuity is required, replacing or requesting the whole resource is undesirable. Several applications using CoAP will need to perform partial resource accesses.

This specification defines the new CoAP methods, FETCH, PATCH and iPATCH, which are used to access and update parts of a resource.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 18, 2017.

Copyright Notice

Table of Contents

1. Introduction

Similar to HTTP, the GET method defined in [RFC7252] for the Constrained Application Protocol (CoAP) only allows the specification of a URI and request parameters in CoAP options, not the transfer of a request payload detailing the request. This leads to some

applications to using POST where actually a cacheable, idempotent, safe request is desired.

Again similar to the original specification of HTTP, the PUT method defined in [RFC7252] only allows to replace a complete resource. This also leads applications to use POST where actually a cacheable, possibly idempotent request is desired.

The present specification adds new CoAP methods: FETCH, to perform the equivalent of a GET with a request body; and the twin methods PATCH and iPATCH, to modify parts of a CoAP resource.

1.1. FETCH

The CoAP GET method [RFC7252] is used to obtain the representation of a resource, where the resource is specified by a URI and additional request parameters can additionally shape the representation. This has been modelled after the HTTP GET operation and the REST model in general.

In HTTP, a resource is often used to search for information, and existing systems varyingly use the HTTP GET and POST methods to perform a search. Often a POST method is used for the sole reason that a larger set of parameters to the search can be supplied in the request body than can comfortably be transferred in the URI with a GET request. The draft [I-D.snell-search-method] proposes a SEARCH method that is similar to GET in most properties but enables sending a request body as with POST. The FETCH method defined in the present specification is inspired by [I-D.snell-search-method], which updates the definition and semantics of the HTTP SEARCH request method previously defined by [RFC5323]. However, there is no intention to limit FETCH to search-type operations, and the resulting properties may not be the same as those of HTTP SEARCH.

A major problem with GET is that the information that controls the request needs to be bundled up in some unspecified way into the URI. Using the request body for this information has a number of advantages:

- The client can specify a media type (and a content encoding), enabling the server to unambiguously interpret the request parameters in the context of that media type. Also, the request body is not limited by the character set limitations of URIs, enabling a more natural (and more efficient) representation of certain domain-specific parameters.

- The request parameters are not limited by the maximum size of the URI. In HTTP, that is a problem as the practical limit for this

size varies. In CoAP, another problem is that the block-wise transfer is not available for transferring large URI options in multiple rounds.

As an alternative to using GET, many implementations make use of the POST method to perform extended requests, even if they are semantically idempotent, safe, and even cacheable, to be able to pass along the input parameters within the request payload as opposed to using the request URI.

The FETCH method provides a solution that spans the gap between the use of GET and POST. As with POST, the input to the FETCH operation is passed along within the payload of the request rather than as part of the request URI. Unlike POST, however the semantics of the FETCH method are more specifically defined.

1.2. PATCH and iPATCH

PATCH is also specified for HTTP in [RFC5789]. Most of the motivation for PATCH described in [RFC5789] also applies here. iPATCH is the idempotent version of PATCH.

The PUT method exists to overwrite a resource with completely new contents, and cannot be used to perform partial changes. When using PUT for partial changes, proxies and caches, and even clients and servers, may get confused as to the result of the operation. PATCH was not adopted in an early design stage of CoAP, however, it has become necessary with the arrival of applications that require partial updates to resources (e.g. [I-D.vanderstok-core-comi]). Using PATCH avoids transferring all data associated with a resource in case of modifications, thereby not burdening the constrained communication medium.

This document relies on knowledge of the PATCH specification for HTTP [RFC5789]. This document provides extracts from [RFC5789] to make independent reading possible.

1.3. Requirements Language

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].

1.4. Terminology and Acronyms

This document uses terminology defined in [RFC5789] and [RFC7252].

Specifically, it uses the terms "safe" and "idempotent" as defined in Section 5.1 of [RFC7252]. (Further discussion of safe and idempotent methods can now be found in Section 4.2.1 and 4.2.2 of [RFC7231], respectively; the implications of idempotency of methods on server implementations are also discussed in Section 4.5 of [RFC7252].)

2. FETCH Method

The CoAP FETCH method is used to obtain a representation of a resource, giving a number of request parameters. Unlike the CoAP GET method, which requests that a server return a representation of the resource identified by the effective request URI (as defined by [RFC7252]), the FETCH method is used by a client to ask the server to produce a representation as described by the request parameters (including the request options and the payload) based on the resource specified by the effective request URI. The payload returned in response to a FETCH cannot be assumed to be a complete representation of the resource identified by the effective request URI, i.e., it cannot be used by a cache as a payload to be returned by a GET request.

Together with the request options, the body of the request (which may be constructed from multiple payloads using the block protocol [RFC7959]) defines the request parameters. With the FETCH method, implementations may submit a request body of any media type that is defined with the semantics of selecting information from a resource in such a FETCH request; it is outside the scope of this document how information about media types admissible for the specific resource is obtained by the client (although we can hint that form relations ([I-D.hartke-core-apps]) might be a preferred way). It is RECOMMENDED that any discovery method that allows a client to find out that the server supports FETCH also provides information what FETCH payload media types are applicable.

FETCH requests are both safe and idempotent with regards to the resource identified by the request URI. That is, the performance of a fetch is not intended to alter the state of the targeted resource. (However, while processing a fetch request, a server can be expected to allocate computing and memory resources or even create additional server resources through which the response to the search can be retrieved.)

A successful response to a FETCH request is expected to provide some indication as to the final disposition of the requested operation.

If a successful response includes a body payload, the payload is expected to describe the results of the FETCH operation.

Depending on the response code as defined by [RFC7252], the response to a FETCH request is cacheable; the request body is part of the cache key. Specifically, 2.05 "Content" response codes, the responses for which are cacheable, are a usual way to respond to a FETCH request. (Note that this aspect differs markedly from [I-D.snell-search-method].) (Note also that caches that cannot use the request payload as part of the cache key will not be able to cache responses to FETCH requests at all.) The Max-Age option in the response has equivalent semantics to its use in a GET.

The semantics of the FETCH method change to a "conditional FETCH" if the request message includes an If-Match, or If-None-Match option ([RFC7252]). A conditional FETCH requests that the query be performed only under the circumstances described by the conditional option(s). It is important to note, however, that such conditions are evaluated against the state of the target resource itself as opposed to the results of the FETCH operation.

2.1. Response Codes

FETCH for CoAP adopts the response codes as specified in sections 5.9 and 12.1.2 of [RFC7252] as well as additional response codes mentioned in Section 2.2.

2.2. Error Handling

A FETCH request may fail under certain known conditions. Beyond the conditions already defined in [RFC7252] for GET, noteworthy ones are:

Malformed FETCH payload: If a server determines that the payload provided with a FETCH request is not properly formatted, it can return a 4.00 (Bad Request) CoAP error. The definition of a malformed payload depends upon the CoAP Content-Format specified with the request.

Unsupported FETCH payload: In case a client sends a payload that is inappropriate for the resource identified by the Request-URI, the server can return a 4.15 (Unsupported Content-Format) CoAP error. The server can determine if the payload is supported by checking the CoAP Content-Format specified with the request.

Unprocessable request: This situation occurs when the payload of a FETCH request is determined as valid, i.e. well-formed and supported, however, the server is unable to or incapable of processing the request. The server can return a 4.22

(Unprocessable Entity) CoAP error. In situations when the server has insufficient computing resources to complete the request successfully, it can return a 4.13 (Request Entity Too Large) CoAP error (see also below). In case there are more specific errors that provide more insight into the problem, then those should be used.

Request too large: If the payload of the FETCH request is larger than a CoAP server can process, then it can return the 4.13 (Request Entity Too Large) CoAP error.

It is possible that other error situations, not mentioned here, are encountered by a CoAP server while processing the FETCH request. In these situations other appropriate CoAP status codes can also be returned.

2.3. Option Numbers

FETCH for CoAP adopts the option numbers as specified in sections 5.10 and 12.2 of [RFC7252].

Generally, options defined for GET act in an analogous way for FETCH. Two specific cases are called out in the rest of this section.

2.3.1. The Content-Format Option

A FETCH request MUST include a Content-Format option (see Section 5.10.3 of [RFC7252]) to specify the media type and content encoding of the request body. (Typically, the media type will specifically have been designed to specify details for a selection or a search on a resource.)

2.3.2. The ETag Option

The ETag Option on a FETCH result has the same semantics as defined in Section 5.10.6 of [RFC7252]. In particular, its use as a response option describes the "tagged representation", which for FETCH is the same as the "selected representation". The FETCH payload is input to that selection process and therefore needs to be part of the cache key. Similarly, the use of ETag as a request option can elicit a 2.03 Valid response if the representation associated with the ETag would still be selected by the FETCH request (including its payload).

2.4. Working with Observe

The Observe option [RFC7641] can be used with a FETCH request as it can be used with a GET request.

2.5. Working with Block

The Block1 option [RFC7959] can be used with a FETCH request as it would be used with a POST request; the Block2 option can then be used as with GET or POST.

2.6. Building FETCH Requests

One property of FETCH that may be non-obvious is that a FETCH request cannot be generated from a link alone, but also needs a way to generate the request payload. Again, form relations ([I-D.hartke-core-apps]) may be able to fill parts of this gap.

2.7. A Simple Example for FETCH

The FETCH method needs a media type for its payload (as expressed by the Content-Format request option) that specifies the search query in a similar detail as is shown for the patch payload in the PATCH example in Section 3.1. ([I-D.snell-search-method] invents a "text/query" format based on some hypothetical SQL dialect for its examples.)

The example below illustrates retrieval of a subset of a JSON [RFC7159] object (the same object as used in Section 3.1). Using a hypothetical media type "application/example-map-keys+json" (with a Content-Format ID of NNN - not defined as this is just an example), the client specifies the items in the object that it wants: it supplies a JSON array giving the map keys for these items. A resource located at "coap://www.example.com/object" can be represented by a JSON document that we will consider as the target of the FETCH. The client wants to learn the contents of the single map key "foo" within this target:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar","baz"]
}
```

FETCH example: JSON document returned by GET

The example FETCH request specifies a single top-level member desired by giving its map key as the sole element of the "example-map-keys" payload:

```
FETCH CoAP://www.example.com/object
Content-Format: NNN (application/example-map-keys+json)
Accept: application/json
[
  "foo"
]
```

FETCH example: Request

The server returns a subset document with just the selected member:

```
2.05 Content
Content-Format: 50 (application/json)
{
  "foo": ["bar","baz"]
}
```

FETCH example: Response with subset JSON document

By the logic of this example, the requester could have entered more than one map key into the request payload array and would have received a more complete subset of the top-level JSON object that is representing the resource.

3. PATCH and iPATCH Methods

The PATCH and iPATCH methods request that a set of changes described in the request payload is applied to the target resource of the request. The set of changes is represented in a format identified by a media type. If the Request-URI does not point to an existing resource, the server MAY create a new resource with that URI, depending on the patch document type (whether it can logically modify a null resource) and permissions, as well as other conditions such as the degree of control the server gives clients in creating new entries in its URI space (see also Section 3.4). Creation of a new resource would result in a 2.01 (Created) Response Code dependent on the patch document type.

Restrictions to a PATCH or iPATCH request can be made by including the If-Match or If-None-Match options in the request (see Section 5.10.8.1 and 5.10.8.2 of [RFC7252]). If the resource could not be created or modified, then an appropriate Error Response Code SHOULD be sent.

The difference between the PUT and PATCH requests is documented in [RFC5789]. When a request is intended to effect a partial update of a given resource, clients cannot use PUT while supplying just the update, but might be able to use PATCH or iPATCH.

The PATCH method is not safe and not idempotent, as with the HTTP PATCH method specified in [RFC5789].

The iPATCH method is not safe but idempotent, as with the CoAP PUT method specified in [RFC7252], Section 5.8.3.

A client can mark a request as idempotent by using the iPATCH method instead of the PATCH method. This is the only difference between the two. The indication of idempotence may enable the server to keep less state about the interaction; some constrained servers may only implement the iPATCH variant for this reason.

PATCH and iPATCH are both atomic. The server MUST apply the entire set of changes atomically and never provide a partially modified representation to a concurrently executed GET request. Given the constrained nature of the servers, most servers will only execute CoAP requests consecutively, thus preventing a concurrent partial overlapping of request modifications. Resuming, modifications MUST NOT be applied to the server state when an error occurs or only a partial execution is possible on the resources present in the server.

The atomicity applies to a single server. When a PATCH or iPATCH request is multicast to a set of servers, each server can either execute all required modifications or not. It is not required that all servers execute all modifications or none. An Atomic Commit protocol that provides multiple server atomicity is out of scope.

A PATCH or iPATCH response can invalidate a cache as with the PUT response. Caching behaviour as function of the successful (2.xx) response codes for PATCH or iPATCH are:

- A 2.01 (Created) response invalidates any cache entry for the resource indicated by the Location-* Options; the payload is a representation of the action result.

- A 2.04 (Changed) response invalidates any cache entry for the target resource; the payload is a representation of the action result.

There is no guarantee that a resource can be modified with PATCH or iPATCH. Servers MUST ensure that a received PATCH body is appropriate for the type of resource identified by the target resource of the request.

It is RECOMMENDED that any discovery method that allows a client to find out that the server supports one of PATCH and iPATCH also provides information what patch payload media types are applicable

and which of the two methods are implemented by the server for each of these media types.

Servers that do not rely on the idempotency of iPATCH can easily support both PATCH and iPATCH, and it is RECOMMENDED they do so. This is inexpensive to do, as, for iPATCH, there is no requirement on the server to check that the client's intention that the request be idempotent is fulfilled (although there is diagnostic value in that check, so a less-constrained implementation may want to perform it).

3.1. Simple Examples for PATCH and iPATCH

The example is taken over from [RFC6902], which specifies a JSON notation for PATCH operations. A resource located at coap://www.example.com/object contains a target JSON document.

JSON document original state:

```
    {
      "x-coord": 256,
      "y-coord": 45,
      "foo": ["bar","baz"]
    }
```

```
REQ: iPATCH CoAP://www.example.com/object
Content-Format: 51 (application/json-patch+json)
    [
      { "op":"replace", "path":"x-coord", "value":45}
    ]
```

```
RET: CoAP 2.04 Changed
```

JSON document final state:

```
    {
      "x-coord": 45,
      "y-coord": 45,
      "foo": ["bar","baz"]
    }
```

This example illustrates use of an idempotent modification to the x-coord member of the existing resource "object". The 2.04 (Changed) response code is conform with the CoAP PUT method.

The same example using the Content-Format application/merge-patch+json from [RFC7396] looks like:

```
JSON document original state:
    {
      "x-coord": 256,
      "y-coord": 45,
      "foo": ["bar","baz"]
    }

REQ: iPATCH CoAP://www.example.com/object
Content-Format: 52 (application/merge-patch+json)
     { "x-coord":45}

RET: CoAP 2.04 Changed

JSON document final state:
    {
      "x-coord": 45,
      "y-coord": 45,
      "foo": ["bar","baz"]
    }
```

The examples show the use of the iPATCH method, but the use of the PATCH method would have led to the same result. Below a non-idempotent modification is shown. Because the action is non-idempotent, iPATCH returns an error, while PATCH executes the action.

```
JSON document original state:
    {
      "x-coord": 256,
      "y-coord": 45,
      "foo": ["bar","baz"]
    }

REQ: iPATCH CoAP://www.example.com/object
Content-Format: 51 (application/json-patch+json)
    [
      { "op":"add","path":"foo/1","value":"bar"}
    ]
RET: CoAP 4.00 Bad Request
Diagnostic payload: Patch format not idempotent

JSON document final state is unchanged

REQ: PATCH CoAP://www.example.com/object
Content-Format: 51 (application/json-patch+json)
    [
      { "op":"add","path":"foo/1","value":"bar"}
    ]
RET: CoAP 2.04 Changed

JSON document final state:
    {
      "x-coord": 45,
      "y-coord": 45,
      "foo": ["bar","bar","baz"]
    }
```

3.2. Response Codes

PATCH and iPATCH for CoAP adopt the response codes as specified in sections 5.9 and 12.1.2 of [RFC7252] and add 4.09 "Conflict" and 4.22 "Unprocessable Entity" with the semantics specified in Section 3.4 of the present specification.

3.3. Option Numbers

PATCH and iPATCH for CoAP adopt the option numbers as specified in sections 5.10 and 12.2 of [RFC7252].

3.4. Error Handling

A PATCH or iPATCH request may fail under certain known conditions. These situations should be dealt with as expressed below.

Malformed PATCH or iPATCH payload: If a server determines that the payload provided with a PATCH or iPATCH request is not properly formatted, it can return a 4.00 (Bad Request) CoAP error. The definition of a malformed payload depends upon the CoAP Content-Format specified with the request.

Unsupported PATCH or iPATCH payload: In case a client sends a payload that is inappropriate for the resource identified by the Request-URI, the server can return a 4.15 (Unsupported Content-Format) CoAP error. The server can determine if the payload is supported by checking the CoAP Content-Format specified with the request.

Unprocessable request: This situation occurs when the payload of a PATCH request is determined as valid, i.e. well-formed and supported, however, the server is unable to or incapable of processing the request. The server can return a 4.22 (Unprocessable Entity) CoAP error. More specific scenarios might include situations when:

* the server has insufficient computing resources to complete the request successfully -- 4.13 (Request Entity Too Large) CoAP Response Code (see below),

* the resource specified in the request becomes invalid by applying the payload -- 4.09 (Conflict) CoAP Response Code (see below)).

In case there are more specific errors that provide more insight into the problem, then those should be used.

Resource not found: The 4.04 (Not Found) error should be returned in case the payload of a PATCH request cannot be applied to a non-existent resource.

Failed precondition: In case the client uses the conditional If-Match or If-None-Match option to define a precondition for the PATCH request, and that precondition fails, then the server can return the 4.12 (Precondition Failed) CoAP error.

Request too large: If the payload of the PATCH request is larger than a CoAP server can process, then it can return the 4.13 (Request Entity Too Large) CoAP error.

Conflicting state: If the modification specified by a PATCH or iPATCH request causes the resource to enter an inconsistent state that the server cannot resolve, the server can return the 4.09 (Conflict) CoAP response. The server SHOULD generate a payload

that includes enough information for a user to recognize the source of the conflict. The server MAY return the actual resource state to provide the client with the means to create a new consistent resource state. Such a situation might be encountered when a structural modification is applied to a configuration data-store, but the structures being modified do not exist.

Concurrent modification: Resource constrained devices might need to process requests in the order they are received. In case requests are received concurrently to modify the same resource but they cannot be queued, the server can return a 5.03 (Service unavailable) CoAP response code.

Conflict handling failure: If the modification implies the reservation of resources or the waiting on conditions to become true, leading to a too long request execution time, the server can return 5.03 (service unavailable) response code.

It is possible that other error situations, not mentioned here, are encountered by a CoAP server while processing the PATCH request. In these situations other appropriate CoAP status codes can also be returned.

4. The New Set of CoAP Methods

Adding three new methods to CoAP's existing four may seem like a major change. However, both FETCH and the two PATCH variants fit well into the REST paradigm and have been anticipated on the HTTP side. Adding both a non-idempotent and an idempotent PATCH variant allows to keep interoperability with HTTP's PATCH method as well as the use/indication of an idempotent PATCH if that is possible, saving significant effort on the server side.

Interestingly, the three new methods fit into the old table of methods with a surprising similarity in the idempotence and safety attributes:

Code	Name	Code	Name	safe	idempotent
0.01	GET	0.05	FETCH	yes	yes
0.02	POST	0.06	PATCH	no	no
0.03	PUT	0.07	iPATCH	no	yes
0.04	DELETE			no	yes

5. Security Considerations

This section analyses the possible threats to the CoAP FETCH and PATCH or iPATCH methods. It is meant to inform protocol and application developers about the security limitations of CoAP FETCH and PATCH or iPATCH as described in this document.

The FETCH method is subject to the same general security considerations as all CoAP methods as described in Section 11 of [RFC7252]. Specifically, the security considerations for FETCH are closest to those of GET, except that the FETCH request carries a payload that may need additional protection. The payload of a FETCH request may reveal more detailed information about the specific portions of a resource of interest to the requester than a GET request for the entire resource would; this may mean that confidentiality protection of the request by DTLS or other means is needed for FETCH where it wouldn't be needed for GET.

The PATCH and iPATCH methods are subject to the same general security considerations as all CoAP methods as described in Section 11 of [RFC7252]. The specific security considerations for PATCH or iPATCH are nearly identical to the security considerations for PUT ([RFC7252]); the security considerations of Section 5 of [RFC5789] also apply to PATCH and iPATCH. Specifically, there is likely to be a need for authorizing requests (possibly through access control and/or authentication) and for ensuring that data is not corrupted through transport errors or through accidental overwrites. The mechanisms used for PUT can be used for PATCH or iPATCH as well.

The new methods defined in the present specification are secured following the CoAP recommendations for the existing methods as specified in section 9 of [RFC7252]. When additional security techniques are standardized for CoAP (e.g., Object Security), these are then also available for securing the new methods.

6. IANA Considerations

IANA is requested to add the following entries to the sub-registry "CoAP Method Codes":

Code	Name	Reference
0.05	FETCH	[RFCthis]
0.06	PATCH	[RFCthis]
0.07	iPATCH	[RFCthis]

The FETCH method is idempotent and safe, and it returns the same response codes that GET can return, plus 4.13 (Request Entity Too Large), 4.15 (Unsupported Content-Format), and 4.22 (Unprocessable Entity) with the semantics specified in Section 2.2.

The PATCH method is neither idempotent nor safe. It returns the same response codes that POST can return, plus 4.09 (Conflict) and 4.22 (Unprocessable Entity) with the semantics specified in Section 3.4.

The iPATCH method is identical to the PATCH method, except that it is idempotent.

IANA is requested to add the following code to the sub-registry "CoAP response codes":

Code	Name	Reference
4.09	Conflict	[RFCthis]
4.22	Unprocessable Entity	[RFCthis]

IANA is requested to add entries to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry:

Media Type	Encoding	ID	Reference
application/json-patch+json		51	[RFC6902]
application/merge-patch+json		52	[RFC7396]

Editors' note (to be removed by RFC editor): RFC 6902 and RFC 7396 are not necessary to implement the present specification, not even an option for it. It was just convenient to use the present document to register content-format identifiers for them (and they are used in examples). Therefore, these references are correctly classified as _informative_.

7. Change log

When published as an RFC, this section needs to be removed.

Version 00 is a composition from draft-vanderstok-core-patch-03 and draft-bormann-core-coap-fetch-00 and replaces these two drafts.

Version 01 added an example for FETCH and is more explicit about some response codes and options.

Version 02 addresses the WGLC comments.

Version 03 addresses the IETF last-call comments.

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>.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <http://www.rfc-editor.org/info/rfc5789>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[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>.

[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>.

[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>.

8.2. Informative References

[RFC5323] Reschke, J., Ed., Reddy, S., Davis, J., and A. Babich, "Web Distributed Authoring and Versioning (WebDAV) SEARCH", RFC 5323, DOI 10.17487/RFC5323, November 2008, <http://www.rfc-editor.org/info/rfc5323>.

[RFC6902] Bryan, P., Ed. and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Patch", RFC 6902, DOI 10.17487/RFC6902, April 2013, <http://www.rfc-editor.org/info/rfc6902>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <http://www.rfc-editor.org/info/rfc7396>.

[I-D.vanderstok-core-comi] Stok, P., Bierman, A., Veillette, M., and A. Pelov, "CoAP Management Interface", draft-vanderstok-core-comi-10 (work in progress), October 2016.

[I-D.hartke-core-apps] Hartke, K., "CoRE Application Descriptions", draft-hartke-core-apps-05 (work in progress), October 2016.

[I-D.snell-search-method] Reschke, J., Malhotra, A., and J. Snell, "HTTP SEARCH Method", draft-snell-search-method-00 (work in progress), April 2015.

Acknowledgements

Klaus Hartke has pointed out some essential differences between CoAP and HTTP concerning PATCH, and found a number of problems in an earlier version of Section 2. We are grateful for discussions with Christian Amsuss, Andy Bierman, Timothy Carey, Paul Duffy, Matthias Kovatsch, Michel Veillette, Michael Verschoor, Thomas Watteyne, and Gengyu Wei. Christian Groves provided detailed comments during the Working-Group Last Call, and Christer Holmberg’s Gen-ART review provided some further editorial improvement. Further last-call reviews were provided by Sheng Jiang and Phillip Hallam-Baker. As usual, the IESG had some very good reviews, we would like to specifically call out those by Alexey Melnikov (responsible AD) and Alissa Cooper.

Authors’ Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

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

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

Anuj Sehgal
Consultant

Email: anuj@iurs.org

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

Z. Shelby
ARM
Z. Vial
Schneider-Electric
M. Koster
SmartThings
C. Groves
Huawei

March 13, 2017

Reusable Interface Definitions for Constrained RESTful Environments
draft-ietf-core-interfaces-09

Abstract

This document defines a set of Constrained RESTful Environments (CoRE) Link Format Interface Descriptions [RFC6690] applicable for use in constrained environments. These include the: Actuator, Paramter, Read-only parameter, Sensor, Batch, Linked Batch and Link List interfaces.

The Batch, Linked Batch and Link List interfaces make use of resource collections. This document further describes how collections relate to interfaces.

Many applications require a set of interface descriptions in order provide the required functionality. This document defines the concept of function sets to specify this set of interfaces and resources.

Editor's note: The git repository for the draft is found at https://github.com/core-wg/interfaces

Editor's note: Two open issues are proposals for: Removing the binding interface in favour of the link list interface. Changing "rel" type from one attribute to two to indicate src and destination.

Copyright Notice

Table of Contents

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format is a standard for doing Web Linking [RFC5988] in constrained environments. SenML [I-D.ietf-core-senml] is a simple data model and representation format for composite and complex structured resources. CoRE Link-Format and SenML can be used by CoAP [RFC7252] or HTTP servers.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop. Machine application clients must be able to adapt to different resource organizations without advance knowledge of the specific data structures hosted by each connected thing. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by CoRE Link Format [RFC6690]. CoRE Link Format additionally defines a link attribute for interface description ("if") that can be used to describe the REST interface of a resource, and may include a link to a description document.

This document defines a set of Link Format interface descriptions for some common design patterns that enable the server side composition and organization, and client side discovery and consumption, of machine resources using Web Linking. A client discovering the "if" link attribute will be able to consume resources based on its knowledge of the expected interface types. In this sense the

Interface Type acts in a similar way as a Content-Format, but as a selector for a high level functional abstraction.

An interface description describes a resource in terms of it's associated content formats, data types, URI templates, REST methods, parameters, and responses. Basic interface descriptions are defined for sensors, actuators, and properties.

A set of collection types is defined for organizing resources for discovery, and for various forms of bulk interaction with resource sets using typed embedding links.

This document first defines the concept of collection interface descriptions. It then defines a number of generic interface descriptions that may be used in contrained environments. Several of these interface descriptions utilise collections.

Whilst this document assumes the use of CoAP [RFC7252], the REST interfaces described can also be realized using HTTP [RFC7230].

2. Terminology

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

This document requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. This document makes use of the following additional terminology:

Gradual Reveal: A REST design where resources are discovered progressively using Web Linking.

Interface Description: The Interface Description describes the generic REST interface to interact with a resource or a set of resources. Its use is described via the Interface Description 'if' attribute which is an opaque string used to provide a name or URI indicating a specific interface definition used to interact with the target resource. One can think of this as describing verbs usable on a resource.

Resource Discovery: The process allowing a web client to identify resources being hosted on a web server.

Service Discovery: The process making it possible for a web client to automatically detect devices and Function Sets offered by these devices on a CoRE network.

3. Collections

3.1. Introduction to Collections

A Collection is a resource which represents one or more related resources. [RFC6573] describes the "item" and "collection" Link Relation. "item" link relation identifies a member of collection. "collection" indicates the collection that an item is a member of. For example: A collection might be a resource representing catalog of products, an item is a resource related to an individual product.

Section 1.2.2/[RFC6690] also describes resource collections.

This document uses the concept of "collection" and applies it to interface descriptions. A collection interface description consists of a set of links and a set of items pointed to by the links which may be sub-resources of the collection resource. The collection interface descriptions described in this document are Link List, Batch and Linked Batch.

The links in a collection are represented in CoRE Link-Format Content-Formats including JSON and CBOR variants, and the items in the collection may be represented by senml, including JSON and CBOR variants. In general, a collection may support items of any available Content-Format.

A particular resource item may be a member of more than one collection at a time by being linked to, but may only be a subresource of one collection.

Some collections may have pre-configured items and links, and some collections may support dynamic creation and removal of items and links. Likewise, modification of items in some collections may be permitted, and not in others.

Collections may support link embedding, which is analogous to an image tag (link) causing the image to display inline in a browser window. Resources pointed to by embedded links in collections may be interacted with using bulk operations on the collection resource. For example, performing a GET on a collection resource may return a single representation containing all of the linked resources.

Links in collections may be selected for processing by a particular request by using Query Filtering as described in CoRE Link-Format [RFC6690].

3.2. Use Cases for Collections

Collections may be used to provide gradual reveal of resources on an endpoint. There may be a small set of links at the .well-known/core location, which may in turn point to other collections of resources that represent device information, device configuration, device management, and various functional clusters of resources on the device.

A collection may provide resource encapsulation, where link embedding may be used to provide a single resource with which a client may interact to obtain a set of related resource values. For example, a collection for manufacturer parameters may consist of manufacturer name, date of manufacture, location of manufacture, and serial number resources which can be read as a single senml data object.

A collection may be used to group a set of like resources for bulk state update or actuation. For example, the brightness control resources of a number of luminaries may be grouped by linking to them in a collection. The collection type may support receiving a single update form a client and sending that update to each resource item in the collection.

Items may be sub-resources of the collection resource. This enables updates to multiple items in the collection to be processed together within the context of the collection resource.

3.3. Content-Formats for Collections

The collection interfaces by default use CoRE Link-Format for the link representations and SenML or text/plain for representations of items. The examples given are for collections that expose resources and links in these formats. In addition, a new "collection" Content-Format is defined based on the SenML framework which represents both links and items in the collection.

The choice of whether to return a representation of the links or of the items or of the collection format is determined by the Accept header option in the request. Likewise, the choice of updating link metadata or item data or the collection resource itself is determined by the Content-Format option in the header of the update request operation.

The default Content-Formats for collection types described in this document are:

Links: application/link-format, application/link-format+json

Items: application/senml+json, text/plain

3.4. Links and Items in Collections

Links use CoRE Link-Format representation by default and may point to any resource reachable from the context of the collection. This includes absolute links and links that point to other network locations if the context of the collection allows. Links to sub-resources in the collection MUST have a path-element starting with the resource name, as per [RFC3986]. Links to resources in the global context MUST start with a root path identifier [RFC5988]. Links to other collections are formed per [RFC3986].

Examples of links:

</sen/>;if="core.lb"': Link to the /sen/ collection describing it as a core.lb type collection (Linked Batch)

</sen/>;rel="grp"': Link to the /sen/ collection indicating that /sen/ is a member of a group in the collection in which the link appears.

<"/sen/temp">;rt="temperature"': An absolute link to the resource at the path /sen/temp

<temp>;rt="temperature": Link to the temp subresource of the collection in which this link appears."

<temp>;anchor="/sen/": A link to the temp subresource of the collection /sen/ which is assumed not to be a subresource of the collection in which the link appears ,but is expected to be identified in the collection by resource name.

Links in the collection MAY be Read, Updated, Added, or Removed using the CoRE Link-Format or JSON Merge-Patch Content-Formats on the collection resource. Reading links uses the GET method and returns an array or list containing the link-values of all selected links. Links may be added to the collection using POST or PATCH methods. Updates to links MUST use the PATCH method and MAY use query filtering to select links for updating. The PATCH method on links MUST use the JSON Merge-Patch Content-Format (application/merge-patch+json) specified in [RFC7396].

Items in the collection SHOULD be represented using the SenML (application/senml+json) or plain text (text/plain) Content-Formats, depending on whether the representation is of a single data point or multiple data points. Items MAY be represented using any supported Content-Format.

Link Embedding enables the bulk processing of items in the collection using a single operation targeting the collection resource. A subset of resources in the collection may be selected for operation using Query Filtering. Bulk Read operations using GET return a SenML representation of all selected resources. Bulk item Update operations using PUT or POST apply the payload document to all selected resource items in the collection, using either a Batch or Group update policy. A Batch update is performed by applying the resource values in the payload document to all resources in the collection that match any resource name in the payload document. Group updates are performed by applying the payload document to each item in the collection. Group updates are indicated by the link relation type rel="grp" in the link.

3.5. Queries on Collections

Collections MAY support query filtering as defined in CoRE Link-Format [RFC6690]. Operations targeting either the links or the items MAY select a subset of links and items in the collection by using query filtering. The Content-Format specified in the request header selects whether links or items are targeted by the operation.

3.6. Observing Collections

Resource Observation via [I-D.ietf-core-dynlink] using CoAP [RFC7252] MAY be supported on items in a collection. A subset of the conditional observe parameters MAY be specified to apply. In most cases pmin and pmax are useful. Resource observation on a collection’s items resource returns the collection representation. Observation Responses, or notifications, SHOULD provide the collection representations in SenML Content-Format. Notifications MAY include multiple observations of the colection resource, with SenML time stamps indicating the observation times.

3.7. Collection Types

There are three collection types defined in this document:

Collection Type	if=	Content-Format
Link List	core.ll	link-format
Batch	core.b	link-format, senml
Linked Batch	core.lb	link-format, senml

Table 1: Collection Type Summary

The interface description defined in this document describes the methods and functions that may be applied to the collections.

4. Interface Descriptions

This section defines REST interfaces for Link List, Batch, Sensor, Parameter and Actuator resources. Variants such as Linked Batch or Read-Only Parameter are also presented. Each type is described along with its Interface Description attribute value and valid methods. These are defined for each interface in the table below. These interfaces can support plain text and/or SenML Media types.

The if= column defines the Interface Description (if=) attribute value to be used in the CoRE Link Format for a resource conforming to that interface. When this value appears in the if= attribute of a link, the resource MUST support the corresponding REST interface described in this section. The resource MAY support additional functionality, which is out of scope for this document. Although these interface descriptions are intended to be used with the CoRE Link Format, they are applicable for use in any REST interface definition.

The Methods column defines the methods supported by that interface, which are described in more detail below.

Interface	if=	Methods	Content-Formats
Link List	core.ll	GET	link-format
Batch	core.b	GET, PUT, POST	link-format, senml
Linked Batch	core.lb	GET, PUT, POST, DELETE	link-format, senml
Sensor	core.s	GET	link-format, text/plain
Parameter	core.p	GET, PUT	link-format, text/plain
Read-only Parameter	core.rp	GET	link-format, text/plain
Actuator	core.a	GET, PUT, POST	link-format, text/plain

Table 2: Interface Description Summary

The following is an example of links in the CoRE Link Format using these interface descriptions. The resource hierarchy is based on a simple resource profile defined in Appendix B. These links are used in the subsequent examples below.

```
Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s/>;rt="simple.sen";if="core.b",
</s/light>;rt="simple.sen.lt";if="core.s",
</s/temp>;rt="simple.sen.tmp";if="core.s";obs,
</s/humidity>;rt="simple.sen.hum";if="core.s",
</a/>;rt="simple.act";if="core.b",
</a/1/led>;rt="simple.act.led";if="core.a",
</a/2/led>;rt="simple.act.led";if="core.a",
</d/>;rt="simple.dev";if="core.ll",
</l/>;if="core.lb",
```

Figure 1: Binding Interface Example

4.1. Link List

The Link List interface is used to retrieve (GET) a list of resources on a web server. The GET request SHOULD contain an Accept option with the application/link-format content format. However if the resource does not support any other form of content-format the Accept option MAY be elided.

Note: The use of an Accept option with application/link-format is recommended even though it is not strictly needed for the link list interface because this interface is extended by the batch and linked batch interfaces where different content-formats are possible.

The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on a web server.

Link List is the base interface to provide gradual reveal of resources on a CoRE web server, hence the root resource of a Function Set SHOULD implement this interface or an extension of this interface.

The following example interacts with a Link List /d containing Parameter sub-resources /d/name, /d/model.

```
Req: GET /d/ (Accept:application/link-format)
Res: 2.05 Content (application/link-format)
</d/name>;rt="simple.dev.n";if="core.p",
</d/model>;rt="simple.dev.mdl";if="core.rp"
```

4.2. Batch

The Batch interface is used to manipulate a collection of sub-resources at the same time. The Batch interface description supports the same methods as its sub-resources, and can be used to read (GET), update (PUT) or apply (POST) the values of those sub-resource with a single resource representation. The sub-resources of a Batch MAY be heterogeneous, a method used on the Batch only applies to sub-resources that support it. For example Sensor interfaces do not support PUT, and thus a PUT request to a Sensor member of that Batch would be ignored. A batch requires the use of SenML Media types in order to support multiple sub-resources.

In addition, the Batch interface is an extension of the Link List interface and in consequence MUST support the same methods. For example: a GET with an Accept:application/link-format on a resource utilizing the batch interface will return the sub-resource links.

The following example interacts with a Batch /s/ with Sensor sub-resources /s/light, /s/temp and /s/humidity.

```
Req: GET /s/
Res: 2.05 Content (application/senml+json)
{"e":[
    { "n": "light", "v": 123, "u": "lx" },
    { "n": "temp", "v": 27.2, "u": "degC" },
    { "n": "humidity", "v": 80, "u": "%RH" }],
}
```

4.3. Linked Batch

The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the web server, a Linked Batch is dynamically controlled by a web client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [RFC5988] and the CoRE Link Format [RFC6690]. A request with a POST method and a content format of application/link-format simply appends new resource links to the collection. The links in the payload MUST reference a resource on the web server with an absolute path. A DELETE request removes the entire collection. All other requests available for a basic Batch are still valid for a Linked Batch.

The following example interacts with a Linked Batch /l/ and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.

```
Req: POST /l/ (Content-Format: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed

Req: GET /l/
Res: 2.05 Content (application/senml+json)
{"e":[
   { "n": "/s/light", "v": 123, "u": "lx" },
   { "n": "/s/temp", "v": 27.2, "u": "degC" },
}

Req: POST /l/ (Content-Format: application/link-format)
</s/humidity>
Res: 2.04 Changed

Req: GET /l/ (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>

Req: GET /l/
Res: 2.05 Content (application/senml+json)
{"e":[
   { "n": "/s/light", "v": 123, "u": "lx" },
   { "n": "/s/temp", "v": 27.2, "u": "degC" },
   { "n": "/s/humidity", "v": 80, "u": "%RH" }],
}

Req: DELETE /l/
Res: 2.02 Deleted
```

4.4. Sensor

The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML. Plain text MAY be used for a single measurement that does not require meta-data. For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

The following are examples of Sensor interface requests in both text/plain and application/senml+json.

```
Req: GET /s/humidity (Accept: text/plain)
Res: 2.05 Content (text/plain)
80

Req: GET /s/humidity (Accept: application/senml+json)
Res: 2.05 Content (application/senml+json)
{"e":[
    { "n": "humidity", "v": 80, "u": "%RH" }],
}
```

4.5. Parameter

The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read (GET) or update (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading and updating a parameter.

```
Req: GET /d/name
Res: 2.05 Content (text/plain)
node5

Req: PUT /d/name (text/plain)
outdoor
Res: 2.04 Changed
```

4.6. Read-only Parameter

The Read-only Parameter interface allows configuration parameters to be read (GET) but not updated. Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading such a parameter.

```
Req: GET /d/model
Res: 2.05 Content (text/plain)
SuperNode200
```

4.7. Actuator

The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or the actuator value can be updated (PUT). In addition, this interface allows the use of POST to change

the state of an actuator, for example to toggle between its possible values. Plain text or SenML Media types MAY be returned from this type of interface. A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to updated.

The following example shows requests for reading, setting and toggling an actuator (turning on a LED).

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0

Req: PUT /a/1/led (text/plain)
1
Res: 2.04 Changed

Req: POST /a/1/led (text/plain)
Res: 2.04 Changed

Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

5. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this document. A server implementing what the client thinks is a resource with one of these interface descriptions could return malformed representations and response codes either by accident or maliciously. A server sending maliciously malformed responses could attempt to take advantage of a poorly implemented client for example to crash the node or perform denial of service.

6. IANA Considerations

This document registers the following CoRE Interface Description (if=) Link Target Attribute Values.

6.1. Link List

Attribute Value: core.ll

Description: The Link List interface is used to retrieve a list of resources on a web server.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.2. Batch

Attribute Value: core.b

Description: The Batch interface is used to manipulate a collection of sub-resources at the same time.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.3. Linked Batch

Attribute Value: core.lb

Description: The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the web server, a Linked Batch is dynamically controlled by a web client.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.4. Sensor

Attribute Value: core.s

Description: The Sensor interface allows the value of a sensor resource to be read.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.5. Parameter

Attribute Value: core.p

Description: The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read or update.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.6. Read-only parameter

Attribute Value: core.rp

Description: The Read-only Parameter interface allows configuration parameters to be read but not updated.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.7. Actuator

Attribute Value: core.a

Description: The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read or the actuator value can be updated. In addition, this interface allows the use of POST to change the state of an actuator, for example to toggle between its possible values.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

7. Acknowledgements

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface descriptions have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this document.

8. Changelog

Changes from -08 to 09:

- Section 3.6: Modified to indicate that the entire collection resource is returned.
- General: Added editor’s note with open issues.

Changes from -07 to 08:

- Section 3.3: Modified Accepts to Accept header option.
- Addressed the editor’s note in Section 4.1 to clarify the use of the Accept option.

Changes from -06 to 07:

- Corrected Figure 1 sub-resource names e.g. tmp to temp and hum to humidity.
- Addressed the editor’s note in Section 4.2.
- Removed section on function sets and profiles as agreed to at the IETF#97.

Changes from -05 to -06:

- Updated the abstract.
- Section 1: Updated introduction.
- Section 2: Alphabetised the order
- Section 2: Removed the collections definition in favour of the complete definition in the collections section.
- Removed section 3 on interfaces in favour of an updated definition in section 1.3.
- General: Changed interface type to interface description as that is the term defined in RFC6690.
- Removed section on future interfaces.
- Section 8: Updated IANA considerations.

- Added Appendix A to discuss current state of the art wrt to collections, function sets etc.

Changes from -04 to -05:

- Removed Link Bindings and Observe attributes. This functionality is now contained in I-D.ietf-core-dynlink.
- Hypermedia collections have been removed. This is covered in a new T2TRG draft.
- The WADL description has been removed.
- Fixed minor typos.
- Updated references.

Changes from -03 to -04:

- Fixed tickets #385 and #386.
- Changed abstract and into to better describe content.
- Focus on Interface and not function set/profiles in intro.
- Changed references from draft-core-observe to RFC7641.
- Moved Function sets and Profiles to section after Interfaces.
- Moved Observe Attributes to the Link Binding section.
- Add a Collection section to describe the collection types.
- Add the Hypermedia Collection Interface Description.

Changes from -02 to -03:

- Added lt and gt to binding format section.
- Added pmin and pmax observe parameters to Observation Attributes.
- Changed the definition of lt and gt to limit crossing.
- Added definitions for getattr and setattr to WADL.
- Added getattr and setattr to observable interfaces.
- Removed query parameters from Observe definition.

- Added observe-cancel definition to WADL and to observable interfaces.

Changes from -01 to -02:

- Updated the date and version, fixed references.
- "Removed pmin and pmax observe parameters "[Ticket #336]"."

Changes from -00 to WG Document -01

- Improvements to the Function Set section.

Changes from -05 to WG Document -00

- Updated the date and version.

Changes from -04 to -05

- Made the Observation control parameters to be treated as resources rather than Observe query parameters. Added Less Than and Greater Than parameters.

Changes from -03 to -04

- Draft refresh

Changes from -02 to -03

- Added Bindings
- Updated all rt= and if= for the new Link Format IANA rules

Changes from -01 to -02

- Defined a Function Set and its guidelines.
- Added the Link List interface.
- Added the Linked Batch interface.
- Improved the WADL interface definition.
- Added a simple profile example.

9. References

9.1. 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>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

9.2. Informative References

[I-D.ietf-core-dynlink]
Shelby, Z., Vial, M., Koster, M., and C. Groves, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-02 (work in progress), February 2017.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-10 (work in progress), March 2017.

[I-D.ietf-core-senml]
Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-05 (work in progress), March 2017.

[OIC-Core]
"OIC Resource Type Specification v1.1.0", 2016, <https://openconnectivity.org/resources/specifications>.

[OIC-SmartHome]
"OIC Smart Home Device Specification v1.1.0", 2016, <https://openconnectivity.org/resources/specifications>.

[OMA-TS-LWM2M]
"Lightweight Machine to Machine Technical Specification", 2016, <http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0>.

[oneM2MTS0008]
"TS 0008 v1.3.2 CoAP Protocol Binding", 2016, <http://www.onem2m.org/technical/published-documents>.

[oneM2MTS0023]
"TS 0023 v2.0.0 Home Appliances Information Model and Mapping", 2016, <http://www.onem2m.org/technical/published-documents>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC6573] Amundsen, M., "The Item and Collection Link Relations", RFC 6573, DOI 10.17487/RFC6573, April 2012, <http://www.rfc-editor.org/info/rfc6573>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[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>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <http://www.rfc-editor.org/info/rfc7396>.

Appendix A. Current Usage of Interfaces and Function Sets

Editor’s note: This appendix will be removed. It is only included for information.

This appendix analyses the current landscape with regards the definition and use of collections, interfaces and function sets/ profiles. This should be considered when considering the scope of this document.

In summary it can be seen that there is a lack of consistancy of the definition and usage of interface description and function sets.

A.1. Constrained RESTful Environments (CoRE) Link Format (IETF)

[RFC6690] assumes that different deployments or application domains will define the appropriate REST Interface Descriptions along with Resource Types to make discovery meaningful. It highlights that collections are often used for these interfaces.

Whilst 3.2/[RFC6690] defines a new Interface Description 'if' attribute the procedures around it are about the naming of the interface not what information should be included in the documentation about the interface.

Function sets are not discussed.

A.2. CoRE Resource Directory (IETF)

[I-D.ietf-core-resource-directory] uses the concepts of collections, interfaces and function sets.

If defines a number of interfaces: discovery, registration, registration update, registration removal, read endpoint links, update endpoint links, registration request interface, removal request interface and lookup interface. However it does not assign an inteface description identifier (if=) to these interfaces.

It does define a resource directory function set which specifies relevant content formats and interfaces to be used between a resource directory and endpoints. However it does not follow the format proposed by this document.

A.3. Open Connectivity Foundation (OCF)

The OIC Core Specification [OIC-Core] most closely aligns with the work in this specification. It makes use of interface descriptions as per [RFC6690] and has registered several interface identifiers (https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#if-link-target-att-value). These interface descriptors are similar to those defined in this specification. From a high level perspective:

```
links list:   OCF (oic.if.ll) -> IETF (core.ll)
              Note: it's called "link list" in the IETF.
linked batch: OCF (oic.if.b) -> IETF (core.lb)
read-only:    OCF (oic.if.r) -> IETF (core.rp)
read-write:   OCF (oic.if.rw) -> IETF (core.p)
actuator:     OCF (oic.if.a) -> IETF (core.a)
sensor:       OCF (oic.if.s) -> IETF (core.s)
batch:        No OCF equivalent -> IETF (core.b)
```

Some of the OCF interfaces make use of collections.

The OIC Core specification does not use the concept of function sets. It does however discuss the concept of profiles. The OCF defines two sets of documents. The core specification documents such as [OIC-Core] and vertical profile specification documents which provide specific information for specific applications. The OIC Smart Home Device Specification [OIC-SmartHome] is one such specification. It provides information on the resource model, discovery and data types.

A.4. oneM2M

OneM2M describes a technology independent functional architecture [oneM2MTS0023]. In this archictecture the reference points between functional entities are called "interfaces". This usage does not match the [RFC6690] concept of interfaces. A more direct comparison is that of 10.2/[oneM2MTS0023] that defines basic procedures and resource type-specific procedures utilising REST type create, retrieve, update, delete, notify actions.

[oneM2MTS0023] does not refer to resource collections however does define "Group Management Procedures" in 10.2.7/[oneM2MTS0023]. It does allow bulk management of member resources.

[oneM2MTS0023] does not use the term "function set". [oneM2MTS0008] describes the binding with the CoAP protocol. In some respects this document provides a profile of the CoAP protocol in terms of the protocol elements that need to be supported. However it does not define any interface descriptions nor collections.

A.5. OMA LWM2M

[OMA-TS-LWM2M] utilises the concept of interfaces. It defines the following interfaces: Bootstrap, Client Registration, Device Management and Service Enablement and Information Reporting. It defines that these have a particular direction (Uplink/Downlink) and indicates the operations that may be applied to the interface (i.e. Request Bootstrap, Write, Delete, Register, Update, De-Register, Create, Read, Write, Delete, Execute, Write Attributes, Discover,

Observe, Cancel Observation, Notify). It then further defines which objects may occur over the interface. In 6/[OMA-TS-LWM2M] resource model, identifier and data formats are described.

Whilst it does not formally describe the use of "collections" the use of a multiple resource TLV allows a hierarchy of resource/sub-resource.

It does not identify the interfaces through an Interface Description (if=) attribute.

It does not use the term function set. Informally the specification could be considered as a function set.

Note: It refers to draft-ietf-core-interfaces-00. It also makes use of the binding/observation attributes from draft-ietf-dynlink-00 but does not refer to that document.

Appendix B. Resource Profile example

The following is a short definition of simple device resource profile. This simplistic profile is for use in the examples of this document.

Functions	Root Path	RT	IF
Device Description	/d	simple.dev	core.ll
Sensors	/s	simple.sen	core.b
Actuators	/a	simple.act	core.b

Table 3: Functional list of resources

Type	Path	RT	IF	Data Type
Name	/d/name	simple.dev.n	core.p	xsd:string
Model	/d/model	simple.dev.mdl	core.rp	xsd:string

Table 4: Device Description Resources

Type	Path	RT	IF	Data Type
Light	/s/light	simple.sen.lt	core.s	xsd:decimal (lux)
Humidity	/s/humidity	simple.sen.hum	core.s	xsd:decimal (%RH)
Temperature	/s/temp	simple.sen.tmp	core.s	xsd:decimal (degC)

Table 5: Sensor Resources

Type	Path	RT	IF	Data Type
LED	/a/{#}/led	simple.act.led	core.a	xsd:boolean

Table 6: Actuator Resources

Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
FINLAND

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

Matthieu Vial
Schneider-Electric
Grenoble
FRANCE

Phone: +33 (0)47657 6522
Email: matthieu.vial@schneider-electric.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smartthings.com

Christian Groves
Huawei
Australia

Email: cngroves.std@gmail.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 29, 2017

K. Li
Alibaba Group
A. Rahman
InterDigital
C. Bormann, Ed.
Universitaet Bremen TZI

April 27, 2017

Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR
draft-ietf-core-links-json-08

Abstract

JavaScript Object Notation, JSON (RFC7159) is a text-based data format which is popular for Web based data exchange. Concise Binary Object Representation, CBOR (RFC7049) is a binary data format which has been optimized for data exchange for the Internet of Things (IoT). For many IoT scenarios, CBOR formats will be preferred since it can help decrease transmission payload sizes as well as implementation code sizes compared to other data formats.

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

Table of Contents

1. Introduction

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

The JavaScript Object Notation (JSON) [RFC7159] is a lightweight, text-based, language-independent data interchange format. JSON is popular in the Web development environment as it is easy for humans to read and write.

The Concise Binary Object Representation (CBOR) [RFC7049] is a binary data format which requires extremely small code size, allows very compact message representation, and provides extensibility without the need for version negotiation. CBOR is especially well suited for IoT environments because of these efficiencies.

When converting between a bespoke syntax such as that defined by [RFC6690] and JSON or CBOR, many small decisions have to be made. If left without guidance, it is likely that a number of slightly incompatible dialects will emerge. This specification defines a common format for representing CoRE Web Linking in JSON and CBOR.

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

1.1. Objectives

This specification has been designed based on the following objectives:

- Canonical mapping
  - lossless conversion in both directions between any pair of [RFC6690], JSON, and CBOR ("round-tripping")
  - but not attempting to ensure that a sequence of conversions from one of the formats through one or both of the others and back to the original would result in a bit-wise identical representation
- The simplest thing that could possibly work
  - Do not cater for RFC 5988 complications caused by HTTP header character set issues [RFC2047]

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.

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

CoAP: Constrained Application Protocol [RFC7252]

CBOR: Concise Binary Object Representation [RFC7049]

CoRE: Constrained RESTful Environments, the field of work underlying [RFC6690], [RFC7049], [RFC7252], [RFC7641], [RFC7959], and [RFC8075]

IoT: Internet of Things

JSON: JavaScript Object Notation [RFC7159]

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

2. Web Links in JSON and CBOR

2.1. Background

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

2.2. Information Model

This section discusses the information model underlying the CORE Link Format payload.

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

We straightforwardly map:

- the collection of Web links to a JSON or CBOR array of links;

- each link to a JSON object or CBOR map, mapping attribute names to attribute values.

In the object representing a "link-value", each target attribute or other parameter ("link-param") is represented by a JSON name/value pair (member). The name is a string representation of the parameter or attribute name (as in "parmname"). The value can be a string, a boolean, or an array of strings or booleans, as described below.

If the attribute value ("ptoken" or "quoted-string") is present, and a Link attribute with this name ("parmname") is present just once in the "link-value", the value is a string representation of the parameter or attribute value ("ptoken" or "quoted-string"). "quoted-string" productions are parsed (i.e, the outer quotes removed and the backslash constructions evaluated) as defined in [RFC6690] and its referenced documents, before placing them in JSON strings (in the representation of which they may gain back additional decorations such as backslashes as defined in [RFC7159]).

If no attribute value ("ptoken" or "quoted-string") is present, the presence of the attribute name is indicated by using the Boolean value "true" as the value.

If a Link attribute ("parmname") is present more than once in a "link-value", its values are then represented as a JSON array of JSON string values or "true"; this array becomes the value of the JSON name/value pair where the attribute name is the JSON name. Attributes occurring just once MUST NOT be represented as JSON arrays but MUST be directly represented as JSON strings or "true". (Note that [RFC6690] has cut down on the use of repeated parameter names; they are still allowed by [RFC5988] though. No attempt has been made to decode the possibly space-separated values for rt=, if=, and rel= into JSON arrays.)

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

As a non-normative summary, the resulting structure can be represented in CBOR Data Definition Language (CDDL) [I-D.greevenbosch-appsawg-cbor-cddl] as:

```
links = [* link]
link = {
  href: tstr    ; resource URI
  * tstr => value
}
value = tstr   ; text value -- the normal case
      / true   ; no value given, just the name
      / [2* tstr/true ] ; repeats for two or more
```

Figure 1: CoRE Link Format Data Model

2.3. Additional Encoding Step for CBOR

The above specification for JSON might have been used as is for the CBOR encoding as well. However, to further reduce message sizes, an extra encoding step is performed: "href" and some commonly occurring attribute names are encoded as small integers.

The substitution is defined in Table 1:

name	encoded value	origin
href	1	[RFC6690], [RFCthis]
rel	2	[RFC5988] Section 5.3
anchor	3	[RFC5988] Section 5.2
rev	4	[RFC5988] Section 5.3
hreflang	5	[RFC5988] Section 5.4
media	6	[RFC5988] Section 5.4
title	7	[RFC5988] Section 5.4
type	8	[RFC5988] Section 5.4
rt	9	[RFC6690] Section 3.1
if	10	[RFC6690] Section 3.2
sz	11	[RFC6690] Section 3.3
ct	12	[RFC7252] Section 7.2.1
obs	13	[RFC7641] Section 6

Table 1: Integer Encoding of common attribute names

This list of substitutions is fixed by the present specification; no future expansion of the list is foreseen. "href" as well as all attribute names in this list MUST be represented by their integer substitutions and MUST NOT use the attribute name in text form.

Adam’s Issue number 2: What should an implementation do that receives a link-format that doesn’t heed the above MUST? Is there an onus to check it?

This leads to the following CDDL representation for the CBOR encoding:

```
links = [* link]
link = {
  href => tstr    ; resource URI
  * label => value
}
href = 1
label = tstr / &(
  rel: 2,       anchor: 3,  rev: 4,
  hreflang: 5,  media: 6,   title: 7,
  type: 8,      rt: 9,      if: 10,
  sz: 11,       ct: 12,     obs: 13,
)
value = tstr   ; text value -- the normal case
      / true   ; no value given, just the name
      / [2* tstr/true ] ; repeats for two or more
```

Figure 2: CoRE Link Format Data Model (CBOR)

2.4. Converting JSON or CBOR to Link-Format

When a JSON or CBOR representation needs to be converted back to link-format, the above process is performed in inverse. Since link-format allows serializing link parameter values both in unqouted form ("ptoken") or in quoted form ("quoted-string"), a decision has to be made for each value. Where the syntax of "ptoken" does not allow the value to be represented, the quoted form clearly needs to be used. However, when both forms are possible, the decision is arbitrary. A work-in-progress revision of [RFC5988], [I-D.nottingham-rfc5988bis], clarifies that this is indeed intended to be the case. However, existing specifications of link attributes, including those in [RFC5988] and [RFC6690], sometimes have made this decision in a specific way by only including one or the other alternative in the ABNF given for a link parameter. This requires a converter to know about all these cases, including those that have not been defined yet at the time of writing the converter. This problem becomes even harder by the fact that there is no central registry of link-attribute names.

Obviously, the conversion back to link-format needs to result in a valid link-format document. The reference implementation in Appendix A has addressed this problem with the following two rules:

- Where a "ptoken" representation is possible, that is used instead of "quoted-string". This rule covers most of the special cases listed above.

- As a special exception to the above rule, the four link attributes "anchor", "title", "rt", and "if" are always expressed as "quoted-string". This rule covers these specific four cases.

This set of rules is based on the hope that future definitions of link attributes will no longer hardcode one or the other serialization.

2.5. Examples

The examples in this section are based on an example on page 15 of [RFC6690] (Figure 3).

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 3: Example from page 15 of [RFC6690]

2.5.1. Link Format to JSON Example

The link-format document in Figure 3 becomes (321 bytes, line breaks shown are not part of the minimally-sized JSON document):

```
"[{"href":"/sensors","ct":"40","title":"Sensor
Index"},{"href":"/sensors/temp","rt":"temperature-
c","if":"sensor"},{"href":"/sensors/light","rt":"light-
lux","if":"sensor"},{"href":"http://www.example.com/sensors/
t123","anchor":"/sensors/
temp","rel":"describedby"},{"href":"/t","anchor":"/sensors/
temp","rel":"alternate"}] "
```

To demonstrate the handling of value-less and array-valued attributes, we extend the link-format example by examples of these (Figure 4; the "obs" attribute is defined in Section 6 of [RFC7641], while the "foo" attribute is for exposition only):

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor";obs,
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby";foo="bar";foo=3;ct=4711,
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 4: Example derived from page 15 of [RFC6690]

The link-format document in Figure 4 becomes the JSON document in Figure 5 (some spacing and indentation added):

```
[{"href":"/sensors","ct":"40","title":"Sensor Index"},
 {"href":"/sensors/temp","rt":"temperature-c","if":"sensor",
  "obs":true},
 {"href":"/sensors/light","rt":"light-lux","if":"sensor"},
 {"href":"http://www.example.com/sensors/t123",
  "anchor":"/sensors/temp","rel":"describedby",
  "foo":["bar","3"],"ct":"4711"},
 {"href":"/t","anchor":"/sensors/temp","rel":"alternate"}]
```

Figure 5: Example derived from page 15 of [RFC6690]

Note that the conversion is unable to convert the string-valued "ct" attribute to a number, which would be the natural type for a Content-Format value; similarly, both "foo" values are treated as strings independently of whether they are quoted or numeric in syntax.

2.5.2. Link Format to CBOR Example

This examples shows conversion from link format to CBOR format.

The link-format document in Figure 3 becomes (in CBOR diagnostic format):

```
[{1: "/sensors", 12: "40", 7: "Sensor Index"},
 {1: "/sensors/temp", 9: "temperature-c", 10: "sensor"},
 {1: "/sensors/light", 9: "light-lux", 10: "sensor"},
 {1: "http://www.example.com/sensors/t123", 3: "/sensors/temp",
  2: "describedby"},
 {1: "/t", 3: "/sensors/temp", 2: "alternate"}]
```

or, in hexadecimal (203 bytes):

```
  85                              # array(number of data items:5)
     a3                           # map(# data item pairs:3)
        01                        # unsigned integer(value:1,"href")
        68                        # text string(8 bytes)
           2f73656e736f7273       # "/sensors"
        0c                        # unsigned integer(value:12,"ct")
        62                        # text(2)
           3430                   # "40"
        07                        # unsigned integer(value:7,"title")
        6c                        # text string(12 bytes)
           53656e736f7220496e646578 # "Sensor Index"
     a3                           # map(# data item pairs:3)
        01                        # unsigned integer(value:1,"href")
```

```
      6d                          # text string(13 bytes)
         2f73656e736f72732f74
         656d70                   # "/sensors/temp"
      09                          # unsigned integer(value:9,"rt")
      6d                          # text string(13 bytes)
         74656d70657261747572
         652d63                   # "temperature-c"
      0a                          # unsigned integer(value:10,"if")
      66                          # text string(6 bytes)
         73656e736f72             # "sensor"
   a3                             # map(# data item pairs:3)
      01                          # unsigned integer(value:1,"href")
      6e                          # text string(14 bytes)
         2f73656e736f72732f6c
         69676874                 # "/sensors/light"
      09                          # unsigned integer(value:9,"rt")
      69                          # text string(9 bytes)
         6c696768742d6c7578       # "light-lux"
      0a                          # unsigned integer(value:10,"if")
      66                          # text string(6 bytes)
         73656e736f72             # "sensor"
   a3                             # map(# data item pairs:3)
      01                          # unsigned integer(value:1,"href")
      78 23                       # text string(35 bytes)
         687474703a2f2f777777
         2e6578616d706c652e63
         6f6d2f73656e736f7273
         2f74313233            # "http://www.example.com/sensors/t123"
      03                          # unsigned integer(value:3,"anchor")
      6d                          # text string(13 bytes)
         2f73656e736f72732f74
         656d70                   # "/sensors/temp"
      02                          # unsigned integer(value:2,"rel")
      6b                          # text string(11 bytes)
         6465736372696265646279   # "describedby"
   a3                             # map(# data item pairs:3)
      01                          # unsigned integer(value:1,"href")
      62                          # text string(12 bytes)
         2f74                     # "/t"
      03                          # unsigned integer(value:3,"anchor")
      6d                          # text string(13 bytes)
         2f73656e736f72732f74
         656d70                   # "/sensors/temp"
      02                          # unsigned integer(value:2,"rel")
      69                          # text string(9 bytes)
         616c7465726e617465       # "alternate"
```

Figure 6: Web Links Encoded in CBOR

3. IANA Considerations

3.1. Media types

This specification registers the following additional Internet Media Types:

Type name: application

Subtype name: link-format+json

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+json" media type are required to conform to the "application/json" Media Type and are therefore subject to the same encoding considerations specified in [RFC7159], Section 11.

Security considerations: See Section 4 of [RFCthis].

Published specification: [RFCthis].

Applications that use this media type: Applications that interchange collections of Web links based on CoRE link format [RFC6690] in JSON.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): TEXT

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

Intended usage: COMMON

Change controller: IESG

and

Type name: application

Subtype name: link-format+cbor

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+cbor" media type are required to conform to the "application/cbor" Media Type and are therefore subject to the same encoding considerations specified in [RFC7049], Section 7.

Security considerations: See Section 4 of [RFCthis].

Published specification: [RFCthis].

Applications that use this media type: Applications that interchange collections of Web links based on CoRE link format [RFC6690] in CBOR.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): CBOR

Person & email address to contact for further information:
Kepeng Li <kepeng.lkp@alibaba-inc.com>

Intended usage: COMMON

Change controller: IESG

3.2. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the above media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. All IDs are assigned from the "Expert Review" (0-255) range. The assigned IDs are show in Table 2.

Media type	ID
application/link-format+cbor	TBD64
application/link-format+json	TBD54

Table 2: CoAP Content-Format IDs

4. Security Considerations

The security considerations relevant to the data model of [RFC6690], as well as those of [RFC7049] and [RFC7159] apply.

5. References

5.1. 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>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

5.2. Informative References

[I-D.greevenbosch-appsawg-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-10 (work in progress), March 2017.

[I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-10 (work in progress), March 2017.

[I-D.nottingham-rfc5988bis] Nottingham, M., "Web Linking", draft-nottingham-rfc5988bis-05 (work in progress), April 2017.

[MNOT11] Nottingham, M., "Linking in JSON", November 2011, <http://www.mnot.net/blog/2011/11/25/linking_in_json>.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <http://www.rfc-editor.org/info/rfc2047>.

[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>.

[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>.

[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>.

[RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <http://www.rfc-editor.org/info/rfc8075>.

[RUBY] "Information technology -- Programming languages -- Ruby", ISO/IEC 30170:2012, April 2012.

Appendix A. Reference implementation

A reference implementation of a converter from [RFC6690] link-format to JSON and CBOR (and back to link-format) in the programming language Ruby [RUBY] is reproduced below.

```
# <CODE BEGINS>
require 'strscan'
require 'json'
require 'cbor-pretty'

class String
  def as_utf8
    force_encoding(Encoding::UTF_8)
  end
end
```

```
   module CoRE
     module Links
       def self.map_to_true(a)
         Hash[a.map{ |t| [t, true]}]
       end

       PTOKENCHAR = %r"[\[\]\w!#-+\--/:<-?^-‘{-˜@]"
       QUOSTRCHAR = %r{(?:[^"\\]|\\.)}    # to be used inside "
       ATTRCHAR   = %r"[\w!#$&+.^‘|˜-]"
       MUSTBEQUOTED = map_to_true(%w{anchor title rt if})
       ANCHORNAME = "href"
       SCANATTR =
   %r{(#{ATTRCHAR}+)(?:=(?:(#{PTOKENCHAR}+)|"(#{QUOSTRCHAR}*)"))?} # "

       RAWMAPPINGS = <<-DATA
     href: 1,   rel: 2,        anchor: 3,
     rev: 4,    hreflang: 5,   media: 6,
     title: 7,  type: 8,       rt: 9,
     if: 10,    sz: 11,        ct: 12,
     obs: 13,
       DATA

       MAPPINGS = Hash.new {|h, k| k}

       RAWMAPPINGS.scan(/([-\w]+)\s*:\s*([-\w]+),/) do |n, v|
         MAPPINGS[n] = Integer(v)
       end

       def self.parse(*args)
         WLNK.parse(*args)
       end

       class WLNK
         attr_accessor :resources
         def initialize(r = [])    # make sure the keys are strings
           @resources = r.to_ary   # make sure it’s an Array
         end
         def self.parse(s, robust = true)
           wl = WLNK.new
           ss = StringScanner.new(s.as_utf8)
           ss.skip(/\s+/) if robust
           while ss.scan(%r{<([^>]+)>})
             res = { ANCHORNAME => ss[1].as_utf8 }
             ss.skip(/\s*/) if robust
             while ss.skip(/;/)
               ss.skip(/\s*/) if robust
               unless ss.scan(SCANATTR)
                 raise ArgumentError, "must have attribute behind ’;’
```

```
                  at: #{ss.peek(20).inspect} (byte #{ss.pos})"
              end
              key = ss[1].as_utf8
              value = ss[2] ||
                      (ss[3] ? ss[3].gsub(/\\(.)/) { $1 } : true)
              if res[key]
                res[key] = Array(res[key]) << value
              else
                res[key] = value
              end
              ss.skip(/\s*/) if robust
            end
            wl.resources << res
            break unless ss.skip(/,/)
            ss.skip(/\s*/) if robust
          end
          ss.skip(/\s*/) if robust
          raise ArgumentError, "link-format unparseable at:
             #{ss.peek(20).inspect} (byte #{ss.pos})" unless ss.eos?
          wl
        end
        def to_json
          JSON.pretty_generate(@resources)
        end
        def to_cbor
          CBOR.encode(@resources.map {|r|
                        Hash[r.map { |k, v| [MAPPINGS[k], v] }]})
        end
        def to_wlnk
          resources.map do |res|
            res = res.dup
            u = res.delete(ANCHORNAME)
            ["<#{u}>", *res.map do |k, v|
               if String === v
                 if MUSTBEQUOTED[k] || v !~ /\A#{PTOKENCHAR}+\z/
                     "#{k}=\"#{v.gsub(/[\\"]/) { |x| "\\#{x}"}}\""
                 else
                   "#{k}=#{v}"
                 end
               else
                 "#{k}"
               end
             end].join(';')
          end.join(",")
        end
      end
    end
  end
```

```
lf = CoRE::Links.parse(ARGF.read)

puts lf.to_json                 # JSON
puts CBOR.pretty(lf.to_cbor)    # CBOR "pretty" binary form
puts lf.to_wlnk                 # RFC 6690 link-format
# <CODE ENDS>
```

Acknowledgements

Special thanks to Bert Greevenbosch who was an author on the initial version of a contributing document as well as the original author on the CDDL notation.

Hannes Tschofenig made many helpful suggestions for improving this document.

Authors' Addresses

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

Email: kepeng.lkp@alibaba-inc.com

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

Phone: +1-514-585-0761
Email: akbar.rahman@interdigital.com

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 4, 2017

G. Selander
J. Mattsson
F. Palombini
Ericsson AB
L. Seitz
SICS Swedish ICT

May 03, 2017

Object Security of CoAP (OSCOAP)
draft-ietf-core-object-security-03

Abstract

This document defines Object Security of CoAP (OSCOAP), a method for application layer protection of the Constrained Application Protocol (CoAP), using the CBOR Object Signing and Encryption (COSE). OSCOAP provides end-to-end encryption, integrity and replay protection to CoAP payload, options, and header fields, as well as a secure message binding. OSCOAP is designed for constrained nodes and networks and can be used across intermediaries and over any layer. The use of OSCOAP is signaled with the CoAP option Object-Security, also defined in this document.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) is a web application protocol, designed for constrained nodes and networks [RFC7228]. CoAP specifies the use of proxies for scalability and efficiency. At the same time CoAP [RFC7252] references DTLS [RFC6347] for security. Proxy operations on CoAP messages require DTLS to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the intended proxy functionality, but is also able to eavesdrop on, or manipulate any part of the CoAP payload and metadata, in transit between client and server. The proxy can also inject, delete, or reorder packages since they are no longer protected by DTLS.

This document defines Object Security of CoAP (OSCOAP), a data object based security protocol, protecting CoAP message exchanges end-to-end, across intermediary nodes. An analysis of end-to-end security for CoAP messages through intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs], this specification addresses the forwarding case. In addition to the core features defined in [RFC7252], OSCOAP supports Observe [RFC7641] and Blockwise [RFC7959].

OSCOAP is designed for constrained nodes and networks and provides an in-layer security protocol for CoAP which does not depend on underlying layers. OSCOAP can be used anywhere that CoAP can be used, including unreliable transport [RFC7228], reliable transport [I-D.ietf-core-coap-tcp-tls], and non-IP transport [I-D.bormann-6lo-coap-802-15-ie]. OSCOAP may also be used to protect group communication for CoAP [I-D.tiloca-core-multicast-oscoap]. The use of OSCOAP does not affect the URI scheme and OSCOAP can therefore be used with any URI scheme defined for CoAP. The application decides the conditions for which OSCOAP is required.

OSCOAP builds on CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg], providing end-to-end encryption, integrity,

replay protection, and secure message binding. A compressed version of COSE is used, see Section 8. The use of OSCOAP is signaled with the CoAP option Object-Security, defined in Section 2. OSCOAP provides protection of CoAP payload, certain options, and header fields. The solution transforms an unprotected CoAP message into a protected CoAP message in the following way: the unprotected CoAP message is protected by including payload (if present), certain options, and header fields in a COSE object. The message fields that have been encrypted are removed from the message whereas the Object-Security option and the compressed COSE object are added, see Figure 1.

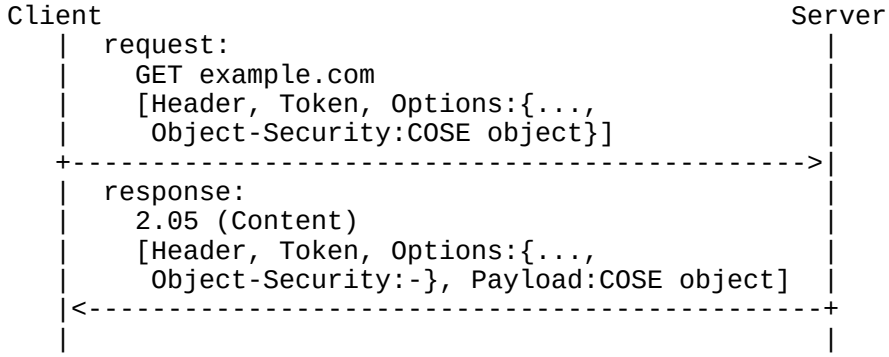


Figure 1: Sketch of OSCOAP

OSCOAP may be used in extremely constrained settings, where CoAP over DTLS may be prohibitive e.g. due to large code size. Alternatively, OSCOAP can be combined with DTLS, thereby enabling end-to-end security of e.g. CoAP payload and options, in combination with hop-by-hop protection of the entire CoAP message, during transport between end-point and intermediary node. Examples of the use of OSCOAP are given in Appendix B.

The message protection provided by OSCOAP can alternatively be applied only to the payload of individual messages. We call this object security of content (OSCON), which is defined in Appendix C.

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]. These words may also appear in this document in lowercase, absent their normative meanings.

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252], Observe [RFC7641], Blockwise [RFC7959], COSE [I-D.ietf-cose-msg], CBOR [RFC7049], CDDL [I-D.greevenbosch-appsawg-cbor-cddl], and constrained environments [RFC7228].

The terms Common/Sender/Recipient Context, Master Secret/Salt, Sender ID/Key/IV, Recepient ID/Key/IV and Context IV are defined in Section 3.1.

2. The Object-Security Option

The Object-Security option (see Figure 2) indicates that OSCOAP is used to protect the CoAP message exchange. The Object-Security option is critical, safe to forward, part of the cache key, not repeatable, and opaque.

No.	C	U	N	R	Name	Format	Length
TBD	x				Object-Security	opaque	0-

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Figure 2: The Object-Security Option

A successful response to a request with the Object-Security option SHALL contain the Object-Security option. A CoAP endpoint SHOULD NOT cache a response to a request with an Object-Security option, since the response is only applicable to the original client's request. The Object-Security option is included in the cache key for backward compatibility with proxies not recognizing the Object-Security option. The effect is that messages with the Object-Security option will never generate cache hits. For Max-Age processing, see Section 4.3.1.1.

The protection is achieved by means of a COSE object (see Section 5), which is compressed and then included in the protected CoAP message. The placement of the COSE object depends on whether the method/ response code allows payload (see [RFC7252]):

- If the method/response code allows payload, then the compressed COSE object Section 8 is the payload of the protected message, and the Object-Security option has length zero. An endpoint receiving a CoAP message with payload, that also contains a non-empty Object-Security option SHALL treat it as malformed and reject it.

- If the method/response code does not allow payload, then the compressed COSE object Section 8 is the value of the Object-Security option and the length of the Object-Security option is equal to the size of the compressed COSE object. An endpoint receiving a CoAP message without payload, that also contains an empty Object-Security option SHALL treat it as malformed and reject it.

The size of the COSE object depends on whether the method/response code allows payload, if the message is a request or response, on the set of options that are included in the unprotected message, the AEAD algorithm, the length of the information identifying the security context, and the length of the sequence number.

3. The Security Context

OSCOAP uses COSE with an Authenticated Encryption with Additional Data (AEAD) algorithm between a CoAP client and a CoAP server. An implementation supporting this specification MAY only implement the client part or MAY only implement the server part.

This specification requires that client and server establish a security context to apply to the COSE objects protecting the CoAP messages. In this section we define the security context, and also specify how to derive the initial security contexts in client and server based on common shared secret and a key derivation function (KDF).

3.1. Security Context Definition

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCOAP. For each endpoint, the security context is composed of a "Common Context", a "Sender Context", and a "Recipient Context".

The endpoints protect messages to send using the Sender Context and verify messages received using the Recipient Context, both contexts being derived from the Common Context and other data. Clients need to be able to retrieve the correct security context to use.

An endpoint uses its Sender ID (SID) to derive its Sender Context, and the other endpoint uses the same ID, now called Recipient ID (RID), to derive its Recipient Context. In communication between two endpoints, the Sender Context of one endpoint matches the Recipient Context of the other endpoint, and vice versa. Thus the two security contexts identified by the same IDs in the two endpoints are not the same, but they are partly mirrored. Retrieval and use of the security context are shown in Figure 3.

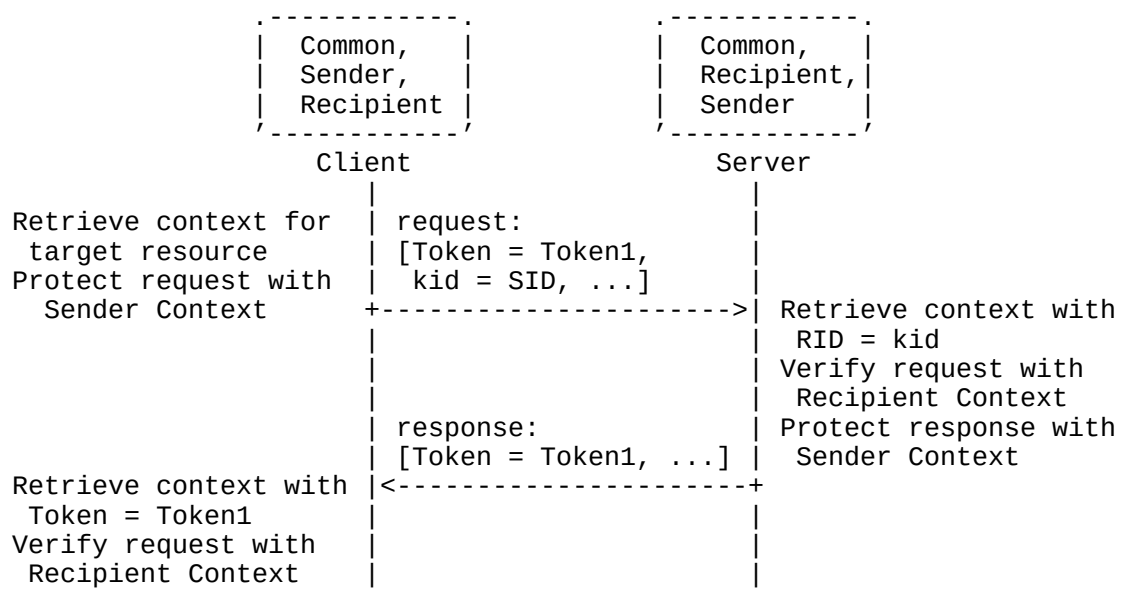


Figure 3: Retrieval and use of the Security Context

The Common Context contains the following parameters:

- Algorithm (Alg). Value that identifies the COSE AEAD algorithm to use for encryption. Its value is immutable once the security context is established.

- Master Secret. Variable length, uniformly random byte string containing the key used to derive traffic keys and IVs. Its value is immutable once the security context is established.

- Master Salt (OPTIONAL). Variable length byte string containing the salt used to derive traffic keys and IVs. Its value is immutable once the security context is established.

The Sender Context contains the following parameters:

- Sender ID. Variable length byte string identifying the Sender Context. Its value is immutable once the security context is established.

- Sender Key. Byte string containing the symmetric key to protect messages to send. Derived from Common Context and Sender ID. Length is determined by Algorithm. Its value is immutable once the security context is established.

- Sender IV. Byte string containing the IV to protect messages to send. Derived from Common Context and Sender ID. Length is determined by Algorithm. Its value is immutable once the security context is established.

- Sequence Number. Non-negative integer used to protect requests and observe responses to send. Used as partial IV [I-D.ietf-cose-msg] to generate unique nonces for the AEAD. Maximum value is determined by Algorithm.

The Recipient Context contains the following parameters:

- Recipient ID. Variable length byte string identifying the Recipient Context. Its value is immutable once the security context is established.

- Recipient Key. Byte string containing the symmetric key to verify messages received. Derived from Common Context and Recipient ID. Length is determined by the Algorithm. Its value is immutable once the security context is established.

- Recipient IV. Byte string containing the IV to verify messages received. Derived from Common Context and Recipient ID. Length is determined by Algorithm. Its value is immutable once the security context is established.

- Replay Window. The replay window to verify requests and observe responses received.

When it is understood which context is referred to (Sender Context or Recipient Context), the term "Context IV" is used to denote the IV currently used with this context.

An endpoint may free up memory by not storing the Sender Key, Sender IV, Recipient Key, and Recipient IV, deriving them from the Common Context when needed. Alternatively, an endpoint may free up memory by not storing the Master Secret and Master Salt after the other parameters have been derived.

The endpoints MAY interchange the client and server roles while maintaining the same security context. When this happens, the former server still protects messages to send using its Sender Context, and verifies messages received using its Recipient Context. The same is also true for the former client. The endpoints MUST NOT change the Sender/Recipient ID. In other words, changing the roles does not change the set of keys to be used.

3.2. Derivation of Security Context Parameters

The parameters in the security context are derived from a small set of input parameters. The following input parameters SHALL be pre-established:

- Master Secret
- Sender ID
- Recipient ID

The following input parameters MAY be pre-established. In case any of these parameters is not pre-established, the default value indicated below is used:

- AEAD Algorithm (Alg)
  - Default is AES-CCM-64-64-128 (COSE abbreviation: 12)
- Master Salt
  - Default is the empty string
- Key Derivation Function (KDF)
  - Default is HKDF SHA-256
- Replay Window Type and Size
  - Default is DTLS-type replay protection with a window size of 32

How the input parameters are pre-established, is application specific. The EDHOC protocol [I-D.selander-ace-cose-ecdhe] enables the establishment of input parameters with the property of forward secrecy and negotiation of KDF and AEAD, it thus provides all necessary pre-requisite steps for using OSCOAP as defined here.

3.2.1. Derivation of Sender Key/IV, Recipient Key/IV

The KDF MUST be one of the HMAC based HKDF [RFC5869] algorithms defined in COSE. HKDF SHA-256 is mandatory to implement. The security context parameters Sender Key/IV and Recipient Key/IV SHALL be derived from the input parameters using the HKDF, which consists of the composition of the HKDF-Extract and HKDF-Expand steps ([RFC5869]):

```
output parameter = HKDF(salt, IKM, info, L)
```

where:

- salt is the Master Salt as defined above

- IKM is the Master Secret is defined above

- info is a CBOR array consisting of:

```
info = [
    id : bstr,
    alg : int,
    type : tstr,
    L : int
]
```

  * id is the Sender ID or Recipient ID

  * type is "Key" or "IV"

- L is the size of the key/IV for the AEAD algorithm used, in octets.

For example, if the algorithm AES-CCM-64-64-128 (see Section 10.2 in [I-D.ietf-cose-msg]) is used, the value for L is 16 for keys and 7 for IVs.

3.2.2. Initial Sequence Numbers and Replay Window

The Sequence Number is initialized to 0. The supported types of replay protection and replay window length is application specific and depends on the lower layers. Default is DTLS-type replay protection with a window size of 32 initiated as described in Section 4.1.2.6 of [RFC6347].

3.3. Requirements on the Security Context Parameters

As collisions may lead to the loss of both confidentiality and integrity, Sender ID SHALL be unique in the set of all security contexts using the same Master Secret. Normally (e.g. when using EDHOC [I-D.selander-ace-cose-ecdhe]) Sender IDs can be very short. Note that Sender IDs of different lengths can be used with the same Master Secret. E.g. the SID with value 0x00 is different from the SID with the value 0x0000. If Sender ID uniqueness cannot be guaranteed, random Sender IDs MUST be used. Random Sender IDs MUST be long enough so that the probability of collisions is negligible.

To enable retrieval of the right Recipient Context, the Recipient ID SHOULD be unique in the sets of all Recipient Contexts used by an endpoint.

The same Master Salt MAY be used with several Master Secrets.

4. Protected CoAP Message Fields

OSCOAP transforms an unprotected CoAP message into a protected CoAP message, and vice versa. This section defines how the CoAP message fields are protected. Note that OSCOAP protects messages from the CoAP Requests/Responses layer only, and not from the Messaging layer (Section 2 of [RFC7252]): this means that RST and ACK empty messages are not protected, while ACK with piggybacked responses are protected using the process defined in this document. All the messages mentioned in this document refer to CON, NON and non-empty ACK messages.

OSCOAP protects as much of the unprotected CoAP message as possible, while still allowing forward proxy operations [I-D.hartke-core-e2e-security-reqs]. Message fields may either be

- Class E: encrypted and integrity protected,
- Class I: integrity protected only, or
- Class U: unprotected.

This section also outlines how the message fields are transferred, a detailed description of the processing is provided in Section 7. Message fields of the unprotected CoAP message are either transferred in the header/options part of the protected CoAP message, or in the plaintext of the COSE object. Depending on which, the location of the message field in the protected CoAP message is called "inner" or "outer":

- Inner message field: message field included in the plaintext of the COSE object of the protected CoAP message (see Section 5.1). The inner message fields are by definition encrypted and integrity protected by the COSE object (Class E).
- Outer message field: message field included in the header or options part of the protected CoAP message. The outer message fields are not encrypted and thus visible to an intermediary, but may be integrity protected by including the message field values in the Additional Authenticated Data (AAD) of the COSE object (see Section 5.2). I.e. outer message fields may be Class I or Class U.

Note that, even though the message formats are slightly different, OSCOAP complies with CoAP over unreliable transport [RFC7252] as well as CoAP over reliable transport [I-D.ietf-core-coap-tcp-tls].

4.1. CoAP Payload

The CoAP Payload SHALL be encrypted and integrity protected (Class E), and thus is an inner message field.

The sending endpoint writes the payload of the unprotected CoAP message into the plaintext of the COSE object.

The receiving endpoint verifies and decrypts the COSE object, and recreates the payload of the unprotected CoAP message.

4.2. CoAP Header

Many CoAP header fields are required to be read and changed during a normal message exchange or when traversing a proxy and thus cannot in general be protected between the endpoints, e.g. CoAP message layer fields such as Message ID.

The CoAP header field Code MUST be sent in plaintext to support RESTful processing, but MUST be integrity protected to prevent an intermediary from changing, e.g. from GET to DELETE (Class I). The CoAP version number MUST be integrity protected to prevent potential future version-based attacks (Class I). Note that while the version number is not sent in each CoAP message over reliable transport [I-D.ietf-core-coap-tcp-tls], its value is known to client and server.

The other CoAP header fields SHALL neither be integrity protected nor encrypted (Class U). All CoAP header fields are thus outer message fields.

The sending endpoint SHALL copy the header fields from the unprotected CoAP message to the header of the protected CoAP message. The receiving endpoint SHALL copy the header fields from the protected CoAP message to the header of the unprotected CoAP message. Both sender and receiver include the CoAP version number and header field Code in the AAD of the COSE object (see Section 5.2).

4.3. CoAP Options

Most options are encrypted and integrity protected (Class E), and thus inner message fields. But to allow certain proxy operations, some options have outer values, i.e. are present as options in the protected CoAP message. Certain options may have both an inner value

and a potentially different outer value, where the inner value is intended for the destination endpoint and the outer value is intended for the proxy.

A summary of how options are protected and processed is shown in Figure 4. Options within each class are protected and processed in a similar way, but certain options which require special processing are indicated by a * in Figure 4 and described in the subsections below.

No.	Name	E	I	U
1	If-Match	x		
3	Uri-Host			x
4	ETag	x		
5	If-None-Match	x		
6	Observe		*	
7	Uri-Port			x
8	Location-Path	x		
11	Uri-Path	x		
12	Content-Format	x		
14	Max-Age	*		
15	Uri-Query	x		
17	Accept	x		
20	Location-Query	x		
23	Block2	*		
27	Block1	*		
28	Size2	*		
35	Proxy-Uri			*
39	Proxy-Scheme			x
60	Size1	*		

E=Encrypt and Integrity Protect, I=Integrity Protect only, U=Unprotected, *=Special

Figure 4: Protection of CoAP Options

Unless specified otherwise, CoAP options not listed in Figure 4 SHALL be encrypted and integrity protected and processed as class E options.

Specifications of new CoAP options SHOULD define how they are processed with OSCOAP. New COAP options SHOULD be of class E and SHOULD NOT have outer values unless a forwarding proxy needs to read that option value. If a certain option has both inner and outer values, the two values SHOULD NOT be the same.

4.3.1. Class E Options

For options in class E (see Figure 4) the option value in the unprotected CoAP message, if present, SHALL be encrypted and integrity protected between the endpoints. Hence the actions resulting from the use of such options is analogous to communicating in a protected manner directly with the endpoint. For example, a client using an If-Match option will not be served by a proxy.

The sending endpoint SHALL write the class E option from the unprotected CoAP message into the plaintext of the COSE object.

Except for the special options described in the subsections, the sending endpoint SHALL NOT use the outer options of class E. However, note that an intermediary may, legitimately or not, add, change or remove the value of an outer option.

Except for the Block options Section 4.3.1.2, the receiving endpoint SHALL discard any outer options of class E from the protected CoAP message and SHALL write the Class E options present in the plaintext of the COSE object into the unprotected CoAP message.

4.3.1.1. Max-Age

An inner Max-Age option, like other class E options, is used as defined in [RFC7252] taking into account that it is not accessible to proxies.

Since OSCOAP binds CoAP responses to requests, a cached response would not be possible to use for any other request. To avoid unnecessary caching, a server MAY add an outer Max-Age option with value zero to protected CoAP responses (see Section 5.6.1 of [RFC7252]). The outer Max-Age option is not integrity protected.

4.3.1.2. The Block Options

Blockwise [RFC7959] is an optional feature. An implementation MAY comply with [RFC7252] and the Object-Security option without implementing [RFC7959].

The Block options (Block1, Block2, Size1 and Size2) MAY be either only inner options, only outer options or both inner and outer options. The inner and outer options are processed independently.

The inner block options are used for endpoint-to-endpoint secure fragmentation of payload into blocks and protection of information about the fragmentation (block number, block size, last block). In this case, the CoAP client fragments the CoAP message as defined in

[RFC7959] before the message is processed by OSCOAP. The CoAP server first processes the OSCOAP message before processing blockwise as defined in [RFC7959].

There SHALL be a security policy defining a maximum unfragmented message size for inner Block options such that messages exceeding this size SHALL be fragmented by the sending endpoint.

Additionally, a proxy may arbitrarily do block fragmentation on any CoAP message, in particular an OSCOAP message, as defined in [RFC7959] and thereby add outer Block options to a block and send on the next hop. The outer block options are thus neither encrypted nor integrity protected.

An endpoint receiving a message with an outer Block option SHALL first process this option according to [RFC7959], until all blocks of the protected CoAP message has been received, or the cumulated message size of the exceeds the maximum unfragmented message size. In the latter case the message SHALL be discarded. In the former case, the processing of the protected CoAP message continues as defined in this document.

If the unprotected CoAP message in turn contains Block options, the receiving endpoint processes this according to [RFC7959].

TODO: Update processing to support multiple concurrently proceeding requests

4.3.2. Class I Options

A Class I option is an outer option and hence visible in the options part of the protected CoAP message. Except for special options described in the subsections, for options in Class I (see Figure 4) the option value SHALL be integrity protected between the endpoints, see (Section 5.2). Unless otherwise specified, the sending endpoint SHALL encode the Class I options in the protected CoAP message as described in Section 4.3.4.

4.3.2.1. Observe

Observe [RFC7641] is an optional feature. An implementation MAY support [RFC7252] and the Object-Security option without supporting [RFC7641]. The Observe option as used here targets the requirements on forwarding of [I-D.hartke-core-e2e-security-reqs] (Section 2.2.1.2).

In order for a proxy to support forwarding of Observe messages, there must be an Observe option present in options part of the protected CoAP message ([RFC7641]), so Observe must have an outer value:

- The Observe option of the unprotected CoAP request SHALL be encoded in the protected CoAP request as described in Section 4.3.4.

To secure the order of the notifications, responses with the Observe option SHALL be integrity protected in the following way:

- The Observe option SHALL be included in the external_aad of the response (see Section 5.2), with value set to the 3 least significant bytes of the Sequence Number of the response.

The Observe option in the CoAP request SHALL NOT be integrity protected, since it may be legitimately removed by a proxy.

If the Observe option is removed from a CoAP request by a proxy, then the server can still verify the request (as a non-Observe request), and produce a non-Observe response. If the OSCOAP client receives a response to an Observe request without an outer Observe value, then it MUST verify the response as a non-Observe response, i.e. not include the Sequence Number of the response in the external_aad.

4.3.3. Class U Options

Options in Class U have outer values and are used to support forward proxy operations. Unless otherwise specified, the sending endpoint SHALL encode the Class U options in the options part of the protected CoAP message as described in Section 4.3.4.

4.3.3.1. Uri-Host, Uri-Port, and Proxy-Scheme

The sending endpoint SHALL copy Uri-Host, Uri-Port, and Proxy-Scheme from the unprotected CoAP message to the options part of the protected CoAP message. When Uri-Host, Uri-Port, or Proxy-Scheme options are present, Proxy-Uri is not used [RFC7252].

4.3.3.2. Proxy-Uri

Proxy-Uri, when present, is split by OSCOAP into class U options and class E options, which are processed accordingly. When Proxy-Uri is used in the unprotected CoAP message, Uri-* are not present [RFC7252].

The sending endpoint SHALL first decompose the Proxy-Uri value of the unprotected CoAP message into the Proxy-Scheme, Uri-Host, Uri-Port,

Uri-Path and Uri-Query options (if present) according to section 6.4 of [RFC7252].

Uri-Path and Uri-Query are class E options and MUST be protected and processed as if obtained from the unprotected CoAP message, see Section 4.3.1.

The value of the Proxy-Uri option of the protected CoAP message MUST be replaced with Proxy-Scheme, Uri-Host and Uri-Port options (if present) composed according to section 6.5 of [RFC7252] and MUST be processed as a class U option, see Section 4.3.3.

An example of how Proxy-Uri is processed is given here. Assume that the unprotected CoAP message contains:

- Proxy-Uri = "coap://example.com/resource?q=1"

During OSCOAP processing, Proxy-Uri is split into:

- Proxy-Scheme = "coap"
- Uri-Host = "example.com"
- Uri-Port = "5863"
- Uri-Path = "resource"
- Uri-Query = "q=1"

Uri-Path and Uri-Query follow the processing defined in Section 4.3.1, and are thus encrypted and transported in the COSE object. The remaining options are composed into the Proxy-Uri included in the options part of the protected CoAP message, which has value:

- Proxy-Uri = "coap://example.com"

4.3.4. Outer Options in the Protected CoAP Message

All options with outer values present in the protected CoAP message, including the Object-Security option, SHALL be encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included outer option value.

5. The COSE Object

This section defines how to use COSE [I-D.ietf-cose-msg] to wrap and protect data in the unprotected CoAP message. OSCOAP uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Additional Data (AEAD) algorithm. The key lengths, IV lengths, and maximum sequence number are algorithm dependent.

The AEAD algorithm AES-CCM-64-64-128 defined in Section 10.2 of [I-D.ietf-cose-msg] is mandatory to implement. For AES-CCM-64-64-128 the length of Sender Key and Recipient Key is 128 bits, the length of nonce, Sender IV, and Recipient IV is 7 bytes. The maximum Sequence Number is specified in Section 10.

The nonce is constructed as described in Section 3.1 of [I-D.ietf-cose-msg], i.e. by padding the partial IV (Sequence Number in network byte order) with zeroes and XORing it with the Context IV (Sender IV or Recipient IV), with the following addition: The most significant bit in the first byte of the Context IV SHALL be flipped for responses, in case there is a unique response (not Observe). In this way, the same sequence number can be reused for requests and corresponding responses, which reduces the size of the responses in the most common case. For detailed processing instructions, see Section 7.

We denote by Plaintext the data that is encrypted and integrity protected, and by Additional Authenticated Data (AAD) the data that is integrity protected only.

The COSE Object SHALL be a COSE_Encrypt0 object with fields defined as follows

- The "protected" field is empty.
- The "unprotected" field includes:
  - The "Partial IV" parameter. The value is set to the Sequence Number. The Partial IV SHALL be of minimum length needed to encode the sequence number. This parameter SHALL be present in requests. In case of Observe (Section 4.3.2.1) the Partial IV SHALL be present in the response, and otherwise the Partial IV SHALL NOT be present in the response.
  - The "kid" parameter. The value is set to the Sender ID (see Section 3). This parameter SHALL be present in requests and SHALL NOT be present in responses.

- The "ciphertext" field is computed from the Plaintext (see Section 5.1) and the Additional Authenticated Data (AAD) (see Section 5.2) following Section 5.2 of [I-D.ietf-cose-msg].

The encryption process is described in Section 5.3 of [I-D.ietf-cose-msg].

5.1. Plaintext

The Plaintext is formatted as a CoAP message without Header (see Figure 5) consisting of:

- all Class E option values Section 4.3.1 present in the unprotected CoAP message (see Section 4.3). The options are encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included Class E option; and

- the Payload of unprotected CoAP message, if present, and in that case prefixed by the one-byte Payload Marker (0xFF).

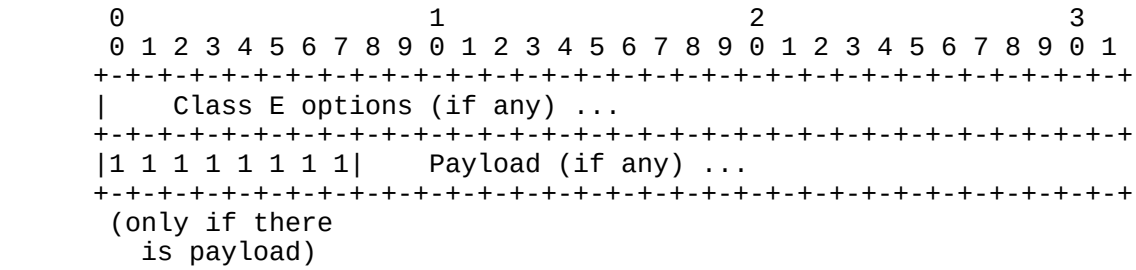


Figure 5: Plaintext

5.2. Additional Authenticated Data

The external_aad SHALL be a CBOR array as defined below:

```
external_aad = [
   ver : uint,
   code : uint,
   options : bstr,
   alg : int,
   request_kid : bstr,
   request_seq : bstr
]
```

where:

- ver: contains the CoAP version number, as defined in Section 3 of [RFC7252].
- code: contains is the CoAP Code of the unprotected CoAP message, as defined in Section 3 of [RFC7252].
- options: contains the Class I options Section 4.3.2 present in the unprotected CoAP message encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included class I option
- alg: contains the Algorithm from the security context used for the exchange (see Section 3.1).
- request_kid: contains the value of the 'kid' in the COSE object of the request (see Section 5).
- request_seq: contains the value of the 'Partial IV' in the COSE object of the request (see Section 5).

6. Sequence Numbers, Replay, Message Binding, and Freshness

Sequence numbers and replay window are initialized as defined in Section 3.2.2.

6.1. AEAD Nonce Uniqueness

An AEAD nonce MUST NOT be used more than once per AEAD key. In order to assure unique nonces, each Sender Context contains a Sequence Number used to protect requests, and - in case of Observe - responses. The maximum sequence number is algorithm dependent, see Section 10. If the Sequence Number exceeds the maximum sequence number, the endpoint MUST NOT process any more messages with the given Sender Context. The endpoint SHOULD acquire a new security context (and consequently inform the other endpoint) before this happens. The latter is out of scope of this document.

6.2. Replay Protection

In order to protect from replay of messages, each Recipient Context contains a Replay Window used to verify request, and - in case of Observe - responses. A receiving endpoint SHALL verify that a Sequence Number (Partial IV) received in the COSE object has not been received before in the Recipient Context. For requests, if this verification fails and the message received is a CON message, the server SHALL respond with a 4.00 Bad Request error message. The diagnostic payload MAY contain the "Replay protection failed" string. For responses, if this verification fails and the message received is

a CON message, the client SHALL respond with an empty ACK and stop processing the response.

The size and type of the Replay Window depends on the use case and lower protocol layers. In case of reliable and ordered transport from endpoint to endpoint, the recipient MAY just store the last received sequence number and require that newly received Sequence Numbers equals the last received Sequence Number + 1.

6.3. Sequence Number and Replay Window State

To prevent reuse of the Nonce/Sequence Number with the same key, or from accepting replayed messages, a node needs to handle the situation of suddenly losing sequence number and replay window state in RAM, e.g. as a result of a reboot.

After boot, a node MAY reject to use existing security contexts from before it booted and MAY establish a new security context with each party it communicates, e.g. using EDHOC [I-D.selander-ace-cose-ecdhe]. However, establishing a fresh security context may have a non-negligible cost in terms of e.g. power consumption.

If a stored security context is to be used after reboot, then the node MUST NOT reuse a previous Sequence Number and MUST NOT accept previously accepted messages.

6.3.1. The Basic Case

To prevent reuse of Sequence Number, the node MAY perform the following procedure during normal operations:

- Before sending a message, the client stores in persistent memory a sequence number associated to the stored security context higher than any sequence number which has been or are being sent using this security context. After boot, the client does not use any lower sequence number in a request than what was persistently stored with that security context.
  - Storing to persistent memory can be costly. Instead of storing a sequence number for each request, the client may store Seq + K to persistent memory every K requests, where Seq is the current sequence number and K > 1. This is a trade-off between the number of storage operations and efficient use of sequence numbers.

To prevent accepting replay of previously received messages, the node MAY perform the following procedure:

- After boot, before verifying a message using a security context stored before boot, the server synchronizes the replay window so that no old messages are being accepted. The server uses the Repeat option [I-D.mattsson-core-coap-actuators] for synchronizing the replay window: For each stored security context, the first time after boot the server receives an OSCOAP request, it generates a pseudo-random nonce and responds with the Repeat option set to the nonce as described in [I-D.mattsson-core-coap-actuators]. If the server receives a repeated OSCOAP request containing the Repeat option and the same nonce, and if the server can verify the request, then the sequence number obtained in the repeated message is set as the lower limit of the replay window.

6.3.2. The Observe Case

To prevent reuse of Sequence Number in case of Observe, the node MAY perform the following procedure during normal operations:

- Before sending a notification, the server stores in persistent memory a sequence number associated to the stored security context higher than any sequence number for which a notification has been or are being sent using this security context. After boot, the server does not use any lower sequence number in an Observe response than what was persistently stored with that security context.

  * Storing to persistent memory can be costly. Instead of storing a sequence number for each notification, the server may store Seq + K to persistent memory every K requests, where Seq is the current sequence number and K > 1. This is a trade-off between the number of storage operations and efficient use of sequence numbers.

Note that a client MAY continue an ongoing observation after reboot using a stored security context. With Observe, the client can only verify the order of the notifications, as they may be delayed. If the client wants to synchronize with a server resource it MAY restart an observation.

6.4. Freshness

For responses without Observe, OSCOAP provides absolute freshness. For requests, and responses with Observe, OSCOAP provides relative freshness in the sense that the sequence numbers allows a recipient to determine the relative order of messages.

For applications having stronger demands on freshness (e.g. control of actuators), OSCOAP needs to be augmented with mechanisms providing absolute freshness [I-D.mattsson-core-coap-actuators].

6.5. Delay and Mismatch Attacks

In order to prevent response delay and mismatch attacks [I-D.mattsson-core-coap-actuators] from on-path attackers and compromised proxies, OSCOAP binds responses to the request by including the request's ID (Sender ID or Recipient ID) and sequence number in the AAD of the response. The server therefore needs to store the request's ID (Sender ID or Recipient ID) and sequence number until all responses have been sent.

7. Processing

7.1. Protecting the Request

Given an unprotected request, the client SHALL perform the following steps to create a protected request:

1. Retrieve the Sender Context associated with the target resource.

2. Compose the Additional Authenticated Data, as described in Section 5.

3. Compose the AEAD nonce by XORing the Context IV (Sender IV) with the partial IV (Sequence Number in network byte order).

4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 8.

5. Format the protected CoAP message according to Section 4. The Object-Security option is added, see Section 4.3.4.

6. Store the association Token - Security Context. The client SHALL be able to find the Recipient Context from the Token in the response.

7. Increment the Sequence Number by one.

7.2. Verifying the Request

A server receiving a request containing the Object-Security option SHALL perform the following steps:

1. Process outer Block options according to [RFC7959], until all blocks of the request have been received, see Section 4.3.1.2.

2. Decompress the COSE Object (Section 8) and retrieve the Recipient Context associated with the Recipient ID in the 'kid' parameter. If the request is a CON message, and:

   * either the decompression or the COSE message fails to decode, the server SHALL respond with a 4.02 Bad Option error message. The diagnostic payload SHOULD contain the string "Failed to decode COSE".

   * the server fails to retrieve a Recipient Context with Recipient ID corresponding to the 'kid' parameter received, the server SHALL respond with a 4.01 Unauthorized error message. The diagnostic payload MAY contain the string "Security context not found".

If the request is a NON message and either the decompression or the COSE message fails to decode, or the server fails to retrieve a Recipient Context with Recipient ID corresponding to the 'kid' parameter received, then the server SHALL stop processing the request.

1. Verify the Sequence Number in the 'Partial IV' parameter, as described in Section 6.

2. Compose the Additional Authenticated Data, as described in Section 5.

3. Compose the AEAD nonce by XORing the Context IV (Recipient IV) with the padded 'Partial IV' parameter, received in the COSE Object.

4. Decrypt the COSE object using the Recipient Key.

   * If decryption fails, the server MUST stop processing the request and, if the request is a CON message, the server MUST respond with a 4.00 Bad Request error message. The diagnostic payload MAY contain the "Decryption failed" string.

   * If decryption succeeds, update the Recipient Replay Window, as described in Section 6.

5. Add decrypted options and payload to the unprotected request, processing the E options as described in (Section 4). The Object-Security option is removed.

6. The unprotected CoAP request is processed according to [RFC7252]

7.3. Protecting the Response

Given an unprotected response, the server SHALL perform the following steps to create a protected response:

1. Retrieve the Sender Context in the Security Context used to verify the request.

2. Compose the Additional Authenticated Data, as described in Section 5.

3. Compose the AEAD nonce

   * If Observe is not used, compose the AEAD nonce by XORing the Context IV (Sender IV with the most significant bit in the first byte flipped) with the padded Partial IV parameter from the request.

   * If Observe is used, compose the AEAD nonce by XORing the Context IV (Sender IV) with the Partial IV of the response (Sequence Number in network byte order).

4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 8.

5. Format the protected CoAP message according to Section 4. The Object-Security option is added, see Section 4.3.4.

6. If Observe is used, increment the Sequence Number by one.

7.4. Verifying the Response

A client receiving a response containing the Object-Security option SHALL perform the following steps:

1. Process outer Block options according to [RFC7959], until all blocks of the protected CoAP message have been received, see Section 4.3.1.2.

2. Retrieve the Recipient Context associated with the Token. Decompress the COSE Object (Section 8). If the response is a CON message and either the decompression or the COSE message fails to decode, then the client SHALL send an empty ACK back and stop processing the response. If the response is a NON message and any of the previous conditions appear, then the client SHALL simply stop processing the response.

1. For Observe notifications, verify the Sequence Number in the 'Partial IV' parameter as described in Section 6.

2. Compose the Additional Authenticated Data, as described in Section 5.

3. Compose the AEAD nonce

   * If the Observe option is not present in the response, compose the AEAD nonce by XORing the Context IV (Recipient IV with the the most significant bit in the first byte flipped) with the padded Partial IV parameter from the request.

   * If the Observe option is present in the response, compose the AEAD nonce by XORing the Context IV (Recipient IV) with the padded Partial IV parameter from the response.

4. Decrypt the COSE object using the Recipient Key.

   * If decryption fails, the client MUST stop processing the response and, if the request is a CON message, the client MUST respond with an empty ACK back.

   * If decryption succeeds and Observe is used, update the Recipient Replay Window, as described in Section 6.

5. Add decrypted options or payload to the unprotected response overwriting any outer E options (see Section 4). The Object-Security option is removed.

   * If Observe is used, replace the Observe value with the 3 least significant bytes in the sequence number.

6. The unprotected CoAP response is processed according to [RFC7252]

8. OSCOAP Compression

The Concise Binary Object Representation (CBOR) [RFC7049] combines very small message sizes with extensibility. The CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg] uses CBOR to create compact encoding of signed and encrypted data. COSE is however constructed to support a large number of different stateless use cases, and is not fully optimized for use as a stateful security protocol, leading to a larger than necessary message expansion. In this section we define a simple stateless compression mechanism for OSCOAP, which significantly reduces the per-packet overhead.

8.1. Encoding of the Object-Security Option

The value of the Object-Security option SHALL be encoded as follows:

- The first byte MUST encode a set of flags and the length of the Partial IV parameter.
  - The three least significant bits encode the Partial IV size. If their value is 0, the Partial IV is not present in the compressed message.
  - The fourth least significant bit is set to 1 if the kid is present in the compressed message.
  - The fifth-eighth least significant bits (= most significant half-byte) are reserved and SHALL be set to zero when not in use.
- The following n bytes (n being the value of the Partial IV size in the first byte) encode the value of the Partial IV, if the Partial IV is present (size not 0).
- The following byte encodes the size of the kid parameter, if the kid is present (flag bit set to 1)
- The following m bytes (m given by the previous byte) encode the value of the kid, if the kid is present (flag bit set to 1)
- The remainining bytes encode the ciphertext.

The presence of Partial IV and kid in requests and responses is specified in Section 5, and summarized in Figure 6.

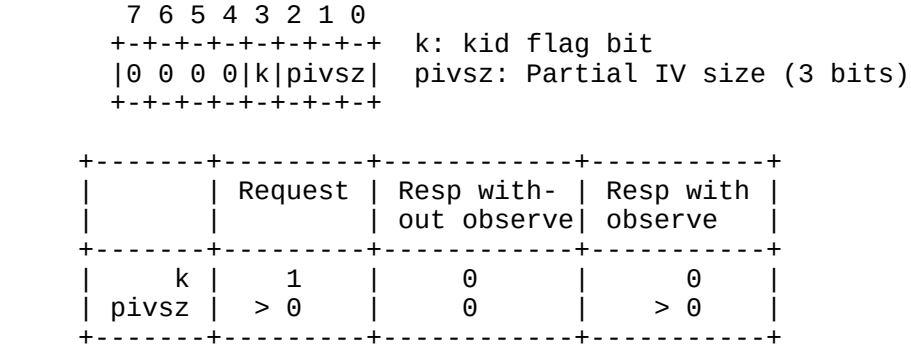


Figure 6: Flag byte for OSCOAP compression

8.2. Examples

This section provides examples of COSE Objects before and after OSCOAP compression.

8.2.1. Example: Request

Before compression:

```
[
h'',
{ 4:h'25', 6:h'05' },
h'aea0155667924dff8a24e4cb35b9'
]
```

```
0x83 40 a2 04 41 25 06 41 05 4e ae a0 15 56 67 92
4d ff 8a 24 e4 cb 35 b9 (24 bytes)
```

After compression:

First byte: 0b00001001 = 0x09

```
0x09 05 01 25 ae a0 15 56 67 92 4d ff 8a 24 e4 cb
35 b9 (18 bytes)
```

8.2.2. Example: Response (without Observe)

Before compression:

```
[
h'',
{},
h'aea0155667924dff8a24e4cb35b9'
]
```

```
0x83 40 a0 4e ae a0 15 56 67 92 4d ff 8a 24 e4 cb
35 b9 (18 bytes)
```

After compression:

First byte: 0b00000000 = 0x00

```
0x00 ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9
(15 bytes)
```

8.2.3. Example: Response (with Observe)

Before compression:

```
[
h'',
{ 6:h'07' },
h'aea0155667924dff8a24e4cb35b9'
]
```

```
0x83 40 a1 06 41 07 4e ae a0 15 56 67 92 4d ff
8a 24 e4 cb 35 b9 (21 bytes)
```

After compression:

First byte: 0b00000001 = 0x01

```
0x01 07 ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9
(16 bytes)
```

9. Web Linking

The use of OSCOAP MAY be indicated by a target attribute "osc" in a web link [RFC5988] to a CoAP resource. This attribute is a hint indicating that the destination of that link is to be accessed using OSCOAP. Note that this is simply a hint, it does not include any security context material or any other information required to run OSCOAP.

A value MUST NOT be given for the "osc" attribute; any present value MUST be ignored by parsers. The "osc" attribute MUST NOT appear more than once in a given link-value; occurrences after the first MUST be ignored by parsers.

10. Security Considerations

In scenarios with intermediary nodes such as proxies or brokers, transport layer security such as DTLS only protects data hop-by-hop. As a consequence the intermediary nodes can read and modify information. The trust model where all intermediate nodes are considered trustworthy is problematic, not only from a privacy perspective, but also from a security perspective, as the intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases, where all the owners of the intermediary nodes are fully trusted, attacks and data breaches make such an architecture brittle.

DTLS protects hop-by-hop the entire CoAP message, including header, options, and payload. OSCOAP protects end-to-end the payload, and all information in the options and header, that is not required for forwarding (see Section 4). DTLS and OSCOAP can be combined, thereby enabling end-to-end security of CoAP payload, in combination with hop-by-hop protection of the entire CoAP message, during transport between end-point and intermediary node.

The CoAP message layer, however, cannot be protected end-to-end through intermediary devices since the parameters Type and Message ID, as well as Token and Token Length may be changed by a proxy. Moreover, messages that are not possible to verify should for security reasons not always be acknowledged but in some cases be silently dropped. This would not comply with CoAP message layer, but does not have an impact on the application layer security solution, since message layer is excluded from that.

The use of COSE to protect CoAP messages as specified in this document requires an established security context. The method to establish the security context described in Section 3.2 is based on a common shared secret material in client and server, which may be obtained e.g. by using EDHOC [I-D.selander-ace-cose-ecdhe] or the ACE framework [I-D.ietf-ace-oauth-authz]. An OSCOAP profile of ACE is described in [I-D.seitz-ace-oscoap-profile].

The mandatory-to-implement AEAD algorithm AES-CCM-64-64-128 is selected for broad applicability in terms of message size (2^64 blocks) and maximum number of messages (2^56). Compatibility with CCM* is achieved by using the algorithm AES-CCM-16-64-128 [I-D.ietf-cose-msg].

Most AEAD algorithms require a unique nonce for each message, for which the sequence numbers in the COSE message field "Partial IV" is used. If the recipient accepts any sequence number larger than the one previously received, then the problem of sequence number synchronization is avoided. With reliable transport it may be defined that only messages with sequence number which are equal to previous sequence number + 1 are accepted. The alternatives to sequence numbers have their issues: very constrained devices may not be able to support accurate time, or to generate and store large numbers of random nonces. The requirement to change key at counter wrap is a complication, but it also forces the user of this specification to think about implementing key renewal.

The maximum sequence number to guarantee nonce uniqueness (Section 6.1) is algorithm dependent. Using AES_CCM, with the maximum sequence number SHALL be 2^(min(nonce length in bits, 56) - 1) - 1. The "-1" in the exponent stems from the same partial IV and

flipped bit of IV (Section 5) is used in request and response. The compression algorithm (Section 8) assumes that the partial IV is 56 bits or less (which is the reason for min(,) in the exponent).

The inner block options enable the sender to split large messages into protected blocks such that the receiving node can verify blocks before having received the complete message. The outer block options allow for arbitrary proxy fragmentation operations that cannot be verified by the endpoints, but can by policy be restricted in size since the encrypted options allow for secure fragmentation of very large messages. A maximum message size (above which the sending endpoint fragments the message and the receiving endpoint discards the message, if complying to the policy) may be obtained as part of normal resource discovery.

Applications need to use a padding scheme if the content of a message can be determined solely from the length of the payload. As an example, the strings "YES" and "NO" even if encrypted can be distinguished from each other as there is no padding supplied by the current set of encryption algorithms. Some information can be determined even from looking at boundary conditions. An example of this would be returning an integer between 0 and 100 where lengths of 1, 2 and 3 will provide information about where in the range things are. Three different methods to deal with this are: 1) ensure that all messages are the same length. For example using 0 and 1 instead of ’yes’ and ’no’. 2) Use a character which is not part of the responses to pad to a fixed length. For example, pad with a space to three characters. 3) Use the PKCS #7 style padding scheme where m bytes are appended each having the value of m. For example, appending a 0 to "YES" and two 1’s to "NO". This style of padding means that all values need to be padded.

11. Privacy Considerations

Privacy threats executed through intermediate nodes are considerably reduced by means of OSCOAP. End-to-end integrity protection and encryption of CoAP payload and all options that are not used for forwarding, provide mitigation against attacks on sensor and actuator communication, which may have a direct impact on the personal sphere.

The unprotected options (Figure 4) may reveal privacy sensitive information. In particular Uri-Host SHOULD NOT contain privacy sensitive information.

CoAP headers sent in plaintext allow for example matching of CON and ACK (CoAP Message Identifier), matching of request and responses (Token) and traffic analysis.

12. IANA Considerations

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

12.1. CoAP Option Numbers Registry

The Object-Security option is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD	Object-Security	[[this document]]

12.2. Media Type Registrations

The "application/oscon" media type is added to the Media Types registry:

Type name: application

Subtype name: oscon

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Appendix C of this document.

Interoperability considerations: N/A

Published specification: [[this document]] (this document)

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): N/A

* Macintosh file type code(s): N/A

Person & email address to contact for further information:
Goeran Selander <goran.selander@ericsson.com>

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander, goran.selander@ericsson.com

12.3. CoAP Content Format Registration

The "application/oscon" content format is added to the CoAP Content Format registry:

Media type	Encoding	ID	Reference
application/oscon	-	70	[[this document]]

13. Acknowledgments

The following individuals provided input to this document: Christian Amsuess, Carsten Bormann, Joakim Brorsson, Martin Gunnarsson, Klaus Hartke, Jim Schaad, Marco Tiloca, and Malisa Vučinić.

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova.

14. References

14.1. Normative References

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

[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>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <http://www.rfc-editor.org/info/rfc5869>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[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>.

[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>.

[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>.

14.2. Informative References

[I-D.bormann-6lo-coap-802-15-ie]
Bormann, C., "Constrained Application Protocol (CoAP) over IEEE 802.15.4 Information Element for IETF", draft-bormann-6lo-coap-802-15-ie-00 (work in progress), April 2016.

[I-D.greevenbosch-appsawg-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-10 (work in progress), March 2017.

[I-D.hartke-core-e2e-security-reqs]
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-02 (work in progress), January 2017.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-06 (work in progress), March 2017.

[I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-08 (work in progress), April 2017.

[I-D.mattsson-core-coap-actuators]
Mattsson, J., Fornehed, J., Selander, G., and F. Palombini, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-02 (work in progress), November 2016.

[I-D.seitz-ace-oscoap-profile]
Seitz, L. and F. Palombini, "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-01 (work in progress), October 2016.

[I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-06 (work in progress), April 2017.

[I-D.tiloca-core-multicast-oscoap]
Tiloca, M., Selander, G., and F. Palombini, "Secure group communication for CoAP", draft-tiloca-core-multicast-oscoap-01 (work in progress), March 2017.

[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>.

Appendix A. Test Vectors

TODO: This section needs to be updated.

Appendix B. Examples

This section gives examples of OSCOAP. The message exchanges are made, based on the assumption that there is a security context established between client and server. For simplicity, these examples only indicate the content of the messages without going into detail of the COSE message format.

B.1. Secure Access to Sensor

This example targets the scenario in Section 3.1 of [I-D.hartke-core-e2e-security-reqs] and illustrates a client requesting the alarm status from a server.

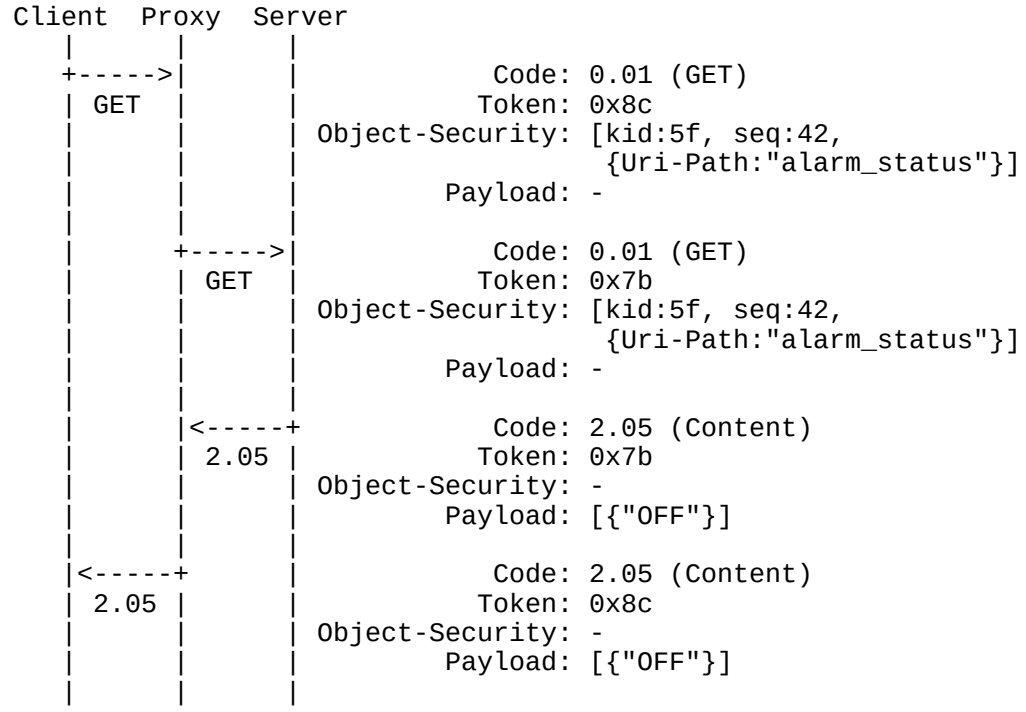


Figure 7: Secure Access to Sensor. Square brackets [...] indicate a COSE object. Curly brackets { ... } indicate encrypted data.

Since the method (GET) doesn't allow payload, the Object-Security option carries the COSE object as its value. Since the response code (Content) allows payload, the COSE object is carried as the CoAP payload.

The COSE header of the request contains an identifier (5f), indicating which security context was used to protect the message and a sequence number (42). The option Uri-Path ("alarm_status") and payload ("OFF") are encrypted.

The server verifies that the sequence number has not been received before. The client verifies that the response is bound to the request.

B.2. Secure Subscribe to Sensor

This example targets the scenario in Section 3.2 of [I-D.hartke-core-e2e-security-reqs] and illustrates a client requesting subscription to a blood sugar measurement resource (GET

/glucose), first receiving the value 220 mg/dl and then a second value 180 mg/dl.

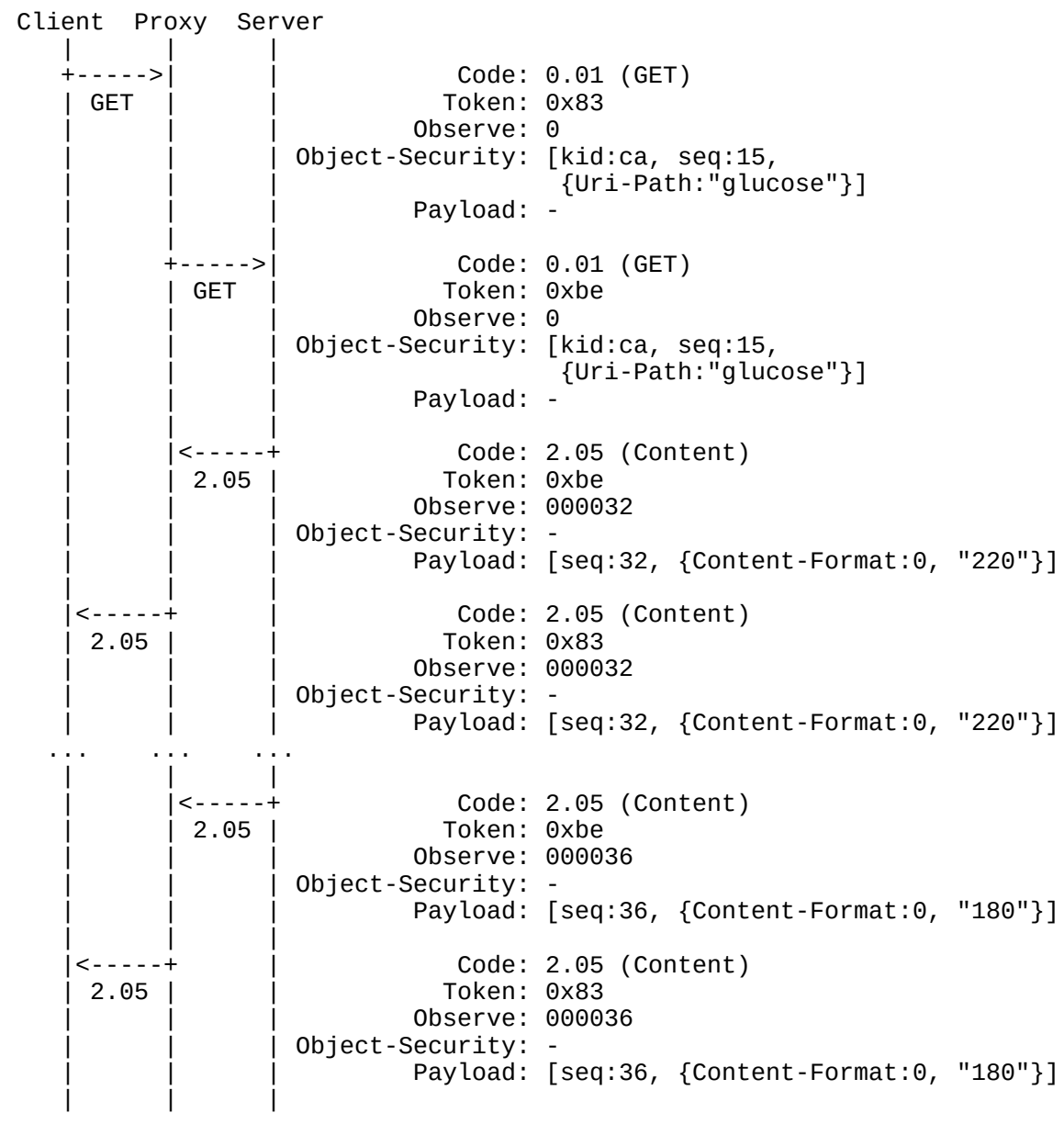


Figure 8: Secure Subscribe to Sensor. Square brackets [...] indicate a COSE object. Curly brackets { ... } indicate encrypted data.

Since the method (GET) doesn't allow payload, the Object-Security option carries the COSE object as its value. Since the response code (Content) allows payload, the COSE object is carried as the CoAP payload.

The COSE header of the request contains an identifier (ca), indicating the security context used to protect the message and a Sequence Number (15). The COSE header of the responses contains sequence numbers (32 and 36). The options Content-Format (0) and the payload ("220" and "180"), are encrypted. The Observe option is integrity protected. The shown Observe values (000032 and 000036) are the ones that the client will see after OSCOAP processing.

The server verifies that the sequence number has not been received before. The client verifies that the sequence number has not been received before and that the responses are bound to the request.

Appendix C. Object Security of Content (OSCON)

TODO: This section needs to be updated.

OSCOAP protects message exchanges end-to-end between a certain client and a certain server, targeting the security requirements for forward proxy of [I-D.hartke-core-e2e-security-reqs]. In contrast, many use cases require one and the same message to be protected for, and verified by, multiple endpoints, see caching proxy section of [I-D.hartke-core-e2e-security-reqs]. Those security requirements can be addressed by protecting essentially the payload/content of individual messages using the COSE format ([I-D.ietf-cose-msg]), rather than the entire request/response message exchange. This is referred to as Object Security of Content (OSCON).

OSCON transforms an unprotected CoAP message into a protected CoAP message in the following way: the payload of the unprotected CoAP message is wrapped by a COSE object, which replaces the payload of the unprotected CoAP message. We call the result the "protected" CoAP message.

The unprotected payload shall be the plaintext/payload of the COSE object. The 'protected' field of the COSE object 'Headers' shall include the context identifier, both for requests and responses. If the unprotected CoAP message includes a Content-Format option, then the COSE object shall include a protected 'content type' field, whose value is set to the unprotected message Content-Format value. The Content-Format option of the protected CoAP message shall be replaced with "application/oscon" (Section 12)

The COSE object shall be protected (encrypted) and verified (decrypted) as described in ([I-D.ietf-cose-msg]).

Most AEAD algorithms require a unique nonce for each message. Sequence numbers for partial IV as specified for OSCOAP may be used for replay protection as described in Section 6. The use of time stamps in the COSE header parameter 'operation time' [I-D.ietf-cose-msg] for freshness may be used.

OSCON shall not be used in cases where CoAP header fields (such as Code or Version) or CoAP options need to be integrity protected or encrypted. OSCON shall not be used in cases which require a secure binding between request and response.

The scenarios in Sections 3.3 - 3.5 of [I-D.hartke-core-e2e-security-reqs] assume multiple recipients for a particular content. In this case the use of symmetric keys does not provide data origin authentication. Therefore the COSE object should in general be protected with a digital signature.

C.1. Overhead OSCON

In general there are four different kinds of modes that need to be supported: message authentication code, digital signature, authenticated encryption, and symmetric encryption + digital signature. The use of digital signature is necessary for applications with many legitimate recipients of a given message, and where data origin authentication is required.

To distinguish between these different cases, the tagged structures of COSE are used (see Section 2 of [I-D.ietf-cose-msg]).

The sizes of COSE messages for selected algorithms are detailed in this section.

The size of the header is shown separately from the size of the MAC/ signature. A 4-byte Context Identifier and a 1-byte Sequence Number are used throughout all examples, with these values:

- Cid: 0xa1534e3c
- Seq: 0xa3

For each scheme, we indicate the fixed length of these two parameters ("Cid+Seq" column) and of the Tag ("MAC"/"SIG"/"TAG"). The "Message OH" column shows the total expansions of the CoAP message size, while the "COSE OH" column is calculated from the previous columns.

Overhead incurring from CBOR encoding is also included in the COSE overhead count.

To make it easier to read, COSE objects are represented using CBOR's diagnostic notation rather than a binary dump.

C.2. MAC Only

This example is based on HMAC-SHA256, with truncation to 8 bytes (HMAC 256/64).

Since the key is implicitly known by the recipient, the COSE_Mac0_Tagged structure is used (Section 6.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
996(                          # COSE_Mac0_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                                {04:h'a1534e3c',
                                 06:h'a3'}
    {},                       # unprotected
    h'',                      # payload
    MAC                       # truncated 8-byte MAC
  ]
)
```

This COSE object encodes to a total size of 26 bytes.

Figure 9 summarizes these results.

Structure	Tid	MAC	COSE OH	Message OH
COSE_Mac0_Tagged	5 B	8 B	13 B	26 B

Figure 9: Message overhead for a 5-byte Tid using HMAC 256/64

C.3. Signature Only

This example is based on ECDSA, with a signature of 64 bytes.

Since only one signature is used, the COSE_Sign1_Tagged structure is used (Section 4.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
            997(                          # COSE_Sign1_Tagged
              [
                h'a20444a1534e3c0641a3', # protected:
                                           {04:h'a1534e3c',
                                            06:h'a3'}
                {},                       # unprotected
                h'',                      # payload
                SIG                       # 64-byte signature
              ]
            )
```

This COSE object encodes to a total size of 83 bytes.

Figure 10 summarizes these results.

Structure	Tid	SIG	COSE OH	Message OH
COSE_Sign1_Tagged	5 B	64 B	14 B	83 bytes

Figure 10: Message overhead for a 5-byte Tid using 64 byte ECDSA signature.

C.4. Authenticated Encryption with Additional Data (AEAD)

This example is based on AES-CCM with the Tag truncated to 8 bytes.

Since the key is implicitly known by the recipient, the COSE_Encrypt0_Tagged structure is used (Section 5.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
993(                          # COSE_Encrypt0_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                               {04:h'a1534e3c',
                                06:h'a3'}
    {},                       # unprotected
    ciphertext                # ciphertext including truncated 8-byte TAG
  ]
)
```

This COSE object encodes to a total size of 25 bytes.

Figure 11 summarizes these results.

Structure	Tid	TAG	COSE OH	Message OH
COSE_Encrypt0_Tagged	5 B	8 B	12 B	25 bytes

Figure 11: Message overhead for a 5-byte Tid using AES_128_CCM_8.

C.5. Symmetric Encryption with Asymmetric Signature (SEAS)

This example is based on AES-CCM and ECDSA with 64 bytes signature. The same assumption on the security context as in Appendix C.4. COSE defines the field 'counter signature w/o headers' that is used here to sign a COSE_Encrypt0_Tagged message (see Section 3 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
993(                        # COSE_Encrypt0_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                               {04:h'a1534e3c',
                                06:h'a3'}
    {9:SIG},                 # unprotected:
                                09: 64 bytes signature
    ciphertext               # ciphertext including truncated 8-byte TAG
  ]
)
```

This COSE object encodes to a total size of 92 bytes.

Figure 12 summarizes these results.

Structure	Tid	TAG	SIG	COSE OH	Message OH
COSE_Encrypt0_Tagged	5 B	8 B	64 B	15 B	92 B

Figure 12: Message overhead for a 5-byte Tid using AES-CCM countersigned with ECDSA.

Authors' Addresses

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com

Ludwig Seitz
SICS Swedish ICT

Email: ludwig@sics.se

CoRE
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant

March 13, 2017

CoRE Resource Directory
draft-ietf-core-resource-directory-10

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Table of Contents

1. Introduction

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

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

This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resource descriptions. Furthermore, new link attributes useful in conjunction with a Resource Directory are defined. Although the examples in this document show the use of

these interfaces with CoAP [RFC7252], they can be applied in an equivalent manner to HTTP [RFC7230].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. The term "byte" is used in its now customary sense as a synonym for "octet".

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

This specification makes use of the following additional terminology:

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

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

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

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

Context
: When registering links to a Resource Directory, the Context refers to the scheme, address, port, and base path for all the links registered on behalf of an endpoint, of the general form

scheme://host:port/path/ where the client may explicitly set the scheme and host, and may supply the port and path as optional parameters. When the context of a registration is explicitly set, the URI resolution rules in [RFC3986] MUST be applied.

Commissioning Tool
: Commissioning Tool (CT) is a device that assists during the installation of the network by assigning values to parameters, naming endpoints and groups, or adapting the installation to the needs of the applications.

RDAO
: Resource Directory Address Option.

3. Architecture and Use Cases

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

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Endpoints proactively register and maintain resource directory registration entries on the RD, which are soft state and need to be periodically refreshed.

An endpoint is provided with interfaces to register, update and remove a resource directory registration entry. It is also possible for an RD to fetch Web Links from endpoints and add them as resource directory entries.

At the first registration of a set of entries, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of the registration entry.

A lookup interface for discovering any of the Web Links held in the RD is provided using the CoRE Link Format.

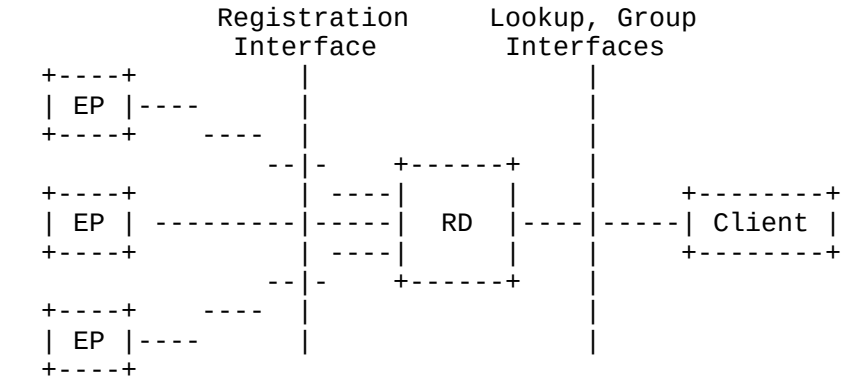


Figure 1: The resource directory architecture.

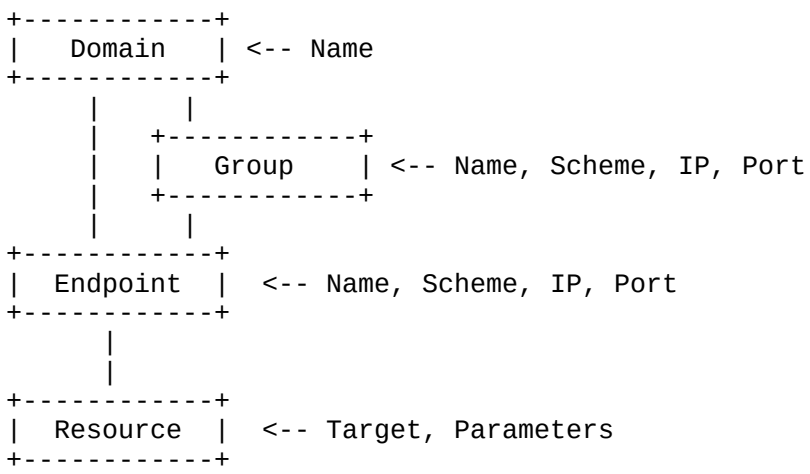


Figure 2: The resource directory information hierarchy.

3.1. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded air interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. From the system design point of

view, the ambition is to design horizontal solutions that can enable utilization of machines in different applications depending on their current availability and capabilities as well as application requirements, thus avoiding silo like solutions. One of the crucial enablers of such design is the ability to discover resources (machines -- endpoints) capable of providing required information at a given time or acting on instructions from the end users.

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

3.2. Use Case: Home and Building Automation

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

3.3. Use Case: Link Catalogues

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

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

Metadata in web link formats like [RFC6690] are supplied by Resource Directories, which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links. External catalogs that are represented in other formats may be converted to common web linking formats for storage and access by Resource Directories. Since it is common practice for these to be URN encoded, simple and lossless structural transforms should generally be sufficient to store external metadata in Resource Directories.

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

4. Finding a Resource Directory

Several mechanisms can be employed for discovering the RD, including assuming a default location (e.g. on an Edge Router in a LoWPAN), assigning an anycast address to the RD, using DHCP, or discovering the RD using .well-known/core and hyperlinks as specified in CoRE Link Format [RFC6690]. Endpoints that want to contact a Resource Directory can obtain candidate IP addresses for such servers in a number of ways.

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

- specific static configuration (e.g., anycast addresses), if any,
- the ABRO option of 6LoWPAN-ND [RFC6775],
- other ND options that happen to point to servers (such as RDNSS),
- DHCPv6 options that might be defined later.
- The IPv6 Neighbor Discovery Resource Directory Address Option described in Section 4.1

In networks with more inexpensive use of multicast, the candidate IP address may be a well-known multicast address, i.e. directory servers are found by simply sending GET requests to that well-known multicast address (see Section 5.2).

Constrained nodes configured in large batches may be configured for an anycast address for the RD. Each target network environment in which some of these preconfigured nodes are to be brought up is then configured with a route for this anycast address that leads to an RD that is appropriate for the environment.

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

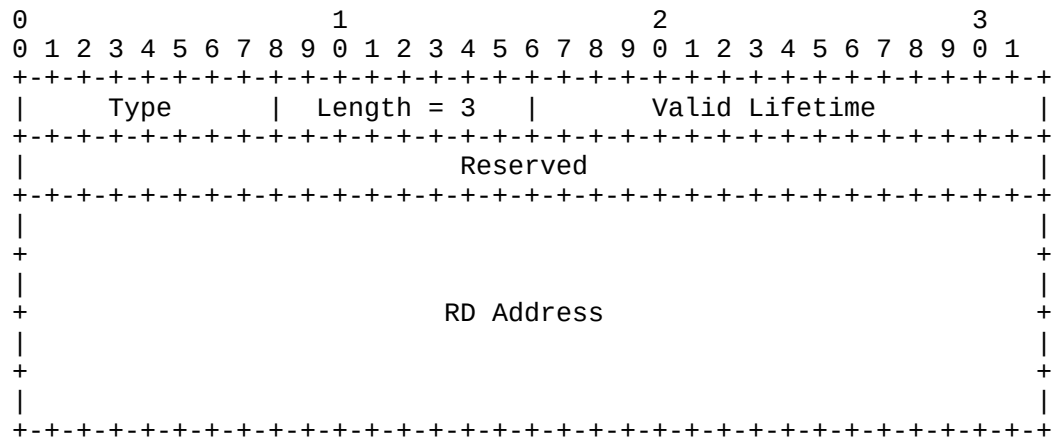
4.1. Resource Directory Address Option (RDAO)

The Resource Directory Option (RDAO) using IPv6 neighbor Discovery (ND) carries information about the address of the Resource Directory (RD). This information is needed when endpoints cannot discover the Resource Directory with link-local multicast address because the endpoint and the RD are separated by a border Router (6LBR). In many circumstances the availability of DHCP cannot be guaranteed either during commissioning of the network. The presence and the use of the RD is essential during commissioning.

It is possible to send multiple RDAO options in one message, indicating as many resource directory addresses.

The lifetime 0x0 means that the RD address is invalid and to be removed.

The RDAO format is:



Fields:

Type: 38

Length: 8-bit unsigned integer. The length of the option in units of 8 bytes. Always 3.

Valid Lifetime: 16-bit unsigned integer. The length of time in units of 60 seconds (relative to the time the packet is received) that this Resource Directory address is valid. A value of all zero bits (0x0) indicates that this Resource Directory address is not valid anymore.

Reserved: This field is unused. It MUST be initialized to zero by the sender and MUST be ignored by the receiver.

RD Address: IPv6 address of the RD.

Figure 3: Resource Directory Address Option

5. Resource Directory

This section defines the required set of REST interfaces between a Resource Directory (RD) and endpoints. Although the examples throughout this section assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. In all definitions in this section, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown.

An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces defined in this section.

5.1. Content Formats

Resource Directory implementations using this specification MUST support the application/link-format content format (ct=40).

Resource Directories implementing this specification MAY support additional content formats.

Any additional content format supported by a Resource Directory implementing this specification MUST have an equivalent serialization in the application/link-format content format.

5.2. Base URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the base URI information for its REST API. This section defines discovery of the RD and its base URI using the well-known interface of the CoRE Link Format [RFC6690]. It is however expected that RDs will also be discoverable via other methods depending on the deployment.

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

A Resource Directory MAY provide hints about the content-formats it supports in the links it exposes or registers, using the "ct" link attribute, as shown in the example below. Clients MAY use these hints to select alternate content-formats for interaction with the Resource Directory.

HTTP does not support multicast and consequently only unicast discovery can be supported using HTTP. Links to Resource Directories MAY be registered in other Resource Directories, and well-known entry

points SHOULD be provided to enable the bootstrapping of unicast discovery.

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

The discovery request interface is specified as follows:

Interaction: EP -> RD

Method: GET

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

URI Template Variables:

rt := Resource Type (optional). MAY contain one of the values "core.rd", "core.rd-lookup*", "core.rd-lookup-d", "core.rd-lookup-res", "core.rd-lookup-ep", "core.rd-lookup-gp", "core.rd-group" or "core.rd*"

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

Content-Format: application/link-format+json (if any)

Content-Format: application/link-format+cbor (if any)

The following response codes are defined for this interface:

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

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

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

Failure: No error response to a multicast request.

HTTP support : YES (Unicast only)

The following example shows an endpoint discovering an RD using this interface, thus learning that the base RD resource is, in this example, at /rd and that the content-format delivered by the server hosting the resource is application/link-format (ct=40). Note that

it is up to the RD to choose its base RD resource, although diagnostics and debugging is facilitated by using the base paths specified here where possible.

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

Res: 2.05 Content
</rd>;rt="core.rd";ct=40,
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct=40,
</rd-lookup/res>;rt="core.rd-lookup-res";ct="40",
</rd-group>;rt="core.rd-group";ct=40

The following example shows the way of indicating that a client may request alternate content-formats. The Content-Format code attribute "ct" MAY include a space-separated sequence of Content-Format codes as specified in Section 7.2.1 of [RFC7252], indicating that multiple content-formats are available. The example below shows the required Content-Format 40 (application/link-format) indicated as well as a more application-specific content format (picked as 65225 in this example; this is in the experimental space, not an assigned value). The base RD resource values /rd, /rd-lookup, and /rd-group are example values.

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

Res: 2.05 Content
</rd>;rt="core.rd";ct="40 65225",
</rd-lookup/res>;rt="core.rd-lookup-res";ct="40 65225",
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct="40 65225",
</rd-group>;rt="core.rd-group";ct="40 65225"

5.3. Registration

After discovering the location of an RD, an endpoint MAY register its resources using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690], JSON CoRE Link Format (application/link-format+json), or CBOR CoRE Link Format (application/link-format+cbor) [I-D.ietf-core-links-json], along with query parameters indicating the name of the endpoint, and optionally its domain and the lifetime of the registration. It is expected that other specifications will define further parameters (see Section 9.3). The RD then creates a new registration resource in the RD and returns its location. An endpoint MUST use that location when refreshing registrations using this interface. Endpoint resources in the RD are kept active for the period indicated by the lifetime parameter. The endpoint is responsible for refreshing the entry within this period using either

the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameters ep and d does not create multiple RD entries. A new registration may be created at any time to supersede an existing registration, replacing the registration parameters and links.

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

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

URI Template Variables:

rd := RD Base URI path (mandatory). This is the path of the RD, as obtained from discovery. The value "rd" is recommended for this variable.

ep := Endpoint name (mandatory). The endpoint name is an identifier that MUST be unique within a domain. The maximum length of this parameter is 63 bytes.

d := Domain (optional). The domain to which this endpoint belongs. The maximum length of this parameter is 63 bytes. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain.

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

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included in the initial registration, a default value of 86400 (24 hours) SHOULD be assumed. If the lt parameter is not included in a registration refresh or update operation, the most recently supplied value SHALL be re-used.

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

endpoint uses an ephemeral port to register with, it MUST include the con: parameter in the registration to provide a valid network path. If the endpoint which is located behind a NAT gateway is registering with a Resource Directory which is on the network service side of the NAT gateway, the endpoint MUST use a persistent port for the outgoing registration in order to provide the NAT gateway with a valid network address for replies and incoming requests.

Content-Format: application/link-format

Content-Format: application/link-format+json

Content-Format: application/link-format+cbor

The following response codes are defined for this interface:

Success: 2.01 "Created" or 201 "Created". The Location header option MUST be included in the response when a new registration resource is created. This Location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration resource. The registration resource location thus returned is for the purpose of updating the lifetime of the registration and for maintaining the content of the registered links, including updating and deleting links.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.09 "Conflict" or 409 "Conflict". Attempt to update the registration content with links resulting in plurality of references; see Section 5.3.4.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

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

```
Req: POST coap://rd.example.com/rd?ep=node1
Content-Format: 40
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

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

A Resource Directory may optionally support HTTP. Here is an example of the same registration operation above, when done using HTTP.

```
Req: POST /rd?ep=node1 HTTP/1.1
Host : example.com
Content-Type: application/link-format
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

Res: 201 Created
Location: /rd/4521
```

5.3.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to a RD as described in Section 5.3. Instead, simple endpoints can implement the Simple Registration approach described in this section. An RD implementing this specification MUST implement Simple Registration. However, there may be security reasons why this form of directory discovery would be disabled.

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

The endpoint then finds one or more addresses of the directory server as described in Section 4.

An endpoint can send (a selection of) hosted resources to a directory server for publication as described in Section 5.3.2.

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

5.3.2. Simple publishing to Resource Directory Server

An endpoint that wants to make itself discoverable occasionally sends a POST request to the "/.well-known/core" URI of any candidate directory server that it finds. The body of the POST request is empty, which triggers the resource directory server to perform GET requests at the requesting server's default discovery URI to obtain the link-format payload to register.

The endpoint MUST include the endpoint name and MAY include the registration parameters d, lt, and et, in the POST request as per Section 5.3.

The following example shows an endpoint using simple publishing, by simply sending an empty POST to a resource directory.

Req:(to RD server from [ff02::1])
POST coap://rd.example.com/.well-known/core?lt=6000;ep=node1

Content-Format: 40

payload:

(empty payload)

Res: 2.04 Changed

(later)

Req: (from RD server to [ff02::1])
GET coap://[ff02::1]/.well-known/core

Accept: 40

Res: 2.05 Content

payload:

</sen/temp>

5.3.3. Third-party registration

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

In a controlled environment (e.g. building control), the Resource Directory can be filled by a third device, called a commissioning

tool. The commissioning tool can fill the Resource Directory from a database or other means. For that purpose the scheme, IP address and port of the registered device is indicated in the Context parameter of the registration described in Section 5.3.

5.3.4. Plurality of link references in a Registration

Plurality of link references within a Registration (registration resource) is an indication of some error condition and should not be allowed.

Plurality of link references exists if, and only if, two or more links in a Registration contain identical context, target, and relation values. This condition would be likely to arise if there were multiple co-ordinators or configuration tools, each with a different set of configuration values for the same resource.

A Resource Directory SHOULD reject a registration, or an operation on a registration, which would result in a plurality of link references within the the context of the registration. There is no requirement in this document for a resource directory to check for plurality of reference between different registrations. Resource Directory operations which are rejected due to reference plurality SHOULD be returned the "Conflict" code, indicating that there is someting wrong with the request.

5.4. Operations on the Registration Resource

After the initial registration, an endpoint should retain the returned location of the Registration Resource for further operations, including refreshing the registration in order to extend the lifetie and "keep-alive" the registration. If the lifetime of the registration expires, the RD SHOULD NOT respond to discovery queries with information from the endpoint. The RD SHOULD continue to provide access to the Registration Resource after a registration time-out occurs in order to enable the registering endpoint to eventually refresh the registration. The RD MAY eventually remove the registration resource for the purpose of resource recovery and garbage collection. If the Registration Resource is removed, the endpoint will need to re-register.

The Registration Resource may also be used to inspect the registration resource using GET, update the registration link contents using PATCH, or cancel the registration using DELETE.

These operations are described in this section.

In accordance with Section 5.3.4, operations which would result in plural link references within the context of a registration resource SHOULD be rejected using the "Conflict" result code.

5.4.1. Registration Update

The update interface is used by an endpoint to refresh or update its registration with an RD. To use the interface, the endpoint sends a POST request to the registration resource returned in the Location header option in the response returned from the intial registration operation.

An update MAY update the lifetime or context registration parameters "lt", "con" as in Section 5.3) if the previous settings are to be retained. Parameters that are not being changed changed SHOULD NOT be included in an update. Adding parameters that have not changed increases the size of the message but does not have any other implications. Parameters MUST be included as query parameters in an update operation as in Section 5.3.

Upon receiving an update request, an RD MUST reset the timeout for that endpoint and update the scheme, IP address and port of the endpoint, using the source address of the update, or the context ("con") parameter if present. If the lifetime parameter "lt" is included in the received update request, the RD MUST update the lifetime of the registration and set the timeout equal to the new lifetime. If the lifetime parameter is not included in the registration update, the most recent setting is re-used for the next registration time-out period.

An update MAY optionally add or replace links for the endpoint by including those links in the payload of the update as a CoRE Link Format document. A link is replaced only if all of the target URI and relation type (if present) and anchor value (if present) match.

If the link payload is included, it SHOULD be checked for reference plurality as described in Section 5.3.4 and rejected with a "Conflict" result if there are plural link references detected.

In addition to the use of POST, as described in this section, there is an alternate way to add, replace, and delete links using PATCH as described in Section 5.4.4.

The update registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

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

URI Template Variables:

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

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, the previous last lifetime set on a previous update or the original registration (falling back to 86400) SHOULD be used.

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

Content-Format: application/link-format (mandatory)

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

Content-Format: application/link-format+cbor (optional)

The following response codes are defined for this interface:

Success: 2.04 "Changed" or 204 "No Content" if the update was successfully processed.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 4.09 "Conflict" or 409 "Conflict". Attempt to update the registration content with links resulting in plurality of references; see Section 5.3.4.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following example shows an endpoint updating its registration at an RD using this interface with the example location value: /rd/4521.

Req: POST /rd/4521

Res: 2.04 Changed

The following example shows an endpoint updating its registration with a new lifetime and context, changing an existing link, and adding a new link using this interface with the example location value /rd/4521. With the initial registration the client set the following values:

- lifetime (lt)=500
- context (con)=coap://local-proxy-old.example.com:5683
- resource= </sensors/temp>;ct=41;rt="foobar";if="sensor"

Req: POST /rd/4521?lt=600&con="coap://local-proxy.example.com:5683"
Content-Format: 40
Payload:
</sensors/temp>;ct=41;rt="temperature-f";if="sensor",
</sensors/door>;ct=41;rt="door";if="sensor"

Res: 2.04 Changed

5.4.2. Registration Removal

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

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

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

The following responses codes are defined for this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.00 "Bad Request" or 400 "Bad request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples shows successful removal of the endpoint from the RD with example location value /rd/4521.

Req: DELETE /rd/4521

Res: 2.02 Deleted

5.4.3. Read Endpoint Links

Some endpoints may wish to manage their links as a collection, and may need to read the current set of links stored in the registration resource, in order to determine link maintenance operations.

One or more links MAY be selected by using query filtering as specified in [RFC6690] Section 4.1

If no links are selected, the Resource Directory SHOULD return an empty payload.

The read request interface is specified as follows:

Interaction: EP -> RD

Method: GET

URI Template: /{+location}{?href,rel,rt,if,ct}

URI Template Variables:

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

href,rel,rt,if,ct := link relations and attributes specified in the query in order to select particular links based on their relations and attributes. "href" denotes the URI target of the link. See [RFC6690] Sec. 4.1

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" upon success with an "application/link-format", "application/link-format+cbor", or "application/link-format+json" payload.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples show successful read of the endpoint links from the RD, with example location value /rd/4521.

Req: GET /rd/4521

Res: 2.01 Content
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

5.4.4. Update Endpoint Links

A PATCH update adds, removes or changes links for the endpoint by including link update information in the payload of the update as a merge-patch+json format [RFC7396] document.

Other PATCH document formats may be used as appropriate for patching the array of objects format of a Registration Resource. In particular, a select-merge patch document format could combine the function of link selection query and link attribute replacement values.

One or more links are selected for update by using query filtering as specified in [RFC6690] Section 4.1

The query filter selects the links to be modified or deleted, by matching the query parameter values to the values of the link attributes.

When the query parameters are not present in the request, the payload specifies links to be added to the target document. When the query parameters are present, the attribute names and values in the query parameters select one or more links on which to apply the PATCH operation.

If no links are selected by the query parameters, the PATCH operation SHOULD NOT update the state of any resource, and SHOULD return a reply of "Changed".

If an attribute name specified in the PATCH document exists in any the set of selected links, all occurrences of the attribute value in the target document MUST be updated using the value from the PATCH payload. If the attribute name is not present in any selected links, the attribute MUST be added to the links.

If the PATCH payload contains plural link references, or processing the PATCH payload would result in plural link references, the request SHOULD be rejected with a "Conflict" result.

If the PATCH payload results in the modification of link target, context, or relation values, that is "href", "rel", or "anchor", the request SHOULD be rejected with a "Conflict" result code.

The update request interface is specified as follows:

Interaction: EP -> RD

Method: PATCH

URI Template: /{+location}{?href,rel,rt,if,ct}

URI Template Variables:

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

href,rel,rt,if,ct := link relations and attributes specified in the query in order to select particular links based on their relations and attributes. "href" denotes the URI target of the link. See [RFC6690] Sec. 4.1

Content-Format: application/merge-patch+json (mandatory)

The following response codes are defined for this interface:

Success: 2.04 "Changed" 0r 204 "No Content" in the update was successfully processed.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration resource does not exist (e.g. may have expired).

Failure: 4.09 "Conflict" or 409 "Conflict". Attempt to update the registration content with links resulting in plurality of references; see Section 5.3.4.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples show an endpoint adding </sensors/humid>, modifying </sensors/temp>, and removing </sensors/light> links in RD using the Update Endpoint Links function with the example location value /rd/4521.

The Registration Resource initial state is:

Req: GET /rd/4521

Res: 2.01 Content
Payload:
</sensors/temp>;ct=41;rt="temperature",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

The following example shows an EP adding the link </sensors/humid>;ct=41;rt="humid-s";if="sensor" to the collection of links at the location /rd/4521.

Req: PATCH /rd/4521

Payload:
[{"href":"/sensors/humid","ct": 41, "rt": "humid-s", "if": "sensor"}]

Content-Format:
application/merge-patch+json

Res: 2.04 Changed

```
Req: GET /rd/4521

Res: 2.01 Content
Payload:
</sensors/temp>;ct=41;rt="temperature",
</sensors/light>;ct=41;rt="light-lux";if="sensor",
</sensors/humid>;ct=41;rt="humid-s";if="sensor"
```

The following example shows an EP modifying all links at the example location /rd/4521 which are identified by href="/sensors/temp", from the initial link-value of </sensors/temp>;rt="temperature" to the new link-value </sensors/temp>;rt="temperature-c";if="sensor" by changing the value of the link attribute "rt" and adding the link attribute if="sensor" using the PATCH operation with the supplied merge-patch+json document payload.

```
Req: PATCH /rd/4521?href=/sensors/temp

Payload:
{"rt": "temperature-c", "if": "sensor"},

Content-Format:
application/merge-patch+json

Res: 2.04 Changed

Req: GET /rd/4521

Res: 2.01 Content
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor",
</sensors/humid>;ct=41;rt="humid-s";if="sensor"
```

This example shows an EP removing all links at the example location /rd/4521 which are identified by href="/sensors/light".

```
Req: PATCH /rd/4521?href=/sensors/light

Payload:
{}

Content-Format:
application/merge-patch+json

Res: 2.04 Changed
```

```
Req: GET /rd/4521

Res: 2.01 Content
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/humid>;ct=41;rt="humid-s";if="sensor"
```

6. RD Groups

This section defines the REST API for the creation, management, and lookup of endpoints for group operations. Similar to endpoint registration entries in the RD, groups may be created or removed. However unlike an endpoint entry, a group entry consists of a list of endpoints and does not have a lifetime associated with it. In order to make use of multicast requests with CoAP, a group MAY have a multicast address associated with it.

6.1. Register a Group

In order to create a group, a commissioning tool (CT) used to configure groups, makes a request to the RD indicating the name of the group to create (or update), optionally the domain the group belongs to, and optionally the multicast address of the group. The registration message includes the list of endpoints that belong to that group.

All the endpoints in the group MUST be registered with the RD before registering a group. If an endpoint is not yet registered to the RD before registering the group, the registration message returns an error. The RD sends a blank target URI for every endpoint link when registering the group.

Configuration of the endpoints themselves is out of scope of this specification. Such an interface for managing the group membership of an endpoint has been defined in [RFC7390].

The registration request interface is specified as follows:

Interaction: CT -> RD

Method: POST

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

URI Template Variables:

rd-group := RD Group Base URI path (mandatory). This is the path of the RD Group REST API. The value "rd-group" is recommended for this variable.

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

d := Domain (optional). The domain to which this group belongs. The maximum length of this parameter is 63 bytes. Optional. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain.

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

Content-Format: application/link-format

Content-Format: application/link-format+json

Content-Format: application/link-format+cbor

The following response codes are defined for this interface:

Success: 2.01 "Created" or 201 "Created". The Location header option MUST be returned in response to a successful group CREATE operation. This Location MUST be a stable identifier generated by the RD as it is used for delete operations of the group registration resource.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". An Endpoint is not registered in the RD (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following example shows an EP registering a group with the name "lights" which has two endpoints to an RD using this interface. The base location value /rd-group is an example of an RD base location.

```
Req: POST coap://rd.example.com/rd-group?gp=lights
Content-Format: 40
Payload:
<>;ep="node1",
<>;ep="node2"

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

6.2. Group Removal

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

The removal request interface is specified as follows:

Interaction: CT -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

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

The following responses codes are defined for this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Group does not exist.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples shows successful removal of the group from the RD with the example location value /rd-group/12.

Req: DELETE /rd-group/12

Res: 2.02 Deleted

7. RD Lookup

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

RD Lookup allows lookups for domains, groups, endpoints and resources using attributes defined in this document and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, a domain lookup MUST return a list of domains, a group lookup MUST return a list of groups, an endpoint lookup MUST return a list of endpoints and a resource lookup MUST return a list of links to resources.

RD Lookup does not expose registration resources directly, but returns link content from registration resource entries which satisfy RD Lookup queries.

The lookup type is selected by a URI endpoint, which is indicated by a Resource Type as per Table 1 below:

Lookup Type	Resource Type	Mandatory
Resource	core.rd-lookup-res	Mandatory
Endpoint	core.rd-lookup-ep	Mandatory
Domain	core.rd-lookup-d	Optional
Group	core.rd-lookup-gp	Optional

Table 1: Lookup Types

Each endpoint and resource lookup result returns respectively the scheme (IP address and port) followed by the path part of the URI of every endpoint and resource inside angle brackets ("<>") and followed by the other parameters.

The target of these links SHOULD be the actual location of the domain, endpoint or resource, but MAY be an intermediate proxy e.g. in the case of an HTTP lookup interface for CoAP endpoints.

The domain lookup returns every lookup domain with a base RD resource value (e.g. "/rd") encapsulated within angle brackets.

In case that a group does not implement any multicast address, the group lookup returns every group lookup with a group base resource value encapsulated within angle brackets (e.g. "/rd/look-up"). Otherwise, the group lookup returns the multicast address of the group inside angle brackets.

Using the Accept Option, the requester can control whether this list is returned in CoRE Link Format ("application/link-format", default) or its alternate content-formats ("application/link-format+json" or "application/link-format+cbor").

The page and count parameters are used to obtain lookup results in specified increments using pagination, where count specifies how many links to return and page specifies which subset of links organized in sequential pages, each containing 'count' links, starting with link zero and page zero. Thus, specifying count of 10 and page of 0 will return the first 10 links in the result set (links 0-9). Count = 10 and page = 1 will return the next 'page' containing links 10-19, and so on.

Multiple query parameters MAY be included in a lookup, all included parameters MUST match for a resource to be returned. The character'*' MAY be included at the end of a parameter value as a wildcard operator.

RD Lookup requests MAY use any set of query parameters to match the registered attributes and relations. In addition, this interface MAY be used with queries that specify domains, endpoints, and groups. For example, a domain lookup filtering on groups would return a list of domains that contain the specified groups. An endpoint lookup filtering on groups would return a list of endpoints that are in the specified groups.

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

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

URI Template Variables:

rd-lookup-base := RD Lookup Base URI path (mandatory). This is the base path for RD Lookup requests. The recommended value for this variable is: "rd-lookup".

lookup-type := ("ep", "res") (mandatory), ("d","gp") (optional) This variable is used to select the type of lookup to perform (endpoint, resource, domain, or group). The values are recommended defaults and MAY use other values as needed. The supported lookup-types SHOULD be listed in .well-known/core using the specified resource types.

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

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

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

gp := Group name (optional). Used for endpoint, group and resource lookups.

page := Page (optional). Parameter can not be used without the count parameter. Results are returned from result set in pages that contain 'count' links starting from index (page * count). Page numbering starts with zero.

count := Count (optional). Number of results is limited to this parameter value. If the page parameter is also present, the response MUST only include 'count' links starting with the (page * count) link in the result set from the query. If the count parameter is not present, then the response MUST return all matching links in the result set. Link numbering starts with zero.

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

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

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

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

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

Content-Format: application/link-format+cbor (optional)

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format", "application/link-format+cbor", or "application/link-format+json" payload containing matching entries for the lookup.

Failure: 4.04 "Not Found" or 404 "Not Found" in case no matching entry is found for a unicast request.

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The examples in this section assume CoAP hosts with a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples.

The following example shows a client performing a resource lookup with the example look-up location /rd-lookup/:

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

Res: 2.05 Content
<coap://[FDFD::123]:61616/temp>;rt="temperature"

The following example shows a client performing an endpoint type lookup:

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

Res: 2.05 Content
<coap://[FDFD::127]:61616>;ep="node5",
<coap://[FDFD::129]:61616>;ep="node7"

The following example shows a client performing a domain lookup:

```
Req: GET /rd-lookup/d

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

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

```
Req: GET /rd-lookup/gp

Res: 2.05 Content
<>;gp="lights1";d="example.com"
<>;gp="lights2";d="ecample.com"
```

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

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

Res: 2.05 Content
<coap://[FDFD::123]:61616>;ep="node1",
<coap://[FDFD::124]:61616>;ep="node2"
```

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

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

Res: 2.05 Content
<>;gp="lights1"
```

The following example shows a client performing a paginated lookup

```
Req: GET /rd-lookup/res?page=0&count=5

Res: 2.05 Content
<coap://[FDFD::123]:61616/res/0>;rt=sensor;ct=60
<coap://[FDFD::123]:61616/res/1>;rt=sensor;ct=60
<coap://[FDFD::123]:61616/res/2>;rt=sensor;ct=60
<coap://[FDFD::123]:61616/res/3>;rt=sensor;ct=60
<coap://[FDFD::123]:61616/res/4>;rt=sensor;ct=60

Req: GET /rd-lookup/res?page=1&count=5

Res: 2.05 Content
<coap://[FDFD::123]:61616/res/5>;rt=sensor;ct=60
<coap://[FDFD::123]:61616/res/6>;rt=sensor;ct=60
<coap://[FDFD::123]:61616/res/7>;rt=sensor;ct=60
<coap://[FDFD::123]:61616/res/8>;rt=sensor;ct=60
<coap://[FDFD::123]:61616/res/9>;rt=sensor;ct=60
```

8. Security Considerations

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

8.1. Endpoint Identification and Authentication

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

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

8.2. Access Control

Access control SHOULD be performed separately for the RD registration, Lookup, and group API base paths, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control

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

8.3. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly become part of a DDoS attack as UDP does not require return routability check. Therefore, an attacker can easily spoof the source IP of the target entity and send requests to such a service which would then respond to the target entity. This can be used for large-scale DDoS attacks on the target. Especially, if the service returns a response that is order of magnitudes larger than the request, the situation becomes even worse as now the attack can be amplified. DNS servers have been widely used for DDoS amplification attacks. There is also a danger that NTP Servers could become implicated in denial-of-service (DoS) attacks since they run on unprotected UDP, there is no return routability check, and they can have a large amplification factor. The responses from the NTP server were found to be 19 times larger than the request. A Resource Directory (RD) which responds to wild-card lookups is potentially vulnerable if run with CoAP over UDP. Since there is no return routability check and the responses can be significantly larger than requests, RDs can unknowingly become part of a DDoS amplification attack.

9. IANA Considerations

9.1. Resource Types

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

9.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the subregistry "IPv6 Neighbor Discovery Option Formats":

- Resource Directory address Option (38)

9.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is

expected that other standards that make use of this interface will define new ones.

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

Initial entries in this sub-registry are as follows:

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

Table 2: RD Parameters

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

10. Examples

Two examples are presented: a Lighting Installation example in Section 10.1 and a LWM2M example in Section 10.2.

10.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the Resource Directory (RD) with a CoAP interface to facilitate the installation and start up of the application code in the lights and sensors. In particular, the example leads to the definition of a

group and the enabling of the corresponding multicast address. No conclusions must be drawn on the realization of actual installation or naming procedures, because the example only "emphasizes" some of the issues that may influence the use of the RD and does not pretend to be normative. The example uses the recommended values for the base resources: "/rd", "/rd-lookup", and "/rd-group".

10.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one endpoint. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager.

At the moment of installation, the network under installation is not necessarily connected to the DNS infra structure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and sensor shown in Table 3 below:

Name	IPv6 address
luminary1	FDFD::ABCD:1
luminary2	FDFD::ABCD:2
Presence sensor	FDFD::ABCD:3
Resource directory	FDFD::ABCD:0

Table 3: interface SLAAC addresses

In Section 10.1.2 the use of resource directory during installation is presented.

10.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have rt: light, and the presence sensor has rt: p-sensor. The endpoints have names which are relevant to the light installation manager. In this case luminary1, luminary2, and the presence sensor are located in room 2-4-015, where luminary1 is located at the window and luminary2 and the presence sensor are located at the door. The endpoint names reflect this physical location. The middle, left and right lamps are accessed via path /light/middle, /light/left, and /light/right respectively. The identifiers relevant to the Resource Directory are shown in Table 4 below:

Name	endpoint	resource path	resource type
luminary1	lm_R2-4-015_wndw	/light/left	light
luminary1	lm_R2-4-015_wndw	/light/middle	light
luminary1	lm_R2-4-015_wndw	/light/right	light
luminary2	lm_R2-4-015_door	/light/left	light
luminary2	lm_R2-4-015_door	/light/middle	light
luminary2	lm_R2-4-015_door	/light/right	light
Presence sensor	ps_R2-4-015_door	/ps	p-sensor

Table 4: Resource Directory identifiers

The CT inserts the endpoints of the luminaries and the sensor in the RD using the Context parameter (con) to specify the interface address:

```
Req: POST coap://[FDFD::ABCD:0]/rd
  ?ep=lm_R2-4-015_wndw&con=coap://[FDFD::ABCD:1]
Payload:
</light/left>;rt="light"; d="R2-4-015",
</light/middle>;rt="light"; d="R2-4-015",
</light/right>;rt="light";d="R2-4-015"

Res: 2.01 Created
Location: /rd/4521

Req: POST coap://[FDFD::ABCD:0]/rd
  ?ep=lm_R2-4-015_door&con=coap://[FDFD::ABCD:2]
Payload:
</light/left>;rt="light"; d="R2-4-015",
</light/middle>;rt="light"; d="R2-4-015",
</light/right>;rt="light"; d="R2-4-015"

Res: 2.01 Created
Location: /rd/4522

Req: POST coap://[FDFD::ABCD:0]/rd
  ?ep=ps_R2-4-015_door&con=coap://[FDFD::ABCD:3]
Payload:
</ps>;rt="p-sensor"; d="R2-4-015"

Res: 2.01 Created
Location: /rd/4523
```

The domain name d="R2-4-015" has been added for an efficient lookup because filtering on "ep" name is more awkward. The same domain name is communicated to the two luminaries and the presence sensor by the CT.

The group is specified in the RD. The Context parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, these two endpoints and the endpoint of the presence sensor are registered as members of the group.

```
Req: POST coap://[FDFD::ABCD:0]/rd-group
?gp=grp_R2-4-015;con="coap//[FF05::1]"
Payload:
<>ep=lm_R2-4-015_wndw,
<>ep=lm_R2-4-015_door,
<>ep=ps_R2-4-015_door

Res: 2.01 Created
Location: /rd-group/501
```

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its domain, queries the RD for the endpoint with rt=light and d=R2-4-015. The RD returns all endpoints in the domain.

```
Req: GET coap://[FDFD::ABCD:0]/rd-lookup/ep
  ?d=R2-4-015;rt=light

Res: 2.05 Content
<coap://[FDFD::ABCD:1]>;
  ep="lm_R2-4-015_wndw",
<coap://[FDFD::ABCD:2]>;
   ep="lm_R2-4-015_door"
```

Knowing its own IPv6 address, the luminary discovers its endpoint name. With the endpoint name the luminary queries the RD for all groups to which the endpoint belongs.

```
Req: GET coap://[FDFD::ABCD:0]/rd-lookup/gp
  ?ep=lm_R2-4-015_wndw

Res: 2.05 Content
<coap://[FF05::1]>;gp="grp_R2-4-015"
```

From the context parameter value, the luminary learns the multicast address of the multicast group.

Alternatively, the CT can communicate the multicast address directly to the luminaries by using the "coap-group" resource specified in [RFC7390].

```
Req: POST //[FDFD::ABCD:1]/coap-group
          Content-Format: application/coap-group+json
       { "a": "[FF05::1]",
          "n": "grp_R2-4-015"}

Res: 2.01 Created
Location-Path: /coap-group/1
```

Dependent on the situation, only the address, "a", or the name, "n", is specified in the coap-group resource.

10.2. OMA Lightweight M2M (LWM2M) Example

This example shows how the OMA LWM2M specification makes use of Resource Directory (RD).

OMA LWM2M is a profile for device services based on CoAP(OMA Name Authority). LWM2M defines a simple object model and a number of abstract interfaces and operations for device management and device service enablement.

An LWM2M server is an instance of an LWM2M middleware service layer, containing a Resource Directory along with other LWM2M interfaces defined by the LWM2M specification.

CoRE Resource Directory (RD) is used to provide the LWM2M Registration interface.

LWM2M does not provide for registration domains and does not currently use the rd-group or rd-lookup interfaces.

The LWM2M specification describes a set of interfaces and a resource model used between a LWM2M device and an LWM2M server. Other interfaces, proxies, and applications are currently out of scope for LWM2M.

The location of the LWM2M Server and RD base URI path is provided by the LWM2M Bootstrap process, so no dynamic discovery of the RD is used. LWM2M Servers and endpoints are not required to implement the ./well-known/core resource.

10.2.1. The LWM2M Object Model

The OMA LWM2M object model is based on a simple 2 level class hierarchy consisting of Objects and Resources.

An LWM2M Resource is a REST endpoint, allowed to be a single value or an array of values of the same data type.

An LWM2M Object is a resource template and container type that encapsulates a set of related resources. An LWM2M Object represents a specific type of information source; for example, there is a LWM2M Device Management object that represents a network connection, containing resources that represent individual properties like radio signal strength.

Since there may potentially be more than one of a given type object, for example more than one network connection, LWM2M defines instances of objects that contain the resources that represent a specific physical thing.

The URI template for LWM2M consists of a base URI followed by Object, Instance, and Resource IDs:

{/base-uri}{/object-id}{/object-instance}{/resource-id}{/resource-instance}

The five variables given here are strings. base-uri can also have the special value "undefined" (sometimes called "null" in RFC 6570). Each of the variables object-instance, resource-id, and resource-instance can be the special value "undefined" only if the values behind it in this sequence also are "undefined". As a special case, object-instance can be "empty" (which is different from "undefined") if resource-id is not "undefined".

base-uri := Base URI for LWM2M resources or "undefined" for default (empty) base URI

object-id := OMNA (OMA Name Authority) registered object ID (0-65535)

object-instance := Object instance identifier (0-65535) or "undefined"/"empty" (see above)) to refer to all instances of an object ID

resource-id := OMNA (OMA Name Authority) registered resource ID (0-65535) or "undefined" to refer to all resources within an instance

resource-instance := Resource instance identifier or "undefined" to refer to single instance of a resource

LWM2M IDs are 16 bit unsigned integers represented in decimal (no leading zeroes except for the value 0) by URI format strings. For example, a LWM2M URI might be:

/1/0/1

The base uri is empty, the Object ID is 1, the instance ID is 0, the resource ID is 1, and the resource instance is "undefined". This example URI points to internal resource 1, which represents the registration lifetime configured, in instance 0 of a type 1 object (LWM2M Server Object).

10.2.2. LWM2M Register Endpoint

LWM2M defines a registration interface based on the REST API, described in Section 5. The base URI path of the LWM2M Resource Directory is specified to be "/rd" as recommended in Section 5.3.

LWM2M endpoints register object IDs, for example </1>, to indicate that a particular object type is supported, and register object instances, for example </1/0>, to indicate that a particular instance of that object type exists.

Resources within the LWM2M object instance are not registered with the RD, but may be discovered by reading the resource links from the object instance using GET with a CoAP Content-Format of application/link-format. Resources may also be read as a structured object by performing a GET to the object instance with a Content-Format of senml+json.

When an LWM2M object or instance is registered, this indicates to the LWM2M server that the object and its resources are available for management and service enablement (REST API) operations.

LWM2M endpoints may use the following RD registration parameters as defined in Table 2 :

ep - Endpoint Name
lt - registration lifetime

Endpoint Name is mandatory, all other registration parameters are optional.

Additional optional LWM2M registration parameters are defined:

Name	Query	Validity	Description
Protocol Binding	b	{"U",UQ","S","SQ","US","UQS"}	Available Protocols
LWM2M Version	ver	1.0	Spec Version
SMS Number	sms		MSISDN

Table 5: LWM2M Additional Registration Parameters

The following RD registration parameters are not currently specified for use in LWM2M:

et - Endpoint Type
con - Context

The endpoint registration must include a payload containing links to all supported objects and existing object instances, optionally including the appropriate link-format relations.

Here is an example LWM2M registration payload:

</1>,</1/0>,</3/0>,</5>

This link format payload indicates that object ID 1 (LWM2M Server Object) is supported, with a single instance 0 existing, object ID 3 (LWM2M Device object) is supported, with a single instance 0 existing, and object 5 (LWM2M Firmware Object) is supported, with no existing instances.

10.2.3. LWM2M Update Endpoint Registration

An LWM2M Registration update proceeds as described in Section 5.4.1, and adds some optional parameter updates:

lt - Registration Lifetime
b - Protocol Binding
sms - MSISDN
link payload - new or modified links

A Registration update is also specified to be used to update the LWM2M server whenever the endpoint’s UDP port or IP address are changed.

10.2.4. LWM2M De-Register Endpoint

LWM2M allows for de-registration using the delete method on the returned location from the initial registration operation. LWM2M de-registration proceeds as described in Section 5.4.2.

11. Acknowledgments

Oscar Novo, Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Mohit Sethi, Sampo Ukkola, Linyi Tian, Chistian Amsuss, and Jan Newmarch have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

12. Changelog

changes from -09 to -10

- removed "ins" and "exp" link-format extensions.
- removed all text concerning DNS-SD.
- removed inconsistency in RDAO text.
- suggestions taken over from various sources
- replaced "Function Set" with "REST API", "base URI", "base path"
- moved simple registration to registration section

changes from -08 to -09

- clarified the "example use" of the base RD resource values /rd, /rd-lookup, and /rd-group.
- changed "ins" ABNF notation.
- various editorial improvements, including in examples
- clarifications for RDAO

changes from -07 to -08

- removed link target value returned from domain and group lookup types

- Maximum length of domain parameter 63 bytes for consistency with group
- removed option for simple POST of link data, don’t require a .well-known/core resource to accept POST data and handle it in a special way; we already have /rd for that
- add IPv6 ND Option for discovery of an RD
- clarify group configuration section 6.1 that endpoints must be registered before including them in a group
- removed all superfluous client-server diagrams
- simplified lighting example
- introduced Commissioning Tool
- RD-Look-up text is extended.

changes from -06 to -07

- added text in the discovery section to allow content format hints to be exposed in the discovery link attributes
- editorial updates to section 9
- update author information
- minor text corrections

Changes from -05 to -06

- added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

- added Update Endpoint Links using PATCH
- http access made explicit in interface specification
- Added http examples

Changes from -03 to -04:

- Added http response codes

- Clarified endpoint name usage
- Add application/link-format+cbor content-format

Changes from -02 to -03:

- Added an example for lighting and DNS integration
- Added an example for RD use in OMA LWM2M
- Added Read Links operation for link inspection by endpoints
- Expanded DNS-SD section
- Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

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

Changes from -00 to -01:

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

Changes from -05 to WG Document -00:

- Updated the version and date.

Changes from -04 to -05:

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

Changes from -03 to -04:

- Added the ins= parameter back for the DNS-SD mapping.
- Integrated the Simple Directory Discovery from Carsten.
- Editorial improvements.
- Fixed the use of ETags.
- Fixed tickets 383 and 372

Changes from -02 to -03:

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

Changes from -01 to -02:

- Added a terminology section.

- Changed the inclusion of an ETag in registration or update to a MAY.
- Added the concept of an RD Domain and a registration parameter for it.
- Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

13. References

13.1. Normative References

[I-D.ietf-core-links-json] Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", draft-ietf-core-links-json-06 (work in progress), July 2016.

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <http://www.rfc-editor.org/info/rfc6570>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <http://www.rfc-editor.org/info/rfc7396>.

13.2. Informative References

[RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <http://www.rfc-editor.org/info/rfc6775>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[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>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <http://www.rfc-editor.org/info/rfc7390>.

Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

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

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Phone: +1-707-502-5136
Email: Michael.Koster@smartthings.com

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

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

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 24, 2017

C. Jennings
Cisco
Z. Shelby
ARM
J. Arkko
A. Keranen
Ericsson
C. Bormann
Universitaet Bremen TZI

May 23, 2017

Media Types for Sensor Measurement Lists (SenML) draft-ietf-core-senml-08

Abstract

This specification defines media types for representing simple sensor measurements and device parameters in the Sensor Measurement Lists (SenML). Representations are defined in JavaScript Object Notation (JSON), Concise Binary Object Representation (CBOR), eXtensible Markup Language (XML), and Efficient XML Interchange (EXI), which share the common SenML data model. A simple sensor, such as a temperature sensor, could use this media type in protocols such as HTTP or CoAP to transport the measurements of the sensor or to be configured.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 24, 2017.

Table of Contents

1. Overview

Connecting sensors to the Internet is not new, and there have been many protocols designed to facilitate it. This specification defines new media types for carrying simple sensor information in a protocol such as HTTP or CoAP. This format was designed so that processors with very limited capabilities could easily encode a sensor measurement into the media type, while at the same time a server parsing the data could relatively efficiently collect a large number of sensor measurements. SenML can be used for a variety of data flow models, most notably data feeds pushed from a sensor to a collector, and the web resource model where the sensor is requested as a resource representation (e.g., "GET /sensor/temperature").

There are many types of more complex measurements and measurements that this media type would not be suitable for. SenML strikes a balance between having some information about the sensor carried with the sensor data so that the data is self describing but it also tries to make that a fairly minimal set of auxiliary information for efficiency reason. Other information about the sensor can be discovered by other methods such as using the CoRE Link Format [RFC6690].

SenML is defined by a data model for measurements and simple meta-data about measurements and devices. The data is structured as a single array that contains a series of SenML Records which can each contain attributes such as an unique identifier for the sensor, the time the measurement was made, the unit the measurement is in, and the current value of the sensor. Serializations for this data model are defined for JSON [RFC7159], CBOR [RFC7049], XML, and Efficient XML Interchange (EXI) [W3C.REC-exi-20140211].

For example, the following shows a measurement from a temperature gauge encoded in the JSON syntax.

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","v":23.1}
]
```

In the example above, the array has a single SenML Record with a measurement for a sensor named "urn:dev:ow:10e2073a01080063" with a current value of 23.1 degrees Celsius.

2. Requirements and Design Goals

The design goal is to be able to send simple sensor measurements in small packets on mesh networks from large numbers of constrained devices. Keeping the total size of payload under 80 bytes makes this easy to use on a wireless mesh network. It is always difficult to define what small code is, but there is a desire to be able to implement this in roughly 1 KB of flash on a 8 bit microprocessor. Experience with power meters and other large scale deployments has indicated that the solution needs to support allowing multiple measurements to be batched into a single HTTP or CoAP request. This "batch" upload capability allows the server side to efficiently support a large number of devices. It also conveniently supports batch transfers from proxies and storage devices, even in situations where the sensor itself sends just a single data item at a time. The multiple measurements could be from multiple related sensors or from the same sensor but at different times.

The basic design is an array with a series of measurements. The following example shows two measurements made at different times. The value of a measurement is in the "v" tag, the time of a measurement is in the "t" tag, the "n" tag has a unique sensor name, and the unit of the measurement is carried in the "u" tag.

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020076e+09,
   "v":23.5},
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020091e+09,
   "v":23.6}
]
```

To keep the messages small, it does not make sense to repeat the "n" tag in each SenML Record so there is a concept of a Base Name which is simply a string that is prepended to the Name field of all elements in that record and any records that follow it. So a more compact form of the example above is the following.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020076e+09,
   "v":23.5},
  {"u":"Cel","t":1.276020091e+09,
   "v":23.6}
]
```

In the above example the Base Name is in the "bn" tag and the "n" tags in each Record are the empty string so they are omitted.

Some devices have accurate time while others do not so SenML supports absolute and relative times. Time is represented in floating point as seconds and values greater than zero represent an absolute time relative to the Unix epoch while values of 0 or less represent a relative time in the past from the current time. A simple sensor with no absolute wall clock time might take a measurement every second, batch up 60 of them, and then send the batch to a server. It would include the relative time each measurement was made compared to the time the batch was sent in each SenML Record. The server might have accurate NTP time and use the time it received the data, and the relative offset, to replace the times in the SenML with absolute times before saving the SenML Pack in a document database.

3. Terminology

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

This document also uses the following terms:

SenML Record: One measurement or configuration instance in time presented using the SenML data model.

SenML Pack: One or more SenML Records in an array structure.

4. SenML Structure and Semantics

Each SenML Pack carries a single array that represents a set of measurements and/or parameters. This array contains a series of SenML Records with several attributes described below. There are two kind of attributes: base and regular. The base attributes can be included in the any SenML Record and they apply to the entries in the Record. Each base attribute also applies to all Records after it up to, but not including, the next Record that has that same base attribute. All base attributes are optional. Regular attributes can be included in any SenML Record and apply only to that Record.

4.1. Base attributes

Base Name: This is a string that is prepended to the names found in the entries.

Base Time: A base time that is added to the time found in an entry.

Base Unit: A base unit that is assumed for all entries, unless otherwise indicated. If a record does not contain a Unit value, then the Base Unit is used. Otherwise the value found in the Unit (if any) is used.

Base Value: A base value is added to the value found in an entry, similar to Base Time.

Base Sum: A base sum is added to the sum found in an entry, similar to Base Time.

Version: Version number of media type format. This attribute is an optional positive integer and defaults to 5 if not present. [RFC Editor: change the default value to 10 when this specification is published as an RFC and remove this note]

4.2. Regular attributes

Name: Name of the sensor or parameter. When appended to the Base Name attribute, this must result in a globally unique identifier for the resource. The name is optional, if the Base Name is present. If the name is missing, Base Name must uniquely identify the resource. This can be used to represent a large array of measurements from the same sensor without having to repeat its identifier on every measurement.

Unit: Units for a measurement value. Optional.

Value Value of the entry. Optional if a Sum value is present, otherwise required. Values are represented using basic data types. This specification defines floating point numbers ("v" field for "Value"), booleans ("vb" for "Boolean Value"), strings ("vs" for "String Value") and binary data ("vd" for "Data Value"). Exactly one value field MUST appear unless there is Sum field in which case it is allowed to have no Value field.

Sum: Integrated sum of the values over time. Optional. This attribute is in the units specified in the Unit value multiplied by seconds.

Time: Time when value was recorded. Optional.

Update Time: An optional time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement. This can be used to detect the failure of sensors or communications path from the sensor.

4.3. Considerations

The SenML format can be extended with further custom attributes. Both new base and regular attributes are allowed. See Section 12.2 for details. Implementations MUST ignore attributes they don't recognize unless that attribute has a label name that ends with the '_' character in which case an error MUST be generated.

Systems reading one of the objects MUST check for the Version attribute. If this value is a version number larger than the version which the system understands, the system SHOULD NOT use this object. This allows the version number to indicate that the object contains mandatory to understand attributes. New version numbers can only be defined in an RFC that updates this specification or it successors.

The Name value is concatenated to the Base Name value to get the name of the sensor. The resulting name needs to uniquely identify and differentiate the sensor from all others. It is RECOMMENDED that the full names are represented as URIs or URNs [RFC2141]. One way to create a unique name is to include some bit string that has guaranteed uniqueness (such as a 1-wire address) that is assigned to the device. Some of the examples in this draft use the device URN type as specified in [I-D.arkko-core-dev-urn]. UUIDs [RFC4122] are another way to generate a unique name. Note that long-term stable unique identifiers are problematic for privacy reasons and should be used with care or avoided as described in [RFC7721].

The resulting concatenated name MUST consist only of characters out of the set "A" to "Z", "a" to "z", "0" to "9", "-", ":", ".", or "_" and it MUST start with a character out of the set "A" to "Z", "a" to "z", or "0" to "9". This restricted character set was chosen so that these names can be directly used as in other types of URI including segments of an HTTP path with no special encoding and can be directly used in many databases and analytic systems. [RFC5952] contains advice on encoding an IPv6 address in a name.

If the Record has no Unit, the Base Unit is used as the Unit. Having no Unit and no Base Unit is allowed.

If either the Base Time or Time value is missing, the missing attribute is considered to have a value of zero. The Base Time and Time values are added together to get the time of measurement. A time of zero indicates that the sensor does not know the absolute

time and the measurement was made roughly "now". A negative value is used to indicate seconds in the past from roughly "now". A positive value is used to indicate the number of seconds, excluding leap seconds, since the start of the year 1970 in UTC.

If only one of the Base Sum or Sum value is present, the missing attribute is considered to have a value of zero. The Base Sum and Sum values are added together to get the sum of measurement. If neither the Base Sum or Sum are present, then the measurement does not have a sum value.

If the Base Value or Value is not present, the missing attribute(s) are considered to have a value of zero. The Base Value and Value are added together to get the value of the measurement.

Representing the statistical characteristics of measurements, such as accuracy, can be very complex. Future specification may add new attributes to provide better information about the statistical properties of the measurement.

4.4. Resolved Records

Sometimes it is useful to be able to refer to a defined normalized format for SenML records. This normalized format tends to get used for big data applications and intermediate forms when converting to other formats.

A SenML Record is referred to as "resolved" if it does not contain any base values and has no relative times, but the base values of the SenML Pack (if any) are applied to the Record. That is, name and base name are concatenated, base time is added to the time of the Record, if the Record did not contain Unit the Base Unit is applied to the record, etc. In addition the records need to be in chronological order. An example of this is show in Section 5.1.4.

Future specification that defines new base attributes need to specify how the attribute is resolved.

4.5. Associating Meta-data

SenML is designed to carry the minimum dynamic information about measurements, and for efficiency reasons does not carry significant static meta-data about the device, object or sensors. Instead, it is assumed that this meta-data is carried out of band. For web resources using SenML Packs, this meta-data can be made available using the CoRE Link Format [RFC6690]. The most obvious use of this link format is to describe that a resource is available in a SenML

format in the first place. The relevant media type indicator is included in the Content-Type (ct=) attribute.

5. JSON Representation (application/senml+json)

The SenML labels (JSON object member names) shown in Table 1 are used in JSON SenML Record attributes.

Name	label	Type
Base Name	bn	String
Base Time	bt	Number
Base Unit	bu	String
Base Value	bv	Number
Base Sum	bs	Number
Version	bver	Number
Name	n	String
Unit	u	String
Value	v	Number
String Value	vs	String
Boolean Value	vb	Boolean
Data Value	vd	String
Value Sum	s	Number
Time	t	Number
Update Time	ut	Number
Link	l	String

Table 1: JSON SenML Labels

The root content consists of an array with one JSON object for each SenML Record. All the fields in the above table MAY occur in the records with the type specified in the table.

Only the UTF-8 form of JSON is allowed. Characters in the String Value are encoded using the escape sequences defined in [RFC7159]. Octets in the Data Value are base64 encoded with URL safe alphabet as defined in Section 5 of [RFC4648].

Systems receiving measurements MUST be able to process the range of floating point numbers that are representable as an IEEE double precision floating point numbers [IEEE.754.1985]. The number of significant digits in any measurement is not relevant, so a reading of 1.1 has exactly the same semantic meaning as 1.10. If the value has an exponent, the "e" MUST be in lower case. The mantissa SHOULD be less than 19 characters long and the exponent SHOULD be less than

5 characters long. This allows time values to have better than micro second precision over the next 100 years.

5.1. Examples

5.1.1. Single Datapoint

The following shows a temperature reading taken approximately "now" by a 1-wire sensor device that was assigned the unique 1-wire address of 10e2073a01080063:

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","v":23.1}
]
```

5.1.2. Multiple Datapoints

The following example shows voltage and current now, i.e., at an unspecified time.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:","n":"voltage","u":"V","v":120.1},
  {"n":"current","u":"A","v":1.2}
]
```

The next example is similar to the above one, but shows current at Tue Jun 8 18:01:16.001 UTC 2010 and at each second for the previous 5 seconds.

```
[
  {"bn":"urn:dev:ow:10e2073a0108006:","bt":1.276020076001e+09,
   "bu":"A","bver":5,
   "n":"voltage","u":"V","v":120.1},
  {"n":"current","t":-5,"v":1.2},
  {"n":"current","t":-4,"v":1.3},
  {"n":"current","t":-3,"v":1.4},
  {"n":"current","t":-2,"v":1.5},
  {"n":"current","t":-1,"v":1.6},
  {"n":"current","v":1.7}
]
```

Note that in some usage scenarios of SenML the implementations MAY store or transmit SenML in a stream-like fashion, where data is collected over time and continuously added to the object. This mode of operation is optional, but systems or protocols using SenML in this fashion MUST specify that they are doing this. SenML defines a separate media type to indicate Sensor Streaming Measurement Lists (SensML) for this usage (see Section 12.3.1). In this situation the

SensML stream can be sent and received in a partial fashion, i.e., a measurement entry can be read as soon as the SenML Record is received and not have to wait for the full SensML Stream to be complete.

For instance, the following stream of measurements may be sent via a long lived HTTP POST from the producer of a SensML to the consumer of that, and each measurement object may be reported at the time it was measured:

```
[
  {"bn":"urn:dev:ow:10e2073a01080063","bt":1.320067464e+09,
   "bu":"%RH","v":21.2},
  {"t":10,"v":21.3},
  {"t":20,"v":21.4},
  {"t":30,"v":21.4},
  {"t":40,"v":21.5},
  {"t":50,"v":21.5},
  {"t":60,"v":21.5},
  {"t":70,"v":21.6},
  {"t":80,"v":21.7},
...
```

5.1.3. Multiple Measurements

The following example shows humidity measurements from a mobile device with a 1-wire address 10e2073a01080063, starting at Mon Oct 31 13:24:24 UTC 2011. The device also provides position data, which is provided in the same measurement or parameter array as separate entries. Note time is used to for correlating data that belongs together, e.g., a measurement and a parameter associated with it. Finally, the device also reports extra data about its battery status at a separate time.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063","bt":1.320067464e+09,
   "bu":"%RH","v":20},
  {"u":"lon","v":24.30621},
  {"u":"lat","v":60.07965},
  {"t":60,"v":20.3},
  {"u":"lon","t":60,"v":24.30622},
  {"u":"lat","t":60,"v":60.07965},
  {"t":120,"v":20.7},
  {"u":"lon","t":120,"v":24.30623},
  {"u":"lat","t":120,"v":60.07966},
  {"u":"%EL","t":150,"v":98},
  {"t":180,"v":21.2},
  {"u":"lon","t":180,"v":24.30628},
  {"u":"lat","t":180,"v":60.07967}
]
```

The size of this example represented in various forms, as well as that form compressed with gzip is given in the following table.

Encoding	Size	Compressed Size
JSON	573	206
XML	649	235
CBOR	254	196
EXI	162	185

Table 2: Size Comparisons

5.1.4. Resolved Data

The following shows the example from the previous section show in resolved format.

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067464e+09,
   "v":20},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067464e+09,
   "v":24.30621},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067464e+09,
   "v":60.07965},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067524e+09,
   "v":20.3},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067524e+09,
   "v":24.30622},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067524e+09,
   "v":60.07965},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067584e+09,
   "v":20.7},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067584e+09,
   "v":24.30623},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067584e+09,
   "v":60.07966},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%EL","t":1.320067614e+09,
   "v":98},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067644e+09,
   "v":21.2},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067644e+09,
   "v":24.30628},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067644e+09,
   "v":60.07967}
]
```

5.1.5. Multiple Data Types

The following example shows a sensor that returns different data types.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:","n":"temp","u":"Cel","v":23.1},
  {"n":"label","vs":"Machine Room"},
  {"n":"open","vb":false},
  {"n":"nfv-reader","vd":"aGkgCg=="}
]
```

5.1.6. Collection of Resources

The following example shows the results from a query to one device that aggregates multiple measurements from another devices. The example assumes that a client has fetched information from a device at 2001:db8::2 by performing a GET operation on http://[2001:db8::2] at Mon Oct 31 16:27:09 UTC 2011, and has gotten two separate values

as a result, a temperature and humidity measurement as well as the results from another device at http://[2001:db8::1] that also had a temperature and humidity. Note that the last record would use the Base Name from the 3rd record but the Base Time from the first record.

```
[
  {"bn":"2001:db8::2","bt":1.320078429e+09,
   "n":"temperature","u":"Cel","v":25.2},
  {"n":"humidity","u":"%RH","v":30},
  {"bn":"2001:db8::1",
   "n":"temperature","u":"Cel","v":12.3},
  {"n":"humidity","u":"%RH","v":67}
]
```

5.1.7. Setting an Actuator

The following example show the SenML that could be used to set the current set point of a typical residential thermostat which has a temperature set point, a switch to turn on and off the heat, and a switch to turn on the fan override.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:"},
  {"n":"temp","u":"Cel","v":23.1},
  {"n":"heat","u":"/","v":1},
  {"n":"fan","u":"/","v":0}
]
```

In the following example two different lights are turned on. It is assumed that the lights are on a 802.1BA network that can guarantee delivery of the messages to the two lights within 15 ms and uses 802.1AS for time synchronization. The controller has set the time of the lights coming on to 20 ms in the future from the current time. This allows both lights to receive the message, wait till that time, then apply the switch command so that both lights come on at the same time.

```
[
  {"bt":1.320078429e+09,"bu":"/","n":"2001:db8::3","v":1},
  {"n":"2001:db8::4","v":1}
]
```

The following shows two lights being turned off using a non deterministic network that has a high odds of delivering a message in less than 100 ms and uses NTP for time synchronization. The current time is 1320078429. The user has just turned off a light switch which is turning off two lights. Both lights are dimmed to 50%

brightness immediately to give the user instant feedback that something is changing. However given the network, the lights will probably dim at somewhat different times. Then 100 ms in the future, both lights will go off at the same time. The instant but not synchronized dimming gives the user the sensation of quick responses and the timed off 100 ms in the future gives the perception of both lights going off at the same time.

```
[
  {"bt":1.320078429e+09,"bu":"/","n":"2001:db8::3","v":0.5},
  {"n":"2001:db8::4","v":0.5},
  {"n":"2001:db8::3","t":0.1,"v":0},
  {"n":"2001:db8::4","t":0.1,"v":0}
]
```

6. CBOR Representation (application/senml+cbor)

The CBOR [RFC7049] representation is equivalent to the JSON representation, with the following changes:

- For JSON Numbers, the CBOR representation can use integers, floating point numbers, or decimal fractions (CBOR Tag 4); however a representation SHOULD be chosen such that when the CBOR value is converted back to an IEEE double precision floating point value, it has exactly the same value as the original Number. For the version number, only an unsigned integer is allowed.

- Characters in the String Value are encoded using a definite length text string (type 3). Octets in the Data Value are encoded using a definite length byte string (type 2) .

- For compactness, the CBOR representation uses integers for the map keys defined in Table 3. This table is conclusive, i.e., there is no intention to define any additional integer map keys; any extensions will use string map keys. This allows translators converting between CBOR and JSON representations to convert also all future labels without needing to update implementations.

Name	Label	CBOR Label
Version	bver	-1
Base Name	bn	-2
Base Time	bt	-3
Base Units	bu	-4
Base Value	bv	-5
Base Sum	bs	-6
Name	n	0
Units	u	1
Value	v	2
String Value	vs	3
Boolean Value	vb	4
Value Sum	s	5
Time	t	6
Update Time	ut	7
Data Value	vd	8
Link	l	9

Table 3: CBOR representation: integers for map keys

- For streaming SensML in CBOR representation, the array containing the records SHOULD be an CBOR indefinite length array while for non streaming SenML, a definite length array MUST be used.

The following example shows a dump of the CBOR example for the same sensor measurement as in Section 5.1.2.

```
0000 87 a7 21 78 1b 75 72 6e 3a 64 65 76 3a 6f 77 3a |..!x.urn:dev:ow:|
0010 31 30 65 32 30 37 33 61 30 31 30 38 30 30 36 3a |10e2073a0108006:|
0020 22 fb 41 d3 03 a1 5b 00 10 62 23 61 41 20 05 00 |".A...[..b#aA ..|
0030 67 76 6f 6c 74 61 67 65 01 61 56 02 fb 40 5e 06 |gvoltage.aV..@^.|
0040 66 66 66 66 66 a3 00 67 63 75 72 72 65 6e 74 06 |fffff..gcurrent.|
0050 24 02 fb 3f f3 33 33 33 33 33 33 a3 00 67 63 75 |$..?.333333..gcu|
0060 72 72 65 6e 74 06 23 02 fb 3f f4 cc cc cc cc cc |rrent.#..?......|
0070 cd a3 00 67 63 75 72 72 65 6e 74 06 22 02 fb 3f |...gcurrent."..?|
0080 f6 66 66 66 66 66 66 a3 00 67 63 75 72 72 65 6e |.ffffff..gcurren|
0090 74 06 21 02 f9 3e 00 a3 00 67 63 75 72 72 65 6e |t.!..>...gcurren|
00a0 74 06 20 02 fb 3f f9 99 99 99 99 99 9a a3 00 67 |t. ..?.........g|
00b0 63 75 72 72 65 6e 74 06 00 02 fb 3f fb 33 33 33 |current....?.333|
00c0 33 33 33                                       |333|
00c3
```

7. XML Representation (application/senml+xml)

A SenML Pack or Stream can also be represented in XML format as defined in this section.

Only the UTF-8 form of XML is allowed. Characters in the String Value are encoded using the escape sequences defined in [RFC7159]. Octets in the Data Value are base64 encoded with URL safe alphabet as defined in Section 5 of [RFC4648].

The following example shows an XML example for the same sensor measurement as in Section 5.1.2.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a0108006:" bt="1.276020076001e+09"
  bu="A" bver="5" n="voltage" u="V" v="120.1"></senml>
  <senml n="current" t="-5" v="1.2"></senml>
  <senml n="current" t="-4" v="1.3"></senml>
  <senml n="current" t="-3" v="1.4"></senml>
  <senml n="current" t="-2" v="1.5"></senml>
  <senml n="current" t="-1" v="1.6"></senml>
  <senml n="current" v="1.7"></senml>
</sensml>
```

The SenML Stream is represented as a sensml tag that contains a series of senml tags for each SenML Record. The SenML Fields are represents as XML attributes. The following table shows the mapping of the SenML labels, which are used for the attribute name, to the attribute types used in the XML senml tags.

Name	Label	Type
Base Name	bn	string
Base Time	bt	double
Base Unit	bu	string
Base Value	bv	double
Base Sum	bs	double
Base Version	bver	int
Name	n	string
Unit	u	string
Value	v	double
String Value	vs	string
Data Value	vd	string
Boolean Value	vb	boolean
Value Sum	s	double
Time	t	double
Update Time	ut	double
Link	l	string

Table 4: XML SenML Labels

The RelaxNG schema for the XML is:

```
default namespace = "urn:ietf:params:xml:ns:senml"
namespace rng = "http://relaxng.org/ns/structure/1.0"

senml = element senml {
  attribute bn { xsd:string }?,
  attribute bt { xsd:double }?,
  attribute bv { xsd:double }?,
  attribute bs { xsd:double }?,
  attribute bu { xsd:string }?,
  attribute bver { xsd:int }?,

  attribute l { xsd:string }?,

  attribute n { xsd:string }?,
  attribute s { xsd:double }?,
  attribute t { xsd:double }?,
  attribute u { xsd:string }?,
  attribute ut { xsd:double }?,

  attribute v { xsd:double }?,
  attribute vb { xsd:boolean }?,
  attribute vs { xsd:string }?,
  attribute vd { xsd:string }?
}

sensml =
  element sensml {
    senml+
}

start = sensml
```

8. EXI Representation (application/senml+exi)

For efficient transmission of SenML over e.g. a constrained network, Efficient XML Interchange (EXI) can be used. This encodes the XML Schema structure of SenML into binary tags and values rather than ASCII text. An EXI representation of SenML SHOULD be made using the strict schema-mode of EXI. This mode however does not allow tag extensions to the schema, and therefore any extensions will be lost in the encoding. For uses where extensions need to be preserved in EXI, the non-strict schema mode of EXI MAY be used.

The EXI header MUST include an "EXI Options", as defined in [W3C.REC-exi-20140211], with an schemaId set to the value of "a" indicating the schema provided in this specification. Future revisions to the schema can change the value of the schemaId to allow for backwards compatibility. When the data will be transported over

CoAP or HTTP, an EXI Cookie SHOULD NOT be used as it simply makes things larger and is redundant to information provided in the Content-Type header.

The following is the XSD Schema to be used for strict schema guided EXI processing. It is generated from the RelaxNG.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="urn:ietf:params:xml:ns:senml"
xmlns:ns1="urn:ietf:params:xml:ns:senml">
  <xs:element name="senml">
    <xs:complexType>
      <xs:attribute name="bn" type="xs:string" />
      <xs:attribute name="bt" type="xs:double" />
      <xs:attribute name="bv" type="xs:double" />
      <xs:attribute name="bs" type="xs:double" />
      <xs:attribute name="bu" type="xs:string" />
      <xs:attribute name="bver" type="xs:int" />
      <xs:attribute name="l" type="xs:string" />
      <xs:attribute name="n" type="xs:string" />
      <xs:attribute name="s" type="xs:double" />
      <xs:attribute name="t" type="xs:double" />
      <xs:attribute name="u" type="xs:string" />
      <xs:attribute name="ut" type="xs:double" />
      <xs:attribute name="v" type="xs:double" />
      <xs:attribute name="vb" type="xs:boolean" />
      <xs:attribute name="vs" type="xs:string" />
      <xs:attribute name="vd" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="sensml">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="ns1:senml" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The following shows a hexdump of the EXI produced from encoding the following XML example. Note this example is the same information as the first example in Section 5.1.2 in JSON format.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a01080063:" n="voltage" u="V"
  v="120.1"></senml>
  <senml n="current" u="A" v="1.2"></senml>
</sensml>
```

Which compresses with EXI to the following displayed in hexdump:

```
0000 a0 30 0d 84 80 79 d5 c9 b8 e9 91 95 d8 e9 bd dc |.0...y.........|
0010 e8 c4 c1 94 c8 c0 dc cd 84 c0 c4 c0 e0 c0 c0 d8 |................|
0020 cc e9 82 5d 9b db 1d 18 59 d9 48 0d 58 ac 42 60 |...]....Y.H.X.B‘|
0030 18 e1 2c 6e ae 4e 4c ad ce 84 06 82 41 90 0e    |..,n.NL.....A..|
003f
```

The above example used the bit packed form of EXI but it is also possible to use a byte packed form of EXI which can makes it easier for a simple sensor to produce valid EXI without really implementing EXI. Consider the example of a temperature sensor that produces a value in tenths of degrees Celsius over a range of 0.0 to 55.0. It would produce an XML SenML file such as:

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml n="urn:dev:ow:10e2073a01080063" u="Cel" v="23.1"></senml>
</sensml>
```

The compressed form, using the byte alignment option of EXI, for the above XML is the following:

```
0000 a0 00 48 80 6c 20 01 07 1d 75 72 6e 3a 64 65 76 |..H.l ...urn:dev|
0010 3a 6f 77 3a 31 30 65 32 30 37 33 61 30 31 30 38 |:ow:10e2073a0108|
0020 30 30 36 33 02 05 43 65 6c 01 00 e7 01 01 00 03 |0063..Cel.......|
0030 01                                              |.|
0031
```

A small temperature sensor devices that only generates this one EXI file does not really need an full EXI implementation. It can simply hard code the output replacing the 1-wire device ID starting at byte 0x20 and going to byte 0x2F with it’s device ID, and replacing the value "0xe7 0x01" at location 0x37 and 0x38 with the current temperature. The EXI Specification [W3C.REC-exi-20140211] contains the full information on how floating point numbers are represented, but for the purpose of this sensor, the temperature can be converted to an integer in tenths of degrees (231 in this example). EXI stores 7 bits of the integer in each byte with the top bit set to one if there are further bytes. So the first bytes at is set to low 7 bits of the integer temperature in tenths of degrees plus 0x80. In this example 231 & 0x7F + 0x80 = 0xE7. The second byte is set to the

integer temperature in tenths of degrees right shifted 7 bits. In this example 231 >> 7 = 0x01.

9. Fragment Identification Methods

A SenML Pack typically consists of multiple SenML Records and for some applications it may be useful to be able to refer with a Fragment Identifier to a single record, or a set of records, in a Pack. The fragment identifier is only interpreted by a client and does not impact retrieval of a representation. The SenML Fragment Identification is modeled after CSV Fragment Identifiers [RFC7111].

To select a single SenML Record, the "rec" scheme followed by a single number is used. For the purpose of numbering records, the first record is at position 1. A range of records can be selected by giving the first and the last record number separated by a '-' character. Instead of the second number, the '*' character can be used to indicate the last Senml Record in the Pack. A set of records can also be selected using a comma separated list of record positions or ranges.

(We use the term "selecting a record" for identifying it as part of the fragment, not in the sense of isolating it from the Pack -- the record still needs to be interpreted as part of the Pack, e.g., using the base values defined in record 1.)

9.1. Fragment Identification Examples

The 3rd SenML Record from "coap://example.com/temp" resource can be selected with:

coap://example.com/temp#rec=3

Records from 3rd to 6th can be selected with:

coap://example.com/temp#rec=3-6

Records from 19th to the last can be selected with:

coap://example.com/temp#rec=19-*

The 3rd and 5th record can be selected with:

coap://example.com/temp#rec=3,5

To select the Records from third to fifth, the 10th record, and all from 19th to the last:

coap://example.com/temp#rec=3-5,10,19-*

10. Usage Considerations

The measurements support sending both the current value of a sensor as well as the an integrated sum. For many types of measurements, the sum is more useful than the current value. For example, an electrical meter that measures the energy a given computer uses will typically want to measure the cumulative amount of energy used. This is less prone to error than reporting the power each second and trying to have something on the server side sum together all the power measurements. If the network between the sensor and the meter goes down over some period of time, when it comes back up, the cumulative sum helps reflect what happened while the network was down. A meter like this would typically report a measurement with the units set to watts, but it would put the sum of energy used in the "s" attribute of the measurement. It might optionally include the current power in the "v" attribute.

While the benefit of using the integrated sum is fairly clear for measurements like power and energy, it is less obvious for something like temperature. Reporting the sum of the temperature makes it easy to compute averages even when the individual temperature values are not reported frequently enough to compute accurate averages. Implementors are encouraged to report the cumulative sum as well as the raw value of a given sensor.

Applications that use the cumulative sum values need to understand they are very loosely defined by this specification, and depending on the particular sensor implementation may behave in unexpected ways. Applications should be able to deal with the following issues:

1. Many sensors will allow the cumulative sums to "wrap" back to zero after the value gets sufficiently large.

2. Some sensors will reset the cumulative sum back to zero when the device is reset, loses power, or is replaced with a different sensor.

3. Applications cannot make assumptions about when the device started accumulating values into the sum.

Typically applications can make some assumptions about specific sensors that will allow them to deal with these problems. A common assumption is that for sensors whose measurement values are always positive, the sum should never get smaller; so if the sum does get smaller, the application will know that one of the situations listed above has happened.

11. CDDL

For reference, the JSON and CBOR representations can be described with the common CDDL [I-D.greevenbosch-appsawg-cbor-cddl] specification in Figure 1.

```
SenML-Pack = [ record, * record]

record = {
  ? bn => tstr,         ; Base Name
  ? bt => numeric,      ; Base Time
  ? bu => tstr,         ; Base Units
  ? bv => numeric,      ; Base value
  ? bs => numeric,      ; Base sum
  ? bver => uint,       ; Base Version
  ? n => tstr,         ; Name
  ? u => tstr,         ; Units
  ? s => numeric,      ; Value Sum
  ? t => numeric,      ; Time
  ? ut => numeric,     ; Update Time
  ? l => tstr,         ; Link
  ? ( v => numeric //  ; Numeric Value
      vs => tstr //    ; String Value
      vb => bool //    ; Boolean Value
      vd => binary-value ) ; Data Value
  * key-value-pair
}

; now define the generic versions
key-value-pair = ( label => value )

label = tstr .regexp  "[A-Za-z0-9][-_:.A-Za-z0-9]*" / uint

value = tstr / binary-value / numeric / bool
numeric = number / decfrac
```

Figure 1: Common CDDL specification for CBOR and JSON SenML

For JSON, we use text labels and base64url-encoded binary data (Figure 2).

```
bver = "bver" n  = "n"   s  = "s"
bn  = "bn"    u  = "u"   t  = "t"
bt  = "bt"    v  = "v"   ut = "ut"
bu  = "bu"    vs = "vs"  vd = "vd"
bv  = "bv"    vb = "vb"  l  = "l"
bs  = "bs"

binary-value = tstr             ; base64url encoded
```

Figure 2: JSON-specific CDDL specification for SenML

For CBOR, we use integer labels and native binary data (Figure 3).

```
bver = -1  n  = 0   s  = 5
bn  = -2   u  = 1   t  = 6
bt  = -3   v  = 2   ut = 7
bu  = -4   vs = 3   vd = 8
bv  = -5   vb = 4   l  = 9
bs  = -6

binary-value = bstr
```

Figure 3: CBOR-specific CDDL specification for SenML

12. IANA Considerations

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

12.1. Units Registry

IANA will create a registry of SenML unit symbols. The primary purpose of this registry is to make sure that symbols uniquely map to give type of measurement. Definitions for many of these units can be found in location such as [NIST811] and [BIPM]. Units marked with an asterisk are NOT RECOMMENDED to be produced by new implementations, but are in active use and SHOULD be implemented by consumers that can use the related base units.

Symbol	Description	Type	Reference
m	meter	float	RFC-AAAA
kg	kilogram	float	RFC-AAAA
g	gram*	float	RFC-AAAA
s	second	float	RFC-AAAA
A	ampere	float	RFC-AAAA
K	kelvin	float	RFC-AAAA

cd	candela	float	RFC-AAAA
mol	mole	float	RFC-AAAA
Hz	hertz	float	RFC-AAAA
rad	radian	float	RFC-AAAA
sr	steradian	float	RFC-AAAA
N	newton	float	RFC-AAAA
Pa	pascal	float	RFC-AAAA
J	joule	float	RFC-AAAA
W	watt	float	RFC-AAAA
C	coulomb	float	RFC-AAAA
V	volt	float	RFC-AAAA
F	farad	float	RFC-AAAA
Ohm	ohm	float	RFC-AAAA
S	siemens	float	RFC-AAAA
Wb	weber	float	RFC-AAAA
T	tesla	float	RFC-AAAA
H	henry	float	RFC-AAAA
Cel	degrees Celsius	float	RFC-AAAA
lm	lumen	float	RFC-AAAA
lx	lux	float	RFC-AAAA
Bq	becquerel	float	RFC-AAAA
Gy	gray	float	RFC-AAAA
Sv	sievert	float	RFC-AAAA
kat	katal	float	RFC-AAAA
m2	square meter (area)	float	RFC-AAAA
m3	cubic meter (volume)	float	RFC-AAAA
l	liter (volume)*	float	RFC-AAAA
m/s	meter per second (velocity)	float	RFC-AAAA
m/s2	meter per square second (acceleration)	float	RFC-AAAA
m3/s	cubic meter per second (flow rate)	float	RFC-AAAA
l/s	liter per second (flow rate)*	float	RFC-AAAA
W/m2	watt per square meter (irradiance)	float	RFC-AAAA
cd/m2	candela per square meter (luminance)	float	RFC-AAAA
bit	bit (information content)	float	RFC-AAAA
bit/s	bit per second (data rate)	float	RFC-AAAA
lat	degrees latitude (note 2)	float	RFC-AAAA
lon	degrees longitude (note 2)	float	RFC-AAAA
pH	pH value (acidity; logarithmic quantity)	float	RFC-AAAA
dB	decibel (logarithmic quantity)	float	RFC-AAAA
Bspl	bel (sound pressure level; logarithmic quantity)*	float	RFC-AAAA
count	1 (counter value)	float	RFC-AAAA
/	1 (Ratio e.g., value of a switch, note 1)	float	RFC-AAAA
%	1 (Ratio e.g., value of a switch,	float	RFC-AAAA

	note 1)*			
%RH	Percentage (Relative Humidity)	float	RFC-AAAA	
%EL	Percentage (remaining battery energy level)	float	RFC-AAAA	
EL	seconds (remaining battery energy level)	float	RFC-AAAA	
1/s	1 per second (event rate)	float	RFC-AAAA	
1/min	1 per minute (event rate, "rpm")*	float	RFC-AAAA	
beat/min	1 per minute (Heart rate in beats per minute)*	float	RFC-AAAA	
beats	1 (Cumulative number of heart beats)*	float	RFC-AAAA	
S/m	Siemens per meter (conductivity)	float	RFC-AAAA	

Table 5

- Note 1: A value of 0.0 indicates the switch is off while 1.0 indicates on and 0.5 would be half on. The preferred name of this unit is "/". For historical reasons, the name "%" is also provided for the same unit - but note that while that name strongly suggests a percentage (0..100) -- it is however NOT a percentage, but the absolute ratio!

- Note 2: Assumed to be in WGS84 unless another reference frame is known for the sensor.

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. Experts should exercise their own good judgment but need to consider the following guidelines:

1. There needs to be a real and compelling use for any new unit to be added.

2. Units should define the semantic information and be chosen carefully. Implementors need to remember that the same word may be used in different real-life contexts. For example, degrees when measuring latitude have no semantic relation to degrees when measuring temperature; thus two different units are needed.

3. These measurements are produced by computers for consumption by computers. The principle is that conversion has to be easily be done when both reading and writing the media type. The value of a single canonical representation outweighs the convenience of easy human representations or loss of precision in a conversion.

4. Use of SI prefixes such as "k" before the unit is not recommended. Instead one can represent the value using scientific notation such a 1.2e3. The "kg" unit is exception to this rule since it is an SI base unit; the "g" unit is provided for legacy compatibility.

5. For a given type of measurement, there will only be one unit type defined. So for length, meters are defined and other lengths such as mile, foot, light year are not allowed. For most cases, the SI unit is preferred.

6. Symbol names that could be easily confused with existing common units or units combined with prefixes should be avoided. For example, selecting a unit name of "mph" to indicate something that had nothing to do with velocity would be a bad choice, as "mph" is commonly used to mean miles per hour.

7. The following should not be used because the are common SI prefixes: Y, Z, E, P, T, G, M, k, h, da, d, c, n, u, p, f, a, z, y, Ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi.

8. The following units should not be used as they are commonly used to represent other measurements Ky, Gal, dyn, etg, P, St, Mx, G, Oe, Gb, sb, Lmb, mph, Ci, R, RAD, REM, gal, bbl, qt, degF, Cal, BTU, HP, pH, B/s, psi, Torr, atm, at, bar, kWh.

9. The unit names are case sensitive and the correct case needs to be used, but symbols that differ only in case should not be allocated.

10. A number after a unit typically indicates the previous unit raised to that power, and the / indicates that the units that follow are the reciprocal. A unit should have only one / in the name.

11. A good list of common units can be found in the Unified Code for Units of Measure [UCUM].

12.2. SenML Label Registry

IANA will create a new registry for SenML labels. The initial content of the registry is:

Name	Label	CBOR	XML Type	ID	Note
Base Name	bn	-2	string	a	RFCXXXX
Base Sum	bs	-6	double	a	RFCXXXX
Base Time	bt	-3	double	a	RFCXXXX
Base Unit	bu	-4	string	a	RFCXXXX
Base Value	bv	-5	double	a	RFCXXXX
Base Version	bver	-1	int	a	RFCXXXX
Boolean Value	vb	4	boolean	a	RFCXXXX
Data Value	vd	8	string	a	RFCXXXX
Name	n	0	string	a	RFCXXXX
String Value	vs	3	string	a	RFCXXXX
Time	t	6	double	a	RFCXXXX
Unit	u	1	string	a	RFCXXXX
Update Time	ut	7	double	a	RFCXXXX
Value	v	2	double	a	RFCXXXX
Value Sum	s	5	double	a	RFCXXXX
Link	l	9	string	a	RFCXXXX

Table 6: SenML Labels

Note to RFC Editor. Please replace RFCXXXX with the number for this RFC.

All new entries must define the Label Name, Label, and XML Type but the CBOR labels SHOULD be left empty as CBOR will use the string encoding for any new labels. The ID fields contains the EXI schemaId value of the first Schema which includes this label or is empty if this label was not intended for use with EXI. The Note field SHOULD contain information about where to find out more information about this label.

The JSON, CBOR, and EXI types are derived from the XML type. All XML numeric types such as double, float, integer and int become a JSON Number. XML boolean and string become a JSON Boolean and String respectively. CBOR represents numeric values with a CBOR type that does not loose any information from the JSON value. EXI uses the XML types.

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. Experts should exercise their own good judgment but need to consider that shorter labels should have more strict review.

All new SenML labels that have "base" semantics (see Section 4.1) MUST start with character 'b'. Regular labels MUST NOT start with that character.

Extensions that add a label that is intended for use with XML need to create a new RelaxNG scheme that includes all the labels in the IANA registry.

Extensions that add a label that is intended for use with EXI need to create a new XSD Schema that includes all the labels in the IANA registry then allocate a new EXI schemaId value. Moving to the next letter in the alphabet is the suggested way to create the new value for the EXI schemaId. Any labels with previously blank ID values SHOULD be updated in the IANA table to have their ID set to this new schemaId value.

Extensions that are mandatory to understand to correctly process the Pack MUST have a label name that ends with the '_' character.

12.3. Media Type Registration

The following registrations are done following the procedure specified in [RFC6838] and [RFC7303]. Clipboard formats are defined for the JSON and XML form of lists but do not make sense for streams or other formats.

Note to RFC Editor - please remove this paragraph. Note that a request for media type review for senml+json was sent to the media-types@iana.org on Sept 21, 2010. A second request for all the types was sent on October 31, 2016.

12.3.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC7159]. See RFC-AAAA for details. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any JSON key value pairs that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+json is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senml

Windows Clipboard Name: "JSON Sensor Measurement List"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-json conforms to public.text

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.2. sensml+json Media Type Registration

Type name: application

Subtype name: sensml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC7159]. See RFC-AAAA for details. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any JSON key value pairs that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+json is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.3. senml+cbor Media Type Registration

Type name: application

Subtype name: senml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049]. See RFC-AAAA for details.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any key value pairs that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the CBOR object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental

information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+cbor is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmlc

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-cbor conforms to public.data

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.4. sensml+cbor Media Type Registration

Type name: application

Subtype name: sensml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049]. See RFC-AAAA for details.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide

security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any key value pairs that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the CBOR object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+cbor is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmlc

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.5. senml+xml Media Type Registration

Type name: application

Subtype name: senml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-xml-20081126]. See RFC-AAAA for details.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any tags or attributes that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" attribute in the senml tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+xml is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmlx

Windows Clipboard Name: "XML Sensor Measurement List"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-xml conforms to public.xml

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.6. sensml+xml Media Type Registration

Type name: application

Subtype name: sensml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-xml-20081126]. See RFC-AAAA for details.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any tags or attributes that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" attribute in the senml tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+xml is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmlx

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.7. senml+exi Media Type Registration

Type name: application

Subtype name: senml+exi

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-exi-20140211]. See RFC-AAAA for details.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any tags or attributes that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" attribute in the senml tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML.

Further information on using schemas to guide the EXI can be found in RFC-AAAA.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+exi is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmle

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-exi conforms to public.data

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.8. sensml+exi Media Type Registration

Type name: application

Subtype name: sensml+exi

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-exi-20140211]. See RFC-AAAA for details.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any tags or attributes that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" attribute in the senml tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML. Further information on using schemas to guide the EXI can be found in RFC-AAAA.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+exi is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmle

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.4. XML Namespace Registration

This document registers the following XML namespaces in the IETF XML registry defined in [RFC3688].

URI: urn:ietf:params:xml:ns:senml

Registrant Contact: The IESG.

XML: N/A, the requested URIs are XML namespaces

12.5. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the SenML media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. All IDs are assigned from the "Expert Review" (0-255) range. The assigned IDs are show in Table 7.

Media type	ID
application/senml+json	TBD
application/sensml+json	TBD
application/senml+cbor	TBD
application/sensml+cbor	TBD
application/senml+xml	TBD
application/sensml+xml	TBD
application/senml+exi	TBD
application/sensml+exi	TBD

Table 7: CoAP Content-Format IDs

13. Security Considerations

See Section 14. Further discussion of security properties can be found in Section 12.3.

14. Privacy Considerations

Sensor data can range from information with almost no security considerations, such as the current temperature in a given city, to highly sensitive medical or location data. This specification provides no security protection for the data but is meant to be used inside another container or transport protocol such as S/MIME or HTTP with TLS that can provide integrity, confidentiality, and authentication information about the source of the data.

15. Acknowledgement

We would like to thank Alexander Pelov, Andrew McClure, Andrew Mcgregor, Bjoern Hoehrmann, Christian Amsuess, Christian Groves, Daniel Peintner, Jan-Piet Mens, Joe Hildebrand, John Klensin, Karl Palsson, Lennart Duhrsen, Lisa Dusseault, Lyndsay Campbell, Martin Thomson, Michael Koster, and Stephen Farrell, for their review comments.

16. References

16.1. Normative References

[BIPM] Bureau International des Poids et Mesures, "The International System of Units (SI)", 8th edition, 2006.

[IEEE.754.1985] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE Standard 754, August 1985.

[NIST811] Thompson, A. and B. Taylor, "Guide for the Use of the International System of Units (SI)", NIST Special Publication 811, 2008.

[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>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[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>.

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <http://www.rfc-editor.org/info/rfc7303>.

[W3C.REC-exi-20140211] Schneider, J., Kamiya, T., Peintner, D., and R. Kyusakov, "Efficient XML Interchange (EXI) Format 1.0 (Second Edition)", World Wide Web Consortium Recommendation REC-exi-20140211, February 2014, <http://www.w3.org/TR/2014/REC-exi-20140211>.

[W3C.REC-xml-20081126] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

16.2. 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-03 (work in progress), July 2012.

[I-D.greevenbosch-appsawg-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-10 (work in progress), March 2017.

[I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", draft-ietf-core-links-json-08 (work in progress), April 2017.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, DOI 10.17487/RFC2141, May 1997, <http://www.rfc-editor.org/info/rfc2141>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <http://www.rfc-editor.org/info/rfc4122>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7111] Hausenblas, M., Wilde, E., and J. Tennison, "URI Fragment Identifiers for the text/csv Media Type", RFC 7111, DOI 10.17487/RFC7111, January 2014, <http://www.rfc-editor.org/info/rfc7111>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <http://www.rfc-editor.org/info/rfc7721>.

[UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure (UCUM)", Regenstrief Institute and Indiana University School of Informatics, 2013, <http://unitsofmeasure.org/ucum.html>.

Appendix A. Links Extension

An attribute to support a link extension for SenML is defined as a string attribute by this specification. The link extension can be used for additional information about a SenML Record. The definition

and usage of the contents of this value are specified in [I-D.ietf-core-links-json].

For JSON and XML the attribute has a label of "l" and a value that is a string.

The following shows an example of the links extension.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:","bt":1.320078429e+09,
   "l":"[{\"href\":\"humidity\",\"foo\":\"bar1\"}",
   "n":"temperature","u":"Cel","v":27.2},
  {"n":"humidity","u":"%RH","v":80}
]
```

Authors' Addresses

Cullen Jennings
Cisco
400 3rd Avenue SW
Calgary, AB T2P 4H2
Canada

Email: fluffy@iii.ca

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

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

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

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

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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: November 3, 2017

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
R. Turner
Landis+Gyr
A. Minaburo
Acklio
A. Somaraju
Tridonic GmbH & Co KG
May 02, 2017

YANG Schema Item iDentifier (SID)
draft-ietf-core-sid-01

Abstract

YANG Schema Item iDentifiers (SID) are globally unique 64-bit numeric identifiers used to identify all items used in YANG. This document defines the semantics, the registration, and assignment processes of SIDs. To enable the implementation of these processes, this document also defines a file format used to persist and publish assigned SIDs.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 3, 2017.

Table of Contents

1. Introduction

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called SID, is encoded using a 64-bit unsigned integer. The following items are identified using SIDs:

- identities
- data nodes
- RPCs and associated input(s) and output(s)
- actions and associated input(s) and output(s)
- notifications and associated information

- YANG modules, submodules and features

To minimize their size, SIDs are often represented as a difference between the current SID and a reference SID. Such difference is called "delta", shorthand for "delta-encoded SID". Conversion from SIDs to deltas and back to SIDs is a stateless process. Each protocol implementing deltas must unambiguously define the reference SID for each YANG item.

SIDs are globally unique numbers, a registration system is used in order to guarantee their uniqueness. SIDs are registered in blocks called "SID ranges". Section 6.1 provide more details about the registration process of SID range(s).

Assignment of SIDs to YANG items can be automated, the recommended process to assign SIDs is as follows:

- A tool extracts the different items defined for a specific YANG module.

- The list of items is ordered in alphabetical order by type and label. Valid types and label formats are described within the 'ietf-sid-file' YANG module defined in Section 4.

- SIDs are assigned sequentially from the entry point up to the size of the registered SID range. This approach is recommended to minimize the serialization overhead, especially when delta encoding is implemented.

- If the number of items exceeds the SID range(s) allocated to a YANG module, an extra range is added for subsequent assignments.

SIDs are assigned permanently, items introduced by a new revision of a YANG module are added to the list of SIDs already assigned. This process can also be automated using the same method described above except that the assignment restart from the highest SID already assigned plus one.

To avoid duplicate assignment of SIDs, the registration of the SIDs assigned to YANG module(s) is recommended. Section 6.2 provide more details about the registration process of YANG modules and associated SIDs. To enable the implementation of this registry, Section 4 defines a standard file format used to store and publish SIDs.

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- action
- feature
- module
- notification
- RPC
- schema node
- schema tree
- submodule

This specification also makes use of the following terminology:

- delta : Difference between the current SID and a reference SID. A reference SID is defined for each context for which deltas are used.
- item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- path: A path is a string that identifies a schema node within the schema tree. A path consists of the list of schema node identifier(s) separated by slashes ("/"). Schema node identifier(s) are always listed from the top-level schema node up to the targeted schema node. (e.g. "/system-state/clock/current-datetime")
- YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

3. ".sid" file lifecycle

YANG is a language designed to model data sent between clients and servers using one of the compatible protocol (e.g. NETCONF [RFC6241], RESCONF [RFC8040] and CoMI [I-D.ietf-core-comi]). A YANG

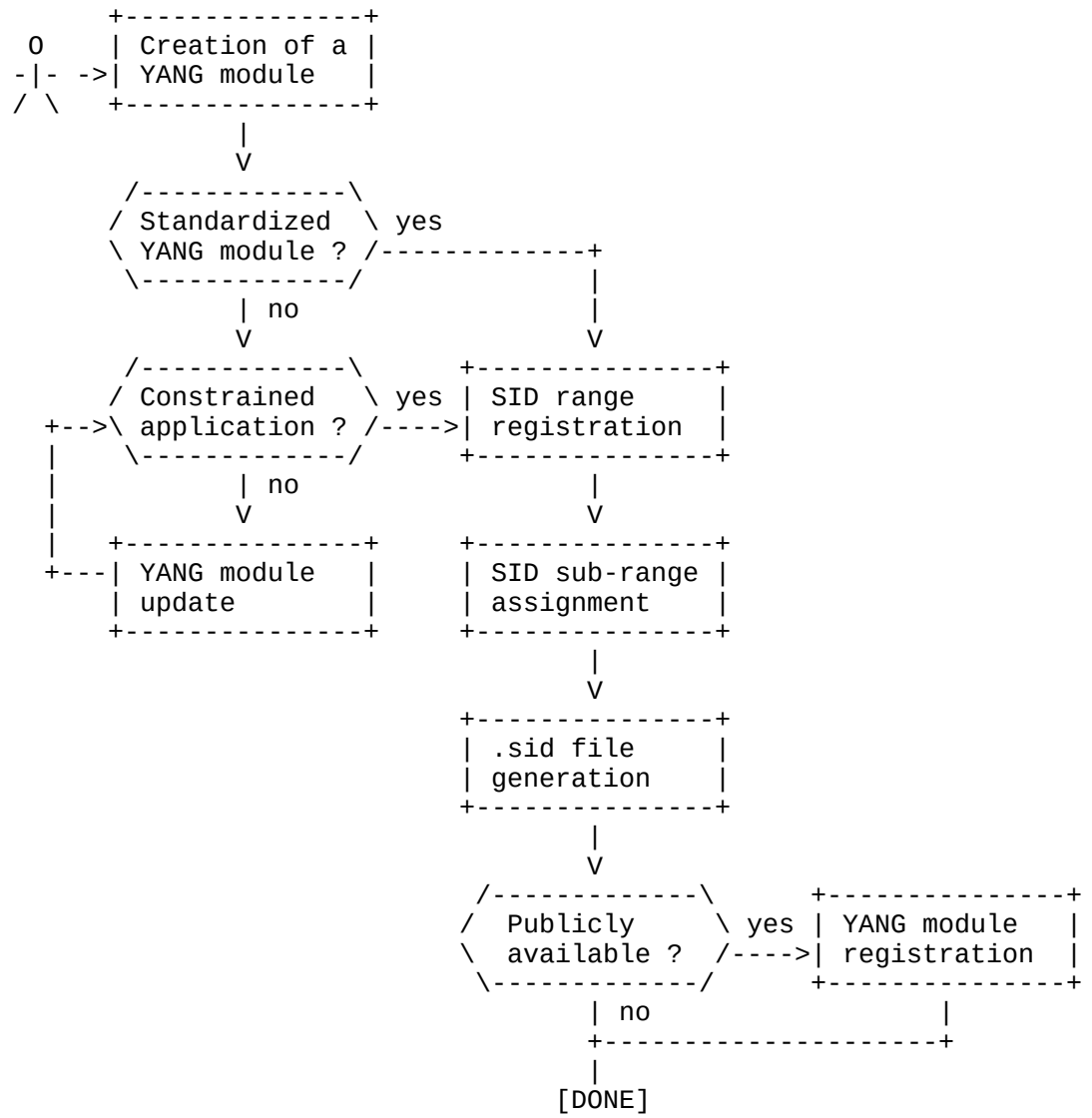
module defines hierarchies of data, including configuration, state data, RPCs, actions and notifications.

YANG modules are not necessary created in the context of constrained applications. YANG modules can be implemented using NETCONF [RFC6241] or RESTCONF [RFC8040] without the need to assign SIDs.

As needed, authors of YANG modules can assign SIDs to their YANG modules. This process starts by the registration of a SID range. Once a SID range is registered, the owner of this range assigns sub-ranges to each YANG module in order to generate the associated ".sid" files. Generation of ".sid" files SHOULD be performed using an automated tool.

Registration of the .sid file associated to a YANG module is optional but recommended to promote interoperability between devices and to avoid duplicate allocation of SIDs to a single YANG module.

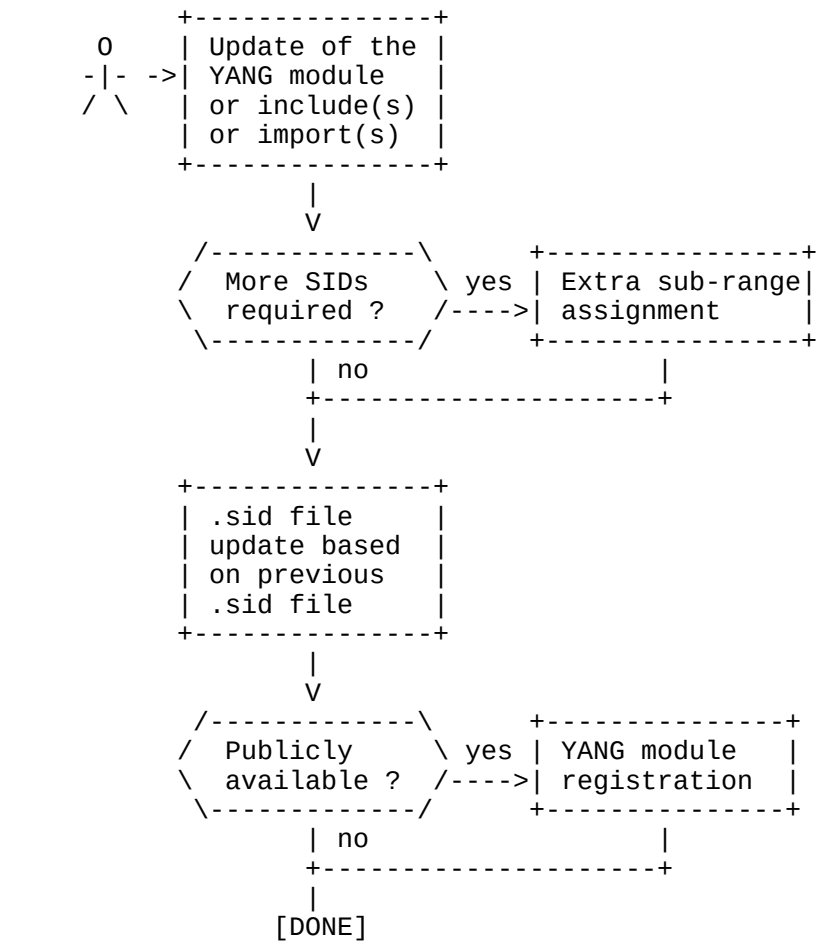
The following activity diagram summarizes the creation of a YANG module and its associated .sid file.



Each time a YANG module or one of its imported module(s) or included sub-module(s) is updated, the ".sid" file MAY need to be updated. This update SHOULD also be performed using an automated tool.

If a new revision requires more SIDs than initially allocated, a new SID range MUST be added to the assignment ranges as defined in the ".sid" file header. These extra SIDs are used for subsequent assignements.

The following activity diagram summarizes the update of a YANG module and its associated .sid file.



4. ".sid" file format

".sid" files are used to persist and publish SIDs assigned to the different YANG items of a specific YANG module. The following YANG module defined the structure of this file, encoding is performed using the rules defined in [RFC7951].

```
<CODE BEGINS> file "ietf-sid-file@2015-12-16.yang"
module ietf-sid-file {
  namespace "urn:ietf:params:xml:ns:yang:ietf-sid-file";
  prefix sid;
```

```
     organization
       "IETF Core Working Group";

     contact
       "Alexander Pelov
        <mailto:a@ackl.io>

        Abhinav Somaraju
        <mailto:abhinav.somaraju@tridonic.com>

        Laurent Toutain
        <Laurent.Toutain@telecom-bretagne.eu>

        Randy Turner
        <mailto:Randy.Turner@landisgyr.com>

        Michel Veillette
        <mailto:michel.veillette@trilliantinc.com>";

     description
       "This module define the structure of the .sid files.
        .sid files contains the identifiers (SIDs) assigned
        to the different items defined in a YANG module.
        SIDs are used to encode a data model defined in YANG
        using CBOR.";

     revision 2015-12-16 {
       description
         "Initial revision.";
       reference
         "RFC XXXX";
         // RFC Ed.: replace XXXX with RFC number assigned to
         // draft-ietf-core-yang-cbor and remove this note
     }

     typedef yang-identifier {
       type string {
         length "1..max";
         pattern '[a-zA-Z_][a-zA-Z0-9\-_.]*';
         pattern '.|..|[^xX].*|.[^mM].*|..[^lL].*';
       }
       description
         "A YANG identifier string as defined by the 'identifier'
          rule in Section 12 of RFC 6020.";
     }

     typedef revision-identifier {
       type string {
```

```
        pattern '\d{4}-\d{2}-\d{2}';
      }
      description
        "Represents a date in YYYY-MM-DD format.";
    }

    leaf module-name {
      type yang-identifier;
      description
        "Name of the module associated with this .sid file.";
    }

    leaf module-revision {
      type revision-identifier;
      description
        "Revision of the module associated with this .sid file.
         This leaf is not present if no revision statement is
         defined in the YANG module.";
    }

    list assigment-ranges {
      key "entry-point";
      description
        "Range(s) of SIDs available for assignment to the
         different items defined by the associated module.";

      leaf entry-point {
        type uint32;
        mandatory true;
        description
          "Lowest SID available for assignment.";
      }

      leaf size {
        type uint16;
        mandatory true;
        description
          "Number of SIDs available for assignment.";
      }
    }

    list items {
      key "type label";
      description
        "List of items defined by the associated YANG module.";

      leaf type {
        type string {
```

```
          pattern 'Module|Submodule|feature|' +
                  'identity$|node$|notification$|rpc$|action$';
        }
        mandatory true;
        description
          "Item type assigned, this field can be set to:
            - 'Module'
            - 'Submodule'
            - 'feature'
            - 'identity'
            - 'node'
            - 'notification'
            - 'rpc'
            - 'action'";
      }

      leaf label {
        type string;
        mandatory true;
        description
          "Label associated to this item, can be set to:
            - a module name
            - a submodule name
            - a feature name
            - a base identity encoded as
              '/<base identity name>'
            - an identity encoded as
              '/<base identity name>/<identity name>'
            - a data node path";
      }

      leaf sid {
        type uint32;
        mandatory true;
        description "Identifier assigned to this YANG item.";
      }
    }
  }
  <CODE ENDS>
```

5. Security Considerations

The security considerations of [RFC7049] and [RFC7950] apply.

This document defines a new type of identifier used to encode data models defined in YANG [RFC7950]. As such, this identifier does not contribute to any new security issues in addition of those identified for the specific protocols or contexts for which it is used.

6. IANA Considerations

6.1. "SID mega-range" registry

The name of this registry is "SID mega-range". This registry is used to delegate the management of block of SIDs for third party's (e.g. SDO, registrar).

Each entry in this registry must include:

- The entry point (first entry) of the registered SID range.
- The size of the registered SID range.
- The contact information of the requesting organization including:
  - Organization name
  - Primary contact name, email address, and phone number
  - Secondary contact name, email address, and phone number

The initial entry in this registry is allocated to IANA:

Entry Point	Size	Organization name
0	1000000	IANA

The IANA policies for future additions to this registry are "Hierarchical Allocation, Expert Review" [RFC5226]. Prior to a first allocation, the requesting organization must demonstrate a functional registry infrastructure. On subsequent allocation request(s), the organization must demonstrate the exhaustion of the prior range. These conditions need to be asserted by the assigned expert(s).

6.1.1. IANA SID Mega-Range Registry

The first million SIDs assigned to IANA is sub-divided as follow:

- The range of 0 to 999 is reserved for future extensions. The IANA policy for this range is "IETF review" [RFC5226].
- The range of 1000 to 59,999 is reserved for YANG modules defined in RFCs. The IANA policy for future additions to this sub-registry is "RFC required" [RFC5226]. Allocation within this

range requires publishing of the associated ".yang" and ".sid" files in the YANG module registry.

- The range of 60,000 to 99,999 is reserved for experimental YANG modules. This range MUST NOT be used in operational deployments since these SIDs are not globally unique which limit their interoperability. The IANA policy for this range is "Experimental use" [RFC5226].

- The range of 100,000 to 999,999 is reserved for standardized YANG modules. The IANA policy for future additions to this sub-registry is "Specification Required" [RFC5226]. Allocation within this range requires publishing of the associated ".yang" and ".sid" files in the YANG module registry.

Entry Point	Size	IANA policy
0	1,000	IETF review
1,000	59,000	RFC required
60,000	40,000	Experimental use
100,000	1,000,000,000	Specification Required

The size of SID range assigned to a YANG module should be between 50% and 60% of the current number of YANG items. This headroom allows assignment within the same range of new YANG items introduced by subsequent revisions. A larger SID range size may be requested by the authors if this recommendation is considered insufficient. It is important to note that an extra SID range can be allocated to existing YANG module if the initial ranges(s) are exhausted.

6.1.2. IANA "RFC SID range assignment" sub-registries

The name of this sub-registry is "RFC SID range assignment". This sub-registry corresponds to the SID entry point 1000, size 59000. Each entry in this sub-registry must include the SID range entry point, the SID range size, the YANG module name, the RFC number.

Initial entries in this registry are as follows:

Entry Point	Size	Module name	Reference
1000	100		Reserved for [I-D.ietf-core-comi]
1100	400	iana-if-type	[RFC7224]
1500	100	ietf-interfaces	[RFC7223]
1600	100	ietf-ip	[RFC7277]
1700	100	ietf-system	[RFC7317]

6.2. "YANG module assignment" registry

The name of this registry is "YANG module assignment". This registry is used to track which YANG modules have been assigned and the specific YANG items assignment. Each entry in this sub-registry must include:

- The YANG module name
- The associated ".yang" file(s)
- The associated ".sid" file

The validity of the ".yang" and ".sid" files added to this registry MUST be verified.

- The syntax of the registered ".yang" and ".sid" files must be valid.
- Each YANG item defined by the registered ".yang" file must have a corresponding SID assigned in the ".sid" file.
- Each SID is assigned to a single YANG item, duplicate assignment is not allowed.
- The SID range(s) defined in the ".sid" file must be unique, must not conflict with any other SID ranges defined in already registered ".sid" files.
- The ownership of the SID range(s) should be verify.

The IANA policy for future additions to this registry is "First Come First Served" as described in [RFC5226].

7. Acknowledgments

The authors would like to thank Carsten Bormann for his help during the development of this document and his useful comments during the review process.

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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <http://www.rfc-editor.org/info/rfc7951>.

8.2. Informative References

[I-D.ietf-core-comi] Stok, P., Bierman, A., Veillette, M., and A. Pelov, "CoAP Management Interface", draft-ietf-core-comi-00 (work in progress), January 2017.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

[RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <http://www.rfc-editor.org/info/rfc7224>.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <http://www.rfc-editor.org/info/rfc7277>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <http://www.rfc-editor.org/info/rfc8040>.

Appendix A. ".sid" file example

The following .sid file (ietf-system@2014-08-06.sid) have been generated using the following yang modules:

- ietf-system@2014-08-06.yang
- ietf-yang-types@2013-07-15.yang
- ietf-inet-types@2013-07-15.yang
- ietf-netconf-acm@2012-02-22.yang
- iana-crypt-hash@2014-04-04.yang


```
{
 "assignment-ranges": [
  {
   "entry-point": 1700,
   "size": 100
  }
 ],
 "module-name": "ietf-system",
 "module-revision": "2014-08-06",
 "items": [
  {
   "type": "Module",
   "label": "ietf-system",
   "sid": 1700
  },
  {
   "type": "feature",
```

```
      "label": "authentication",
      "sid": 1701
     },
     {
      "type": "feature",
      "label": "dns-udp-tcp-port",
      "sid": 1702
     },
     {
      "type": "feature",
      "label": "local-users",
      "sid": 1703
     },
     {
      "type": "feature",
      "label": "ntp",
      "sid": 1704
     },
     {
      "type": "feature",
      "label": "ntp-udp-port",
      "sid": 1705
     },
     {
      "type": "feature",
      "label": "radius",
      "sid": 1706
     },
     {
      "type": "feature",
      "label": "radius-authentication",
      "sid": 1707
     },
     {
      "type": "feature",
      "label": "timezone-name",
      "sid": 1708
     },
     {
      "type": "identity",
      "label": "/authentication-method",
      "sid": 1709
     },
     {
      "type": "identity",
      "label": "/authentication-method/local-users",
      "sid": 1710
     },
```

```
{
 "type": "identity",
 "label": "/authentication-method/radius",
 "sid": 1711
},
{
 "type": "identity",
 "label": "/radius-authentication-type",
 "sid": 1712
},
{
 "type": "identity",
 "label": "/radius-authentication-type/radius-chap",
 "sid": 1713
},
{
 "type": "identity",
 "label": "/radius-authentication-type/radius-pap",
 "sid": 1714
},
{
 "type": "node",
 "label": "/system",
 "sid": 1715
},
{
 "type": "node",
 "label": "/system-state",
 "sid": 1716
},
{
 "type": "node",
 "label": "/system-state/clock",
 "sid": 1717
},
{
 "type": "node",
 "label": "/system-state/clock/boot-datetime",
 "sid": 1718
},
{
 "type": "node",
 "label": "/system-state/clock/current-datetime",
 "sid": 1719
},
{
 "type": "node",
 "label": "/system-state/platform",
```

```
      "sid": 1720
     },
     {
      "type": "node",
      "label": "/system-state/platform/machine",
      "sid": 1721
     },
     {
      "type": "node",
      "label": "/system-state/platform/os-name",
      "sid": 1722
     },
     {
      "type": "node",
      "label": "/system-state/platform/os-release",
      "sid": 1723
     },
     {
      "type": "node",
      "label": "/system-state/platform/os-version",
      "sid": 1724
     },
     {
      "type": "node",
      "label": "/system/authentication",
      "sid": 1725
     },
     {
      "type": "node",
      "label": "/system/authentication/user",
      "sid": 1726
     },
     {
      "type": "node",
      "label": "/system/authentication/user-authentication-order",
      "sid": 1727
     },
     {
      "type": "node",
      "label": "/system/authentication/user/authorized-key",
      "sid": 1728
     },
     {
      "type": "node",
      "label": "/system/authentication/user/authorized-key/algorithm",
      "sid": 1729
     },
     {
```

```
      "type": "node",
      "label": "/system/authentication/user/authorized-key/key-data",
      "sid": 1730
     },
     {
      "type": "node",
      "label": "/system/authentication/user/authorized-key/name",
      "sid": 1731
     },
     {
      "type": "node",
      "label": "/system/authentication/user/name",
      "sid": 1732
     },
     {
      "type": "node",
      "label": "/system/authentication/user/password",
      "sid": 1733
     },
     {
      "type": "node",
      "label": "/system/clock",
      "sid": 1734
     },
     {
      "type": "node",
      "label": "/system/clock/timezone-name",
      "sid": 1735
     },
     {
      "type": "node",
      "label": "/system/clock/timezone-utc-offset",
      "sid": 1736
     },
     {
      "type": "node",
      "label": "/system/contact",
      "sid": 1737
     },
     {
      "type": "node",
      "label": "/system/dns-resolver",
      "sid": 1738
     },
     {
      "type": "node",
      "label": "/system/dns-resolver/options",
      "sid": 1739
```

```
    },
    {
     "type": "node",
     "label": "/system/dns-resolver/options/attempts",
     "sid": 1740
    },
    {
     "type": "node",
     "label": "/system/dns-resolver/options/timeout",
     "sid": 1741
    },
    {
     "type": "node",
     "label": "/system/dns-resolver/search",
     "sid": 1742
    },
    {
     "type": "node",
     "label": "/system/dns-resolver/server",
     "sid": 1743
    },
    {
     "type": "node",
     "label": "/system/dns-resolver/server/name",
     "sid": 1744
    },
    {
     "type": "node",
     "label": "/system/dns-resolver/server/udp-and-tcp",
     "sid": 1745
    },
    {
     "type": "node",
     "label": "/system/dns-resolver/server/udp-and-tcp/address",
     "sid": 1746
    },
    {
     "type": "node",
     "label": "/system/dns-resolver/server/udp-and-tcp/port",
     "sid": 1747
    },
    {
     "type": "node",
     "label": "/system/hostname",
     "sid": 1748
    },
    {
     "type": "node",
```

```
      "label": "/system/location",
      "sid": 1749
     },
     {
      "type": "node",
      "label": "/system/ntp",
      "sid": 1750
     },
     {
      "type": "node",
      "label": "/system/ntp/enabled",
      "sid": 1751
     },
     {
      "type": "node",
      "label": "/system/ntp/server",
      "sid": 1752
     },
     {
      "type": "node",
      "label": "/system/ntp/server/association-type",
      "sid": 1753
     },
     {
      "type": "node",
      "label": "/system/ntp/server/iburst",
      "sid": 1754
     },
     {
      "type": "node",
      "label": "/system/ntp/server/name",
      "sid": 1755
     },
     {
      "type": "node",
      "label": "/system/ntp/server/prefer",
      "sid": 1756
     },
     {
      "type": "node",
      "label": "/system/ntp/server/udp",
      "sid": 1757
     },
     {
      "type": "node",
      "label": "/system/ntp/server/udp/address",
      "sid": 1758
     },
```

```
{
 "type": "node",
 "label": "/system/ntp/server/udp/port",
 "sid": 1759
},
{
 "type": "node",
 "label": "/system/radius",
 "sid": 1760
},
{
 "type": "node",
 "label": "/system/radius/options",
 "sid": 1761
},
{
 "type": "node",
 "label": "/system/radius/options/attempts",
 "sid": 1762
},
{
 "type": "node",
 "label": "/system/radius/options/timeout",
 "sid": 1763
},
{
 "type": "node",
 "label": "/system/radius/server",
 "sid": 1764
},
{
 "type": "node",
 "label": "/system/radius/server/authentication-type",
 "sid": 1765
},
{
 "type": "node",
 "label": "/system/radius/server/name",
 "sid": 1766
},
{
 "type": "node",
 "label": "/system/radius/server/udp",
 "sid": 1767
},
{
 "type": "node",
 "label": "/system/radius/server/udp/address",
```

```
      "sid": 1768
     },
     {
      "type": "node",
      "label": "/system/radius/server/udp/authentication-port",
      "sid": 1769
     },
     {
      "type": "node",
      "label": "/system/radius/server/udp/shared-secret",
      "sid": 1770
     },
     {
      "type": "rpc",
      "label": "/set-current-datetime",
      "sid": 1771
     },
     {
      "type": "rpc",
      "label": "/set-current-datetime/input/current-datetime",
      "sid": 1772
     },
     {
      "type": "rpc",
      "label": "/system-restart",
      "sid": 1773
     },
     {
      "type": "rpc",
      "label": "/system-shutdown",
      "sid": 1774
     }
    ]
   }
```


Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov (editor)
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Randy Turner
Landis+Gyr
30000 Mill Creek Ave
Suite 100
Alpharetta, GA 30022
US

Phone: ++16782581292
Email: randy.turner@landisgyr.com
URI: http://www.landisgyr.com/

Ana Minaburo
Acklio
2bis rue de la chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: August 11, 2017

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
A. Somaraju
Tridonic GmbH & Co KG
R. Turner
Landis+Gyr
A. Minaburo
Acklio

February 07, 2017

CBOR Encoding of Data Modeled with YANG
draft-ietf-core-yang-cbor-04

Abstract

This document defines encoding rules for serializing configuration data, state data, RPC input and RPC output, Action input, Action output and notifications defined within YANG modules using the Concise Binary Object Representation (CBOR) [RFC7049].

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 11, 2017.

Table of Contents

1. Introduction

The specification of the YANG 1.1 data modelling language [RFC7950] defines an XML encoding for data instances, i.e. contents of configuration datastores, state data, RPC inputs and outputs, action inputs and outputs, and event notifications.

A new set of encoding rules has been defined to allow the use of the same data models in environments based on the JavaScript Object Notation (JSON) Data Interchange Format [RFC7159]. This is accomplished in the JSON Encoding of Data Modeled with YANG specification [RFC7951].

The aim of this document is to define a set of encoding rules for the Concise Binary Object Representation (CBOR) [RFC7049]. The resulting encoding is more compact compared to XML and JSON and more suitable for Constrained Nodes and/or Constrained Networks as defined by [RFC7228].

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- action
- anydata
- anyxml
- data node
- data tree
- feature
- identity
- module
- notification

- RPC
- schema node
- schema tree
- submodule

The following terms are defined in [RFC7951]:

- member name
- name of an identity
- namespace-qualified

This specification also makes use of the following terminology:

- child: A schema node defined within a collection such as a container, a list, a case, a notification, an RPC input, an RPC output, an action input, an action output.
- delta: Difference between the current SID and a reference SID. A reference SID is defined for each context for which deltas are used.
- item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- parent: The collection in which a schema node is defined.
- YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

2.1. YANG Schema Item iDentifier (SID)

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called YANG Schema Item iDentifier (SID), is encoded using an unsigned integer. The following items are identified using SIDs:

- identities
- data nodes

- RPCs and associated input(s) and output(s)
- actions and associated input(s) and output(s)
- notifications and associated information
- YANG modules, submodules and features

To minimize its size, in certain positions, SIDs are represented using a (signed) delta from a reference SID and the current SID. Conversion from SIDs to deltas and back to SIDs are stateless processes solely based on the data serialized or deserialized.

Mechanisms and processes used to assign SIDs to YANG items and to guarantee their uniqueness is outside the scope of the present specification. If SIDs are to be used, the present specification is used in conjunction with a specification defining this management. One example for such a specification is under development as [I-D.ietf-core-sid].

2.2. CBOR diagnostic notation

Within this document, CBOR binary contents are represented using an equivalent textual form called CBOR diagnostic notation as defined in [RFC7049] section 6. This notation is used strictly for documentation purposes and is never used in the data serialization. Table 1 below provides a summary of this notation.

CBOR content	CBOR type	Diagnostic notation	Example	CBOR encoding
Unsigned integer	0	Decimal digits	123	18 7b
Negative integer	1	Decimal digits prefixed by a minus sign	-123	38 7a
Byte string	2	Hexadecimal value enclosed between single quotes and prefixed by an 'h'	h'f15c'	42 f15c
Text string	3	String of Unicode characters enclosed between double quotes	"txt"	63 747874
Array	4	Comma-separated list of values within square brackets	[1, 2]	82 01 02
Map	5	Comma-separated list of key : value pairs within curly braces	{ 1: 123, 2: 456 }	a2 01187b 021901c8
Boolean	7/20	false	false	f4
	7/21	true	true	f5
Null	7/22	null	null	f6
Not assigned	7/23	undefined	undefined	f7

Table 1: CBOR diagnostic notation summary

The following extensions to the CBOR diagnostic notation are supported:

- Any text within and including a pair of slashes is considered a comment.

- Deltas are visualized as numbers preceded by a '+' or '-' sign. The use of the '+' sign for positive deltas represents an extension to the CBOR diagnostic notation as defined by [RFC7049] section 6.

3. Properties of the CBOR Encoding

This document defines CBOR encoding rules for YANG schema trees and their subtrees.

Basic schema nodes such as leaf, leaf-list, list, anydata and anyxml can be encoded standalone. In this case, only the value of this

schema node is encoded in CBOR. Identification of this value needs to be provided by some external means when required.

A collection such as container, list instance, notification, RPC input, RPC output, action input and action output is serialized using a CBOR map in which each child schema node is encoded using a key and a value. This specification supports two type of CBOR keys; YANG Schema Item iDentifier (SID) as defined in Section 2.1 and member names as defined in [RFC7951]. Each of these key types is encoded using a specific CBOR type which allows their interpretation during the deserialization process. The end user of this mapping specification (e.g. RESTCONF [RFC8040], CoMI [I-D.ietf-core-comi]) can mandate the use of a specific key type.

In order to minimize the size of the encoded data, the proposed mapping avoids any unnecessary meta-information beyond those natively supported by CBOR. For instance, CBOR tags are used solely in the case of anyxml data nodes and the union datatype to distinguish explicitly the use of different YANG datatypes encoded using the same CBOR major type.

4. Encoding of YANG Data Node Instances

Schema node instances defined using the YANG modeling language are encoded using CBOR [RFC7049] based on the rules defined in this section. We assume that the reader is already familiar with both YANG [RFC7950] and CBOR [RFC7049].

4.1. The 'leaf' Data Node

Leafs MUST be encoded based on the encoding rules specified in Section 5.

4.2. The 'container' Data Node

Collections such as containers, list instances, notifications, RPC inputs, RPC outputs, action inputs and action outputs MUST be encoded using a CBOR map data item (major type 5). A map is comprised of pairs of data items, with each data item consisting of a key and a value. Each key within the CBOR map is set to a data node identifier, each value is set to the value of this data node instance according to the instance datatype.

This specification supports two type of CBOR keys; SID as defined in Section 2.1 encoded as deltas and member names as defined in [RFC7951] encoded using CBOR text strings. The use of CBOR byte strings for keys is reserved for future extensions.

4.2.1. SIDs as keys

Keys implemented using SIDs MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value. Keys are represented as the delta of the associated SID, delta values are computed as follows:

- The delta value is equal to the SID of the current schema node minus the SID of the parent schema node. When no parent exists in the context of use of this container, the delta is set to the SID of the current schema node (i.e., a parent with SID equal to zero is assumed).

- Delta values may result in a negative number, clients and servers MUST support both unsigned and negative deltas.

The following example shows the encoding of a 'system' container instance.

Definition example from [RFC7317]:

```
typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]
             \d{2}:\d{2})';
  }
}

container clock {
  leaf current-datetime {
    type date-and-time;
  }

  leaf boot-datetime {
    type date-and-time;
  }
}
```

CBOR diagnostic notation:

```
{
  1717 : {                              / clock  (SID 1717) /
    +2 : "2015-10-02T14:47:24Z-05:00",  / current-datetime (SID 1719) /
    +1 : "2015-09-15T09:12:58Z-05:00"   / boot-datetime (SID 1718) /
  }
}
```

CBOR encoding:

```
a1                                      # map(1)
   19 06b5                              # unsigned(1717)
   a2                                   # map(2)
      02                                # unsigned(2)
      78 1a                             # text(26)
      323031352d31302d30325431343a34373a32345a2d30353a3030
      01                                # unsigned(1)
      78 1a                             # text(26)
      323031352d30392d31355430393a31323a35385a2d30353a3030
```

4.2.2. Member names as keys

Keys implemented using member names MUST be encoded using a CBOR text string data item (major type 3). A namespace-qualified member name MUST be used for all members of a top-level collection, and then also whenever the namespaces of the schema node and its parent are different. In all other cases, the simple form of the member name MUST be used. Names and namespaces are defined in [RFC7951] section 4.

The following example shows the encoding of a 'system' container instance using names. This example is described in Section 4.2.1.

CBOR diagnostic notation:

```
{
  "ietf-system:clock" : {
    "current-datetime" : "2015-10-02T14:47:24Z-05:00",
    "boot-datetime" : "2015-09-15T09:12:58Z-05:00"
  }
}
```

CBOR encoding:

```
a1                                      # map(1)
   71                                   # text(17)
      696574662d73797374656d3a636c6f636b    # "ietf-system:clock"
   a2                                   # map(2)
      70                                # text(16)
         63757272656e742d6461746574696d65   # "current-datetime"
      78 1a                             # text(26)
         323031352d31302d30325431343a34373a32345a2d30353a3030
      6d                                # text(13)
         626f6f742d6461746574696d65         # "boot-datetime"
      78 1a                             # text(26)
         323031352d30392d31355430393a31323a35385a2d30353a3030
```

4.3. The 'leaf-list' Data Node

A leaf-list MUST be encoded using a CBOR array data item (major type 4). Each entry of this array MUST be encoded using the rules defined by the YANG type specified.

The following example shows the encoding a 'search' leaf-list instance containing the two entries, "ietf.org" and "ieee.org".

Definition example [RFC7317]:

```
typedef domain-name {
  type string {
    length "1..253";
    pattern '((([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9].)
             *([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9]\.?
             )|\.';
  }
}

leaf-list search {
  type domain-name;
  ordered-by user;
}
```

CBOR diagnostic notation: ["ietf.org", "ieee.org"]

CBOR encoding: 82 68 696574662e6f7267 68 696565652e6f7267

4.4. The 'list' Data Node

A list MUST be encoded using a CBOR array data item (major type 4). Each list instance within this CBOR array is encoded using a CBOR map data item (major type 5) based on the same rules as a YANG container as defined in Section 4.2.

4.4.1. SIDs as keys

The following example show the encoding of a 'server' list instance using SIDs. It is important to note that the protocol or method using this mapping may carry a parent SID or may have the knowledge of this parent SID based on its context. In these cases, delta encoding can be performed based on this parent SID which minimizes the size of the encoded data.

Definition example from [RFC7317]:

```
   list server {
     key name;

     leaf name {
       type string;
     }
     choice transport {
       case udp {
         container udp {
           leaf address {
             type host;
             mandatory true;
           }
           leaf port {
             type port-number;
           }
         }
       }
     }
     leaf association-type {
       type enumeration {
         enum server;
         enum peer;
         enum pool;
       }
       default server;
     }
     leaf iburst {
       type boolean;
       default false;
     }
     leaf prefer {
       type boolean;
       default false;
     }
   }
```

CBOR diagnostic notation:

```
[
  {
    1755 : "NRC TIC server",          / name (SID 1755) /
    1757 : {                          / udp (SID 1757) /
      +1 : "tic.nrc.ca",              / address (SID 1758) /
      +2 : 123                        / port (SID 1759) /
    },
    1753 : 0,                         / association-type (SID 1753) /
    1754 : false,                     / iburst (SID 1754) /
    1756 : true                       / prefer (SID 1756) /
  },
  {
    1755 : "NRC TAC server",          / name (SID 1755) /
    1757 : {                          / udp (SID 1757) /
      +1 : "tac.nrc.ca"               / address (SID 1758) /
    }
  }
]
```

CBOR encoding:

```
82                                  # array(2)
   a5                               # map(5)
      19 06db                       # unsigned(1755)
      6e                            # text(14)
         4e52432054494320736572766572 # "NRC TIC server"
      19 06dd                       # unsigned(1757)
      a2                            # map(2)
         01                         # unsigned(1)
         6a                         # text(10)
            7469632e6e72632e6361    # "tic.nrc.ca"
         02                         # unsigned(2)
         18 7b                      # unsigned(123)
      19 06d9                       # unsigned(1753)
      00                            # unsigned(0)
      19 06da                       # unsigned(1754)
      f4                            # primitive(20)
      19 06dc                       # unsigned(1756)
      f5                            # primitive(21)
   a2                               # map(2)
      19 06db                       # unsigned(1755)
      6e                            # text(14)
         4e52432054414320736572766572 # "NRC TAC server"
      19 06dd                       # unsigned(1757)
      a1                            # map(1)
         01                         # unsigned(1)
         6a                         # text(10)
            7461632e6e72632e6361    # "tac.nrc.ca"
```

4.4.2. Member names as keys

The following example shows the encoding of a 'server' list instance using names. This example is described in Section 4.4.1.

CBOR diagnostic notation:

```
[
  {
    "ietf-system:name" : "NRC TIC server",
    "ietf-system:udp" : {
      "address" : "tic.nrc.ca",
      "port" : 123
    },
    "ietf-system:association-type" : 0,
    "ietf-system:iburst" : false,
    "ietf-system:prefer" : true
  },
  {
    "ietf-system:name" : "NRC TAC server",
    "ietf-system:udp" : {
      "address" : "tac.nrc.ca"
    }
  }
]
```

CBOR encoding:

```
82                                              # array(2)
   a5                                           # map(5)
      70                                        # text(16)
         696574662d73797374656d3a6e616d65       # "ietf-system:name"
      6e                                        # text(14)
         4e5243205449432073657276657200         # "NRC TIC server"
      6f                                        # text(15)
         696574662d73797374656d3a756470         # "ietf-system:udp"
      a2                                        # map(2)
         67                                     # text(7)
            61646472657373                      # "address"
         6a                                     # text(10)
            7469632e6e72632e6361                # "tic.nrc.ca"
         64                                     # text(4)
            706f7274                            # "port"
         18 7b                                  # unsigned(123)
      78 1c                                     # text(28)
         696574662d73797374656d3a6173736f63696174696f6e2d74797065
      00                                        # unsigned(0)
      72                                        # text(18)
         696574662d73797374656d3a696275727374 # "ietf-system:iburst"
      f4                                        # primitive(20)
      72                                        # text(18)
         696574662d73797374656d3a707265666572 # "ietf-system:prefer"
      f5                                        # primitive(21)
   a2                                           # map(2)
      70                                        # text(16)
         696574662d73797374656d3a6e616d65       # "ietf-system:name"
      6e                                        # text(14)
         4e5243205441432073657276657200         # "NRC TAC server"
      6f                                        # text(15)
         696574662d73797374656d3a756470         # "ietf-system:udp"
      a1                                        # map(1)
         67                                     # text(7)
            61646472657373                      # "address"
         6a                                     # text(10)
            7461632e6e72632e6361                # "tac.nrc.ca"
```

4.5. The 'anydata' Data Node

An anydata serves as a container for an arbitrary set of schema nodes that otherwise appear as normal YANG-modeled data. An anydata instance is encoded using the same rules as a container, i.e., CBOR map. The requirement that anydata content can be modeled by YANG implies the following:

- Keys of any inner data nodes MUST be set to valid deltas or member names.

- The CBOR array MUST contain either unique scalar values (as a leaf-list, see Section 4.3), or maps (as a list, see Section 4.4).

- Values MUST follow the encoding rules of one of the datatypes listed in Section 5.

The following example shows a possible use of anydata. In this example, an anydata is used to define a data node containing a notification event, this data node can be part of a YANG list to create an event logger.

Definition example:

```
anydata event;
```

This example also assumes the assistance of the following notification.

```
module example-port {
  ...

  notification example-port-fault {  # SID 2600
    leaf port-name {                 # SID 2601
      type string;
    }
    leaf port-fault {                # SID 2601
      type string;
    }
  }
}
```

CBOR diagnostic notation:

```
{
  2601 : "0/4/21",       / port-name /
  2602 : "Open pin 2"    / port-fault /
}
```

CBOR encoding:

```
a2                          # map(2)
   19 0a29                  # unsigned(2601)
   66                       # text(6)
      302f342f3231          # "0/4/21"
   19 0a2a                  # unsigned(2602)
   6a                       # text(10)
      4f70656e2070696e2032 # "Open pin 2"
```

4.6. The 'anyxml' Data Node

An anyxml schema node is used to serialize an arbitrary CBOR content, i.e., its value can be any CBOR binary object. anyxml value may contain CBOR data items tagged with one of the tag listed in Section 7.1, these tags shall be supported.

The following example shows a valid CBOR encoded instance.

Definition example from [RFC7951]:

anyxml bar;

CBOR diagnostic notation: [true, null, true]

CBOR encoding: 83 f5 f6 f5

5. Representing YANG Data Types in CBOR

The CBOR encoding of an instance of a leaf or leaf-list data node depends on the built-in type of that data node. The following sub-section defined the CBOR encoding of each built-in type supported by YANG as listed in [RFC7950] section 4.2.4. Each subsection shows an example value assigned to a data node instance of the discussed built-in type.

5.1. The unsigned integer Types

Leafs of type uint8, uint16, uint32 and uint64 MUST be encoded using a CBOR unsigned integer data item (major type 0).

The following example shows the encoding of a 'mtu' leaf instance set to 1280 bytes.

Definition example from [RFC7277]:

```
leaf mtu {
  type uint16 {
    range "68..max";
  }
}
```

CBOR diagnostic notation: 1280

CBOR encoding: 19 0500

5.2. The integer Types

Leafs of type int8, int16, int32 and int64 MUST be encoded using either CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value.

The following example shows the encoding of a 'timezone-utc-offset' leaf instance set to -300 minutes.

Definition example from [RFC7317]:

```
leaf timezone-utc-offset {
  type int16 {
    range "-1500 .. 1500";
  }
}
```

CBOR diagnostic notation: -300

CBOR encoding: 39 012b

5.3. The 'decimal64' Type

Leafs of type decimal64 MUST be encoded using a decimal fraction as defined in [RFC7049] section 2.4.3.

The following example shows the encoding of a 'my-decimal' leaf instance set to 2.57.

Definition example from [RFC7317]:

```
leaf my-decimal {
  type decimal64 {
    fraction-digits 2;
    range "1 .. 3.14 | 10 | 20..max";
  }
}
```

CBOR diagnostic notation: 4([-2, 257])

CBOR encoding: c4 82 21 19 0101

5.4. The 'string' Type

Leafs of type string MUST be encoded using a CBOR text string data item (major type 3).

The following example shows the encoding of a ’name’ leaf instance set to "eth0".

Definition example from [RFC7223]:

```
leaf name {
  type string;
}
```

CBOR diagnostic notation: "eth0"

CBOR encoding: 64 65746830

5.5. The ’boolean’ Type

Leafs of type boolean MUST be encoded using a CBOR true (major type 7, additional information 21) or false data item (major type 7, additional information 20).

The following example shows the encoding of an ’enabled’ leaf instance set to ’true’.

Definition example from [RFC7317]:

```
leaf enabled {
  type boolean;
}
```

CBOR diagnostic notation: true

CBOR encoding: f5

5.6. The ’enumeration’ Type

Leafs of type enumeration MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value. Enumeration values are either explicitly assigned using the YANG statement ’value’ or automatically assigned based on the algorithm defined in [RFC7950] section 9.6.4.2.

The following example shows the encoding of an ’oper-status’ leaf instance set to ’testing’.

Definition example from [RFC7317]:

```
leaf oper-status {
  type enumeration {
    enum up { value 1; }
    enum down { value 2; }
    enum testing { value 3; }
    enum unknown { value 4; }
    enum dormant { value 5; }
    enum not-present { value 6; }
    enum lower-layer-down { value 7; }
  }
}
```

CBOR diagnostic notation: 3

CBOR encoding: 03

5.7. The 'bits' Type

Leafs of type bits MUST be encoded using a CBOR byte string data item (major type 2). Bits position are either explicitly assigned using the YANG statement 'position' or automatically assigned based on the algorithm defined in [RFC7950] section 9.7.4.2.

Bits position 0 to 7 are assigned to the first byte within the byte string, bits 8 to 15 to the second byte, and subsequent bytes are assigned similarly. Within each byte, bits are assigned from least to most significant.

The following example shows the encoding of a 'mybits' leaf instance with the 'disable-nagle' and '10-Mb-only' flags set.

Definition example from [RFC7950]:

```
leaf mybits {
  type bits {
    bit disable-nagle {
      position 0;
    }
    bit auto-sense-speed {
      position 1;
    }
    bit 10-Mb-only {
      position 2;
    }
  }
}
```

CBOR diagnostic notation: h'05'

CBOR encoding: 41 05

5.8. The 'binary' Type

Leafs of type binary MUST be encoded using a CBOR byte string data item (major type 2).

The following example shows the encoding of an 'aes128-key' leaf instance set to 0x1f1ce6a3f42660d888d92a4d8030476e.

Definition example:

```
leaf aes128-key {
  type binary {
    length 16;
  }
}
```

CBOR diagnostic notation: h'1f1ce6a3f42660d888d92a4d8030476e'

CBOR encoding: 50 1f1ce6a3f42660d888d92a4d8030476e

5.9. The 'leafref' Type

Leafs of type leafref MUST be encoded using the rules of the schema node referenced by the 'path' YANG statement.

The following example shows the encoding of an 'interface-state-ref' leaf instance set to "eth1".

Definition example from [RFC7223]:

```
typedef interface-state-ref {
  type leafref {
    path "/interfaces-state/interface/name";
  }
}

container interfaces-state {
  list interface {
    key "name";
    leaf name {
      type string;
    }
    leaf-list higher-layer-if {
      type interface-state-ref;
    }
  }
}
```

CBOR diagnostic notation: "eth1"

CBOR encoding: 64 65746831

5.10. The 'identityref' Type

This specification supports two approaches for encoding identityref, a YANG Schema Item iDentifier (SID) as defined in Section 2.1 or a name as defined in [RFC7951] section 6.8.

5.10.1. SIDs as identityref

When schema nodes of type identityref are implemented using SIDs, they MUST be encoded using a CBOR unsigned integer data item (major type 0). (Note that no delta mechanism is employed for SIDs as identityref.)

The following example shows the encoding of a 'type' leaf instance set to the value 'iana-if-type:ethernetCsmacd' (SID 1180).

Definition example from [RFC7317]:

```
identity interface-type {
}

identity iana-interface-type {
  base interface-type;
}

identity ethernetCsmacd {
  base iana-interface-type;
}

leaf type {
  type identityref {
    base interface-type;
  }
}
```

CBOR diagnostic notation: 1180

CBOR encoding: 19 049c

5.10.2. Name as identityref

Alternatively, an identityref may be encoded using a name as defined in [RFC7951] section 6.8. When names are used, identityref MUST be encoded using a CBOR text string data item (major type 3). If the identity is defined in another module than the leaf node containing the identityref value, the namespace-qualified form MUST be used. Otherwise, both the simple and namespace-qualified forms are permitted. Names and namespaces are defined in [RFC7951] section 4.

The following example shows the encoding of the identity 'iana-if-type:ethernetCsmacd' using its name. This example is described in Section 5.10.1.

CBOR diagnostic notation: "iana-if-type:ethernetCsmacd"

CBOR encoding: 78 1b
69616e612d69662d747970653a65746865726e657443736d616364

5.11. The 'empty' Type

Leafs of type empty MUST be encoded using the CBOR null value (major type 7, additional information 22).

The following example shows the encoding of a 'is-router' leaf instance when present.

Definition example from [RFC7277]:

```
leaf is-router {
  type empty;
}
```

CBOR diagnostic notation: null

CBOR encoding: f6

5.12. The 'union' Type

Leafs of type union MUST be encoded using the rules associated with one of the types listed. When used in a union, the following YANG datatypes are prefixed by CBOR tag to avoid confusion between different YANG datatypes encoded using the same CBOR major type.

- bits
- enumeration
- identityref
- instance-identifier

See Section 7.1 for more information about these CBOR tags.

The following example shows the encoding of an 'ip-address' leaf instance when set to "2001:db8:a0b:12f0::1".

Definition example from [RFC7317]:

```
typedef ipv4-address {
  type string {
  pattern '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}
           ([0-9][1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(%[\p{N}
           \p{L}]+)?';
  }
}

typedef ipv6-address {
  type string {
    pattern '((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}((([0-9a
             -fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|(((25[0-5]|2[0-4][0
             -9]|[01]?[0-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0
             -9]?[0-9])))(%[\p{N}\p{L}]+)?';
    pattern '(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*))|((([^:]+:)*[^:]+)
             ?::(([^:]+:)*[^:]+)?)(%.+)?';
  }
}

typedef ip-address {
  type union {
    type ipv4-address;
    type ipv6-address;
  }
}

leaf address {
  type inet:ip-address;
}
```

CBOR diagnostic notation: "2001:db8:a0b:12f0::1"

CBOR encoding: 74 323030313a6462383a6130623a313266303a3a31

5.13. The 'instance-identifier' Type

This specification supports two approaches for encoding an instance-identifier, one based on YANG Schema Item iDentifier (SID) as defined in Section 2.1 and one based on names as defined in [RFC7951] section 6.11.

5.13.1. SIDs as instance-identifier

SIDs uniquely identify a data node. In the case of a single instance data node, a data node defined at the root of a YANG module or submodule or data nodes defined within a container, the SID is sufficient to identify this instance.

In the case of a data node member of a YANG list, a SID is combined with the list key(s) to identify each instance within the YANG list(s).

Single instance data nodes MUST be encoded using a CBOR unsigned integer data item (major type 0) and set to the targeted data node SID.

Data nodes member of a YANG list MUST be encoded using a CBOR array data item (major type 4) containing the following entries:

- The first entry MUST be encoded as a CBOR unsigned integer data item (major type 0) and set to the targeted data node SID.

- The following entries MUST contain the value of each key required to identify the instance of the targeted data node. These keys MUST be ordered as defined in the 'key' YANG statement, starting from top level list, and follow by each of the subordinate list(s).

First example:

The following example shows the encoding of a leaf instance of type instance-identifier which identifies the data node "/system/contact" (SID 1737).

Definition example from [RFC7317]:

```
container system {

  leaf contact {
    type string;
  }

  leaf hostname {
    type inet:domain-name;
  }
}
```

CBOR diagnostic notation: 1737

CBOR encoding: 19 06c9

Second example:

The following example shows the encoding of a leaf instance of type instance-identifier which identify the data node instance

"/system/authentication/user/authorized-key/key-data" (SID 1730) for user name "bob" and authorized-key "admin".

Definition example from [RFC7317]:

```
list user {
  key name;

  leaf name {
    type string;
  }
  leaf password {
    type ianach:crypt-hash;
  }

  list authorized-key {
    key name;

    leaf name {
      type string;
    }
    leaf algorithm {
      type string;
    }
    leaf key-data {
      type binary;
  }
}
```

CBOR diagnostic notation: [1730, "bob", "admin"]

CBOR encoding:

```
83                      # array(3)
   19 06c2              # unsigned(1730)
   63                   # text(3)
      626f62            # "bob"
   65                   # text(5)
      61646d696e        # "admin"
```

Third example:

The following example shows the encoding of a leaf instance of type instance-identifier which identify the list instance "/system/authentication/user" (SID 1726) corresponding to the user name "jack".

CBOR diagnostic notation: [1726, "jack"]

CBOR encoding:

```
82                      # array(2)
   19 06be              # unsigned(1726)
   64                   # text(4)
      6a61636b          # "jack"
```

5.13.2. Names as instance-identifier

The use of names as instance-identifier is defined in [RFC7951] section 6.11. The resulting xpath MUST be encoded using a CBOR text string data item (major type 3).

First example:

This example is described in Section 5.13.1.

CBOR diagnostic notation: "/ietf-system:system/contact"

CBOR encoding:

```
78 1c 2f20696574662d73797374656d3a73797374656d2f636f6e74616374
```

Second example:

This example is described in Section 5.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name='bob']/authorized-key [name='admin']/key-data"

CBOR encoding:

```
78 59
   2f696574662d73797374656d3a73797374656d2f61757468656e74696361
   74696f6e2f757365725b6e616d653d27626f62275d2f617574686f72697a
   65642d6b65795b6e616d653d2761646d696e275d2f6b65792d64617461
```

Third example:

This example is described in Section 5.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name='bob']"

CBOR encoding:

```
78 33
   2f696574662d73797374656d3a73797374656d2f61757468656e74696361
   74696f6e2f757365725b6e616d653d27626f62275d
```

6. Security Considerations

The security considerations of [RFC7049] and [RFC7950] apply.

This document defines an alternative encoding for data modeled in the YANG data modeling language. As such, this encoding does not contribute any new security issues in addition of those identified for the specific protocol or context for which it is used.

To minimize security risks, software on the receiving side SHOULD reject all messages that do not comply to the rules of this document and reply with an appropriate error message to the sender.

7. IANA Considerations

7.1. Tags Registry

This specification requires the assignment of CBOR tags for the following YANG datatypes. These tags are added to the Tags Registry as defined in section 7.2 of [RFC7049].

Tag	Data Item	Semantics	Reference
40	bits	YANG bits datatype	RFC XXXX
41	enumeration	YANG enumeration datatype	RFC XXXX
42	identityref	YANG identityref datatype	RFC XXXX
43	instance-identifier	YANG instance-identifier datatype	RFC XXXX

// RFC Ed.: update Tag values using allocated tags if needed and remove this note // RFC Ed.: replace XXXX with RFC number and remove this note

8. Acknowledgments

This document has been largely inspired by the extensive works done by Andy Bierman and Peter van der Stok on [I-D.ietf-core-comi]. [RFC7951] has also been a critical input to this work. The authors would like to thank the authors and contributors to these two drafts.

The authors would also like to acknowledge the review, feedback, and comments from Ladislav Lhotka and Juergen Schoenwaelder.

9. References

9.1. 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>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

9.2. Informative References

[I-D.ietf-core-comi] Stok, P., Bierman, A., Veillette, M., and A. Pelov, "CoAP Management Interface", draft-ietf-core-comi-00 (work in progress), January 2017.

[I-D.ietf-core-sid] Somaraju, A., Veillette, M., Pelov, A., Turner, R., and A. Minaburo, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-00 (work in progress), October 2016.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

[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>.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <http://www.rfc-editor.org/info/rfc7277>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <http://www.rfc-editor.org/info/rfc7951>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <http://www.rfc-editor.org/info/rfc8040>.

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov (editor)
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Randy Turner
Landis+Gyr
30000 Mill Creek Ave
Suite 100
Alpharetta, GA 30022
US

Phone: ++16782581292
Email: randy.turner@landisgyr.com
URI: http://www.landisgyr.com/

Ana Minaburo
Acklio
2bis rue de la chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 31, 2017

J. Mattsson
J. Fornehed
G. Selander
F. Palombini
Ericsson

November 27, 2016

Controlling Actuators with CoAP
draft-mattsson-core-coap-actuators-02

Abstract

Being able to trust information from sensors and to securely control actuators is essential in a world of connected and networking things interacting with the physical world. In this memo we show that just using COAP with a security protocol like DTLS, TLS, or OSCOAP is not enough. We describe several serious attacks any on-path attacker can do, and discusses tougher requirements and mechanisms to mitigate the attacks. While this document is focused on actuators, one of the attacks applies equally well to sensors using DTLS.

Table of Contents

1. Introduction

Being able to trust information from sensors and to securely control actuators is essential in a world of connected and networking things interacting with the physical world. One protocol used to interact with sensors and actuators is the Constrained Application Protocol (CoAP) [RFC7252]. Any Internet-of-Things (IoT) deployment valuing security and privacy would use a security protocol such as DTLS [RFC6347], TLS [RFC5246], or OSCOAP [I-D.selander-ace-object-security] to protect CoAP, where the choice of security protocol depends on the transport protocol and the presence of intermediaries. The use of CoAP over UDP and DTLS is specified in [RFC6347] and the use of CoAP over TCP and TLS is specified in [I-D.ietf-core-coap-tcp-tls]. OSCOAP protects CoAP end-to-end with the use of COSE [I-D.ietf-cose-msg] and the CoAP Object-Security option [I-D.selander-ace-object-security], and can therefore be used over any transport. In this document we show that protecting CoAP with a security protocol is not enough to securely control actuators. We describe several serious attacks any on-path attacker (i.e. not only "trusted" intermediaries) can do, and discusses tougher requirements and mechanisms to mitigate the attacks. The request delay attack (valid for DTLS, TLS, and OSCOAP and described in Section 2.2) lets an attacker control an actuator at a much later time than the client anticipated. The response delay and mismatch attack (valid for DTLS and described in Section 2.3) lets an attacker

respond to a client with a response meant for an older request. In Section 3, a new CoAP Option, the Repeat Option, mitigating the delay attack in specified.

2. Attacks

Internet-of-Things (IoT) deployments valuing security and privacy, MUST use a security protocol such as DTLS, TLS, or OSCOAP to protect CoAP. This is especially true for deployments of actuators where attacks often (but not always) have serious consequences. The attacks described in this section are made under the assumption that CoAP is already protected with a security protocol such as DTLS, TLS, or OSCOAP, as an attacker otherwise can easily forge false requests and responses.

2.1. The Block Attack

An on-path attacker can block the delivery of any number of requests or responses. The attack can also be performed by an attacker jamming the lower layer radio protocol. This is true even if a security protocol like DTLS, TLS, or OSCOAP is used. Encryption makes selective blocking of messages harder, but not impossible or even infeasible. With DTLS and TLS, proxies have access to the complete CoAP message, and with OSCOAP, the CoAP header and several CoAP options are not encrypted. In both security protocols, the IP-addresses, ports, and CoAP message lengths are available to all on-path attackers, which may be enough to determine the server, resource, and command. The block attack is illustrated in Figure 1 and 2.

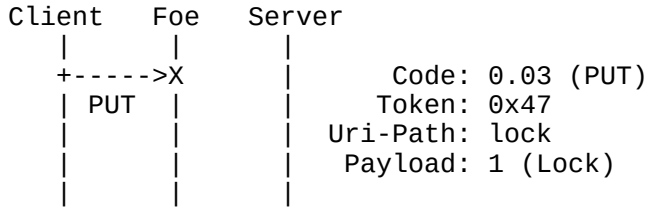


Figure 1: Blocking a Request

Where ’X’ means the attacker is blocking delivery of the message.

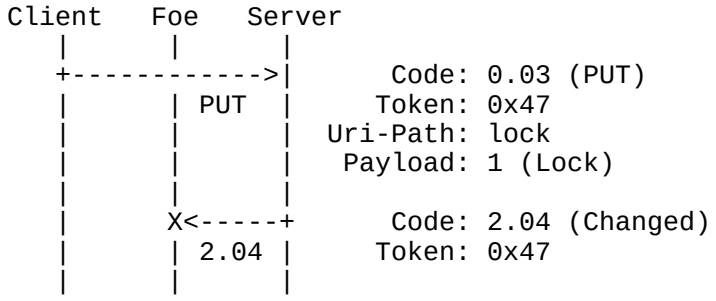


Figure 2: Blocking a Response

While blocking requests to, or responses from, a sensor is just a denial of service attack, blocking a request to, or a response from, an actuator results in the client losing information about the server's status. If the actuator e.g. is a lock (door, car, etc.), the attack results in the client not knowing (except by using out-of-band information) whether the lock is unlocked or locked, just like the observer in the famous Schrodinger's cat thought experiment. Due to the nature of the attack, the client cannot distinguish the attack from connectivity problems, offline servers, or unexpected behavior from middle boxes such as NATs and firewalls.

Remedy: Any IoT deployment of actuators where confirmation is important MUST notify the user upon reception of the response, or warn the user when a response is not received.

2.2. The Request Delay Attack

An on-path attacker may not only block packets, but can also delay the delivery of any packet (request or response) by a chosen amount of time. If CoAP is used over a reliable and ordered transport such as TCP with TLS or OSCOAP, no messages can be delivered before the delayed message. If CoAP is used over an unreliable and unordered transport such as UDP with DTLS, or OSCOAP, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. When CoAP is used over UDP, both DTLS and OSCOAP allow out-of-order delivery and uses sequence numbers together with a replay window to protect against replay attacks. The replay window has a default length of 64 in both DTLS and OSCOAP. The attacker can control the replay window by blocking some or all other packets. By first delaying a request, and then later, after delivery, blocking the response to the request, the client is not made aware of the delayed delivery except by the missing response. The server has in general, no way of knowing that the request was delayed and will therefore happily process the request.

If some wireless low-level protocol is used, the attack can also be performed by the attacker simultaneously recording what the client transmits while at the same time jamming the server. The request delay attack is illustrated in Figure 3.

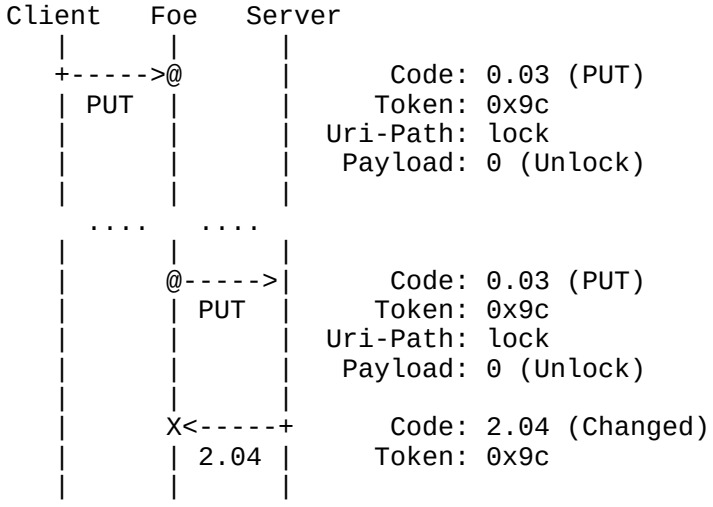


Figure 3: Delaying a Request

Where '@' means the attacker is storing and later forwarding the message (@ may alternatively be seen as a wormhole connecting two points in time).

While an attacker delaying a request to a sensor is often not a security problem, an attacker delaying a request to an actuator performing an action is often a serious problem. A request to an actuator (for example a request to unlock a lock) is often only meant to be valid for a short time frame, and if the request does not reach the actuator during this short timeframe, the request should not be fulfilled. In the unlock example, if the client does not get any response and does not physically see the lock opening, the user is likely to walk away, calling the locksmith (or the IT-support).

If a non-zero replay window is used (the default when CoAP is used over UDP), the attacker can let the client interact with the actuator before delivering the delayed request to the server (illustrated in Figure 4). In the lock example, the attacker may store the first "unlock" request for later use. The client will likely resend the request with the same token. If DTLS is used, the resent packet will have a different sequence number and the attacker can forward it. If OSCOAP is used, resent packets will have the same sequence number and the attacker must block them all until the client sends a new message

with a new sequence number (not shown in Figure 4). After a while when the client has locked the door again, the attacker can deliver the delayed "unlock" message to the door, a very serious attack.

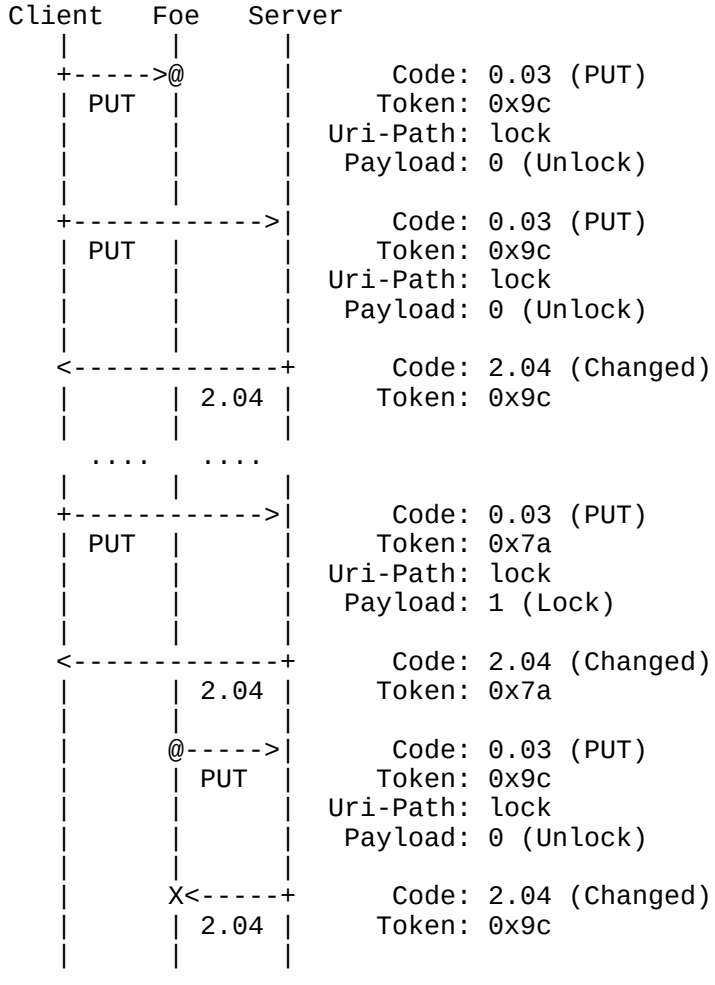


Figure 4: Delaying Request with Reordering

While the second attack (Figure 4) can be mitigated by using a replay window of length zero, the first attack (Figure 3) cannot. A solution must enable the server to verify that the request was received within a certain time frame after it was sent. This can be accomplished with either a challenge-response pattern, by exchanging timestamps, or by only allowing requests a short period after client authentication. Requiring a fresh client authentication (such as a new TLS/DTLS handshake or an EDHOC key exchange

[I-D.selander-ace-cose-ecdhe]) mitigates the problem, but requires larger messages and more processing than a dedicated solution. Security solutions based on timestamps require exactly synchronized time, and this is hard to control with complications such as time zones and daylight saving. Even if the clocks are synchronized at one point in time, they may easily get out-of-sync and an attacker may even be able to affect the client or the server time in various ways such as setting up a fake NTP server, broadcasting false time signals to radio controlled clocks, or expose one of them to a strong gravity field. As soon as client falsely believes it is time synchronized with the server, delay attacks are possible. A challenge response mechanism is much more failure proof and easy to analyze. The challenge and response may be sent in a CoAP option or in the CoAP payload. One such mechanism, the CoAP Replay Option, is specified in Section 3.

Remedy: The CoAP Replay Option specified in Section 3 SHALL be used for controlling actuators unless another application specific challenge-response or timestamp mechanism is used.

2.3. The Response Delay and Mismatch Attack

The following attack can be performed if CoAP is protected by a security protocol where the response is not bound to the request in any way except by the CoAP token. This would include most general security protocols, such as DTLS and IPsec, but not OSCOAP. The attacker performs the attack by delaying delivery of a response until the client sends a request with the same token. As long as the response is inside the replay window (which the attacker can make sure by blocking later responses), the response will be accepted by the client as a valid response to the later request. CoAP [RFC7252] uses a client generated token that the server echoes to match responses to request, but does not give any guidelines for the use of token with DTLS, except that the tokens currently "in use" SHOULD (not SHALL) be unique.

The attack can be performed by an attacker on the wire, or an attacker simultaneously recording what the server transmits while at the same time jamming the client. The response delay and mismatch attack is illustrated in Figure 5.

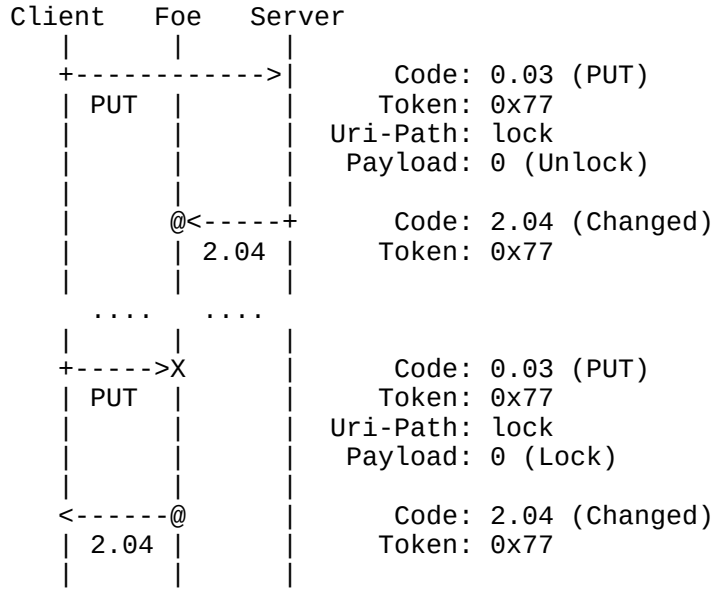


Figure 5: Delaying and Mismatching Response to PUT

If we once again take a lock as an example, the security consequences may be severe as the client receives a response message likely to be interpreted as confirmation of a locked door, while the received response message is in fact confirming an earlier unlock of the door. As the client is likely to leave the (believed to be locked) door unattended, the attacker may enter the home, enterprise, or car protected by the lock.

The same attack may be performed on sensors, also this with serious consequences. As illustrated in Figure 6, an attacker may convince the client that the lock is locked, when it in fact is not. The "Unlock" request may be also be sent by another client authorized to control the lock.

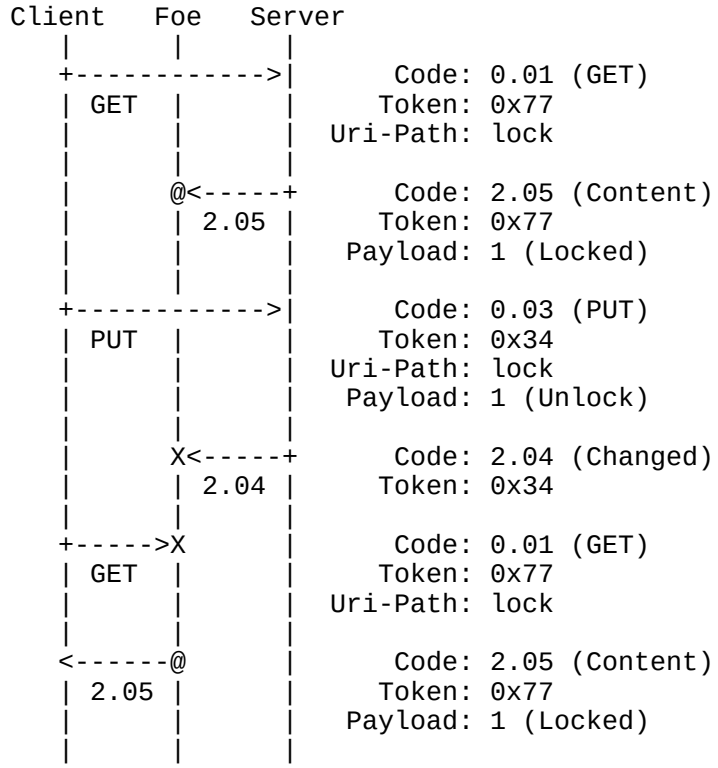


Figure 6: Delaying and Mismatching Response to GET

As illustrated in Figure 7, an attacker may even mix responses from different resources as long as the two resources share the same DTLS connection on some part of the path towards the client. This can happen if the resources are located behind a common gateway, or are served by the same CoAP proxy. An on-path attacker (not necessarily a DTLS endpoint such as a proxy) may e.g. deceive a client that the living room is on fire by responding with an earlier delayed response from the oven (temperatures in degree Celsius).

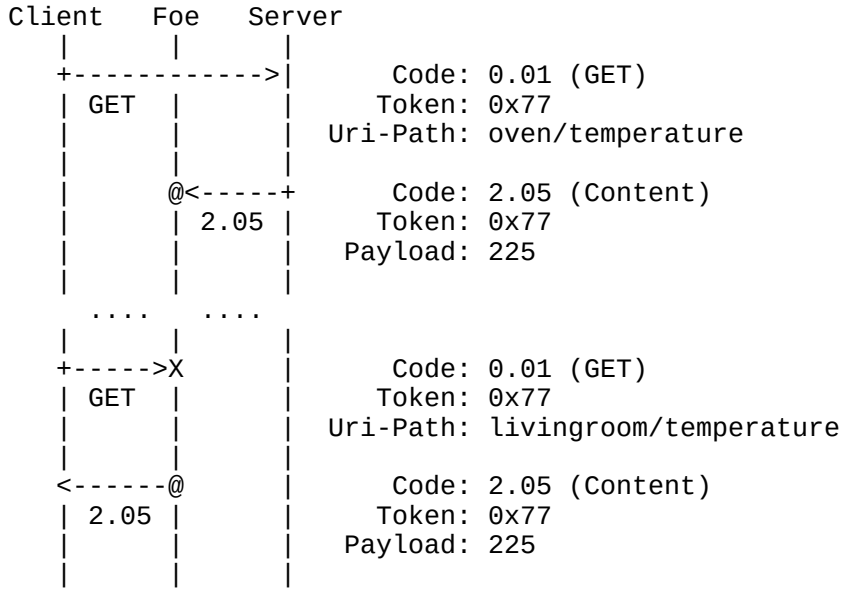


Figure 7: Delaying and Mismatching Response from other resource

Remedy: If CoAP is protected with a security protocol not providing bindings between requests and responses (e.g. DTLS) the client MUST NOT reuse any tokens for a given source/destination which the client has not received responses to. The easiest way to accomplish this is to implement the token as a counter and never reuse any tokens at all, this approach SHOULD be followed.

2.4. The Relay Attack

Yet another type of attack can be performed in deployments where actuator actions are triggered automatically based on proximity and without any user interaction, e.g. a car (the client) constantly polling for the car key (the server) and unlocking both doors and engine as soon as the car key responds. An attacker (or pair of attackers) may simply relay the CoAP messages out-of-band, using for examples some other radio technology. By doing this, the actuator (i.e. the car) believes that the client is close by and performs actions based on that false assumption. The attack is illustrated in Figure 8. In this example the car is using an application specific challenge-response mechanism transferred as CoAP payloads.

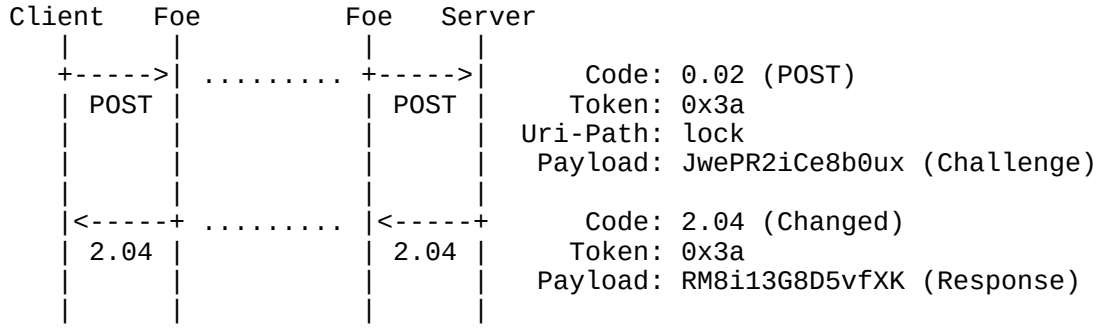


Figure 8: Relay Attack (the client is the actuator)

The consequences may be severe, and in the case of a car, lead to the attacker unlocking and driving away with the car, an attack that unfortunately is happening in practice.

Remedy: Getting a response over a short-range radio MUST NOT be taken as proof of proximity and therefore MUST NOT be used to take actions based on such proximity. Any automatically triggered mechanisms relying on proximity MUST use other stronger mechanisms to guarantee proximity. Mechanisms that MAY be used are: measuring the round-trip time and calculate the maximum possible distance based on the speed of light, or using radio with an extremely short range like NFC (centimeters instead of meters). Another option is to including geographical coordinates (from e.g. GPS) in the messages and calculate proximity based on these, but in this case the location measurements MUST be very precise and the system MUST make sure that an attacker cannot influence the location estimation, something that is very hard in practice.

3. The Repeat Option

The Repeat Option is a challenge-response mechanism for CoAP, binding a resent request to an earlier 4.03 forbidden response. The challenge (for the client) is simply to echo the Repeat Option value in a new request. The Repeat Option enables the server to verify the freshness of a request, thus mitigating the Delay Attack described in Section 2.2. An example message flow is illustrated in Figure 9.

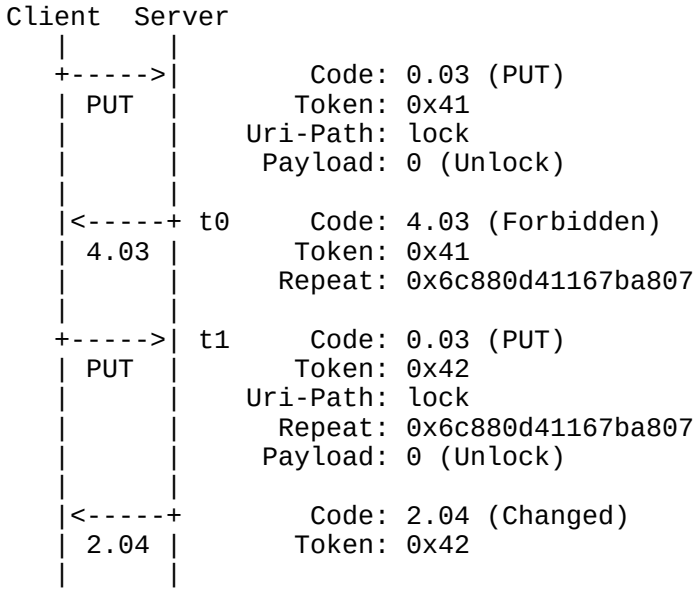


Figure 9: The Repeat Option

The Repeat Option may be used for all Methods and Response Codes. In responses, the value MUST be a (pseudo-)random bit string with a length of at least 64 bits. A new (pseudo-)random bit string MUST be generated for each response. In requests, the Repeat Option MUST echo the value from a previously received response.

The Repeat Option is critical, Safe-to-Forward, not part of the Cache-Key, and not repeatable.

Upon receiving a request without the Repeat Option to a resource with freshness requirements, the server sends a 4.03 Forbidden response with a Repeat Option and stores the option value and the response transmit time t0.

Upon receiving a 4.03 Forbidden response with the Repeat Option, the client SHOULD resend the request, echoing the Repeat Option value.

Upon receiving a request with the Repeat Option, the server verifies that the option value equals the previously sent value; otherwise the request is not processed further. The server calculates the round-trip time RTT = (t1 - t0), where t1 is the request receive time. The server MUST only accept requests with a round-trip time below a certain threshold T, i.e. RTT < T, otherwise the request is not processed further, and an error message MAY be send. The threshold T is application specific.

EDITORS NOTE: The mechanism described above is secure and gives the server freshness guarantee independently of what the client does. The disadvantages are that the mechanism always takes two round-trips and that the server has to save the option value and the time t0. Two different solutions involving time overcomes these disadvantages:

- The server may simply send the client the current time in its timescale, i.e. a timestamp (option value = t0). The client may then use this timestamp to estimate the current time in the servers timescale when sending future requests (i.e. not echoing). This approach has the benefit of reducing round-trips and server state, but has the security problems discussed in Section 2.2.

- The server may instead of a pseudorandom value send an encrypted timestamp (option value = E(k, t0)). CTR-mode would from a security point be like sending (value = t0). ECB-mode or CCM-mode would work, but would expand the value length. With CCM, the server might also bind the option value to request (value = AEAD(k, t0, parts of request)). This approach does not reduce the number of round-trips but eliminates server state.

TODO: Update the Repeat Option to use a combination of these two solutions instead.

4. IANA Considerations

This document defines the following Option Number, whose value have been assigned to the CoAP Option Numbers Registry defined by [RFC7252].

Number	Name
29	Repeat

5. Security Considerations

The whole document can be seen as security considerations for CoAP.

6. Acknowledgements

The authors would like to thank Carsten Bormann, Klaus Hartke, Ari Keranen, Matthias Kovatsch, Sandeep Kumar, and Andras Mehes for their valuable comments and feedback.

7. References

7.1. Normative References

[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>.

7.2. Informative References

[I-D.ietf-core-coap-tcp-tls] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-05 (work in progress), October 2016.

[I-D.ietf-cose-msg] Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

[I-D.selander-ace-cose-ecdhe] Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-04 (work in progress), October 2016.

[I-D.selander-ace-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-06 (work in progress), October 2016.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

Authors' Addresses

John Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

John Fornehed
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.fornehed@ericsson.com

Goran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

J. Mattsson
Ericsson AB

March 13, 2017

Message Size Overhead of CoAP Security Protocols
draft-mattsson-core-security-overhead-00

Abstract

This document analyzes and compares per-packet message size overheads when using different security protocols to secure CoAP. The analyzed security protocols are DTLS 1.2, DTLS 1.3, TLS 1.2, TLS 1.3, and OSCOAP. DTLS and TLS are analyzed with and without compression. DTLS are analyzed with two different alternatives for header compression.

Table of Contents

1. Introduction

This document analyzes and compares per-packet message size overheads when using different security protocols to secure CoAP over UPD [RFC7252] and TCP [I-D.ietf-core-coap-tcp-tls]. The analyzed security protocols are DTLS 1.2 [RFC6347], DTLS 1.3 [I-D.rescorla-tls-dtls13], TLS 1.2 [RFC5246], TLS 1.3 [I-D.ietf-tls-tls13], and OSCOAP [I-D.ietf-core-object-security]. The DTLS and TLS record layers are analyzed with and without compression. DTLS are analyzed with two different alternatives ([RFC7400] and [raza-6lo-compressed-dtls]) for header compression.

2. Overhead of Security Protocols

To enable comparison, all the overhead calculations in this section use AES-CCM with a tag length of 8 bytes, a plaintext of 6 bytes, and the sequence number ’05’. This follows the example in [RFC7400], Figure 16.

2.1. DTLS 1.2

This example is taken directly from [RFC7400], Figure 16. The nonce follow the strict profiling given in [RFC7925].

DTLS 1.2 Record Layer (35 bytes, 29 bytes overhead):
17 fe fd 00 01 00 00 00 00 00 05 00 16 00 01 00
00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4
cb 35 b9

Content type:
17
Version:
fe fd
Epoch:
00 01
Sequence number:
00 00 00 00 00 05
Length:
00 16
Nonce:
00 01 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.2 gives 29 bytes overhead.

2.2. DTLS 1.2 with 6LoWPAN-GHC

Note that the compressed overhead is dependent on the parameters epoch, sequence number, and length. The following is only an example.

Note that the sequence number ’01’ used in [RFC7400], Figure 15 gives an exceptionally small overhead that is not representative at all.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.2 Record Layer (22 bytes, 16 bytes overhead):
b0 c3 03 05 00 16 f2 0e ae a0 15 56 67 92 4d ff
8a 24 e4 cb 35 b9

Compressed DTLS 1.2 Record Layer Header and Nonce:
b0 c3 03 05 00 16 f2 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.2 with the above parameters (epoch, sequence number, length) gives 16 bytes overhead.

2.3. DTLS 1.2 with raza-6lo-compressed-dtls

Note that the compressed overhead is dependent on the parameters epoch and sequence number. The following is only an example.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with raza-6lo-compressed-dtls.

Compressed DTLS 1.2 Record Layer (19 bytes, 13 bytes overhead):
90 17 01 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4
cb 35 b9

NHC
90
Compressed DTLS 1.2 Record Layer Header and Nonce:
17 01 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with raza-6lo-compressed-dtls, DTLS 1.2 with the above parameters (epoch, sequence number) gives 13 bytes overhead.

2.4. DTLS 1.3

The only change compared to DTLS 1.2 is that the DTLS 1.3 record layer does not have an explicit nonce.

DTLS 1.3 Record Layer (27 bytes, 21 bytes overhead):
17 fe fd 00 01 00 00 00 00 00 05 00 0e ae a0 15
56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Version:
fe fd
Epoch:
00 01
Sequence number:
00 00 00 00 00 05
Length:
00 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 gives 21 bytes overhead.

2.5. DTLS 1.3 with 6LoWPAN-GHC

Note that the overhead is dependent on the parameters epoch, sequence number, and length. The following is only an example.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.3 Record Layer (20 bytes, 14 bytes overhead):
b0 c3 11 05 00 0e ae a0 15 56 67 92 4d ff 8a 24
e4 cb 35 b9

Compressed DTLS 1.3 Record Layer Header and Nonce:
b0 c3 11 05 00 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.3 with the above parameters (epoch, sequence number, length) gives 14 bytes overhead.

2.6. DTLS 1.3 with raza-6lo-compressed-dtls

Note that the compressed overhead is dependent on the parameters epoch and sequence number. The following is only an example.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with raza-6lo-compressed-dtls.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with raza-6lo-compressed-dtls.

Compressed DTLS 1.3 Record Layer (19 bytes, 13 bytes overhead):
90 17 01 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4
cb 35 b9

NHC
90
Compressed DTLS 1.3 Record Layer Header and Nonce:
17 01 00 05
c3 03 05 00 16 f2 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with raza-6lo-compressed-dtls, DTLS 1.3 with the above parameters (epoch, sequence number) gives 13 bytes overhead.

2.7. TLS 1.2

The changes compared to DTLS 1.2 is that the TLS 1.2 record layer does not have epoch and sequence number, and that the version is different.

TLS 1.2 Record Layer (27 bytes, 21 byte overhead):
17 03 03 00 16 00 00 00 00 00 00 00 05 ae a0 15
56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Version:
03 03
Length:
00 16
Nonce:
00 00 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

TLS 1.2 gives 21 bytes overhead.

2.8. TLS 1.2 with 6LoWPAN-GHC

Note that the overhead is dependent on the parameters epoch, sequence number, and length. The following is only an example.

Note that this header compression is not available when TLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed TLS 1.2 Record Layer (23 bytes, 17 bytes overhead):
05 17 03 03 00 16 85 0f 05 ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9

Compressed TLS 1.2 Record Layer Header and Nonce:
05 17 03 03 00 16 85 0f 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, TLS 1.2 with the above parameters (epoch, sequence number, length) gives 17 bytes overhead.

2.9. TLS 1.3

The change compared to TLS 1.2 is that the TLS 1.3 record layer uses a different version.

TLS 1.3 Record Layer (27 bytes, 21 byte overhead):
17 03 01 00 16 00 00 00 00 00 00 00 05 ae a0 15
56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Version:
03 01
Length:
00 16
Nonce:
00 00 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

TLS 1.3 gives 21 bytes overhead.

2.10. TLS 1.3 with 6LoWPAN-GHC

Note that the overhead is dependent on the parameters epoch, sequence number, and length. The following is only an example.

Note that this header compression is not available when TLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed TLS 1.3 Record Layer (23 bytes, 17 bytes overhead):
02 17 03 c3 01 16 85 0f 05 ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9

Compressed TLS 1.3 Record Layer Header and Nonce:
02 17 03 c3 01 16 85 0f 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, TLS 1.3 with the above parameters (epoch, sequence number, length) gives 17 bytes overhead.

2.11. OSCOAP

Note that the overhead is dependent on the included CoAP Option numbers, if the CoAP method allows payload, as well as the length of the OSCOAP parameters Sender ID and sequence number. The below

calculation uses Method = POST, Option Delta = ’9’, and Sender ID = ’25’, and is only an example.

OSCOAP Request (19 bytes, 13 bytes overhead):
90 19 05 41 25 ae a0 15 56 67 92 4d ff 8a 24 e4
cb 35 b9

CoAP Delta and Option Length:
90
Compressed COSE Header:
19 05 41 25
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

OSCOAP Response (15 bytes, 9 bytes overhead):
90 ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP Delta and Option Length:
90
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

OSCOAP with the above parameters gives 13 bytes overhead for requests and 9 bytes overhead for responses.

Unlike DTLS and TLS, OSCOAP has much smaller overhead for responses than requests.

3. Overhead with Different Sequence Numbers

The compression overhead (GHC) is dependent on the parameters epoch, sequence number, and length. The following overheads should be representative for sequence numbers with the same length.

The compression overhead (raza-6lo-compressed-dtls) is dependent on the length of the parameters epoch and sequence number. The following overheads apply for all sequence numbers with the same length.

The OSCOAP overhead is dependent on the included CoAP Option numbers, if the CoAP method allows payload, as well as the length of the OSCOAP parameters Sender ID and sequence number.

Sequence Number	'05'	'1005'	'100005'
DTLS 1.2	29	29	29
DTLS 1.3	21	21	21
TLS 1.2	21	21	21
TLS 1.3	21	21	21
DTLS 1.2 (GHC)	16	16	17
DTLS 1.2 (Raza)	13	13	14
DTLS 1.3 (GHC)	14	14	15
DTLS 1.3 (Raza)	13	13	14
TLS 1.2 (GHC)	17	18	19
TLS 1.3 (GHC)	17	18	19
OSCOAP Request	13	14	15
OSCOAP Response	9	9	9

Figure 1: Overhead as a function of sequence number

4. Summary

DTLS 1.2 has quite a large overhead as it uses an explicit sequence number and an explicit nonce. DTLS 1.3, TLS 1.2, and TLS 1.3 have significantly less overhead.

Both DTLS compression methods provides very good compression. raza-6lo-compressed-dtls achieves slightly better compression but requires state. GHC is stateless but provides slightly worse compression. As DTLS 1.3 uses the same version number as DTLS 1.2, both GHC and raza-6lo-compressed-dtls works well also for DTLS 1.3.

The Generic Header Compression (6LoWPAN-GHC) is not very generic (the static dictionary is more or less a DTLS record layer) and the compression of TLS is significantly worse than the compression of DTLS. Similar compression levels as for DTLS could be achieved also for TLS, but this would require different static dictionaries for each version of TLS (as TLS 1.2 and TLS 1.3 uses different version numbers).

The header compression is not available when (D)TLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC or raza-6lo-compressed-dtls.

OSCOAP has much lower overhead than DTLS and TLS. The overhead of OSCOAP is smaller than DTLS over 6LoWPAN with compression, and this small overhead is achieved even on deployments without 6LoWPAN or 6LoWPAN without DTLS compression. OSCOAP is lightweight because it makes use of some excellent features in CoAP, CBOR, and COSE.

5. Security Considerations

This document is purely informational.

6. Acknowledgments

The authors want to thank Ari Keraenen for reviewing previous versions of the draft.

7. Informative References

[I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-07 (work in progress), March 2017.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-01 (work in progress), December 2016.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-19 (work in progress), March 2017.

[I-D.rescorla-tls-dtls13]
Rescorla, E. and H. Tschofenig, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-rescorla-tls-dtls13-00 (work in progress), October 2016.

[raza-6lo-compressed-dtls]
Raza, S., Shafagh, H., and O. Dupont, "Compression of Record and Handshake Headers for Constrained Environments", March 2017, <http://shahidraza.info/draft-raza-6lo-compressed.txt>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[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>.

[RFC7400] Bormann, C., "6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7400, DOI 10.17487/RFC7400, November 2014, <http://www.rfc-editor.org/info/rfc7400>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <http://www.rfc-editor.org/info/rfc7925>.

Author’s Address

John Mattsson
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

ANIMA WG
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

M. Pritikin
P. Kampanakis
Cisco Systems

October 31, 2016

BRSKI over CoAP
draft-pritikin-coap-bootstrap-01

Abstract

This document provides an initial discussion of Bootstrapping of Remote Secure key infrastructures (BRSKI) when the device being bootstrapped speaks CoAP. The HTTPS REST methods leveraged by BRSKI are mapped to CoAP methods. Fragmentation management of large messages during EST certificate enrollment is addressed.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 4, 2017.

Table of Contents

1. Introduction

Many IoT and other devices are expected to use CoAP over UDP extensively. Bootstrapping these devices without requiring a full TCP stack is an often raised requirement for [I-D.ietf-anima-bootstrapping-keyinfra]. BRSKI provides REST methods over TLS that can be functional in a UDP setting with the folling necessary additions:

DTLS: Because the CoAP use of DTLS includes support for large handshake messages there is little to describe here. BRSKI and EST [RFC7030] are expanded to include DTLS.

REST: The mapping of BRSKI and EST messages to CoAP REST calls is described.

Fragmentation: Use of block chaining to support fragmentation of large BRSKI and EST messages is described.

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].

3. Scope of solution

The definition of BRSKI over DTLS and CoAP is not intended to expand the scope of BRSKI to highly constrained devices. (ref: [RFC7228]). Instead it is intended to ensure that bootstrapping works for less constained devices that choose to limit their communications stack to UDP/CoAP.

The BRSKI document details extensions to EST as well as making section 5.7 requirements on EST flows. This document's references to BRSKI are intended to include all BRSKI extensions and all existing EST messages. This document could replace BRSKI -03 section 5.7.5. [[TODO: making this section 5.8 might make the most sense.]]

Support for Observe CoAP options (https://tools.ietf.org/html/rfc7641) in Blocks with BRSKI is not supported in the current BRSKI/EST message flows and is thus out-of-scope for this discussion. Observe options could be used by the server to notify clients about a change in the cacerts or csr attributes (resources) and might be an area of future work.

4. DTLS

COAP was designed to avoid fragmentation. DTLS is used to secure COAP messages. When using DTLS, even though it can be avoided by using pre-shared keys or ECC ciphersuites, sometimes fragmentation will be needed. During the DTLS handshake, if fragmentation is necessary, "DTLS provides a mechanism for fragmenting a handshake message over a number of records, each of which can be transmitted separately, thus avoiding IP fragmentation" [RFC6347].

Within BRSKI and EST when "TLS" is referred to, it is understood that CoAP security is provided using DTLS instead. No other changes are necessary (all provisional modes etc are the same as for TLS).

In a constrained CoAP environment, endpoints can't afford to establish a DTLS connection for every EST transaction. Authenticating and negotiating DTLS keys requires resources on low-end endpoints and consumes valuable bandwidth. The DTLS connection SHOULD remain open for persistent EST connections. For example, an EST cacerts request that is followed by an simpleenroll request can use the same authenticated DTLS connection. Given that after a successfull enrollment, it is more likely that a new EST transaction will take place after a significant amount of time, the DTLS connections SHOULD only be kept alive for EST messages that are relatively close to each other.

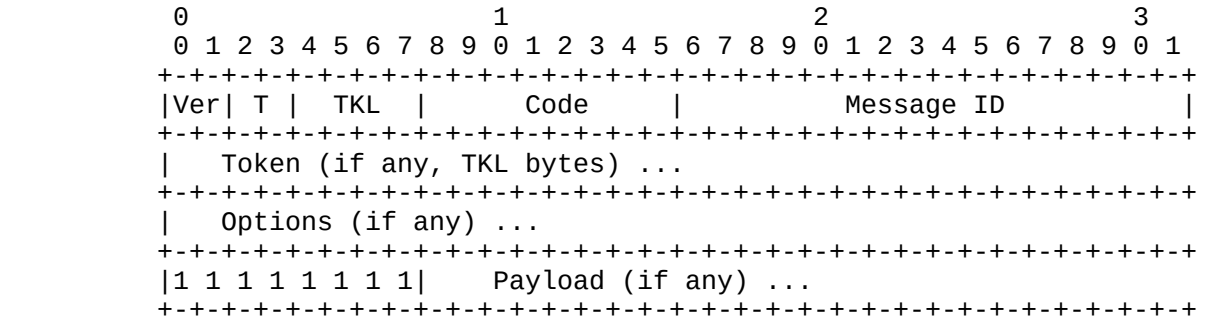
5. Message Bindings

This section describes BRSKI to CoAP message mappings.

CoAP defines confirmed (CON), acknowledgements (ACK), reset (RST) and non-corfirmed (NON) message types. For confirmable messages, the responses are CoAP ACKs or RSTs. All /cacerts, /simpleenroll, /simplereenroll, /csrattrs, /fullcmc and /serverkeygen EST messages expect a response, so they are all COAP CON messages.

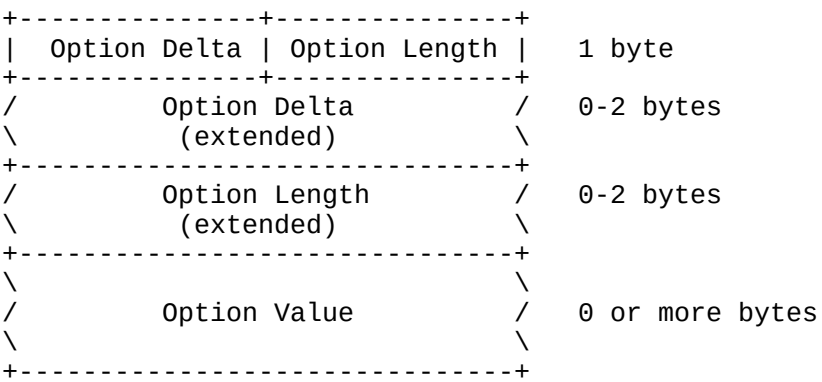
The HTTP responses in BRSKI are translated to COAP Response Codes as explained in [RFC7252] section 5.3.1

A CoAP message has the following fields ([I-D.ietf-core-block]):



Then Ver, TKL, Token, Message ID are not affected in BRSKI. Their use is the same as in CoAP.

The options that can be used in a CoAP header have the following format (from [I-D.ietf-core-block] Figure 8):



Options are used to convey Uri-Host, Uri-Path, Uri-Port, Content-Format and more in COAP. For BRSKI, COAP Options are used to communicate the HTTP fields used in the BRSKI REST messages.

BRSKI URLs are HTTPS based (https://), in CoAP these will be assumed to be transformed to coaps (coaps://)

Some examples of how an BRSKI message would be translated in CoAP follow. [[TODO: This section to be expanded to ensure it covers all BRSKI edge conditions.]]

5.1. cacerts

First let's see how a get cacerts message in EST would be in CoAP. The HTTPS cacerts message can be

```
GET /.well-known/est/cacerts HTTP/1.1
   User-Agent: curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0
               OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
   Host: 192.0.2.1:8085
   Accept: */*
```

The corresponding CoAP fields would be:

```
Ver = 1
T = 0 (CON)
Code = 0x01 (0.01 is GET)
Options
 Option1 (Uri-Host)
   Option Delta = 0x3
   Option Length = 0x9
   Option Value = 192.0.2.1
 Option2 (Uri-Port)
   Option Delta = 0xA
   Option Length = 0x4
   Option Value = 8085
 Option3 (Uri-Path)
   Option Delta = 0xD
   Option Length = 0xD
   Extended Option Delta = 0x08
   Extended Option Length = 0x14
   Option Value = /.well-known/est/cacerts HTTP/1.1
Payload = [Empty]
```

Now let's say we have a 200 OK response with a cert in EST:

```
HTTP/1.1 200 OK
Status: 200 OK
Content-Type: application/pkcs7-mime
Content-Transfer-Encoding: base64 (TODO: Verify if we need a new
                                   option registry for Encoding?)
Content-Length: 4246 (TODO: this example overflows and would
                      need fragmentation. Choose a better example.
                      Regardless we might need an CoAP option for
                      the content-length ie the CoAP payload?)

MIIMOQYJKoZIhvcNAQcCoIIMKjCCDCYCAQExADALBgkqhkiG9w0BBwGgggwMMIIC
+zCCAeOgAwIBAgIJAJpY3nUZO3qcMA0GCSqGSIb3DQEBBQUAMBsxGTAXBgNVBAMT
...
```

The corresponding CoAP fields would be:

```
Ver = 1
T = 2 (ACK)
Code = 0x21 (TODO: Maybe we need to create a 0x200 response code.)
Options
  Option1 (Content-Format)
    Option Delta = 0xC
    Option Length = 0xD
    Extended Option Length = 0x09
    Option Value = <number for application/pkcs7-mime>
              (TODO: We need a new CoAP IANA registered value
              application/pkcs7-mime; smime-type=certs-only,
              application/csrattrs, application/pkcs10,
              application/pkcs8,
              application/pkcs12 )
Payload = MIIMOQYJKoZIhvcNAQcCoIIMKjCCDCYCAQExA \
           DALBgkqhkiG9w0BBwGgggwMMIIC...
```

5.2. enroll / reenroll

[[TODO: username/password authentication can be described here but is not a primary focus for BRSKI. It is important for generic EST exchanges but would an endpoint device with sufficient user interface to allow username/password input from an end user be required to use CoAP instead of a full HTTPS exchange?]]

[[TODO: We might need a new Option for the Retry-After response message. We might need a new Option for the WWW-Authenticate response.]]

5.3. csrattr

5.4. requestauditttoken, requestauditlog

6. Data Fragmentation

Even though ECC certificates are small in size, they can vary greatly based on signature algorithms, key sizes, and OID fields used. Even with ECC certs, BRSKI CoAP messages carrying such certificates can still exceed sizes in MTU of 1280 for IPv6 or 60-80 bytes for 6LoWPAN [RFC4919]) (section 2 of [I-D.ietf-core-block]). For 256-bit curves, common ECDSA cert sizes are 500-1000 bytes which could fluctuate further based on the algorithms, OIDs, SANs and cert fields. For 384-bit curves, ECDSA certs increase in size and can sometimes reach 1.5KB. Additionally, there are times when the EST cacert response from the server can include multiple certs that amount large payloads.

CoAP RFC section 4.6 describes the possible payload sizes: "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". Also "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; per [RFC0791], 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". Thus, after the DTLS connection is established, fragmentation will sometimes be needed for the CoAP messages which involve certificate enrollment and management. A fragmentation solution for BRSKI and EST CoAP message is required.

The [I-D.ietf-core-block] document describes how fragmentation can be done by using a pair of Block options added to the CoAP flow. Block1 options are used by the client PUT and POST requests. A Block1 in a client request requires a Block1 option in the responses. A Block2 comes from a server response that will also need Block2 from the client to acknowledge the block and get the rest of blocks from the server. So, Block1 is used when a request (POST for example) is done in BRSKI over CoAP with a payload that needs fragmentation. Then the server responds with Block1 option to acknowledge the fragment-blocks. Block2 is used when a BRSKI server response is big and needs fragmentation. The Block2 acknowledgements are requests with the same options as the initial request and a Block2 option. "To influence the block size used in a response, the requester MAY also use the Block2 Option on the initial request, giving the desired size, a block number of zero and an M bit of zero". As explained in see section 2.8 of [I-D.ietf-core-block]), blockwise transfers SHOULD

be used in Confirmable COAP messages to avoid the exacerbation of lost blocks.

In a scenario with a big BRSKI POST request we might have Block1 options from client to server and Block2 from server to client. In this case the Block1 blocks get completed and then the Block2 comes the other direction.

The BLOCK draft also defines Size1 and Size2 options. These are used to convey the size of the resources in the requests or responses. The Size1 response MAY be parsed by the client as an size indication of the Block2 resource in the server response or by the server as a request for a size estimate by the client. Similarly, Size2 option defined in BLOCK should be parsed by the server as an indication of the size of the resource carried in Block1 options and by the client as a maximum size expected in the 4.13 (Request Entity Too Large) response to a request.

6.1. Fragmented response example with Block2

An example of a server cacerts response that exceeds the MTU is:

An example of a server cacerts response that exceeds the MTU is
HTTP/1.1 200 OK
Status: 200 OK
Content-Type: application/pkcs7-mime; smime-type=certs-only
Content-Transfer-Encoding: base64
Content-Length: 1122

MIIDOAYJKoZIhvcNAQcCoIIDKTCCAyUCAQExADALBgkqhkiG9w0BBwGgggMLMIID
BzCCAe+gAwIBAgIBFTANBgkqhkiG9w0BAQUFADAbMRkwFwYDVQQDExBlc3RFeGFt
cGxlQ0EgTndOMB4XDTEzMDUwOTIzMTU1M1oXDTE0MDUwOTIzMTU1M1owHzEdMBsG
A1UEAxMUZGVtb3N0ZXA0IDEzNjgxNDEzNTIwggEiMA0GCSqGSIb3DQEBAQUAA4IB
DwAwggEKAoIBAQClNp+kdz+Nj8XpEp9kaumWxDZ3eFYJpQKz9ddD5e5OzUeCm103
ZIXQIxc0eVtMCatnRr3dnZRCAxGjwbqoB3eKt29/XSQffVv+odbyw0WdkQOIbntC
Qry8YdcBZ+8LjI/N7M2krmjmoSLmLwU2V4aNKf0YMLR5Krmah3Ik31jmYCSvwTnv
6mx6pr2pTJ82JavhTEIIt/fAYq1RYhkM1CXoBL+yhEoDanN7TzC94skfS3VV+f53
J9SkUxTYcy1Rw0k3VXfxWwy+cSKEPREl7I6k0YeKtDEVAgBIEYM/L1S69RXTLuji
rwnqSRjOquzkAkD31BE961KZCxeYGrhxaR4PAgMBAAGjUjBQMA4GA1UdDwEB/wQE
AwIEsDAdBgNVHQ4EFgQU/qDdB6ii6icQ8wGMXvy1jfE4xtUwHwYDVR0jBBgwFoAU
scRp5lujBKfYl6OLO7+5arIyQjwwDQYJKoZIhvcNAQEFBQADggEBACmxg1hvL6+7
a+lFTARoxainBx5gxdZ9omSb0L+qL+4PDvg/+KHzKsDnMCrcU6M4YP5n0EDKmGa6
4lY8fbET4tt7juJg6ixb95/760Th0vuctwkGr6+D6ETTfqyHnrbhX3lAhnB+0Ja7
o1gv4CWxh1I8aRaTXdpOHORvN0SMXdcrlCys2vrtOl+LjR2a3kajJO6eQ5leOdzF
QlZfOPhaLWen0e2BLNJI0vsC2Fa+2LMCnfC38XfGALa5A8e7fNHXWZBjXZLBCza3
rEs9Mlh2CjA/ocSC/WxmMvd+Eqnt/FpggRy+F8IZSRvBaRUCtGE1lgDmu6AFUxce
R4POrT2xz8ChADEA

Block options in CoAP messages can contain fields, SZX, M and NUM which are not affected by BRSKI.

Let’s assume that the cacerts message will need to be broken up to 3 messages. The first Block2 will be:

```
  Ver = 1
  T = 2 (ACK)
  Code = 0x21 (2.01 success message.
         TODO: Do we need to create a 0x200 respond code.)
  Options
    Option1 (Content-Format)
      Option Delta = 0xC
      Option Length = 0xD
      Extended Option Length = 0x09
      Option Value = <number for application/pkcs7-mime;
                      smime-type=certs-only>
                 (TODO: We need a new CoAP IANA registered value
                 application/pkcs7-mime, application/csrattrs,
                 application/pkcs10, application/pkcs8,
                 application/pkcs12 )
    Option2 (Block2)
      Option Delta = 0xD
      Option Length = 0x1
      Extended Option Delta = 0x16
      Option Value = 0x0D
  Payload = MIIMOQYJKoZIhvcNAQcCoIIMKjCCDCYC \
            AQExADALBgkqhkiG9w0BBwGgggwMMIIC... (512 bytes)
```

The second Block2:

```
     Ver = 1
     T = 2 (means ACK)
     Code = 0x21
     Options
       Option1 (Content-Format)
         Option Delta = 0xC
         Option Length = 0xD
         Extended Option Length = 0x09
         Option Value = <number for application/pkcs7-mime;
                         smime-type=certs-only>
                    (TODO: We need a new CoAP IANA registered value
                    application/pkcs7-mime, application/csrattrs,
                    application/pkcs10, application/pkcs8,
                    application/pkcs12 )
       Option2 (Block2)
         Option Delta = 0xD
         Option Length = 0x1
         Extended Option Delta = 0x16
         Option Value = 0x1D
     Payload = ... (512 bytes)
```

The third and final Block2:

```
     Ver = 1
     T = 2 (means ACK)
     Code = 0x21
     Options
       Option1 (Content-Format)
         Option Delta = 0xC
         Option Length = 0xD
         Extended Option Length = 0x09
         Option Value = <number for application/pkcs7-mime;
                          smime-type=certs-only>
                    (TODO: We need a new CoAP IANA registered value
                    application/pkcs7-mime, application/csrattrs,
                    application/pkcs10, application/pkcs8,
                    application/pkcs12 )
       Option2 (Block2)
         Option Delta = 0xD
         Option Length = 0x1
         Extended Option Delta = 0x16
         Option Value = 0x25
     Payload = ...
```

[[TODO: This section to be populated. It will address how proxying can take place by an entity that resides at the edge of the CoAP network, such as the Registrar, and can reach the BRSKI server residing in a traditional "TCP setting".]]

[[TODO: This section to be populated. It will address transmission parameters for BRSKI described in sections 4.7 and 4.8 of the CoAP draft. BRSKI does not impose any unique parameters that affect the CoAP parameters in Table 2 and 3 in the CoAP draft but the ones in CoAP could be affecting BRSKI. For example the processing delay of CAs could be less then 2s, but in this case they should send an CoAP ACK every 2s while processing.]]

[[TODO: This section to be populated. This document describes an existing protocol moved to CoAP and there should not be additional security concerns added beyond the protocol’s or CoAP’s specifics security considerations.]]

-01:

Added more binding usecases and Block examples subsections to be expanded on later. Made various text improvements and clarifications.

Added two clarifying sentences in the relevant sections to address Brian C.’s comments and explain that message fragmentation can be avoided to a degree in DTLS and that fragments of block transfers should be confirmed to prevent exacerbated fragment loss in constrained networks.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., and S. Bjarnason, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-03 (work in progress), June 2016.

[I-D.ietf-core-block]
Bormann, D. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-20 (work in progress), April 2016.

[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>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <http://www.rfc-editor.org/info/rfc7030>.

[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>.

[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>.

Authors' Addresses

Max Pritikin
Cisco Systems

Email: pritikin@cisco.com

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: November 5, 2017

B. Silverajan
TUT
M. Ocak
Ericsson
May 4, 2017

CoAP Protocol Negotiation
draft-silverajan-core-coap-protocol-negotiation-05

Abstract

CoAP has been standardised as an application-level REST-based protocol. When multiple transport protocols exist for exchanging CoAP resource representations, this document introduces a way forward for CoAP endpoints as well as intermediaries to agree upon alternate transport and protocol configurations as well as URIs for CoAP messaging, using the CoRE Resource Directory.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] allows clients, origin servers and proxies, to exchange and manipulate resource representations using REST-based methods over UDP or DTLS. CoAP messaging is however being extended to use other alternative underlying transports. These include reliable transports such as TCP, WebSockets and TLS. In addition, the use of SMS as a CoAP transport remains a possibility for simple communication in cellular networks.

When CoAP-based endpoints and proxies possess the ability to perform CoAP messaging over multiple transports, significant benefits can be obtained if communicating client endpoints can discover that multiple transport bindings may exist on an origin server over which CoAP resources can be retrieved. This allows a client to understand and possibly subsitute a different transport protocol configuration for the same CoAP resources on the origin server, based on the preferences of the communicating peers. Inevitably, if two CoAP endpoints reside in distinctly separate networks with orthogonal

transports, a CoAP proxy node is needed between the two networks so that CoAP Requests and Responses can be exchanged properly.

A URI in CoAP, however, serves two purposes simultaneously. It firstly functions as a locator, by specifying the network location of the endpoint hosting the resource, and the underlying transport used by CoAP for accessing the resource representation. It secondly identifies the name of the specific resource found at that endpoint together with its namespace, or resource path. A single CoAP URI cannot be used to express the identity of the resource independently of alternate underlying transports or protocol configuration. Multiple URIs can result for a single CoAP resource representations if:

- the authority components of the URI differ, owing to the same physical host exposing several network endpoints. For example, "coap://example.org/sensors/temperature" and "coap://example.net/sensors/temperature"

- the scheme components of the URI differ, owing to the origin server exposing several underlying transport alternatives. For example, "coap://example.org/sensors/temperature" and "coap+tcp://example.org/sensors/temperature"

- the path components of the URI differ, should an origin server also allow alternative transport endpoint such as the WebSocket protocol, to be expressed using the path. For example, "coap://example.org/sensors/temperature" and "coap+ws://example.org/ws-endpoint/sensors/temperature"

Without a priori knowledge, clients would be unable to ascertain if two or more URIs provided by an origin server are associated to the same representation or not. Consequently, a communication mechanism needs to be conceived to allow an origin server to properly capture the relationship between these alternate representations or locations and then subsequently supply this information to clients. This also goes some way in limiting URI aliasing [WWWArchv1].

In order to support CoAP clients, proxies and servers wishing to use CoAP over multiple transports, this draft proposes the following:

- An ability for servers to register supported CoAP transports to a CoRE Resource Directory [I-D.ietf-core-resource-directory] with optional registration lifetime values

- A means for CoAP clients to interact with a CoRE resource directory interface for requesting and discovering alternative transports and locations of CoAP resources

- o New Resource Directory parameter types enabling the above-mentioned features.

(Note: Although previous versions of this draft provided a mechanism for CoAP clients to directly interact with, discover, use and possibly even negotiate an alternative transport for CoAP-based communication directly with an origin server, discussions at the CoRE Working Group yielded new insights about problems with the proposed approach [CoREWG96]. The current version consequently adopts the usage of the CoRE Resource Directory. Future work is planned on performing discovery and negotiation without the RD as well.)

2. Aim

The following simple scenarios aim to better portray how CoAP protocol negotiation benefits communicating nodes

2.1. Overcoming Middlebox Issues

Discovering which transports are available is important for a client to determine the optimal alternative to perform CoAP messaging according to its needs, particularly when separated from a CoAP server via a NAT. It is well-known that some firewalls as well as many NATs, particularly home gateways, hinder the proper operation of UDP traffic. NAT bindings for UDP-based traffic do not have as long timeouts as TCP-based traffic.

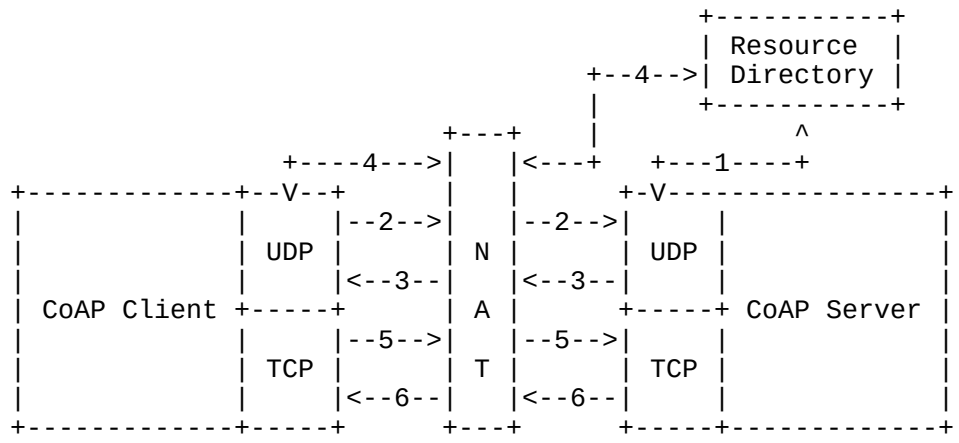


Figure 1: CoAP Client initially accesses CoAP Server over UDP and then switching to TCP

Figure 1 depicts such a scenario. Step 1 depicts the CoAP Server registering its transports to a Resource Directory. A CoAP client uses UDP initially for accessing a CoAP Server in Step 2 and receives a response in Step 3. Subsequently a CoAP client, residing behind a NAT, performs a lookup on the Resource Directory in Step 4 to discover alternative transports offered by the server. Steps 5 and 6 illustrate the client then deciding to use TCP for CoAP messaging instead of UDP to set up an Observe relationship for a resource at the CoAP Server, in order to avoid incoming packets containing resource updates being discarded by the NAT.

2.2. Better resource caching and serving in proxies

Figure 2 outlines a more complex example of intermediate nodes such as CoAP-based proxies to intelligently cache and respond to CoAP or HTTP clients with the same resource representation requested over alternative transports or server endpoints. As with the earlier example, the CoAP Server registers its transports to a Resource Directory (This is assumed to be performed beforehand and not depicted in the figure, for brevity)

In this example, a CoAP over WebSockets client successfully obtains a response from a CoAP forward proxy to retrieve a resource representation from an origin server using UDP, by supplying the CoAP server’s endpoint address and resource in a Proxy-URI option. Arrow 1 represents a GET request to "coap+ws://proxy.example.com" which

subsequently retrieves the resource from the CoAP server using the URI "coap://example.org/sensors/temperature", shown as arrow 2.

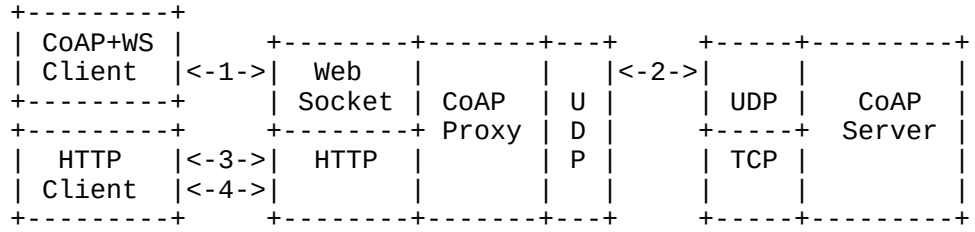


Figure 2: Proxying and returning a resource's alternate cached representations to multiple clients

Subsequently, assume an HTTP client requests the same resource, but instead specifies a CoAP over TCP alternative URI instead. Arrow 3 represents this event, where the HTTP client performs a GET request to "http://proxy.example.com/coap+tcp://example.org/sensors/temperature". When the proxy receives the request, instead of immediately retrieving the temperature resource again over TCP, it first verifies from the Resource Directory whether the cached resource retrieved over UDP is a valid equivalent representation of the resource requested by the HTTP client over TCP. Upon confirmation, the proxy is able to supply the same cached representation to the HTTP client as well (arrow 4).

3. Node Types based on Transport Availability

In [RFC7228], Tables 1, 3 and 4 introduced classification schemes for devices, in terms of their resource constraints, energy limitations and communication power. For this document, in addition to these capabilities, it seems useful to also identify devices based on their transport capabilities.

Name	Transport Availability
T0	Single transport
T1	Multiple transports, with one or more active at any point in time
T2	Multiple active and persistent transports at all times

Table 1: Classes of Available Transports

Type T0 nodes possess the capability of exactly 1 type of transport channel for CoAP, at all times. These include both active and sleepy nodes, which may choose to perform duty cycling for power saving.

Type T1 nodes possess multiple different transports, and can retrieve or expose CoAP resources over any or all of these transports. However, not all transports are constantly active and certain transport channels and interfaces could be kept in a mostly-off state for energy-efficiency, such as when using CoAP over SMS.

Type T2 nodes possess more than 1 transport, and multiple transports are simultaneously active at all times in a persistent manner. CoAP proxy nodes which allow CoAP endpoints from disparate transports to communicate with each other, are a good example of this.

4. New Resource Directory Parameters

In order to allow resource interactions between clients and servers with multiple locations or transports, the registration, update and lookup interfaces of the CoRE Resource Directory need to be extended. In this section two new RD parameters, "at" and "tt" are introduced. Both are optional CoAP features. If supported, they occur at the granularity level of an origin server, ie. they cannot be applied selectively on some resources only. When absent, it is assumed that the server does not support multiple transports or locations.

4.1. The ’at’ RD parameter

A CoAP server wishing to advertise its resources over multiple transports does so by using a new "at" parameter to register a list of CoAP alternative transport URIs during registration with a

Resource Directory. Such a URI would contain the schemes, addresses as well as any ports or paths at which the server is available.

Name	Query	Validity	Description
CoAP Transport URI List	at	URI	Comma separated list of URIs (scheme, address, port, and path) available at the server

Table 2: The "at" RD parameter

The "at" parameter extends the Resource Directory's Registration and Update interfaces.

The following example shows a type T1 endpoint registering its resources and advertising its ability to use TCP as an alternative transport:

```
Req: POST coap:/rd.example.com/rd
 ?ep=node1&at=coap+tcp://server.example.com,coap+ws://server.example.com:5683
/ws/
Content-Format: 40
Payload:
</sensors/temp>;ct=41;rt="temperature-f";if="sensor",
</sensors/door>;ct=41;rt="door";if="sensor"

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

The next example shows the same endpoint updating its registration with a new lifetime and the availability of a single alternative transport for CoAP (in this case WebSockets):

```
   Req: POST /rd/4521?lt=600&at=coap+ws://server.example.com:5683/ws/
   Content-Format: 40
   Payload:
   </sensors/temp>;ct=41;rt="temperature-f";if="sensor",
   </sensors/door>;ct=41;rt="door";if="sensor"

   Res: 2.04 Changed
```

4.2. The ’tt’ RD parameter

A CoAP client wishing to perform a look-up on the Resource Directory for CoAP servers supporting multiple transports does so by using a new "tt" parameter to query for CoAP alternative transport URIs.

Name	Query	Validity	Description
CoAP Transport Type	tt		Transport type requested by the client

Table 3: The "tt" RD parameter

The "tt" parameter extends the Resource Directory’s rd-lookup interface.

The following example shows a client performing a lookup for endpoints supporting TCP:

```
Req: GET /rd-lookup/ep?tt=tcp

Res: 2.05 Content
<coap+tcp://[FDFD::123]:61616>;ep="node5",
<coap+tcp://[FDFD::123]:61616>;ep="node7"
```

The next example shows a client performing a lookup for all transports supported by a specific endpoint:

```
Req: GET /rd-lookup/ep?ep=node5&tt=*

Res: 2.05 Content
<coap+tcp://[FDFD::123]:61616>;ep="node5",
<coap+ws://[FDFD::123]:61616>;ep="node5"
```

5. IANA Considerations

This document requests the registration of new RD parameter types "at" and "tt".

6. Security Considerations

When multiple transports, locations and representations are used, some obvious risks are present both at the origin server as well as by requesting clients.

When a client is presented with alternate URIs for retrieving resources, it presents an opportunity for attackers to mount a series of attacks, either by hijacking communication and masquerading as an alternate location or by using a man-in-the-middle attack on TLS-based communication to a server and redirecting traffic to an alternate location. A malicious or compromised server could also be used for reflective denial-of-service attacks on innocent third parties. Moreover, clients may obtain web links to alternate URIs containing weaker security properties than the existing session.

7. Acknowledgements

Thanks to Klaus Hartke for comments and reviewing this draft, and Teemu Savolainen for initial discussions about protocol negotations and lifetime values. Zach Shelby provided significant suggestions on how the Resource Directory can be employed and extended in place of link attributes and relation types.

8. References

8.1. Normative References

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-10 (work in progress), March 2017.

[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>.

[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>.

[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>.

8.2. Informative References

[CoREWG96]
https://www.ietf.org/proceedings/96/minutes/minutes-96-core, "IETF96 CoRE minutes", July 2016.

[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>.

[WWWArchv1]
http://www.w3.org/TR/webarch/#uri-aliases, "Architecture of the World Wide Web, Volume One", December 2004.

Appendix A. Change Log

A.1. From -04 to -05

Freshness update

A.2. From -03 to -04

Removed previously introduced link attribute and relation types

Initial foray with Resource Directory support

A.3. From -02 to -03

Added new author

Rewrite of "Introduction" section

Added new Aims Section

Added new Section on Node Types

Introduced "al" Active Lifetime link attribute

Added new Section on Observing transports and resources

Security and IANA considerations sections populated

A.4. From -01 to -02

Freshness update.

A.5. From -00 to -01

Reworked "Introduction" section, added "Rationale", and "Goals" sections.

Authors' Addresses

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi

Mert Ocak
Ericsson
Hirsalantie 11
02420 Jorvas
Finland

Email: mert.ocak@ericsson.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

D. Thaler
Microsoft

October 31, 2016

COAP Redirects
draft-thaler-core-redirect-01

Abstract

This document allows a Constrained Application Protocol (CoAP) server to redirect a client to a new URI. The primary use case is to allow a client using multicast CoAP discovery to learn a COAPS endpoint of the server, without the server revealing privacy-sensitive information. This improves security and privacy in environments with untrusted clients.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a specialized web transfer protocol for use with constrained nodes and constrained networks. When COAP nodes can appear on a network that allows untrusted clients, security and privacy issues can arise, as discussed in Section 11 of [RFC7252].

This document focuses on a solution for a specific use case: preventing privacy-sensitive information from being passed to untrusted clients, especially as part of resource discovery. The resource discovery phase is important because DTLS is not used with multicast COAP.

The specific relevant threats are:

- Correlation across location: If a COAP server can move between multiple networks in which an attacker has a presence, the attacker can potentially correlate responses from the COAP server across the two locations and determine that the same entity is moving between those two locations. This can even be used to identify individuals, such as when the COAP server is in a wearable device.

- Correlation across time: If a COAP server is available periodically in the same location over a long time, an attacker in that location can potentially correlate reponses over time and determine that it is the same entity, even though the IP address and layer-2 address may be different. This can even be used to identify individuals, such as when the COAP server is in a wearable device.

- Fingerprinting: Device-specific vulnerability exploitation can be most easily accomplished if an attacker can easily narrow down what software the server runs. Information returned via multicast service discovery can facilitate such fingerprinting.

For more discussion of these threats, see Section 5.2 of [RFC6973], Section 3 of [RFC7721], and [I-D.winfaa-intarea-broadcast-consider].

To mitigate these threats, this document defines the ability for a server to redirect a client to another URI. Specifically, the expected use is that in response to an unsecured COAP request, a privacy-sensitive server could be configured to simply respond by redirecting the client to a COAPS endpoint, thus allowing the client to discover a unicast endpoint, but not to discover any privacy-sensitive information without establishing a secured unicast connection.

By comparison, HTTP (Section 6.4.2 of [RFC7231]) redirects with 301 (Moved Permanently) and a Location header containing the new URI. COAP, on the other hand, defines Location-Path and Location-Query COAP options [RFC7252] for those components of the URI, but did not define options for the other URI components. [ListDiscussion] explains:

> While early drafts of CoAP did have some forms of redirection, we found that the use cases most people had in mind did not call for redirects. The main reason is that in a CoRE world, URIs are usually found through a discovery process, and these URIs can be made to point to the right place right away.

The use case motivating this document, however, is specifically for redirects as part of the discovery process itself.

1.1. Example

Existing clients conforming to the OIC 1.1 Core spec [OIC1.1Core] sections 10 and 11.3.5 do discovery by sending a multicast CoAP GET for "/oic/res". Existing servers will respond with links to a set of resources, but that information might be privacy-sensitive in some cases. For example, it might contain sufficient a unique identifier

of the server, or information sufficient for an attacker to determine what version of what software it runs. (A sample response can be found in section 10.2 of [OIC1.1Core].) Hence a privacy-sensitive server needs a way to be discovered by trusted clients without revealing privacy-sensitive information to untrusted observers. A redirect allows a client to send the same request, thus not increasing the amount of multicast traffic on the network.

For example, consider a network with a privacy-sensitive server, and a legacy server. A client wants to efficiently discover both servers. The client can send a single multicast GET for "/oic/res", and the legacy server would send a unicast response with the requested data, whereas the privacy-sensitive server would respond with a unicast redirect to "coaps://<ipaddr>:<port>/oic/res". The client can then generate a unicast GET over coaps to get the actual data, if permitted, from the privacy-sensitive server. This mechanism keeps the latency and number of messages to a minimum.

2. Alternatives Considered

This section discusses why existing alternatives are not sufficient.

2.1. Just use normal multicast discovery

Normal multicast discovery is susceptible to the threats discussed earlier. Another approach would be for multicast discovery to return only generic information that is the same for every device, and hence does not reveal any privacy related information or allow fingerprinting. This is undesirable since the resource handler would have to return different information based on whether the client is authenticated vs. unauthenticated, and thus is complex and error prone to implement and maintain.

2.2. Just use a resource directory

A resource directory could be used and only provide data to authenticated clients. However, the same problem still remains as to how to discover the resource directory itself. One could potentially use an alternate discovery protocol such as DNS-SD, but this introduces additional complexity when clients otherwise just use COAP for both discovery and communication. In addition, requiring a resource directory to be implemented, deployed, and maintained in a constrained environment presents an extra deployment burden that is desirable to avoid.

2.3. Use Alternative-Address

Section 4.5 of [I-D.ietf-core-coap-tcp-tls] provides an Alternative-Address option, which can be used to redirect the client to another transport address. However, it states:

> The Alternative-Address elective option requests the peer to instead open a connection of the same kind as the present connection to the alternative transport address given. Its value is in the form "authority" as defined in Section 3.2 of [RFC3986].

Thus, Alternative-Address can indicate another authority component, but it explicitly requires the same URI scheme to be used, so it cannot be used to redirect from coap to coaps.

3. Redirects

3.1. Option Definitions

The following additional options are defined.

Number	Name	Format	Length	Base Value
TBD	Location-Scheme	string	0-255	(none)
TBD	Location-Authority	string	0-255	(none)

3.1.1. Location-Scheme and Location-Authority

Section 5.10.7 of [RFC7252] states:

> The options that are used to compute the relative URI-reference are collectively called Location-* options. Beyond Location-Path and Location-Query, more Location-* options may be defined in the future and have been reserved option numbers 128, 132, 136, and 140.

The Location-Scheme and Location-Authority options are subject to all rules for Location-* options discussed in [RFC7252].

Together with Location-Path and Location-Query, the Location-Scheme and Location-Authority Options indicate a relative URI that contains either of an absolute path, a query string, or both. A combination of these options is included in a 3.01 (Moved Permanently) response to indicate the new location of the requested resource relative to the request URI.

If a response with Location-Scheme and/or Location-Authority Options passes through a cache that interprets these options and the implied URI identifies one or more currently stored responses, those entries MUST be marked as not fresh.

The Location-Scheme and Location-Authority Option can contain any character sequence conforming to the scheme and authority components defined in [RFC3986].

3.2. Response Codes

This specification adds the following response code.

3.2.1. 3.01 Moved Permanently

This Response Code indicates that the target resource has been assigned a new permanent URI and any future references to this resource ought to use the indicated effective URI.

The server MUST include in the response any of the following options whose values differ between the requested URI and the new effective URI: Location-Scheme, Location-Authority, Location-Path, and Location-Query. The client SHOULD use the Location field value for automatic redirection.

A 3.01 response is cacheable. Caches can use the Max-Age Option to determine freshness. A 3.01 response cannot be validated.

4. IANA Considerations

This document adds the following option numbers to the "CoAP Option Numbers" registry defined by [RFC7252]:

Number	Name	Reference
TBD	Location-Scheme	I-D.thaler-core-redirect
TBD	Location-Authority	I-D.thaler-core-redirect

NOTE: Section 5.10.7 of [RFC7252] reserves option numbers 128, 132, 136, and 140 for new Location-* options. Thus, the option numbers should be assigned from that set.

This document adds the following response codes to the "CoAP Response Codes" registry defined by [RFC7252]:

Code	Description	Reference
3.01	Moved Permanently	I-D.thaler-core-redirect

5. Security Considerations

The use case for this document is specifically to mitigate privacy concerns by allowing a request to an unsecured URI to be redirected to a secured URI.

Preventing identifying information from being observed by untrusted clients doing multicast discovery is necessary but not sufficient to mitigate the privacy issues discussed in Section 1. That is, one must also use an authentication scheme for subsequent unicast messages that does not reveal a stable identifier to clients before authentication is complete. Mutual authentication schemes exist (e.g., [Balfanz]) that only reveal the identity of both endpoints if authentication succeeds, but they may not yet be available in current standards and popular code bases.

6. References

6.1. 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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[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>.

6.2. Informative References

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <http://www.rfc-editor.org/info/rfc6973>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <http://www.rfc-editor.org/info/rfc7721>.

[I-D.ietf-core-coap-tcp-tls] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-05 (work in progress), October 2016.

[I-D.winfaa-intarea-broadcast-consider] Winter, R., Faath, M., and F. Weisshaar, "Privacy considerations for IP broadcast and multicast protocol designers", draft-winfaa-intarea-broadcast-consider-03 (work in progress), September 2016.

[Balfanz] Balfanz, D., Durfee, G., Shankar, N., Smetters, D., Staddon, J., and H-C. Wong, "Secret Handshakes From Pairing-based Key Agreements", May 2003, <http://ieeexplore.ieee.org/document/1199336>.

[ListDiscussion] Bormann, C., "Question about Location and redirection", Symposium on Security and Privacy 2003, October 2013, <https://www.ietf.org/mail-archive/web/core/current/msg04867.html>.

[OIC1.1Core] Open Connectivity Foundation, "OIC Core Specification V1.1.0", 2016, <https://openconnectivity.org/wp-content/uploads/2016/10/OIC_1.1-Specification.zip>.

Author’s Address

Dave Thaler
Microsoft
One Microsoft Way
Redmond, WA 98052
USA

Email: dthaler@microsoft.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

M. Tiloca
RISE SICS AB
G. Selander
F. Palombini
Ericsson AB

March 13, 2017

Secure group communication for CoAP
draft-tiloca-core-multicast-oscoap-01

Abstract

This document describes a method for application layer protection of messages exchanged with the Constrained Application Protocol (CoAP) in a group communication context. The proposed approach relies on Object Security of CoAP (OSCOAP) and the CBOR Object Signing and Encryption (COSE) format. All security requirements fulfilled by OSCOAP are maintained for multicast CoAP request messages and related unicast CoAP response messages. Source authentication of all messages exchanged within the group is ensured, by means of digital signatures produced through asymmetric private keys of sender devices and embedded in the protected CoAP messages.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol specifically designed for constrained devices and networks.

[RFC7390] enables group communication for CoAP, addressing use cases where deployed devices benefit from a group communication model for example to limit latencies and improve performance. Use cases include lighting control, integrated building control, software and firmware updates, parameter and configuration updates, commissioning of constrained networks, and emergency multicast. [RFC7390]

recognizes the importance to introduce a secure mode for CoAP group communication. This specification defines such a mode.

Object Security of CoAP (OSCOAP)[I-D.ietf-core-object-security] describes a security protocol based on the exchange of protected CoAP messages. OSCOAP builds on CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg] and provides end-to-end encryption, integrity, and replay protection across intermediate modes. To this end, a CoAP message is protected by including payload (if any), certain options, and header fields in a COSE object, which finally replaces the authenticated and encrypted fields in the protected message.

This document describes multicast OSCOAP, providing end-to-end security of CoAP messages exchanged between members of a multicast group. In particular, the described approach defines how OSCOAP should be used in a group communication context, while fulfilling the same security requirements. That is, end-to-end security is assured for multicast CoAP requests sent by multicaster nodes to the group and for related unicast CoAP responses sent as reply by multiple listener nodes. Multicast OSCOAP provides source authentication of all CoAP messages exchanged within the group, by means of digital signatures produced through asymmetric private keys of sender devices and embedded in the protected CoAP messages. As in OSCOAP, it is still possible to simultaneously rely on DTLS to protect hop-by-hop communication between a multicaster node and a proxy (and vice versa), and between a proxy and a listener node (and vice versa).

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]. These words may also appear in this document in lowercase, absent their normative meanings.

Readers are expected to be familiar with the terms and concepts described in [RFC7252], [RFC7390] and [RFC7641].

Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

Terminology for protection and processing of CoAP messages through OSCOAP, such as "Security Context", "Master Secret", "Master Salt", is defined in [I-D.ietf-core-object-security].

This document refers also to the following terminology.

- Keying material: data that is necessary to establish and maintain secure communication among member of a multicast group. This includes, for instance, keys, key pairs, and IVs [RFC4949].

- Group Manager (GM): entity responsible for creating a multicast group, establishing and provisioning security contexts among authorized group members, and managing the joining of new group members. A GM can be responsible for multiple multicast groups, while it is not required to be an actual group member and to take part in the group communication. The GM may also be responsible for renewing/updating security contexts and related keying material. Any message exchange with the GM MUST be secured and based on different secure channels for different endpoints.

- Multicaster: member of a multicast group that sends multicast CoAP messagges intended for all members of the group. In a 1-to-N multicast group, only a single multicaster transmits data to the group; in an M-to-N multicast group (where M and N do not necessarily have the same value), M group members are multicasters.

- Listener: member of a multicast group that receives multicast CoAP messages when listening to the multicast IP address associated to the multicast group. A listener MAY reply back, by sending a unicast response message to the multicaster which has sent the multicast message.

- Group request: multicast CoAP request message sent by a multicaster in the group to all listeners in the group through multicast IP.

- Group response: unicast CoAP response message sent back by a listener in the group as a response to a group request received from a multicaster.

- Source authentication: evidence that a received message in the group originated from a specifically identified group member. This also provides assurances that the message was not tampered with by any other group member or an adversary outside the group.

2. Requirements

The following security requirements are out of the scope of this document and are assumed to be already fulfilled.

- Establishment and management of a security context: a security context must be established among the group members by the Group Manager which manages the multicast group. A secure mechanism

must be used to generate, revoke and (re-)distribute keying material, multicast security policies and security parameters in the multicast group. The actual establishment and management of the security context is out of the scope of this document, and it is anticipated that an activity in IETF dedicated to the design of a generic key management scheme will include this feature, preferably based on [RFC3740][RFC4046][RFC4535].

- Multicast data security ciphersuite: all group members MUST agree on a ciphersuite to provide authenticity, integrity and confidentiality of messages in the multicast group. The ciphersuite is specified as part of the security context.

- Backward security: a new device joining the multicast group should not have access to any old security contexts used before its joining. This ensures that a new group member is not able to decrypt confidential data sent before it has joined the group. The adopted key management scheme should ensure that the security context is updated to ensure backward confidentiality. The actual mechanism to update the security context and renew the group keying material upon a group member’s joining has to be defined as part of the group key management scheme.

- Forward security: entities that leave the multicast group should not have access to any future security contexts or message exchanged within the group after their leaving. This ensures that a former group member is not able to decrypt confidential data sent within the group anymore. Also, it ensures that a former member is not able to send encrypted and/or integrity protected messages to the group anymore. The actual mechanism to update the security context and renew the group keying material upon a group member’s leaving has to be defined as part of the group key management scheme.

The following security requirements need to be fulfilled by the approach described in this document:

- Multicast communication topology: this document considers both 1-to-N (one multicaster and multiple listeners) and M-to-N (multiple multicasters and multiple listeners) communication topologies. The 1-to-N communication topology is the simplest group communication scenario that would serve the needs of a typical LLN. For instance, in the lighting control use case, switches are the only entities responsible for sending commands to a group of lighting devices. In more advanced lighting control use cases, a M-to-N communication topology would be required, for instance in case multiple sensors (presence or day-light) are responsible to trigger events to a group of lighting devices.

- Multicast group size: security solutions for group communication SHOULD be able to adequately support different, possibly large, group sizes. Group size is the combination of the number of multicasters and listeners in a multicast group, with possible overlap (i.e. a multicaster MAY also be a listener at the same time). In the use cases mentioned in this document, the number of multicasters (normally the controlling devices) is expected to be much smaller than the number of listeners (i.e. the controlled devices). A security solution for group communication that supports 1 to 50 multicasters would be able to properly cover the group sizes required for most use cases that are relevant for this document. The total number of group members is expected to be in the range of 2 to 100 devices. Groups larger than that SHOULD be divided into smaller independent multicast groups, e.g. by grouping lights in a building on a per floor basis.

- Data replay protection: it MUST NOT be possible to replay a group request message or a group response message, which would disrupt the correct communication in the group and the activity of group members.

- Group-level data confidentiality: messages sent within the multicast group SHALL be encrypted. In fact, some control commands and/or associated responses could pose unforeseen security and privacy risks to the system users, when sent as plaintext. In particular, data confidentiality MAY be required if privacy sensitive data is exchanged in the group. This document considers group-level data confidentiality since messages are encrypted at a group level, i.e. in such a way that they can be decrypted by any member of the multicast group, but not by an external adversary or other external entities.

- Source authentication: messages sent within the multicast group SHALL be authenticated. That is, it is essential to ensure that a message is originated by a member of the group in the first place (group authentication), and in particular by a specific member of the group (source authentication). The approach proposed in this document provides both group authentication and source authentication, both for group requests originated by multicasters and group responses originated by listeners. In order to provide source authentication, outgoing messages are signed by the respective originator group member by means of its own asymmetric private key. The resulting signature is included in the COSE object.

- Message integrity: messages sent within the multicast group SHOULD be integrity protected. That is, it is essential to ensure that a message has not been tampered with by an external adversary or

other external entities which are not group members. Message integrity is provided through the same means used to provide source authentication.

3. Scope Description

An endpoint joins a multicast group by explicitly interacting with the responsible Group Manager. The actual join process MAY be based on the ACE framework [I-D.ietf-ace-oauth-authz] and the OSCOAP profile of ACE [I-D.seitz-ace-oscoap-profile], as discussed in Appendix A.

An endpoint registered as member of a group can behave as a multicaster and/or as a listener. As a multicaster, it can transmit multicast request messages to the group. As a listener, it receives multicast request messages from any multicaster in the group, and possibly replies by transmitting unicast response messages. A number of use cases that benefit from secure group communication are discussed in Appendix B. Upon joining the group, endpoints are not required to know how many and what endpoints are active in the same group.

An endpoint which is registered as member of a group is identified by an endpoint ID, which is not necessarily related to any protocol-relevant identifiers, such as IP addresses. The Group Manager generates and manages endpoint IDs in order to ensure their uniqueness within a same multicast group. That is, there cannot be multiple endpoints that belong to the same group and are associated to a same endpoint ID.

In order to participate in the secure group communication, an endpoint needs to maintain additional information elements, stored in its own security context. Those include keying material used to protect and verify group messages, as well as the public keys of other endpoints in the groups, in order to verify digital signatures of secure messages and ensure their source authenticity. These pieces of information are provided by the Group Manager through out-of-band means or other pre-established secure channels. Further details about establishment, revocation and renewal of the security context and keying material are out of the scope of this document.

According to [RFC7390], any possible proxy entity is supposed to know about the multicasters in the group and to not perform aggregation of response messages. Also, every multicaster expects and is able to handle multiple unicast response messages associated to a given multicast request message.

4. Security Context

To support multicast communication secured with OSCOAP, each endpoint registered as member of a multicast group maintains a Security Context as defined in Section 3 of [I-D.ietf-core-object-security]. In particular, each endpoint in a group stores:

1. one Common Context, received from the Group Manager upon joining the multicast group and shared by all the endpoints in the group. The Common Context contains the COSE AEAD algorithm, the Master Secret and, optionally, the Master Salt used to derive endpoint-based keying material (see Section 3.2 of [I-D.ietf-core-object-security]). All the endpoints in the group agree on the same COSE AEAD algorithm. Besides, in addition to what is defined in [I-D.ietf-core-object-security], the Common Context stores the following parameters:

   * Context Identifier (Cid). Variable length byte string that identifies the Security Context. The Cid used in a multicast group is determined by the responsible Group Manager and does not change over time. A Cid MUST be unique in the sets of all the multicast groups associated to the same Group Manager. The choice of the Cid for a given group's Security Context is application specific, but it is RECOMMENDED to use 64-bit long pseudo-random Cids, in order to have globally unique Context Identifiers. It is the role of the application to specify how to handle possible collisions.

   * Counter signature algorithm. Value that identifies the algorithm used for source authenticating messages sent within the group. Its value is immutable once the security context is established. All the endpoints in the group agree on the same counter signature algorithm.

2. one Sender Context, used to secure outgoing messages. In particular, the Sender Context is initialized according to Section 3 of [I-D.ietf-core-object-security], once the endpoint has joined the multicast group. Besides, in addition to what is defined in [I-D.ietf-core-object-security], the Sender Context stores also the endpoint's asymmetric public-private key pair;

3. one Recipient Context for each distinct endpoint from which messages are received, used to process such incoming secure messages. The endpoint creates a new Recipient Context upon receiving an incoming message from another endpoint in the group for the first time. Besides, in addition to what is defined in [I-D.ietf-core-object-security], each Recipient Context stores also the public key of the associated other endpoint from which

secure messages are received. Possible approaches to provision and retrieve public keys of group members are discussed in Section 7.4.

The Sender Key/IV stored in the Sender Context and the Recipient Keys/IVs stored in the Recipient Contexts are derived according to the same scheme defined in Section 3.2 of [I-D.ietf-core-object-security].

The 3-tuple (Cid, Sender ID, Partial IV) is called Transaction Identifier (Tid), and SHALL be unique for each Master Secret. The Tid is used as a unique challenge in the COSE object of the protected CoAP request. The Tid is part of the Additional Authenticated Data (AAD, see Section of 5.2 of [I-D.ietf-core-object-security]) of the protected CoAP response message, which is how unicast responses are bound to multicast requests.

5. The COSE Object

When creating a protected CoAP message, an endpoint in the group computes the COSE object as defined in Section 5 of [I-D.ietf-core-object-security], with the following modifications.

1. The value of the "Partial IV" parameter in the "protected" field is set to the Sequence Number and SHALL be present in both multicast requests and unicast responses. Specifically, a multicaster endpoint sets the value of "Partial IV" to the Sequence Number from its own Sender Context, upon sending a multicast request message. Similarly, a listener endpoint sets the value of "Partial IV" to the Sequence Number from its own Sender Context, upon sending a unicast response message.

2. The value of the "kid" parameter in the "protected" field is set to the Sender ID of the endpoint and SHALL be present in both multicast requests and unicast responses.

3. The "protected" field of the "Headers" field SHALL include also the following parameter:

   * gid : its value is set to the Context Identifier (Cid) of the group’s Security Context. This parameter is optional if the message is a CoAP response.

4. The Additional Authenticated Data (AAD) considered to compute the COSE object is extended. In particular, the "external_aad" considered for secure response messages SHALL include also the following parameter:

* cid : bstr, contains the Context Idenfier (Cid) of the Security Context considered to protect the request message (which is same as the Cid considered to protect the response message).

5. Before transmitting any secure CoAP message, the sender endpoint uses its own private key to create a counter signature of the COSE_Encrypt0 object (Appendix C.4 of [I-D.ietf-cose-msg]). Then, the counter signature is included in the Header of the COSE object in its "unprotected" field.

6. Message Processing

Each multicast request message and unicast response message is protected and processed as specified in [I-D.ietf-core-object-security], with the modifications described in the following sections. Furthermore, error handling and processing of invalid messages are performed according to the same principles adopted in [I-D.ietf-core-object-security].

6.1. Protecting the Request

A multicaster endpoint transmits a secure multicast request message as described in Section 7.1 of [I-D.ietf-core-object-security], with the following modifications:

1. The multicaster endpoint stores the association Token - Cid. That is, it SHALL be able to find the correct Security Context used to protect the multicast request and verify the unicast response(s) by using the CoAP Token considered in the message exchange.

2. The multicaster endpoint computes the COSE object as defined in Section 5 of this specification.

6.2. Verifying the Request

Upon receiving a secure multicast request message, a listener endpoint proceeds as described in Section 7.2 of [I-D.ietf-core-object-security], with the following modifications:

1. The listener endpoint retrieves the Context Identifier from the "gid" parameter of the received COSE object, and uses it to identify the correct group’s Security Context.

2. The listener endpoint retrieves the Sender ID from the header of the COSE object. Then, the Sender ID is used to retrieve the correct Recipient Context associated to the multicaster endpoint and used to process the request message. When receiving a secure

multicast CoAP request message from that multicaster endpoint for the first time, the listener endpoint creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], and includes the multicaster endpoint's public key.

3. The listener endpoint retrieves the corresponding public key of the multicaster endpoint from the associated Recipient Context and uses it to verify the counter signature, before proceeding with the verification and decryption of the secure request message.

6.3. Protecting the Response

A listener endpoint that has received a multicast request message MAY reply with a secure unicast response message, which is protected as described in Section 7.3 of [I-D.ietf-core-object-security], with the following modifications:

1. The listener endpoint considers the Transaction Identifier (Tid) as defined in Section 4 of this specification.

2. The listener endpoint computes the COSE object as defined in Section 5 of this specification.

6.4. Verifying the Response

Upon receiving a secure unicast response message, a multicaster endpoint proceeds as described in Section 7.4 of [I-D.ietf-core-object-security], with the following modifications:

1. The multicaster endpoint considers the Security Context identified by the Token of the received response message.

2. The multicaster endpoint retrieves the Sender ID from the header of the COSE object. Then, the Sender ID is used to retrieve the correct Recipient Context associated to the listener endpoint and used to process the response message. When receiving a secure CoAP response message from that listener endpoint for the first time, the multicaster endpoint creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], and includes the listener endpoint's public key.

3. The multicaster endpoint retrieves the corresponding public key of the listener endpoint from the associated Recipient Context and uses it to verify the counter signature, before proceeding

with the verification and decryption of the secure response message.

The mapping between unicast response messages from listener endpoints and the associated multicast request message from a multicaster endpoint relies on the same principle adopted in [I-D.ietf-core-object-security]. That is, it is based on the Transaction Identifier (Tid) associated to the secure multicast request message, which is considered by listener endpoints as part of the Additional Authenticated Data when protecting their own response message.

7. Security Considerations

Specific security aspects to be taken into account are discussed below.

7.1. Group-level Security

The approach described in this document relies on commonly shared group keying material to protect communication within a multicast group. This means that messages are encrypted at a group level (group-level data confidentiality), i.e. they can be decrypted by any member of the multicast group, but not by an external adversary or other external entities.

In addition, it is required that all group members are trusted, i.e. they do not forward the content of group messages to unauthorized entities. However, in many use cases, the devices in the multicast group belong to a common authority and are configured by a commissioner. For instance, in a professional lighting scenario, the roles of multicaster and listener are configured by the lighting commissioner, and devices strictly follow those roles.

Furthermore, the presented approach SHOULD take into consideration the risk of compromise of group members. Such a risk is reduced when multicast groups are deployed in physically secured locations, like lighting inside office buildings. The adoption of key management schemes for secure revocation and renewal of security contexts group keying material SHOULD be considered.

7.2. Management of Group Keying Material

As stated in Section 2, it is important to adopt a group key management scheme that SHOULD update the security context and keying material in the group, before a new endpoint joins the group or after a currently present endpoint leaves the group. This is necessary in

order to preserve backward confidentiality and forward confidentiality in the multicast group.

Especially in dynamic, large-scale, multicast groups where endpoints can join and leave at any time, it is important that the considered group key management scheme is efficient and highly scalable with the group size, in order to limit the impact on performance due to the security context and keying material update.

7.3. Late Joining Endpoints

Upon joining the multicast group when the system is fully operative, listeners are not aware of the current sequence number values used by different multicasters to transmit multicast request messages. This means that, when such listeners receive a secure multicast message from a multicaster, they are not able to verify if that message is fresh and has not been replayed.

In order to address this issue, upon receiving a multicast message from a particular multicaster for the first time, late joining listeners can initialize their last-seen sequence number in their Recipient Context associated to that multicaster. However, after that they drop the message, without delivering it to the application layer. This provides a reference point to identify if future multicast messages from the same multicaster are fresher than the last one seen. As an alternative, a late joining listener can directly contact the multicaster, and explicitly request a confirmation of the sequence number in the first received multicast message.

A possible different approach considers the GM as an additional listener in the multicast group. Then, the GM can maintain the sequence number values of each multicaster in the group. When late joiners send a request to the GM to join the group, the GM can provide them with the list of sequence number values to be stored in the Recipient Contexts associated to the appropriate multicasters.

7.4. Provisioning of Public Keys

Upon receiving a secure CoAP message, a recipient endpoint relies on the sender endpoint’s public key, in order to verify the counter signature conveyed in the COSE Object.

If not already stored in the Recipient Context associated to the sender endpoint, the recipient endpoint retrieves the public key from a trusted key repository. In such a case, the correct binding between the sender endpoint and the retrieved public key MUST be assured, for instance by means of public key certificates. Further

details about how this requirement can be fulfilled are out of the scope of this document.

Alternatively, the Group Manager can be configured to store public keys of group members and provide them upon request. In such a case, upon joining a multicast group, an endpoint provides the Group Manager with its own public key, by means of the same secure channel used to carry out the join procedure. After that, the Group Manager MUST perform a proof-of-possession challenge-response with the joining endpoint, in order to verify that it actually owns the associated private key. In case of success, the Group Manager stores the received public key as associated to the joining endpoint and its endpoint ID. From then on, that public key will be available for secure and trusted delivery to other endpoints in the multicast group.

Note that in simple, less dynamic, multicast groups, it can be convenient for the Group Manager to provide an endpoint upon its joining with the public keys associated to all endpoints currently present in the group.

8. IANA Considerations

This document has no actions for IANA.

9. Acknowledgments

The authors sincerely thank Rolf Blom, Carsten Bormann, John Mattsson and Jim Schaad for their feedback and comments.

10. References

10.1. Normative References

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-01 (work in progress), December 2016.

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

[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>.

[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>.

[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>.

10.2. Informative References

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-05 (work in progress), February 2017.

[I-D.seitz-ace-oscoap-profile] Seitz, L. and F. Palombini, "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-01 (work in progress), October 2016.

[I-D.selander-ace-cose-ecdhe] Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-04 (work in progress), October 2016.

[RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, DOI 10.17487/RFC3740, March 2004, <http://www.rfc-editor.org/info/rfc3740>.

[RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, DOI 10.17487/RFC4046, April 2005, <http://www.rfc-editor.org/info/rfc4046>.

[RFC4535] Harney, H., Meth, U., Colegrove, A., and G. Gross, "GSAKMP: Group Secure Association Key Management Protocol", RFC 4535, DOI 10.17487/RFC4535, June 2006, <http://www.rfc-editor.org/info/rfc4535>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <http://www.rfc-editor.org/info/rfc4944>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <http://www.rfc-editor.org/info/rfc6282>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[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>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <http://www.rfc-editor.org/info/rfc7390>.

Appendix A. Group Joining Based on the ACE Framework

The join process to register an endpoint as a new member of a multicast group MAY be based on the ACE framework [I-D.ietf-ace-oauth-authz] and the OSCOAP profile of ACE [I-D.seitz-ace-oscoap-profile]. With reference to the terminology defined in OAuth 2.0 [RFC6749]:

- The joining endpoint acts as Client;
- The Group Manager acts as Resource Server, exporting one join-resource for each multicast group it is responsible for;
- An Authorization Server enables and enforces authorized access of the joining endpoint to the Group Manager and its join-resources.

Then, in accordance with [I-D.seitz-ace-oscoap-profile], the joining endpoint and the Group Manager rely on OSCOAP [I-D.ietf-core-object-security] for secure communication and consider Ephemeral Diffie-Hellman Over COSE (EDHOC)

[I-D.selander-ace-cose-ecdhe] as a possible method to establish key material.

The joining endpoint sends to the Group Manager an OSCOAP request to access the join-resource associated to the multicast group to join. The Group Manager replies with an OSCOAP response including the Common Context associated to that group (see Section 4). In case the Group Manager is configured to store the public keys of group members, the joining endpoint additionally provides the Group Manager with its own public key, and MAY request from the Group Manager the public keys of the endpoints currently present in the group (see Section 7.4).

Both the joining endpoint and the Group Manager MUST adopt secure communication also for any message exchange with the Authorization Server. To this end, different alternatives are possible, including OSCOAP and DTLS [RFC6347].

Appendix B. List of Use Cases

Group Communication for CoAP [RFC7390] provides the necessary background for multicast-based CoAP communication, with particular reference to low-power and lossy networks (LLNs) and resource constrained environments. The interested reader is encouraged to first read [RFC7390] to understand the non-security related details. This section discusses a number of use cases that benefit from secure group communication. Specific security requirements for these use cases are discussed in Section 2.

- Lighting control: consider a building equipped with IP-connected lighting devices, switches, and border routers. The devices are organized into groups according to their physical location in the building. For instance, lighting devices and switches in a room or corridor can be configured as members of a single multicast group. Switches are then used to control the lighting devices by sending on/off/dimming commands to all lighting devices in a group, while border routers connected to an IP network backbone (which is also multicast-enabled) can be used to interconnect routers in the building. Consequently, this would also enable logical multicast groups to be formed even if devices in the lighting group may be physically in different subnets (e.g. on wired and wireless networks). Connectivity between ligthing devices may be realized, for instance, by means of IPv6 and (border) routers supporting 6LoWPAN [RFC4944][RFC6282]. Group communication enables synchronous operation of a group of connected lights, ensuring that the light preset (e.g. dimming level or color) of a large group of luminaires are changed at the same perceived time. This is especially useful for providing a

visual synchronicity of light effects to the user. Devices may reply back to the switches that issue on/off/dimming commands, in order to report about the execution of the requested operation (e.g. OK, failure, error) and their current operational status.

- Integrated building control: enabling Building Automation and Control Systems (BACSs) to control multiple heating, ventilation and air-conditioning units to pre-defined presets. Controlled units can be organized into multicast groups in order to reflect their physical position in the building, e.g. devices in the same room can be configured as members of a single multicast group. Furthermore, controlled units are expected to possibly reply back to the BACS issuing control commands, in order to report about the execution of the requested operation (e.g. OK, failure, error) and their current operational status.

- Software and firmware updates: software and firmware updates often comprise quite a large amount of data. Therefore, it can overload a LLN that is otherwise typically used to deal with only small amounts of data, on an infrequent base. Rather than sending software and firmware updates as unicast messages to each individual device, multicasting such updated data to a larger group of devices at once displays a number of benefits. For instance, it can significantly reduce the network load and decrease the overall time latency for propagating this data to all devices. Even if the complete whole update process itself is secured, securing the individual messages is important, in case updates consist of relatively large amounts of data. In fact, checking individual received data piecemeal for tampering avoids that devices store large amounts of partially corrupted data and that they detect tampering hereof only after all data has been received. Devices receiving software and firmware updates are expected to possibly reply back, in order to provide a feedback about the execution of the update operation (e.g. OK, failure, error) and their current operational status.

- Parameter and configuration update: by means of multicast communication, it is possible to update the settings of a group of similar devices, both simultaneously and efficiently. Possible parameters are related, for instance, to network load management or network access controls. Devices receiving parameter and configuration updates are expected to possibly reply back, to provide a feedback about the execution of the update operation (e.g. OK, failure, error) and their current operational status.

- Commissioning of LLNs systems: a commissioning device is responsible for querying all devices in the local network or a selected subset of them, in order to discover their presence, and

be aware of their capabilities, default configuration, and operating conditions. Queried devices displaying similarities in their capabilities and features, or sharing a common physical location can be configured as members of a single multicast group. Queried devices are expected to reply back to the commissioning device, in order to notify their presence, and provide the requested information and their current operational status.

- o Emergency multicast: a particular emergency related information (e.g. natural disaster) is generated and multicast by an emergency notifier, and relayed to multiple devices. The latters may reply back to the emergency notifier, in order to provide their feedback and local information related to the ongoing emergency.

Authors' Addresses

Marco Tiloca
RISE SICS AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Goeran Selander
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

CORE Working Group
Internet Draft
Intended status: Experimental

P. Urien
Telecom ParisTech

December 22 2016

Expires: June 2017

Identity Modules for CoAP
draft-urien-core-identity-module-coap-01.txt

Abstract

This document defines identity modules based on Secure Elements processing DTLS/TLS stacks for CoAP devices. The expected benefits of these secure microcontrollers are the following :
- Secure storage of pre-share keys or private keys
- Trusted simple or mutual authentication between CoAP devices and CoAP clients.
- The device identity is enforced by a non cloneable chip.
- Trusted cryptographic support.
- Low power consumption for DTLS/TLS processing.

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.

Status of this Memo

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

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

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

This Internet-Draft will expire on June 2017.

.

Table of Contents

1 Overview

The CoAP [CoAP] protocol MAY be secured by the DTLS protocol [DTLS] over an UDP/IP stack; the TLS support [TLS] is also under definition [CoAP-TLS] over a TCP/IP stack.

According to [CoAP] four security modes are available, NoSec, PreSharedKey, RawPublicKey, and Certificate. When DTLS is used with the PreShareKey or Certificate modes there is a need to store secrets such as symmetric or asymmetric keys, which authenticate the CoAP device.

In that case a Secure Element (SE) MAY be used in order to fully run the DTLS or TLS protocol. According to the data throughput or other security considerations the DTLS/TLS session MAY be exported from the secure element after the exchange of the finished messages.

This class of Secure Element is referred by this draft as an identity module (IdMod).

The expected benefits of identity modules are the following :

- Secure storage of pre-share keys or private keys
- Trusted simple or mutual authentication between the CoAP device and the CoAP client.
- The device identity is enforced by a non cloneable identity module.
- Trusted cryptographic support.
- Low power consumption for DTLS/TLS processing.

2 What is a Secure Element

A Secure Element (SE) is a tamper resistant microcontroller (see figure 1) equipped with host interfaces such as [ISO7816], SPI (Serial Peripheral Interface) or I2C (Inter Integrated Circuit).

The typical area size of these electronic chips is about 5x5 mm2. They comprise CPU (8, 16, 32 bits), ROM (a few hundred KB), non volatile memory (EEPROM, FLASH, a few hundred KB) and RAM (a few ten KB). Security is enforced by multiple hardware and logical countermeasures.

According to the [EUROSMART] association height billion of such secure devices were shipped in 2013. Secure elements are widely deployed for electronic payment (EMV cards), telecommunication (SIM modules), identity (electronic passports), ticketing, and access control (PKCS15 cards).

Most of secure elements include a Java Virtual Machine (JVM) and therefore are able to execute embedded program written in the JAVACARD language. Because these devices are dedicated to security

purposes they support numerous cryptographic resources such as digest functions (MD5, SHA1, SHA2...), symmetric cipher (3xDES, AES) or asymmetric procedures (RSA, ECC).

A set of Global Platform [GP] standards control the lifecycle of embedded software, i.e. application downloading, activation and deletion.

As an illustration a typical low cost Secure Element has the following characteristics:

- JAVACARD operating system;
- Compliant with the GP (Global Platform) standards;
- 160 KB of ROM;
- 72 KB of EEPROM;
- 4KB of RAM;
- Embedded crypto-processor;
- 3xDES, AES, RSA, ECC;
- Certification according to Common Criteria (CC) EAL5+ level;
- Security Certificates from payment operators.

According to the state of art, TLS/DTLS stacks may run in secure elements, for example written as a javacard applications.

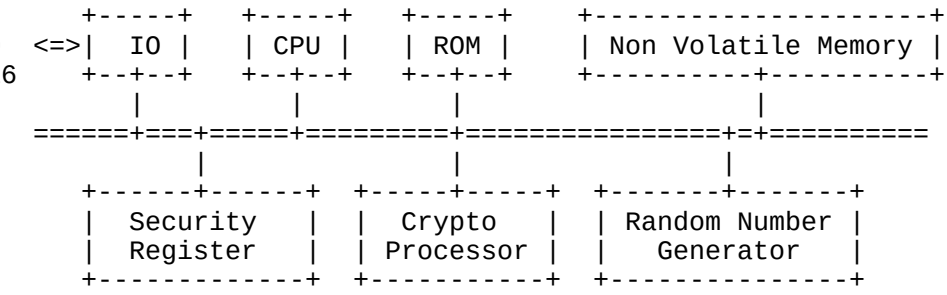


Figure 1. A typical hardware architecture of a Secure Element

3 Identity Module for CoAP

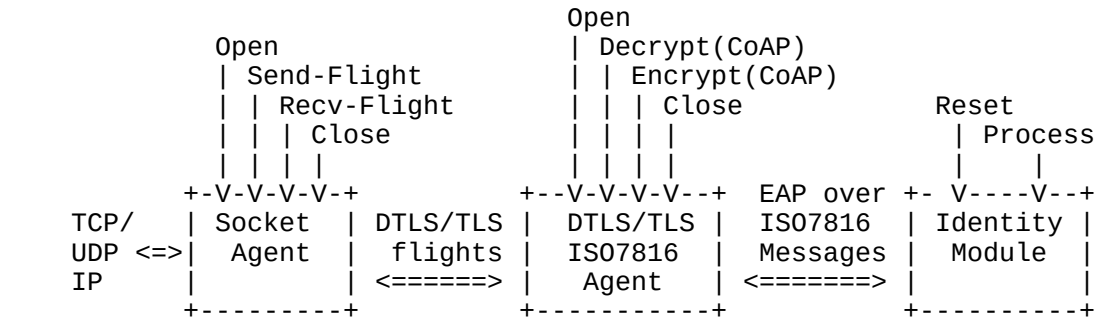


Figure 2. CoAP Identity module framework

ISO7816 interface for Secure Elements providing TLS/DTLS stacks are detailed in [DTLS/TLS-SM]. The Identity module MUST support two commands Reset and Process.

TLS/DTLS packets are transported by the EAP protocol over ISO7816 messages. This mechanism previously detailed by [EAPSC] provides a double segmentation procedure thanks to EAP and ISO7816 facilities.

A DTLS/TLS-ISO7816 software agent sends and receives DTLS/TLS flights to/from sockets over EAP/ISO7816 messages to/from the identity module. Conceptually this component interface SHOULD have four procedures Open, Close, Encrypt and Decrypt.

A socket software agent extracts and send DTLS/TLS flights from/to UDP/TCP packets. Conceptually this component interface SHOULD have four procedures Open, Close, Recv-Flight, Send-Flight.

4 DTLS/TLS profile for CoAP security modules

To be done.

5 IANA Considerations

This draft does not require any action from IANA.

6 References

6.1 Normative References

[TLS] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 5746, August 2008

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

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

[CoAP-TLS] A TCP and TLS Transport for the Constrained Application Protocol (CoAP), draft-ietf-core-coap-tcp-tls-02, April 2016.

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO).

6.2 Informative References

[GP] Global Platform Standards, http://www.globalplatform.org

[EUROSMART] The EUROSMART association, http://www.eurosmart.com

[DTLS/TLS-SM] Urien, P., "TLS and DTLS Security Modules", draft-urien-uta-tls-dtls-security-module-03.txt, December 2016

[EAPSC] Urien, P., "EAP Support in Smartcard", draft-urien-eap-smartcard-32.txt, December 2016

7 Authors' Addresses

Pascal Urien
Telecom ParisTech
23 avenue d'Italie
75013 Paris Phone: NA
France Email: Pascal.Urien@telecom-paristech.fr

CORE Working Group
Internet Draft
Intended status: Experimental

P. Urien
Telecom ParisTech

December 8 2016

Expires: June 2017

Remote APDU Call Secure (RACS)
draft-urien-core-racs-08.txt

Abstract

This document describes the Remote APDU Call Protocol Secure (RACS) protocol, dedicated to Grid of Secure Elements (GoSE). These servers host Secure Elements (SE), i.e. tamper resistant chips offering secure storage and cryptographic resources.

Secure Elements are microcontrollers whose chip area is about 25mm2; they deliver trusted computing services in constrained environments.

RACS supports commands for GoSE inventory and data exchange with secure elements. It is designed according to the representational State Transfer (REST) architecture. RACS resources are identified by dedicated URIs. An HTTP interface is also supported.

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.

Status of this Memo

.

Table of Contents

1 Overview

This document describes the Remote APDU Call Protocol Secure (RACS) protocol, dedicated to Grids of Secure Elements (GoSE). These servers host Secure Elements (SE), i.e. tamper resistant chips offering secure storage and cryptographic resources.

Secure Elements are microcontrollers whose chip area is about 25mm2; they deliver trusted computing services in constrained environments.

RACS supports commands for GoSE inventory and data exchange with secure elements.

RACS is designed according to the representational State Transfer (REST) architecture [REST], which encompasses the following features:
- Client-Server architecture.
- Stateless interaction.
- Cache operation on the client side.
- Uniform interface.
- Layered system.
- Code On Demand.

1.1 What is a Secure Element

A Secure Element (SE) is a tamper resistant microcontroller equipped with host interfaces such as [ISO7816], SPI (Serial Peripheral Interface) or I2C (Inter Integrated Circuit).

The typical area size of these electronic chips is about 25mm2. They comprise CPU (8, 16, 32 bits), ROM (a few hundred KB), nonvolatile memory (EEPROM, FLASH, a few hundred KB) and RAM (a few ten KB). Security is enforced by multiple hardware and logical countermeasures.

According to the [EUROSMART] association height billion of such secure devices were shipped in 2013. Secure elements are widely deployed for electronic payment (EMV cards), telecommunication (SIM modules), identity (electronic passports), ticketing, and access control.

Most of secure elements include a Java Virtual Machine and therefore are able to execute embedded program written in the JAVACARD language. Because these devices are dedicated to security purposes they support numerous cryptographic resources such as digest functions (MD5, SHA1, SHA2...), symmetric cipher (3xDES, AES) or asymmetric procedures (RSA, ECC).

A set of Global Platform [GP] standards control the lifecycle of embedded software, i.e. application downloading, activation and deletion.

As an illustration a typical Secure Element has the following characteristics:

- JAVACARD operating system;
- Compliant with the GP (Global Platform) standards;
- 160 KB of ROM;
- 72 KB of EEPROM;
- 4KB of RAM;
- Embedded crypto-processor;
- 3xDES, AES, RSA, ECC;
- Certification according to Common Criteria (CC) EAL5+ level;
- Security Certificates from payment operators.

1.2 Grid Of Secure Elements (GoSE)

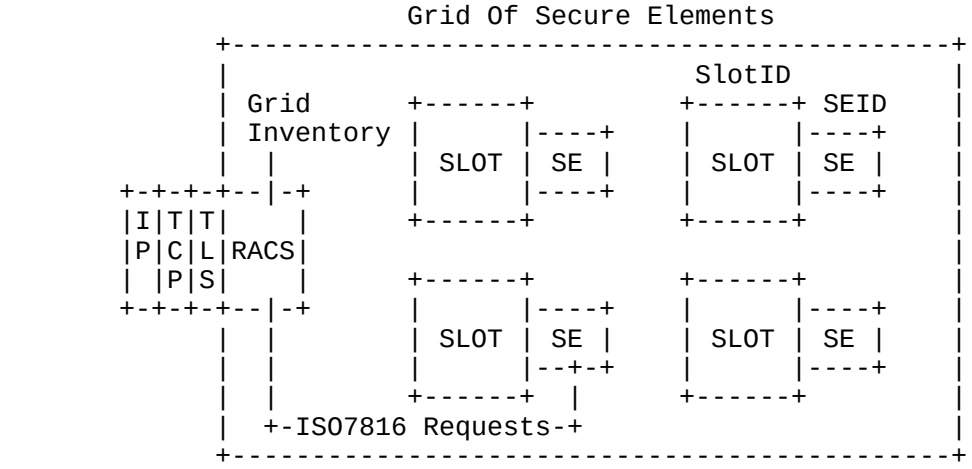


Figure 1. Architecture of a Grid of Secure Elements

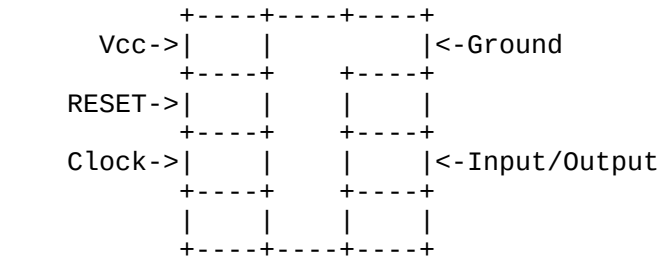


Figure 2. Illustration of an ISO7816 Secure Element

A grid of Secure Elements (GoSE) is a server hosting a set of secure elements.

The goal of these platforms is to deliver trusted services over the Internet. These services are available in two functional planes,
- The user plane, which provides trusted computing and secure storage.
- The management plane, which manages the lifecycle (downloading, activation, deletion) of applications hosted by the Secure Element.

A grid of Secure Elements offers services similar to HSM (Hardware Secure Module), but may be managed by a plurality of administrators, dealing with specific secure microcontrollers.

According to this draft all accesses to a GoSE require the TCP transport and are secured by the TLS [TLS 1.0] [TLS 1.1] [TLS 2.0] protocol.

The RACS protocol provides all the features needed for the remote use of secure elements, i.e.
- Inventory of secure elements
- Information exchange with the secure elements

1.3 Secure Element Identifier (SEID)

Every secure element needs a physical slot that provides electrical feeding and communication resources. This electrical interface is for example realized by a socket soldered on an electronic board, or a CAD (Card Acceptance Device, i.e. a reader) supporting host buses such as USB.

Within the GoSE each slot is identified by a SlotID (slot identifier) attribute, which may be a socket number or a CAD name.

The SEID (Secure Element IDentifier) is a unique identifier indicating that a given SE is hosted by a GoSE. It also implicitly refers the physical slot (SlotID) to which the SE is plugged.

The GoSE manages an internal table that establishes the relationship between SlotIDs and SEIDs.

Therefore three parameters are needed for remote communication with secure element, the IP address of the GoSE, the associated TCP port, and the SEID.

1.3.1 SlotID example

According to the PC/SC (Personal Computer/Smart Card) standard [PS/SC], a smart card reader MAY include a serial number. This attribute (VENDOR-IFD-SERIAL) is associated to the tag 0x0103 in the class VENDOR-INFO.

1.3.2 SEID for Secure Elements

According to the Global Platform standard [GP] the Issuer Security Domain (ISD) manages applications lifecycle (downloading, activation, deletion). The command 'initialize update' is used to start a mutual authentication between the administration entity and the secure element; it collects a set of data whose first ten bytes are called the 'key diversification data'. This information is used to compute symmetric keys, and according for example to [EMV] MAY comprise a serial number.

1.4 APDUs

According to the [ISO7816] standards secure element process ISO7816 request messages and return ISO7816 response messages, named APDUs (application protocol data unit).

1.4.1 ISO7816 APDU request

An APDU request comprises two parts: a header and an optional body.

The header is a set of four or five bytes noted CLA INS P1 P2 P3

- CLA indicates the class of the request, and is usually bound to standardization committee (00 for example means ISO request).
-INS indicates the type of request, for example B0 for reading or D0 for writing.
- P1 P2 gives additional information for the request (such index in a file or identifier of cryptographic procedures)
- P3 indicates the length of the request body (from P3=01 to P3=FF), or the size of the expected response body (a null value meaning 256 bytes). Short ISO7816 requests may comprise only 4 bytes
- The body may be empty. Its maximum size is 255 bytes

1.4.2 ISO7816 APDU response

An APDU response comprises two parts an optional body and a mandatory status word.

- The optional body is made of 256 bytes at the most.

- The response ends by a two byte status noted SW. SW1 refers the most significant byte and SW2 the less significant byte.

An error free operation is usually associated to the 9000 status word. Following are some interpretations of the tuple SW1, SW2 according to various standards:

- ’61’ ’xx’, indicates that xx bytes (modulus 256) are ready for reading. Operation result MUST be fetched by the ISO Get Response APDU (CLA=00, INS=C0, P1=P2=00, P3=XX)
- ’9F’ ’xx’, indicates that xx bytes (modulus 256) are ready for reading. Operation result MUST be fetched by the ISO Get Response APDU (CLA=00, INS=C0, P1=P2=00, P3=XX)
- ’6C’ ’XX’, the P3 value is wrong, request must be performed again with the LE parameter value sets to ’XX’
- ’6E’ ’XX’, wrong instruction class (CLA) given in the request
- ’6D’ ’XX’, unknown instruction code (INS) given in the request
- ’6B’ ’XX’, incorrect parameter P1 or P2
- ’67’ ’XX’, incorrect parameter P3
- ’6F’ ’XX’, technical problem, not implemented...

2 The RACS protocol

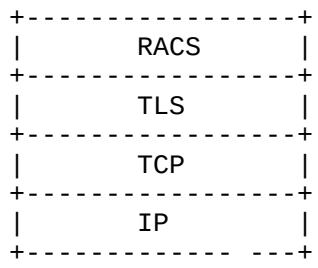


Figure 2. The RACS stack

The RACS protocol works over the TCP transport layer and is secured by the TLS protocol. The TLS client (i.e. the RACS client) MUST be authenticated by a certificate.

One of the main targets of the RACS protocol is to efficiently push a set of ISO7816 requests towards a secure element in order to perform cryptographic operations in the user’s plane. In that case a RACS request typically comprises a prefix made with multiple ISO7816 requests and a suffix that collects the result of a cryptographic procedure.

The mandatory use of TLS with mutual authentication based on certificate provides a simple and elegant way to establish the credentials of a RACS client over the GoSE. It also enables an easy splitting between users’ and administrators’ privileges.

2.1 Structure of RACS request

A RACS request is a set of command lines, encoded according to the ASCII format. Each line ends by the Cr (carriage return) and line feed (Lf) characters. The RACS protocol is case sensitive.

Each command is a set of tokens (i.e. words) separated by space (0x20) character(s).

The first token of each line is the command to be executed.

A command line MAY comprise other tokens, which are called the command parameters.

A RACS request MUST start by a BEGIN command and MUST end by an END command.

Each command line is associated to an implicit line number. The BEGIN line is associated to the zero line number.

The processing of a RACS request is stopped after the first error. In that case the returned response contained the error status induced by the last executed command.

2.2 Structure of a RACS response

A RACS response is a set of lines, encoded according to the ASCII format. Each line ends by the Cr (carriage return) and line feed (Lf) characters. The RACS protocol is case sensitive.

Each line is a set of tokens (i.e. words) separated by space (0x20) character(s).

The first token of each line is the header.

The second token of response each line is associated command line number

A response line MAY comprise other tokens, which are called the response parameters.

Three classes of headers are defined BEGIN, END and Status.

A RACS response MUST start by a BEGIN header and MUST end by an END header. It comprises one or several status lines.

2.2.1 BEGIN Header

This header starts a response message.

It comprises an optional parameter, an identifier associated to a previous request message.

2.2.2 END Header

This header ends a response message.

2.2.3 Status line

A status header indicates a status line.

It begins by the character '+' in case of success or '-' if an error occurred during the RACS request execution. It is followed by an ASCII encoded integer, which is the value of the status.

The second mandatory token of a status line is the command line number (starting from zero)

A status line MAY comprise other tokens, which are called the response parameters.

2.2.4 Examples of RACS responses:

```
BEGIN CrLf
+001 000 Success CrLf
END CrLf

BEGIN moon1969 CrLf
-301 007 Illegal command, BEGIN condition not satisfied at line 7
END CrLf

BEGIN Asterix237 CrLf
+006 001 [ISO7816-Response] CrLf
END CrLf

BEGIN CrLf
-100 002 Unknown command at line 2 CrLf
END CrLf

BEGIN CrLf
-606 001 Unauthorized command APDU command at line 1
END CrLf

BEGIN CrLf
-706 001 SEID Already in use, APDU command at line 1
END CrLf
```

2.3 RACS request commands

2.3.1 BEGIN

This command starts a request message. A response message is returned if an error is detected.

An optional parameter is the request identifier, which MUST be echoed in the parameter of the first response line (i.e. starting by the BEGIN header).

2.3.2 END

This command ends a request message. It returns the response message triggered by the last command.

```
Example1
========
Request:
BEGIN CrLf
END CrLf

Response:
BEGIN CrLf
+001 000 Success CrLf
END CrLF

Example2
========
Request:
BEGIN Marignan1515 CrLf
APDU ASTERIX-CRYPTO-MODULE [ISO7816-Request] CrLf
END CrLf

Response:
BEGIN Marignan1515 CrLf
+006 001 [ISO7816-Response] CrLf
END CrLf
```

2.3.3 The APPEND parameter

The APPEND parameter MAY be used in all command lines, excepted BEGIN and END. The APPEND parameter MUST be the last parameter of a command line.
By default a response message returns only the last status line.
When APPEND is inserted, the command line, if executed, MUST produce a status line.

```
Example

Request:
BEGIN SanchoPanza CrLf
APDU 100 [ISO7816-Request-1] CrLf
APDU 100 [ISO7816-Request-2] CrLf
END CrLf

Response:
BEGIN SanchoPanza CrLf
+006 002 [ISO7816-Response-2] CrLf
END CrLf

Request:
BEGIN DonQuichotte CrLf
APDU 100 [ISO7816-Request-1] APPEND CrLf
APDU 100 [ISO7816-Request-2] APPEND CrLf
END CrLf
```

```
Response:
BEGIN DonQuichotte CrLf
+006 001 [ISO7816-Response-1] CrLf
+006 002 [ISO7816-Response-2] CrLf
END CrLf
```

2.3.4 GET-VERSION

This command requests the current version of the RACS protocol. The returned response is the current version encoded by two integer separated by the '.' character. The first integer indicates the major version and the second integer gives the minor version.

This draft version is 0.2

```
Example
=======
Request:
BEGIN CrLf
GET-VERSION CrLf
END CrLf

Response:
BEGIN CrLf
+002 001 1.0 CrLf
END CrLf
```

2.3.5 SET-VERSION

This command sets the version to be used for the RACS request. An error status is returned by the response if an error occurred.

```
Example 1
=========
Request:
BEGIN CrLf
SET-VERSION 2.0 CrLf
END CrLf

Response:
BEGIN CrLf
-403 001 Error line 1 RACS 2.0 is not supported CrLf
END CrLf

Example 2
=========
Request:
BEGIN CrLf
SET-VERSION 1.0 CrLf
END CrLf
```

Response:
BEGIN CrLf
+003 001 RACS 1.0 has been activated CrLf
END CrLf

2.3.6 LIST

This command requests the list of SEID plugged in the GoSE.

It returns a list of SEIDs separated by space (0x20) character(s).

Some SEID attributes MAY be built from a prefix and an integer suffix (such as SE#100 in which SE# is the suffix and 100 is the integer suffix. A list of non-consecutive SEID MAY be encoded as prefix[i1;i2;..;ip] where i1,i2,ip indicates the integer suffix. A list of consecutive SEID could be encoded as prefix[i1-ip] where i1,i2,ip indicates the integer suffix.

Example 1
=========
Request:
BEGIN CrLf
LIST CrLf
END CrLf

Response:
BEGIN CrLf
+004 001 SEID1 SEID2 CR LF
END CrLf

Example 2
=========
Request:
BEGIN CrLf
LIST CrLf
END CrLf

Response:
BEGIN CrLf
+004 001 Device[1000-2000] SerialNumber[567;789;243] CrLf
END CrLf

2.3.7 RESET

This command resets a secure element. The first parameter gives the secure element identifier (SEID). An optional second parameter specifies a warm reset. The default behavior is a cold reset. The response status indicates the success or the failure of this operation.

Syntax: RESET SEID [WARM] CrLf

Example 1
=========
Request:
BEGIN CrLf
RESET device#45 CrLf
END CrLf

Response:
BEGIN CrLf
+005 001 device#45 Reset Done
END CrLf

Example 2
=========
Request:
BEGIN CrLf
RESET device#45 CrLf
END CrLf

Response:
BEGIN CrLf
-705 001 error device#45 is already in use
END CrLf

Example 3
=========
Request:
BEGIN CrLf
RESET device#45 WARM CrLf
END CrLf

Response:
BEGIN CrLf
+005 001 device#45 Warm Reset Done CrLf
END CrLf

2.3.8 APDU

This command sends an ISO7816 request to a secure element or a set of ISO7816 commands.

The first parameter specifies the SEID.
The second parameter is an ISO7816 request.
Three optional parameters are available; they MUST be located after the second parameter.

- CONTINUE=value, indicates that the next RACS command will be executed only if the ISO7816 status word (SW) is equal to a given value. Otherwise an error status is returned.
- MORE=value, indicates that a FETCH request will be performed (i.e. a new ISO7816 request will be sent) if the first byte of the ISO7816 status word (SW1) is equal to a given value.
- FETCH=value fixes the four bytes of the ISO7816 FETCH request (i.e. CLA INS P1 P2). The default value (when FETCH is omitted) is 00C00000 (CLA=00, INS=C0, P1=00, P2=00)

When the options CONTINUE and MORE are simultaneously set the SW1 byte is first checked. If there is no match then the SW word is afterwards checked.

The ISO7816 6Cxx status MUST be autonomously processed by the GoSE.

SYNTAX
APDU SEID ISO7816-REQUEST [CONTINUE=SW] [MORE=SW1] [FETCH=CMD] CrLf

The returned response is the ISO7816 response. If multiple ISO7816 requests are executed (due to the MORE option), the bodies are concatenated in the response, which ends by the last ISO7816 status word.

The pseudo code of the APDU command is the following :

```
1. BODY = empty;
2. SW   = empty;
3. DoIt = true;
3. Do
4. { iso7816-response = send(iso7816-request);
5.   body || sw1 || sw2 = iso7816-response;
6.   If ( (first request) && (iso7816-request.size==5) &&
          (body==empty) && (sw1==6C) )
8.   { iso7816-request.P3 = sw2 ; }
6.   Else
7.   { SW = sw1 || sw2
8.     BODY = BODY || body;
9.     If (sw1 == MORE)
10.    { iso7816-request = FETCH || sw2 ; }
11.    Else
12.    { DoIt=false;}
13.   }
14. }
15. While (DoIt == true)

16. iso7816-response = BODY || SW ;
17. If (SW != CONTINUE) Error   ;
18. Else                No Error;
```

```
Example 1
=========
Request:
BEGIN CrLf
APDU SEID ISO7816-REQUEST CrLf
END CrLf

Response:
BEGIN CrLf
+006 001 ISO7816-RESPONSE CrLf
END CrLf


Example 2
=========
Request:
BEGIN CrLf
APDU SEID ISO7816-REQUEST CrLf
END CrLf

Response:
BEGIN CrLf
-706 001 error SEID is already used CrLf
END CrLf


Example 3
=========
Request:
BEGIN CrLf
APDU SEID ISO7816-REQUEST CrLf
END CrLf

Response:
BEGIN CrLf
-606 001 error access unauthorized access CrLf
END CrLf


Example 4
=========
BEGIN CrLf
APDU SEID ISO7816-REQUEST-1 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQUEST-2 CrLf
END CrLf

Response:
BEGIN CrLf
+006 002 ISO7816-RESPONSE-2 CrLf
END CrLf
```

Example 5
=========
BEGIN CrLf
APDU SEID ISO7816-REQUEST-1 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQUEST-2 CrLf
END CrLf

Response:
BEGIN CrLf
-006 001 Request Error line 1 wrong SW CrLf
END CrLf

Example 6
=========
BEGIN CrLf
APDU SEID ISO7816-REQ-1 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQ-2 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQ-3 CONTINUE=9000 MORE=61 FETCH=00C00000 CrLf
END CrLf

Response:
BEGIN CrLf
+006 003 ISO7816-RESP-3 CrLf
END CrLf

Multiple ISO7816 requests have been performed by the third APDU command according to the following scenario :
- the ISO7816-REQ-3 request has been forwarded to the secure element (SEID)
- the ISO 7816 response comprises a body (body-0) and a status word (SW-0) whose first byte is 0x61, and the second byte is SW2-0
- the FETCH command CLA=00, INS=00, P1=00, P2=00, P3=SW2-0 is sent to the secure element
- the ISO 7816 response comprises a body (body-1) and a status word (SW-1) set to 9000

The RACS response is set to
+006 003 body-0 || body-1 || SW-1 CrLf
where ||indicates a concatenation operation.

2.3.9 SHUTDOWN

This command powers down a secure element. The first parameter gives the secure element identifier (SEID).

Syntax: SHUTDOWN SEID CrLf

Example
=========
Request:
BEGIN Goodbye CrLf
SHUTDOWN device#45 CrLf
END CrLf

Response:
BEGIN Goodbye CrLf
+007 001 device#45 has been powered down CrLf
END CrLf

2.3.10 POWERON

This command powers up a secure element. The first parameter gives the secure element identifier (SEID).

Syntax: POWERON SEID CrLf

Example 1
=========
Request:
BEGIN CrLf
POWERON device#45 CrLf
END CrLf

Response:
BEGIN CrLf
+008 001 device#45 Has been powered up CrLf
END CrLf

Example 2
=========
Request:
BEGIN CrLf
POWERON device#45 CrLf
END CrLf

Response:
BEGIN CrLf
-708 001 error device#45 is already in use CrLf
END CrLf

Example 3
=========
Request:
BEGIN CrLf
POWERON device#45 CrLf
END CrLf

Response:
BEGIN CrLf
-608 001 error unauthorized access CrLf
END CrLf

2.3.11 ECHO

This command echoes a token. The first parameter is the token (word) to be echoed by the response.

Syntax: ECHO SEID CrLf

Example 1
=========
Request:
BEGIN TestEcho CrLf
ECHO Hello CrLf
END CrLf

Response:
BEGIN TestEcho CrLf
+009 001 Hello CrLf
END CrLf

Example 2
=========
Request:
BEGIN ResetSEID CrLf
POWERON device#45 CrLf
ECHO Done CrLf
END CrLf

Response:
BEGIN ResetSEID CrLf
+009 001 Done CrLf
END CrLf

2.4 Status header encoding

The first token of a response line is the status header. It begins by a '+' or a '-' character, and comprises three decimal digits (xyz).

The first digit (x) MUST indicates an event class.
The second and third digits (yz) MAY indicate a command class.

2.4.1 Event class

This draft only defines the meaning of the first digit located at the left most side.

+0yz: No error
-0yz: Command execution error
-1yz: Unknown command, the command is not defined by this draft
-2yz: Not implemented command
-3yz: Illegal command, the command can’t be executed
-4yz: Not supported parameter or parameter illegal value
-5yz: Parameter syntax error or parameter missing
-6yz: Unauthorized command
-7yz: Already in use, a session with this SE is already opened
-8yz: Hardware error
-9yz: System error

2.4.2 Command class

The second and third digits (yz) MAY indicates the command that trigged the current line status

01 BEGIN
02 GET-VERSION
03 SET-VERSION
04 LIST
05 RESET
06 APDU
07 SHUTDOWN
08 POWERON
09 ECHO

3 URI for the GoSE

The URI addressing the resources hosted by the GoSE is represented by the string:

RACS://GoSE-Name:port/?request

where request is the RACS request to be forwarded to a the GoSE.

RACS command lines are encoded in a way similar to the INPUT field of an HTML form. Each command is associated to an INPUT name, the remaining of the command line i.e. a set of ASCII characters, is written according to the URL encoding rules. End of line characters, i.e. carriage return (Cr) and line feed (Lf) are omitted.

As a consequence a request is written to the following syntax cmd1=cmd1-parameters&cmd2=cmd2-parameters

Example:
RACS://GoSE-Name:port/?BEGIN=&APDU=SEID%20[ISO7816-REQUEST]&END=

4 HTTP interface

A GoSE SHOULD support an HTTP interface. RACS requests/responses are transported by HTTP messages. The use of TLS is mandatory.

4.1 HTTPS Request

https://GoSE-Name:port/RACS?request

where request is the RACS request to be forwarded to a secure element (SEID)

The RACS request is associated to an HTML form whose name is "RACS". The request command lines are encoded as the INPUT field of an HTML form. Each command is associated to an INPUT name, the remaining of the command line i.e. a set of ASCII characters is written according to the URL encoding rules. End of line characters, i.e. carriage return (Cr) and line feed (Lf) are omitted.

As a consequence a RACS request is written as https://GoSE-Name/RACS?cmd1=cmd1-parameters&cmd2=cmd2-parameters

Example:

https://GoSE-Name/RACS?BEGIN=&APDU=SEID%20[ISO7816-REQUEST]&END=

4.2 HTTPS response

The RACS response is returned in an XML document.

The root element of the document is <RACS-Response>

The optional parameter of the BEGIN header, is the content of the <begin> element.

Each status line is the content of the <Cmd-Response> element, which includes the following information :

- The status header is the content of the <status> element.

- The line number is the content of the <line> element.

- The other parameters of the status line are the content of the <parameters> element.

The END header is associated to the element <end>

End of line, i.e. carriage return (Cr) and line feed (Lf) characters are omitted.

As a consequence a RACS response is written as :

```
<RACS-Response>
<begin>Optionnal-ID</begin>
<Cmd-Response
<status>+000</status>
<line>001</line>
<parameters>other parameters of the RACS response</parameters>
</Cmd-Response>
<end></end>
</RACS-Response>
```

5 Security Considerations

5.1 Authorization

A RACS client MUST be authenticated by an X509 certificate.

The GoSE software MUST provide a mean to establish a list of SEIDs that can be accessed from a client whose identity is the CommonName (CN) attribute of its certificate. It MAY allocate a UserID (UID), i.e. an integer index from the certficate common name.

5.2 Secure Element access

The GoSE MUST manage a unique session identifier (SID) for each TLS session. The SID is bound to the client’s certificate CommonName (SID(CN))

A secure element has two states, unlocked and locked. In the locked state the secure element may be only used by the SID that previously locked it.

The first authorized command that successfully accesses to a SEID (either POWERON ,RESET, APDU) locks a secure element (SEID) with the current session (SID).

The SHUTDOWN command MUST unlock a secure element (SEID).

The end of a TLS session MUST unlock all the secure elements locked by the session.

5.3 Applications security policy

According to the [ISO7816] standards each Application embedded within a secure element (associated to a SEID) is identified by an AID parameter (16 bytes at the most)

The RACS server SHOULD support the following facilities

5.3.1 Users-Table

Each CN (the Users-Table primary key) is associated to a list of SEIDs whose access is authorized.

5.3.2 SEID-Table

Each AID (the SEID-Table primary key) is associated to a list of CNs whose access is authorized.

5.3.3 APDU-Table

For a given AID and an authorized CN, an APDU-Table MAY be available. This table acts as a firewall, which defined a set of forbidden ISO7816 commands.

For example this filter could be expressed as a set of the four first bytes of an APDU-Prefix (CLA INS P1 P2) and a four bytes Mask An ISO7816-Request is firewall if:

ISO7816-Request AND Mask IsEQUAL to APDU-Prefix

5.4 Overview of the security policy

The summary of the security policy is illustrated by the figure 3.

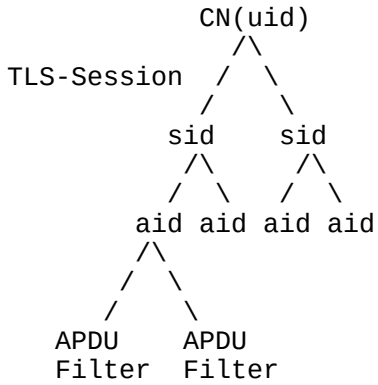


Figure 3. Summary of the security policy

6 IANA Considerations

This draft does not require any action from IANA.

7 References

7.1 Normative References

[TLS 1.0] Dierks, T., C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999

[TLS 1.1] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006

[TLS 1.2] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 5746, August 2008

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO)

7.2 Informative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[GP] Global Platform Standards, http://www.globalplatform.org

[EUROSMART] The EUROSMART association, http://www.eurosmart.com

[PC/SC] The PC/SC workgroup, http://www.pcscworkgroup.com

[EMV] EMV Card Personalization Specification, Version 1.1, July 2007

8 Authors' Addresses

Pascal Urien
Telecom ParisTech
23 avenue d'Italie
75013 Paris Phone: NA
France Email: Pascal.Urien@telecom-paristech.fr

anima
Internet-Draft
Intended status: Standards Track
Expires: May 2, 2017

S. Kumar
Philips Lighting Research
P. van der Stok
Consultant

October 29, 2016

EST based on DTLS secured CoAP (EST-coaps)
draft-vanderstok-core-coap-est-00

Abstract

Low-resource devices in a Low-power and Lossy Network (LLN) can operate in a mesh network using the IPv6 over Low-power Personal Area Networks (6LoWPAN) and IEEE 802.15.4 link-layer standards. Provisioning these devices in a secure manner with keys (often called security bootstrapping) used to encrypt and authenticate messages is the subject of Bootstrapping of Remote Secure Key Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra]. Enrollment over Secure Transport (EST) [RFC7030], based on TLS and HTTP, is used for BRSKI. This document defines how low-resource devices are expected to use EST over DTLS and CoAP. 6LoWPAN fragmentation management and minor extensions to CoAP are needed to enable EST over DTLS-secured CoAP (EST-coaps).

Note

Many of the concepts in this document are taken over from [RFC7030]. Consequently, much text is directly traceable to [RFC7030]. The same document structure is followed to point out the differences and commonalities between EST and EST-coaps.

Status of This Memo

Table of Contents

1. Introduction

IPv6 over Low-power Wireless Personal Area Networks (6LoWPANs) [RFC4944] on IEEE 802.15.4 [ieee802.15.4] wireless networks is becoming common in many professional application domains such as lighting controls. However commissioning of such networks suffers from a lack of standardized secure bootstrapping mechanisms for these networks.

Although IEEE 802.15.4 defines how security can be enabled between nodes within a single mesh network, it does not specify the provisioning and management of the keys. Therefore securing a 6LoWPAN network with devices from multiple manufacturers with

different provisioning techniques is often tedious and time consuming.

Bootstrapping of Remote Secure Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra] addresses the issue of bootstrapping networked devices in the context of Autonomic Networking Integrated Model and Approach (ANIMA). However, BRSKI has not been developed specifically for low-resource devices in constrained networks. These networks use DTLS [RFC6347], CoAP [RFC7252], and UDP instead of TLS [RFC5246], HTTP [RFC7230] and TCP. BRSKI relies on Enrollment over Secure Transport (EST) [RFC7030] for the provisioning of the operational domain certificates. Replacing the EST invocations of TLS and HTTP by DTLS and CoAP invocations enables applying BRSKI on CoAP-based low-resource devices.

The Figure 1 below shows the EST-coaps architecture.

Figure 1: EST-coaps protocol layers

Although EST-coaps paves the way for the utilization of BRSKI for constrained devices on constrained networks, some devices will not have enough resources to handle the large payloads that come with EST-coaps. It is up to the network designer to decide which devices execute the BRSKI protocol and which not.

EST-coaps is designed for use in professional control networks such as lighting. The autonomic bootstrapping is interesting because it reduces the manual intervention during the commissioning of the

network. Typing in passwords is contrary to this wish. Therefore, the password authentication of EST is not supported in EST-coaps.

In the constrained devices context it is very unlikely that full PKI request messages will be used. For that reason, full PKI messages are not supported in EST-coaps.

Because the relatively large messages involved in EST cannot be readily transported over constrained (6LoWPAN, LLN) wireless networks, this document defines the use of CoAP Block-Wise Transfer ("Block") [RFC7959] combined with DTLS to fragment EST messages at the application layer.

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].

All the terminology from EST [RFC7030] is included in this document by reference.

2. Operational Scenarios Overview

Only the differences to EST with respect to operational scenarios are described in this section. EST-coaps server authentication differs from EST as follows:

- Replacement of TLS by DTLS and HTTP by CoAP, resulting in:
  - DTLS-secured CoAP sessions between EST-coaps client and EST-coaps server.
- Only certificate-based client authentication is supported, with as result:
  - The EST-coaps client does not support manual authentication (as described in Section 4.4.1 of [RFC7030])
  - The EST-coaps client does not support authentication at the application layer.
- EST-coaps does not support full PKI request messages [RFC5272].

The following EST-coaps protocol parts are supported as described for the equivalent EST parts:

1. Request of client certificates by submitting a enrollment request to EST-coaps server.

2. Renewal of existing client certificates by submitting a re-enrollment request to EST-coaps server.

3. Request of certificate with key pair generated by EST-coaps server.

4. The EST-coaps client can request the attributes needed for enrollment before the enrollment request is issued"

3. Protocol Design and Layering

The EST-coaps protocol design follows closely the EST design, excluding some aspects that are not relevant for automatic bootstrapping of constrained devices within a professional context. The parts supported by EST-coaps are:

Message types:

* Simple PKI messages.

* CA certificate retrieval.

* CSR Attributes Request.

* Server-generated key request.

CoAP with Block-Wise Transfer:

* CoAP Block-Wise Transfer header Options for control of the transfer of larger EST messages.

DTLS for transport security:

* Authentication of the EST-coaps server.

* Authentication of the EST-coaps client.

* Communication integrity and confidentiality.

* Channel-binding information for linking proof-of-identity with message-based proof-of-possession (OPTIONAL).

Given that CoAP and DTLS can provide proof of identity for EST-coaps clients and server, simple PKI messages can be used conformant to section 3.1 of [RFC5272]. EST-coaps supports the certificate types

and Trust Anchors (TA) that are specified for EST in section 3 of [RFC7030].

The EST-coaps server URI is identical to the EST URI (except for replacing the scheme https by coaps):

coaps://www.example.com/.well-known/est
coaps://www.example.com/.well-known/est/arbitraryLabel1

See Figure 5 in section 3.2.2 of [RFC7030] for the path-suffixes (operations) that are supported by EST.

EST-coaps uses CoAP to transfer EST messages, aided by Block-Wise Transfer [RFC7959] to transport CoAP messages in blocks thus avoiding (excessive) 6LoWPAN fragmentation of UDP datagrams. The use of "Block" is specified in Section 3.2.

The content-format (media type equivalent) of the CoAP message determines which EST message is transported in the CoAP payload. The media types specified in the HTTP Content-Type header(see section 3.2.2 of [RFC7030]) are in EST-coaps specified by the Content-Format Option (12) of CoAP. The combination of URI path-suffix and content-format used MUST map to an allowed combination of path-suffix and media type as defined for EST.

EST-coaps is designed for use between low-resource devices using CoAP and hence does not need to send base64-encoded data. Simple binary coding is more efficient (30% less payload compared to base64) and well supported by CoAP. Therefore, the content formats specification in Section 5 requires the use of binary encoding for all EST-coaps CoAP payloads.

The functions of TLS specified for EST are in EST-coaps mapped to the equivalent DTLS functions. However, DTLS sessions SHOULD remain open for persistent EST-coaps connections to reduce storage load. For example, a cacerts request followed by an enrollments request SHOULD use the same DTLS session.

The mandatory cipher suite for DTLS is TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 defined in [RFC7251] which is the mandatory-to-implement cipher suite in CoAP. Additionally the curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. DTLS implementations MUST use the Supported Elliptic Curves and Supported Point Formats Extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090] can be used as an implementation method.

3.1. CoAP response codes

Section 5.9 of [RFC7252] specifies the mapping of HTTP response codes to CoAP response codes. Every time the HTTP response code 200 is specified in [RFC7030] in response to a GET request, in EST-coaps the equivalent CoAP response code 2.05 MUST be used. Response code HTTP 202 in EST is mapped as indicated below; while other HTTP 2xx response codes are not used by EST. For the following HTTP 4xx error codes that may occur: 400, 401, 403, 404, 405, 406, 412, 413, 415 ; the equivalent CoAP response code for EST-coaps is 4.xx. For the HTTP 5xx error codes: 500, 501, 502, 503, 504 the equivalent CoAP response code is 5.xx.

HTTP response code 202 needs a different treatment from the one described for [RFC7030]. A new CoAP response code 2.06 is needed. When the EST over CoAP request cannot be treated immediately, a CoAP response code 2.06 Delayed is returned with Content-Format: application/link-format described in [RFC6690]. The payload of the response contains a link to receive the delayed response. ALTERNATIVE (to discuss) : a 2.06 Delayed response without payload and the link to receive the delayed response indicated using the Location-Path and Location-Query Options.

The waiting client may send GET requests to the returned link. When the response is not available, the server returns response code 2.06 with again the link for the client to query. When the response is available, the server returns the response code 2.05 Content with a payload containing the requested response in the appropriate content format.

3.2. Message fragmentation using Block

DTLS defines fragmentation only for the handshake part and not for secure data exchange (DTLS records). [RFC6347] states "Each DTLS record MUST fit within a single datagram". In order to avoid using IP fragmentation, which is not supported by 6LoWPAN, invokers of the DTLS record layer MUST size DTLS records so that they fit within any Path MTU estimates obtained from the record layer. In addition, invokers residing on a 6LoWPAN over IEEE 802.15.4 network SHOULD attempt to size CoAP messages such that each DTLS record will fit within one or two IEEE 802.15.4 frames only by choosing the appropriate block sizes.

Certificates can vary greatly in size dependent on signature algorithms and key sizes. For a 256-bit curve, common ECDSA sizes fluctuate between 500 bytes and 1 KB. Some EST messages may be several kilobytes in size. Given non-existence of IP fragmentation in 6LoWPAN networks and its 1280 bytes MTU, EST-coaps needs to be

able to fragment EST messages into multiple DTLS datagrams with each DTLS datagram containing a block of CoAP payload data. Further considering the small payload size available to a CoAP message, which can be as low as 68 bytes in case the message needs to fit into a single IEEE 802.15.4 frame, fine-grained fragmentation of EST messages is essential.

For CoAP, [RFC7959] specifies the "Block1" option for fragmentation of the request payload and the "Block2" option for fragmentation of the return payload. The CoAP client MAY specify the Block1 size and MAY also specify the Block2 size. The CoAP server MAY specify the Block2 size, but not the Block1 size.

Examples of fragmented messages are shown in Appendix A.

3.3. CoAP message headers

EST-coaps uses CoAP payload blocks that each fit in a single DTLS record i.e. UDP datagram without causing IP fragmentation. The returned CoAP response codes are specified in Section 3.1. The CoAP Token value is not specified by EST-coaps and may be chosen by the CoAP client according to [RFC7252].

An example HTTP request message cacerts in EST will look like:

```
REQ:
        GET /.well-known/est/cacerts HTTP/1.1
           Host: 192.0.2.1:8085
           Accept: */*

RES:
        HTTP/1.1 200 OK
        Status: 200 OK
        Content-Type: application/pkcs7-mime
        Content-Transfer-Encoding : base64
        Content-Length: 4246
        payload
```

The corresponding EST-coaps request looks like:

```
REQ:
        GET coaps://[192.0.2.1:8085]/.well-known/est/cacerts

RES:
        2.05 Content (Content-Format: application/pkcs7-mime)
        {payload}
```

4. Protocol Exchange Details

The EST-coaps client MUST be configured with an implicit TA database or an explicit TA database. The authentication of the EST-coaps server by the EST-coaps client is based on Certificate authentication in the DTLS handshake.

The authentication of the EST-coaps client is based on client certificate in the DTLS handshake. This can either be

- DTLS with a previously issued client certificate (e.g., an existing certificate issued by the EST CA);
- DTLS with a previously installed certificate (e.g., manufacturer-installed certificate or a certificate issued by some other party);

The details on checking the validity of the certificates are identical to EST.

The other protocol aspects such as simple enrollment (re-enrollment), certificate attributes and CA certificate request are similar to EST with the exception that these are performed on coaps (CoAP+DTLS) as the transport. The required content-formats for these request and response messages are defined in Section 5. The CoAP response codes are defined in Section 3.1.

EST-coaps does not support full PKI Requests. Consequently, the fullcmc request of section 4.3 of [RFC7030] and response MUST NOT be supported by EST-coaps.

5. IANA Considerations

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are needed for the below media types. These can be registered either in the Expert Review range (0-255) or IETF Review range (256-9999).

1.

   * application/pkcs7-mime
   * Type name: application
   * Subtype name: pkcs7-mime
   * smime-type: certs-only

   * ID: TBD1

   * Required parameters: None

   * Optional parameters: None

   * Encoding considerations: Binary

   * Security considerations: As defined in this specification

   * Published specification: [RFC5751]

   * Applications that use this media type: ANIMA Bootstrap (BRSKI) and EST

2.

   * application/pkcs8

   * Type name: application

   * Subtype name: pkcs8

   * ID: TBD2

   * Required parameters: None

   * Optional parameters: None

   * Encoding considerations: Binary

   * Security considerations: As defined in this specification

   * Published specification: [RFC5958]

   * Applications that use this media type: ANIMA Bootstrap (BRSKI) and EST

3.

   * application/csrattrs

   * Type name: application

   * Subtype name: csrattrs

   * ID: TBD3

* Required parameters: None
* Optional parameters: None
* Encoding considerations: Binary
* Security considerations: As defined in this specification
* Published specification: [RFC7030]
* Applications that use this media type: ANIMA Bootstrap (BRSKI) and EST

4.

* application/pkcs10
* Type name: application
* Subtype name: pkcs10
* ID: TBD4
* Required parameters: None
* Optional parameters: None
* Encoding considerations: binary
* Security considerations: As defined in this specification
* Published specification: [RFC5967]
* Applications that use this media type: ANIMA bootstrap (BRSKI) and EST

Additions to the sub-registry "CoAP Response Code", within the "CoRE Parameters" registry are needed for the following response codes:

o Code: 2.06

o Description: Delayed

o Reference: this document

6. Security Considerations

The security considerations mentioned in EST applies also to EST-coaps.

7. Acknowledgements

The authors are very grateful to Klaus Hartke for his detailed explanations on the use of Block with DTLS. The authors would like to thank Esko Dijk and Michael Verschoor for the valuable discussions that helped in shaping the solution.

8. Change Log

9. References

9.1. Normative References

[I-D.ietf-anima-bootstrapping-keyinfra] Pritikin, M., Richardson, M., Behringer, M., and S. Bjarnason, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-03 (work in progress), June 2016.

[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>.

[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, DOI 10.17487/RFC4492, May 2006, <http://www.rfc-editor.org/info/rfc4492>.

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <http://www.rfc-editor.org/info/rfc5272>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <http://www.rfc-editor.org/info/rfc5751>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <http://www.rfc-editor.org/info/rfc5958>.

[RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, DOI 10.17487/RFC5967, August 2010, <http://www.rfc-editor.org/info/rfc5967>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <http://www.rfc-editor.org/info/rfc6090>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <http://www.rfc-editor.org/info/rfc7030>.

[RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS", RFC 7251, DOI 10.17487/RFC7251, June 2014, <http://www.rfc-editor.org/info/rfc7251>.

[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>.

[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>.

9.2. Informative References

[ieee802.15.4] Institute of Electrical and Electronics Engineers, , "IEEE Standard 802.15.4-2006", 2006.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <http://www.rfc-editor.org/info/rfc4944>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

Appendix A. Operational Scenario Example Messages

This appendix provides detailed examples of the messages using DTLS and BLOCK option Block2. The minimum PMTU is 1280 bytes, which is the example value assumed for the DTLS datagram size. The example block length is taken as 64 which gives an SZX value of 2.

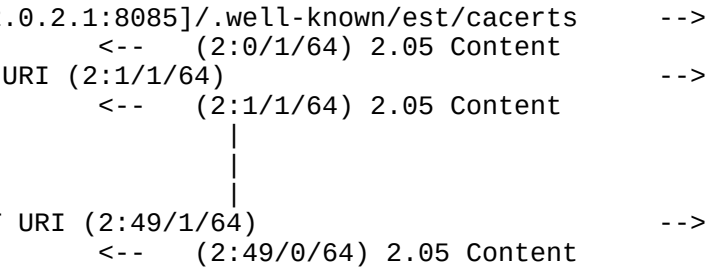
The following is an example of a valid /cacerts exchange.

During the initial DTLS handshake, the client can ignore the optional server-generated "certificate request" and can instead proceed with the CoAP GET request. The content length of the cacerts response in appendix A.1 of [RFC7030] is 4246 bytes using base64. This leads to a length of 3185 bytes in binary. The CoAP message adds around 10 bytes, the DTLS record 29 bytes.

To avoid IP fragmentation, the CoAP block option is used and an MTU of 127 is assumed to stay within one IEEE 802.15.4 packet. To stay below the MTU of 127, the payload is split in 50 packets with a payload of 64 bytes each. Fifty times the client sends an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP Request. The server returns an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP response.

The CoAP request-response exchange with block option is shown below. Block option is shown in a decomposed way indicating the kind of Block option (2 in this case because used in the response) followed by a colon, and then the block number (NUM), the more bit (M = 0 means last block), and block size exponent (2**(SZX+4)) separated by slashes. The Length 64 is used with SZX= 2 to avoid IP fragmentation.

The CoAP Request is sent with confirmable (CON) option and the content format of the Response is /application/cacerts.



Authors' Addresses

Sandeep S. Kumar
Philips Lighting Research
High Tech Campus 7
Eindhoven 5656 AE
NL

Email: ietf@sandeep.de

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

CoRE
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2017

P. van der Stok, Ed.
consultant
J. Jimenez
Ericsson
October 27, 2016

Mapping from LWM2M model to CoMI YANG model
draft-vanderstok-core-yang-lwm2m-00

Abstract

This document defines a set of rules to convert a LWM2M xml-based device specification to a YANG MODULE. The invocation of the server executing the converted YANG code makes use of CoMI. The mapping from the original LWM2M URI to the corresponding CoMI URI is presented.

Note

Discussion and suggestions for improvement are requested, and should be sent to roll@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 30, 2017.

Copyright Notice

Table of Contents

1. Introduction

Standardization organizations define interfaces hosted by processors to manipulate the connected equipment. Examples of such standardization organizations are BACnet, KNX, ZigBee, oBIX, OMA/ IPSO, and many others. These organizations plan to move to resource based interfaces. The data models proposed by these organizations are hierarchical models that can be specified in XML and describe classes with attributes and operations that can be instantiated to objects. An example is the OMA LWM2M (see [OMNA]) Object model, that standardizes eight numbered object types for device management. IPSO (see [IPSO]) expands those objects to handle applications. This document describes rules to translate xml specifications of the LWM2M/IPSO organizations to YANG [RFC7950], and the invocation of the YANG based server according to the CoAP Management Interface (CoMI) specification [I-D.vanderstok-core-comi].

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].

The following terms are defined in [RFC6241] and are not redefined here:

- client
- configuration data
- server
- state data

The following terms are defined in [RFC7950] and are not redefined here:

- data model
- data node

The terminology for describing YANG data models is found in [RFC7950].

1.1.1. Tree Diagrams

A simplified graphical representation of the data model is used in the YANG modules specified in this document. The meaning of the symbols in these diagrams is as follows:

Brackets "[" and "]" enclose list keys.

Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).

Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Conversion rules LWM2M to YANG

LWM2M objects are typed, where each type is identified with a number. The object provides one or more instances which are numbered. An instance is composed of resources, also identified with numbers. An instance on a host can be accessed with the example URI: coap+lwm2m://example.com/object/instance/resource, where resource, instance and object are numbers, specified by the LWM2M specification.

When using YANG, the object identifiers, followed by the resource identifier, are YANG strings instead of numbers. The format of the instance identifier depends on the YANG model that is chosen to specify LWM2M objects.

For an automatic translation from the XML LWM2M specification to a YANG specification, the following rules apply for access, optional, units, and range specifications:

- The optional/mandatory aspect of the LWM2M resource is covered by the leaf's mandatory "false/true" statement of YANG as specified in section 7.6.5 of [RFC7950]. The YANG statement "mandatory = TRUE" means that(given the right conditions) the leaf must exist.

- The R,W access aspects of a data item are translated using the YANG "config" statement as specified in section 7.21.1 of [RFC7950]. If "config" is "true", the Data nodes are part of configuration datastores, resulting in RW access. If "config" is "false", the data nodes are not part of configuration datastores, resulting in R access. YANG does not provide facilities to specify W access only.

- When the YANG RPC is specified, E access is meant. In section 7.14 of [RFC7950] RPCs are modelled for NETCONF using YANG input and output parameters. When input parameters are added, EW access is meant; when output parameters are added, ER access is meant and with both input and output parameters ERW access is meant.

- The YANG ACTION is specified in section 7.15 of [RFC7950]. Contrary to RPC, ACTION statement is associated with a data node. The data node has E access. Input leafs of the data node have W access, and output leafs have R access.

- To specify the range of a data resource the YANG range statement, specified in section 9.2.4 of [RFC7950], is used.

- YANG range can be used in a straightforward fashion for items of type integer. The range of decimal64, used for float, is less

straightforward. The possible ranges are restricted by the fraction-digits which specifies the size of the fraction part of the float (see section 9.3.4. of [RFC7950].

- The YANG units statement, specified in section 7.3.3 of [RFC7950], is used to express the units.

- The attributes of the YANG leaf need to be presented in the order: "type", possibly qualified with "range", "units", "config", "mandatory", and finally "Description".

- In the presented YANG specification the LWM2M resources are specified as leafs of a YANG list.

YANG lists may contain key leafs which uniquely identify an instance in a list. By specifying a key leaf (for example called "instance") that contains the list instance number, the YANG list instance can be uniquely referenced by the instance number. Accordingly, OMA objects are modelled as YANG lists. The value of the "instance" leaf in the list is equal to the instance number of the OMA object. The numbering of the instances does not need to be consecutive. The OMA resources are the other leafs of the YANG list.

Choices need to be made how to represent the numbered object ID, and resource ID as YANG identifiers. YANG identifiers are strings and cannot be represented by numbers.

The YANG identifier strings need to be mapped to numbered identifiers. The appendices show 3 ways to represent the LWM2M device ID and resource ID in the YANG specification.

- In Appendix A, Yang Identifiers are modelled as strings that start with string "ID" followed by the identifier number (see module humidityID).

- In Appendix B, Yang Identifiers of objects and resources are modelled as strings that are equivalent to the OMA object- and resource- name (see module humidityNM).

- In Appendix C, the OMA device is modelled as a YANG container composed of an identifier and a list of instances. The list is composed of an instance number and a set of resource containers. The resource container is composed of the pair (attribute identifier number, IPSO resource specification)(see module humidityLF).

Below the tree diagrams (see Section 1.1.1 for an explanation of the syntax) of the three valid YANG modules are shown.

```
   module: ietf-yang-humidityID
   +--ro ID3304* [instance]
   +--ro instance                              uint16
   +--ro ID5700                                decimal64
   +--ro ID5701?                               string
   +--ro ID5601?                               decimal64
   +--ro ID5602?                               decimal64
   +--ro ID5603?                               decimal64
   +--ro ID5604?                               decimal64
   +---x ID5605


   module: ietf-yang-humidityNM
   +--ro IPSO-humidity* [instance]
   +--ro instance                               uint16
   +--ro Sensor_Value                           decimal64
   +--ro Units?                                 string
   +--ro Min_Measured_Value?                    decimal64
   +--ro Max_Measured_Value?                    decimal64
   +--ro Min_Range_Value?                       decimal64
   +--ro Max_Range_Value?                       decimal64
   +---x Reset_Min_and_Max_measured_values

   module: ietf-yang-humidityLF
   +--rw IPSO-humidity
   +--ro identifier    uint16
   +--ro resources* [instance]
   +--ro instance                               uint16
   +--ro Sensor_Value
   |  +--ro identifier?                         uint16
   |  +--ro content                             decimal64
   +--ro Units
   |  +--ro identifier?                         uint16
   |  +--ro content?                            string
   +--ro Min_Measured_Value
   |  +--ro identifier?                         uint16
   |  +--ro content?                            decimal64
   +--ro Max_Measured_Value
   |  +--ro identifier?                         uint16
   |  +--ro content?                            decimal64
   +--ro Min_Range_Value
   |  +--ro identifier?                         uint16
   |  +--ro content?                            decimal64
   +--ro Max_Range_Value
   |  +--ro identifier?                         uint16
   |  +--ro content?                            decimal64
   +--ro Reset_Min_and_Max_measured_values
   +--ro identifier?                            uint16
```

```
   +---x reset
```

Module humidityLF of Appendix C is the most complex one and is not recommended. Module humidityID of Appendix A works but is a bit forced approach and lacks the resource name. Module humidityNM of Appendix B is the most natural approach where the YANG identifiers are equal to the device (type) and resource names.

CoMI [I-D.vanderstok-core-comi] uses a conversion from names to numbers to reduce the request URI size, and the payload of the server requests and answers. The LWM2M organization specifies both the names and the numbers of the devices and resources. The number of the resource is not unique and for the CoMI identifier the resource number needs to be prefixed by the device number to be unique.

3. URI convention

The invocation URI of a LWM2M resource looks like:

coap+lwm2M://example.com/object/instance/resource

In this section it is assumed that the YANG mapping of the module humidityNM of Appendix B is used.

When YANG is used, the LWM2M resource invocation can follow the RESTCONF convention using http, or the CoMI convention using CoAP.

When using RESTCONF (see section 3.5.3 of [I-D.ietf-netconf-restconf] the invocation of object with instance = number will look like:

http://example.com/object/instance=number/resource

In the case of CoMI the object/resource numbers are used, and not the names, to reduce the payload size. The instance is specified in a query parameter. Consequently, the LWM2M resource on a server executing a YANG specification, is accessed according to the CoMI specification with:

coap://example.com/identifier?k=number

The identifier is a composition of the object number and the resource number. Assume that n is smallest number for which 10**(n+1)/resource >= 1. The value of the identifier = (object*10**n)+resource.

Assume that the IPSO-humidity/Sensor_Value 3304/5700 numbers are composed to the numeric identifier 33045700. According to [RFC4648],

the identifier is represented in base64 which leads to B-DzE. The URI for the CoMI invocation of instance 0 of IPSO-humidity/ Sensor_value will look like:

coap://example.com/B-DzE?k=0

For LWM2M objects with only one instance, the k=0 can be omitted.

4. observation and notification

An LWM2M server uses "observe" to receive notification from the server. This remains unchanged with YANG servers and CoMI.

5. Payload format

The payload of the request and the response follows the payload format specified for CoMI. The content format is CBOR [RFC7049]. The YANG objects are returned as maps containing (identifier, value) pairs. Where the identifier is the numeric identifier discussed in Section 3. and the value is of the type specified by the YANG specification of the server. The CBOR encoding of the YANG types is specified in [I-D.ietf-core-yang-cbor].

6. YANG extensions to LWM2M

By adding keys leafs to a list object, YANG allows additionally the selection of instances by the contents of the key leafs.

The FETCH method of CoAP makes it possible to request multiple resource instances in one request.

The notification statement of YANG encourages a more flexible specification of notifications.

7. Security considerations

To be filled in

8. Acknowledgements

We are grateful to

9. Changelog

NO changes from nothing to version 00

10. References

10.1. 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>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[I-D.vanderstok-core-comi] Stok, P. and A. Bierman, "CoAP Management Interface", draft-vanderstok-core-comi-09 (work in progress), March 2016.

[I-D.ietf-core-yang-cbor] Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-02 (work in progress), July 2016.

10.2. Informative References

[I-D.ietf-netconf-restconf] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-17 (work in progress), September 2016.

[OMNA] "Open Mobile Naming Authority (OMNA)", Web http://http://technical.openmobilealliance.org/Technical/technical-information/omna.

[IPSO] "IP for Smart Objects (IPSO)",
Web http://ipso-alliance.github.io/pub/.

Appendix A. YANG identifiers as IDnumbers

Yang Identifiers are modelled as string that starts with ID followed by the identifier number. The device object is modelled as a list that contains multiple instances.

<CODE BEGINS> file "ietf-humidityID@2016-07-25.yang"

```
module ietf-humidityID{

yang-version 1.1;  // needed for action

namespace
"urn:ietf:params:xml:ns:yang:ietf-humidityID";

prefix humid;

organization
"IPSO";

contact
"WG Web:   http://tools.ietf.org/wg/core/
WG List:  mailto:core@ietf.org

WG Chair: Carsten Bormann
mailto:cabo@tzi.org

WG Chair: Jaime Jimenez
mailto:jaime.jimenez@ericsson.com

Editor:   Peter van der Stok
mailto:consultancy@vanderstok.org

Editor:   Jaime Jimenez
mailto:jaime.jimenez@ericsson.com";

description
"This module contains information about the operation of the
IPSO LWM2M humidity sensor with ID 3304.

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

Redistribution and use in source and binary forms, with or
without modification, is permitted pursuant to, and subject
to the license terms contained in, the Simplified BSD License
```

```
   set forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents
           (http://trustee.ietf.org/license-info).

   This version of this YANG module is part of RFC XXXX; see
   the RFC itself for full legal notices.";

           revision "2016-07-25" {
           description "Initial revision.";
           reference
   "I-D:draft-vanderstok-core-yang-lwm2m: YANG language applied
   to the LWM2M IPSO humidity sensor specification";
           }

           list ID3304 {
           key instance;
           config  false;     // should be same for key leaf
           description
           "IPSO humidity: The humidity sensor is composed of
            a set of instances";
           leaf instance {
           type uint16{
           range "0..1";   // only one instance zero (0)
           }
           config false;    // R access
           mandatory "true";
           description
           "the number of the humidity sensor instance";
           }
           leaf ID5700 {
           type decimal64{    // YANG has no float
           fraction-digits 2;
           range "10.0 .. 66.6";}
           config false;    // R access
           mandatory "true";
           description
           "Sensor Value: Last or Current Measured Value
              from the Sensor";
           }
           leaf ID5701 {
           type string;
           units "Defined by 'Units' resource";
           config false;    // R access
           description
           "Units: Measurement unit definition
               e.g. 'Cel' for temperature in Celsius";
           }
           leaf ID5601 {
```

```
            type decimal64{    // YANG has no float
            fraction-digits 2;
            range "10.0 .. 66.6";}
            units "Defined by 'Units' resource";
            config false;   // R access
            description
            "Min Measured Value: The minimum value measured
              by the sensor since power ON or reset";
            }
            leaf ID5602 {
            type decimal64{    // YANG has no float
            fraction-digits 2;
            range "10.0 .. 66.6";}
            units "Defined by 'Units' resource";
            config false;    // R access
            description
            "Max Measured Value: The maximum value measured
               by the sensor since power ON or reset";
            }
            leaf ID5603 {
            type decimal64{    // YANG has no float
            fraction-digits 2;
            range "10.0 .. 66.6";}
            units "Defined by 'Units' resource";
            config false;    // R access
            description
            "Min Range Value: The minimum value that
               can be measured by the sensor";
            }
            leaf ID5604 {
            type decimal64{    // YANG has no float
            fraction-digits 2;
            range "10.0 .. 66.6";}
            units "Defined by 'Units' resource";
            config false;    // R access
            description
            "Max Range Value: The maximum value that
               can be measured by the sensor";
            }
            action ID5605 {
            //E access: this is an RPC
            //    without input and output parameters
            description
            "Reset Min and Max measured values: Reset the
               Min and Max measured values to current value";
            }
            }  // list ID3304
            } // module ietf-yang-humidity
```

<CODE ENDS>

Appendix B. YANG identifiers as resource names

Yang Identifiers are modelled as strings that represent the resource name. The device object is modelled as a list with multiple instances.

```
<CODE BEGINS> file "ietf-humidityNM@2016-07-25.yang"

module ietf-humidityNM{

yang-version 1.1;  // needed for action

namespace
            "urn:ietf:params:xml:ns:yang:ietf-humidityNM";

prefix humid;

            organization
            "IPSO";

            contact
            "WG Web:   http://tools.ietf.org/wg/core/
            WG List:  mailto:core@ietf.org

            WG Chair: Carsten Bormann
            mailto:cabo@tzi.org

            WG Chair: Jaime Jimenez
            mailto:jaime.jimenez@ericsson.com

            Editor:   Peter van der Stok
            mailto:consultancy@vanderstok.org

            Editor:   Jaime Jimenez
            mailto:jaime.jimenez@ericsson.com";

            description
  "This module contains information about the
operation of the IPSO LWM2M humidity sensor with ID 3304.

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

Redistribution and use in source and binary forms, with or
without modification, is permitted pursuant to, and subject
```

```
   to the license terms contained in, the Simplified BSD License
   set forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents
         (http://trustee.ietf.org/license-info).

   This version of this YANG module is part of RFC XXXX; see
   the RFC itself for full legal notices.";

               revision "2016-07-25" {
               description "Initial revision.";
               reference
               "I-D:draft-vanderstok-core-yang-lwm2m:
   YANG language applied to the LWM2M IPSO humidity sensor
   specification";
               }


               list IPSO-humidity {
               key instance;
               config   false;   //  should be same as key leaf
               description
               "3304: The humidity sensor is composed of
                 a set of instances";
               leaf instance {
               type uint16{
               range "0..1";   // only one instance zero (0)
               }
               config false;    // R access
               mandatory "true";
               description
               "the number of the humidity sensor instance";
               }
               leaf Sensor_Value  {
               type decimal64{    // YANG has no float
               fraction-digits 2;
               range "10.0 .. 66.6";}
               units "Defined by 'Units' resource";
               config false;    // R access
               mandatory "true";
               description
               "5700: Last or Current Measured Value
                  from the Sensor";
               }
               leaf Units {
               type string;
               units "Defined by 'Units' resource";
               config false;    // R access
               description
```

```
                "5701: Measurement unit definition
                    e.g. 'Cel' for temperature in Celsius";
                }
                leaf Min_Measured_Value {
                type decimal64{    // YANG has no float
                fraction-digits 2;
                range "10.0 .. 66.6";}
                units "Defined by 'Units' resource";
                config false;    // R access
                description
                "5601: The minimum value measured by
                   the sensor since power ON or reset";
                }
                leaf Max_Measured_Value {
                type decimal64{    // YANG has no float
                fraction-digits 2;
                range "10.0 .. 66.6";}
                units "Defined by 'Units' resource";
                config false;    // R access
                description
                "5602: The maximum value measured
                    by the sensor since power ON or reset";
                }
                leaf Min_Range_Value {
                type decimal64{    // YANG has no float
                fraction-digits 2;
                range "10.0 .. 66.6";}
                units "Defined by 'Units' resource";
                config false;    // R access
                description
                "5603: The minimum value that can be measured
                   by the sensor";
                }
                leaf Max_Range_Value{
                type decimal64{    // YANG has no float
                fraction-digits 2;
                range "10.0 .. 66.6";}
                units "Defined by 'Units' resource";
                config false;    // R access
                description
                "5604: The maximum value that can be measured
                   by the sensor";
                }
                action Reset_Min_and_Max_measured_values {
                // E access: this is an RPC
                // without input and output parameter
                description
                "5605: Reset the Min and Max measured values
```

```
                to current value";
              }  // rpc
              }  // list ID3304
              } // module ietf-yang-humidity

              <CODE ENDS>
```

Appendix C. YANG identifiers as additional leaf

The device object is modelled as a container composed of an identifier and a list of instances. The list instance is composed of an instance number and a set of resource containers. The resource container is composed of the pair (attribute identifier number, IPSO resource specification).

```
 <CODE BEGINS> file "ietf-humidityLF@2016-07-25.yang"

 module ietf-humidityLF{

 yang-version 1.1;  // needed for rpc

 namespace
   "urn:ietf:params:xml:ns:yang:ietf-humidityLF";

 prefix humid;

                organization
                "IPSO";

                contact
                "WG Web:   http://tools.ietf.org/wg/core/
                WG List:  mailto:core@ietf.org

                WG Chair: Carsten Bormann
                mailto:cabo@tzi.org

                WG Chair: Jaime Jimenez
                mailto:jaime.jimenez@ericsson.com

                Editor:   Peter van der Stok
                mailto:consultancy@vanderstok.org

                Editor:   Jaime Jimenez
                mailto:jaime.jimenez@ericsson.com";
```

```
                description
                "This module contains information about the
   operation of the IPSO LWM2M humidity sensor with ID 3304.

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

   Redistribution and use in source and binary forms, with or
   without modification, is permitted pursuant to, and subject
   to the license terms contained in, the Simplified BSD License
   set forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents
   (http://trustee.ietf.org/license-info).

   This version of this YANG module is part of RFC XXXX;
   see the RFC itself for full legal notices.";

                revision "2016-07-25" {
                description "Initial revision.";
                reference
                "I-D:draft-vanderstok-core-yang-lwm2m:
   YANG language applied to the LWM2M IPSO humidity sensor
   specification";
                }

                container IPSO-humidity{
                description
                "Device separated in identifier and list";
                leaf identifier{
                type uint16;  // fixed to 3304
                config false;
                mandatory "true";
                description
                "the LWM2M identification number of the device";
                }
                list resources {
                key instance;
                config  false;   // should be same as key leaf
                description
                "3304: The humidity sensor is composed of
                   a set of instances";
                leaf instance {
                type uint16{
                range "0..1";   // only one instance zero (0)
                }
                config false;    // R access
                mandatory "true";
                description
```

```
"the number of the humidity sensor instance";
}  // instance number
container Sensor_Value  {
description
"Resource separated in identifier and content";
leaf identifier{
type uint16;   // fixed to 5700
config false;  // R access
description
"identifier should contain the value 5700";
}
leaf content{
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
mandatory "true";
description
"5700: Last or Current Measured Value
  from the Sensor";
}  // content
}  // container
container Units {
description
"Resource separated in identifier and content";
leaf identifier{
type uint16;   // fixed to 5701
config false;  // R access
description
"identifier should contain the value 5701";
}
leaf content{
type string;
units "Defined by 'Units' resource";
config false;    // R access
description
"5701: Measurement unit definition
  e.g. 'Cel' for temperature in Celsius";
}  // content
}  // container
container Min_Measured_Value {
description
"Resource separated in identifier and content";
leaf identifier{
type uint16;   // fixed to 5601
config false;  // R access
description
```

```
"identifier should contain the value 5601";
} // identifier
leaf content{
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"5601: The minimum value measured by the
  sensor since power ON or reset";
} // content
}
container Max_Measured_Value {
description
"Resource separated in identifier and content";
leaf identifier{
type uint16;   // fixed to 5602
config false;  // R access
description
"identifier should contain the value 5602";
}
leaf content{
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"5602: The maximum value measured by
  the sensor since power ON or reset";
} // content
} // container
container Min_Range_Value {
description
"Resource separated in identifier and content";
leaf identifier{
type uint16;   // fixed to 5603
config false;  // R access
description
"identifier should contain the value 5603";
} // identifier
leaf content{
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
```

```
                    description
                    "5603: The minimum value that can be measured
                       by the sensor";
                    }  // content
                    }  // container
                    container Max_Range_Value{
                    description
                    "Resource separated in identifier and content";
                    leaf identifier{
                    type uint16;   // fixed to 5604
                    config false;  // R access
                    description
                    "identifier should contain the value 5604";
                    }  // identifier
                    leaf content{
                    type decimal64{    // YANG has no float
                    fraction-digits 2;
                    range "10.0 .. 66.6";}
                    units "Defined by 'Units' resource";
                    config false;    // R access
                    description
                    "5604: The maximum value that can be measured
                      by the sensor";
                    }  // content
                    }
                    container Reset_Min_and_Max_measured_values {
                    description
                    "Resource separated in identifier and action";
                    leaf identifier{
                    type uint16;   // fixed to 5605
                    config false;  // R access
                    description
                    "identifier should contain the value 5605";
                    }
                    action reset{
   // E access: this is an RPC without input and output parameters
                    description
                    "5605: Reset the Min and Max measured values to
                     current value";
                    } // action  reset
                    } // container Reset_min_and_max
                    }  // list resources
                    }  // container IPSO-humidity (3304)
                    } // module ietf-yang-humidity

                    <CODE ENDS>
```

Authors' Addresses

Peter van der Stok (editor)
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Jaime Jimenez
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Phone: +358-442992827(Finland)
Email: jaime.jimenez@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: August 4, 2017

M. Veillette, Ed.
Trilliant Networks Inc.

January 31, 2017

Constrained YANG Module Library
draft-veillette-core-yang-library-00

Abstract

This document describes a YANG library that provides information about all the YANG modules used by a constrained network management server (e.g., a CoAP Management Interface (CoMI) server). Simple caching mechanisms are provided to allow clients to minimize retrieval of this information.

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 4, 2017.

Copyright Notice

Table of Contents

1. Introduction

The YANG library specified in this document is available to clients of a given server to discover the YANG modules supported by this constrained network management server. A CoMI server provides a link to this library in the /c/mod.uri resource. The following YANG module information is provided to client applications to fully utilize the YANG data modeling language:

- module list: The list of YANG modules implemented by a server, each module is identified by its assigned YANG Schema Item iDentifier (SID) and revision.

- submodule list: The list of YANG submodules included by each module, each submodule is identified by its assigned SID and revision.

- feature list: The list of features supported by the server, each feature is identified by its assigned SID.

- deviation list: The list of YANG modules used for deviation statements associated with each YANG module, each module is identified by its assigned SID and revision.

1.1. Major differences between ietf-constrained-yang-library and ietf-yang-library

YANG module ietf-constrained-yang-library targets the same functionality and shares the same approach as YANG module ietf-yang-library. The following changes with respect to ietf-yang-library are specified to make ietf-constrained-yang-library compatible with SID [I-D.ietf-core-yang-cbor] used by CoMI [I-D.vanderstok-core-comi] and to improve its applicability to constrained devices and networks.

- YANG module ietf-constrained-yang-library extends the caching mechanism supported by ietf-yang-library to multiple servers. This is accomplished by supporting the identityref datatype for "module-set-id". This enables the use of a managed identifier (i.e. a SID) to identify a specific assembly of YANG modules, deviations and features implemented by a group of constrained servers.

- Modules, sub-modules, deviations and features are identified using a numerical value (SID) instead of a string (yang-identifier).

- The "namespace" leaf, not required for SIDs, but mandatory in ietf-yang-library is not included in ietf-constrained-yang-library.

- Schemas can be located using the already available module or sub-module identifier (SID) and revision. For this reason, support of module and sub-module schema URIs have been removed.

- To minimize their size, each revision date is encoded in binary.

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- module

- submodule

- feature

- deviation

The following terms are defined in [I-D.ietf-core-yang-cbor]:

- YANG Schema Item iDentifier (SID)

The following terms are defined in [I-D.vanderstok-core-comi]:

- client
- server

The following terms are used within this document:

- library: a collection of YANG modules used by a server.

3. Overview

The "ietf-constrained-yang-library" module provides information about the YANG library used by a given server. This module is defined using YANG version 1 as defined by [RFC7950], but it supports the description of YANG modules written in any revision of YANG.

3.1. Tree diagram

A simplified graphical representation of the YANG module specified in this document (ietf-constrained-yang-library) is provided below. The meaning of the symbols in this diagram is as follows:

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

```
module: ietf-constrained-yang-library
   +--ro modules-state
      +--ro module-set-id    union
      +--ro module* [sid revision]
         +--ro sid                 sid
         +--ro revision            revision
         +--ro feature*            sid
         +--ro deviation* [sid revision]
         |  +--ro sid         sid
         |  +--ro revision    revision
         +--ro conformance-type    enumeration
         +--ro submodule* [sid revision]
            +--ro sid         sid
            +--ro revision    revision
notifications:
   +---n yang-library-change
      +--ro module-set-id    -> /modules-state/module-set-id
```

3.2. Description

3.2.1. modules-state

This mandatory container specifies the module set identifier and the list of modules supported by the server.

3.2.2. modules-state/module-set-id

This mandatory leaf contains an identifier representing the current set of modules and submodules used by a server. This identifier is server-specific when implemented as unit32 or can be used by multiple servers when implemented as identityref. The value of this leaf MUST change whenever the set of modules and submodules in the library changes. There is no requirement that the same set always results in the same module-set-id value.

This leaf allows a client to fetch the module list once, cache it, and only re-fetch it if the value of this leaf has been changed.

If the value of this leaf changes, the server also generates a "yang-library-change" notification, with the new value of "module-set-id".

3.2.3. modules-state/module

This mandatory list contains one entry for each YANG module supported by the server. There MUST be an entry in this list for each revision of each YANG module that is used by the server. It is possible for multiple revisions of the same module to be imported, in addition to an entry for the revision that is implemented by the server.

4. YANG Module "ietf-constrained-yang-library"

RFC Ed.: update the date below with the date of RFC publication and remove this note.

```
<CODE BEGINS> file "ietf-constrained-yang-library@2017-01-20.yang"
module ietf-constrained-yang-library {
  namespace "urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library";
  prefix "lib";

  organization
    "IETF CORE (Constrained RESTful Environments) Working Group";

  contact
    "WG Web:   <http://datatracker.ietf.org/wg/core/>

     WG List:  <mailto:core@ietf.org>

     WG Chair: Carsten Bormann
               <mailto:cabo@tzi.org>

     WG Chair: Jaime Jimenez
               <mailto:jaime.jimenez@ericsson.com>

     Editor:   Michel Veillette
               <mailto:michel.veillette@trilliantinc.com>";

  description
    "This module contains the list of YANG modules and submodules
    implemented by a server.

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

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject
     to the license terms contained in, the Simplified BSD License
     set forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (http://trustee.ietf.org/license-info).

     This version of this YANG module is part of RFC XXXX; see
     the RFC itself for full legal notices.";

  // RFC Ed.: replace XXXX with actual RFC number and remove
  // this note.

  // RFC Ed.: update the date below with the date of the RFC
```

```
  // publication and remove this note.

  revision 2017-01-20 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: Constrained YANG Module Library.";
  }

  /*
   * Typedefs
   */

  typedef revision {
    type binary {
      length "4";
    }
    description
      "Revision date encoded as a binary string as follow:
      - First byte = Year divided by 100
      - Second byte = Year modulo 100 (0 to 99)
      - Third byte = Month (1 = January to 12 = december)
      - Forth byte = Day (1 to 31)";
  }

  typedef sid {
    type uint64;
    description
      "Identifier assigned to different YANG items such as
      data nodes, RPCs and actions, notifications, modules,
      sub-modules, features and deviations.";
  }

  /*
   * Groupings
   */

  grouping identification-info {
    description
      "YANG modules and submodules identification information.";

    leaf sid {
      type sid;
      mandatory true;
      description
        "SID assigned to this module or submodule.";
    }
```

```
    leaf revision {
      type revision;
      description
        "Revision date assigned to this module or submodule.
        A zero-length binary string is used if no revision statement
        is present in the YANG module or submodule.";
    }
  }

  identity module-set {
      description
        "Base identity from which shared module-set identifiers
        are derived.";
    }

  /*
   * Operational state data nodes
   */

  container modules-state {
    config false;
    description
      "Contains information about the different data models
      implemented by the server.";

    leaf module-set-id {
      type union {
        type uint32;
        type identityref {
          base "lib:module-set";
        }
      }
      mandatory true;
      description
        "Identifier representing the current set of modules
        and submodules listed in the 'module' list. This
        identifier is server-specific when implemented as
        unit32 or shared between multiple servers when
        implemented as identityref. The server MUST change
        the value of this leaf each time the content of the
        'module' list instance change.";
    }

    list module {
      key "sid revision";
      description
        "Each entry represents one revision of one module
         currently supported by the server.";
```

```
      uses identification-info;

      leaf-list feature {
        type sid;
        description
          "List of YANG features from this module that are
          supported by the server, regardless whether
          they are defined in the module or in any included
          submodule.";
      }

      list deviation {
        key "sid revision";
        description
          "List of YANG deviation modules used by this server
          to modify the conformance of the module associated
          with this entry.  Note that the same module can be
          used for deviations for multiple modules, so the
          same entry MAY appear within multiple 'module' entries.

          The deviation module MUST be present in the 'module'
          list, with the same sid and revision values.
          The 'conformance-type' value will be 'implement' for
          the deviation module.";

        uses identification-info;
      }

      leaf conformance-type {
        type enumeration {
          enum implement {
            value 0;
            description
              "Indicates that the server implements one or more
              protocol-accessible objects defined in the YANG
              module identified in this entry.  This includes
              deviation statements defined in the module.

              For YANG version 1.1 modules, there is at most one
              module entry with conformance type 'implement' for a
              particular module, since YANG 1.1 requires that
              at most one revision of a module is implemented.

              For YANG version 1 modules, there SHOULD NOT be more
              than one module entry for a particular module.";
          }
          enum import {
            value 1;
```

```
            description
              "Indicates that the server imports reusable definitions
              from the specified revision of the module, but does
              not implement any protocol accessible objects from
              this revision.

              Multiple module entries for the same module MAY
              exist. This can occur if multiple modules import the
              same module, but specify different revision-dates in
              the import statements.";
          }
        }
        mandatory true;
        description
          "Indicates the type of conformance the server is claiming
          for the YANG module identified by this entry.";
      }

      list submodule {
        key "sid revision";
        description
          "Each entry represents one submodule within the
           parent module.";
        uses identification-info;
      }
    }
  }

  /*
   * Notifications
   */

  notification yang-library-change {
    description
      "Generated when the set of modules and submodules supported
      by the server has changed.";

    leaf module-set-id {
      type leafref {
        path "/lib:modules-state/lib:module-set-id";
      }
      mandatory true;
      description
        "Contains the module-set-id value representing the
        set of modules and submodules supported by the server
        at the time the notification is generated.";
    }
  }
```

```
}
<CODE ENDS>
```

5. IANA Considerations

5.1. YANG Module Registry

This document registers one YANG module in the YANG Module Names registry [RFC7950].

name: ietf-constrained-yang-library

namespace: urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library

prefix: lib

reference: RFC XXXX

// RFC Ed.: replace XXXX with RFC number and remove this note

6. Security Considerations

This YANG module is designed to be accessed via the CoMI protocol [I-D.vanderstok-core-comi]. Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access to these data nodes.

Specifically, the 'module' list may help an attacker to identify the server capabilities and server implementations with known bugs. Server vulnerabilities may be specific to particular modules, module revisions, module features, or even module deviations. This information is included in each module entry. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the module list information will help an attacker identify server implementations with such a defect, in order to launch a denial of service attack on the device.

7. Acknowledgments

The YANG module defined by this memo have been derived from an already existing YANG module, ietf-yang-library [RFC7895], we will like to thanks to the authors of this YANG module. A special thank Andy Bierman for his initial recommendations for the creation of this YANG module.

8. References

8.1. Normative References

[I-D.ietf-core-yang-cbor]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-03 (work in progress), October 2016.

[I-D.vanderstok-core-comi]
Stok, P., Bierman, A., Veillette, M., and A. Pelov, "CoAP Management Interface", draft-vanderstok-core-comi-11 (work in progress), January 2017.

[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>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

8.2. Informative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-18 (work in progress), October 2016.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

Author’s Address

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Core
Internet Draft
Intended status: Standards Track
Expires: September 14, 2017

P. Wang
C. Pu
H. Wang
Y. Yang
L. Shao
Chongqing University of
Posts and Telecommunications
J. Hou
Huawei Technologies
March 13, 2017

OPC UA Message Transmission Method over CoAP
draft-wang-core-opcua-transmission-01

Abstract

OPC Unified Architecture (OPC UA) is a data exchange standard that provides interoperability in industrial automation. With the coming Industry 4.0, it is of great importance to implement the exchange of semantic information utilizing OPC UA Transmitting in CoAP. This document provides some transmission methods for message of OPC UA over CoAP.

Table of Contents

1. Introduction

Internet of things is one of the attractive applications for CoAP [RFC7252]. Utilizing OPC UA [IEC TR 62541-1] Transmitting over CoAP could meet the demand for industry 4.0 based on the exchange of semantic information [I-D.wang-core-opcua-transmition-requirements]. Similar to OPC UA, CoAP message is exchanged in server/client mode. However, their transmission is not the same. Driven by this, to use OPC UA Transmitting over CoAP, the major problem to be solved is how OPC UA packets are transmitted over CoAP. For the transport layer of OPC UA, the main message transmission method is TCP or HTTP, and CoAP's design inspiration comes from HTTP, thus, there are some

connections in transmission method between them. This document provides some transmission methods for message of OPC UA over CoAP, so that a communication could be established between OPC UA client and OPC UA server.

1.1. Conventions and Terminology

The keywords "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].

OPC: OLE for Process Control

OPC UA: OPC Unified Architecture

SOAP: Simple Object Access Protocol

2. Overview of OPC UA

OPC Unified Architecture (OPC UA), standardized as IEC 62541, is a client-server communication protocol developed by OPC Foundation for safe, reliable data exchange in industrial automation. It is the evolution product of OPC (OLE for Process Control, where OLE denotes Object Linking and Embedding), the widely used standard process for automation technology, and is of great importance in realizing industry 4.0. By introducing Service-oriented architecture (SOA), OPC UA enables an open, cross-platform communication with the advantages of web services, robust security and integrated data model.

2.1. Protocol Stack

OPC UA is an application layer protocol that can be built on an existing layer 5, 6 or 7 protocol such TCP/IP, TLS or HTTP. The OPC UA application layer consists of four sublayers: UA Application, Serialization Layer, Secure Channel Layer and Transport Layer (see Figure 1).

Serialization Layer includes two kinds of data encoding methods: UA Binary and UA XML. The UA XML, based on SOAP/HTTP or SOAP/HTTPS, is firewall friendly. On the other hand, the UA Binary, with least overhead and resource cost, offers an optimized speed and throughput.

The security layer varies according to the selected encoding format. For the HTTPS-based situation, security is guaranteed at TLS but Security Channel should still be presented even empty. It is

worthwhile noting that the communication based on SOAP/HTTP has been deprecated since 2015 due to the lack of industrial approbation in the WS Secure Conversation.

For the transport layer (not the layer in OSI 7 layer model), options can be UA TCP, HTTPS, SOAP/HTTPS, and SOAP/HTTP. OPC UA defines a UA TCP protocol, which differs from HTTP in two main features: the allowance of responses to be returned in any order and to be returned on a different TCP transport end-point. In addition, UA TCP defines the interaction with the upper security channel.

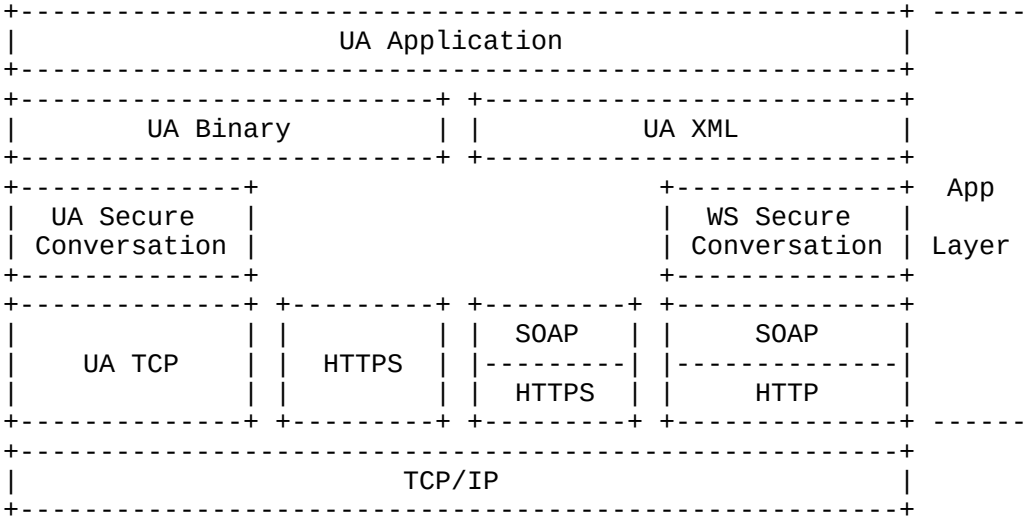


Figure 1: Layering of OPC UA over TCP/IP

2.2. Request/Response Model

The message exchange in UA binary mode is illustrated in Figure 2. After opening the socket, the client starts the connection with the server by using "hello" (HEL) and "acknowledge" (ACK) messages. Afterwards, a pair of messages is needed to open the security channel and define the encryption property. Then another two pairs of messages are exchanged so as to create and activate a session between the client and the server respectively. After these steps, the connection is initiated and the client can send request messages for services. When the request/response process is finished, a reverse process is required for disconnection.

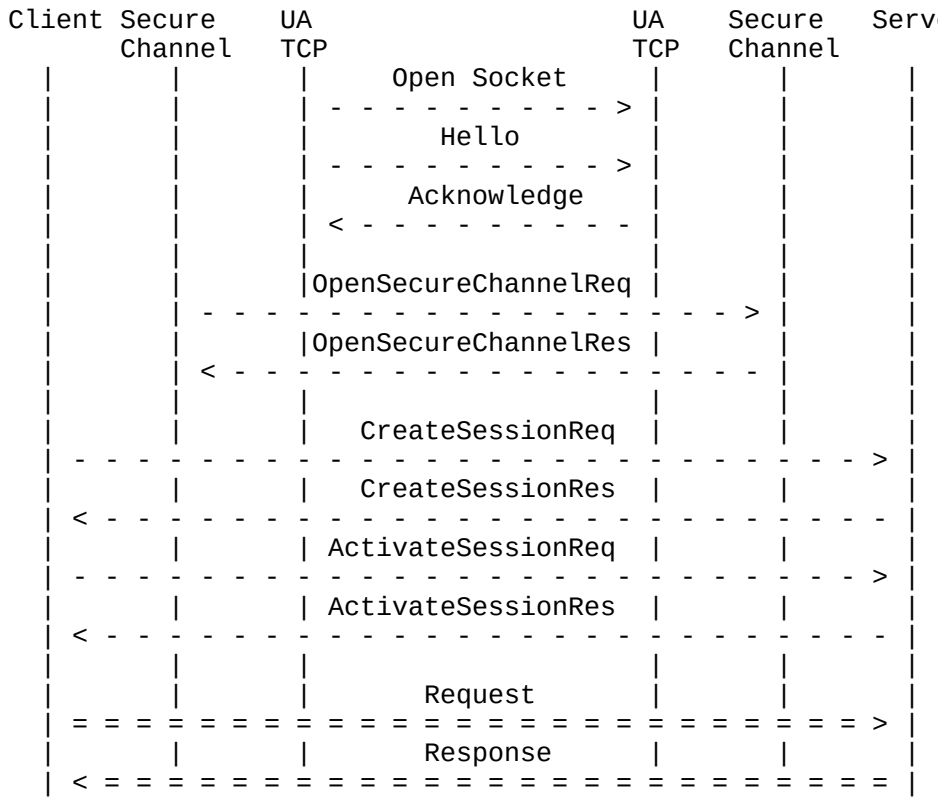


Figure 2: Request/Response Process of UA TCP

3. Specification of OPC UA over CoAP

As mentioned in section 2.1, OPC UA communications can be processed through four options, among which two are related to HTTPS: HTTPS => UA Binary; HTTPS => SOAP => UA XML. HTTPS is a security-guaranteed protocol consisting of a HTTP layer over Transport Layer Security (TLS), thus the UA Security Channel can be left empty.

Constrained Application Protocol (CoAP) is an application layer protocol for constrained nodes and networks, which is designed to easily translate to HTTP for integration with the web. Although CoAP is built on the unreliable transport layer UDP, it offers a security mode binding to Datagram Transport Layer Security (DTLS). This document proposes a transmission scheme based on CoAPs (CoAP + DTLS) for constrained scenarios. The transmission based on CoAP over

Transport Layer Security (TLS) is also possible. Such "CoAP + TLS" transmission scheme is under development [I-D.ietf-core-coap-tcp-tls] and would be covered in the future version.

The protocol stack of the CoAP based OPC UA is illustrated in Figure 3 including two options at Serialization Layer: UA Binary and UA XML. OPC UA packets are encoded in either binary or xml format, and the option field in the CoAP header can specify parameters that support both formats. Therefore, according to the format specified by the CoAP header, the entire packet of the OPC UA can be encapsulated in the payload of the CoAP message for direct transmission.

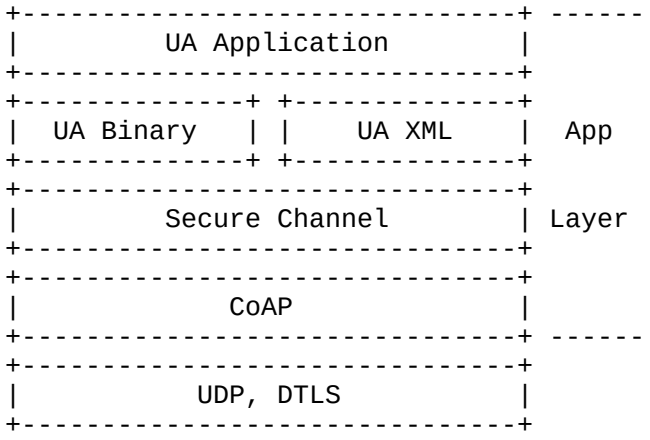


Figure 3: Layering of OPC UA over UDP

Both binary and XML encoding modes are based on the CoAP with an empty UA secure channel in between. For the XML encoding mode, since CoAP layer supports XML encoding format, the SOAP layer in the original stack is not needed.

4. Transmission scheme

4.1. Proxy for OPC UA-CoAP

OPC UA is a protocol mainly for application layer, which defines many services for the different needs of industrial applications. Message is exchanged mainly through server/client mode, utilizing TCP or HTTPS. When security is ignored, OPC UA can be considered to support HTTP transmission. CoAP’s design inspiration comes mainly from HTTP, the two can be mapped between each other to meet the

needs of some special scenes [RFC8075]. Combined with the characteristics of OPC UA and CoAP, a CoAP proxy can be established between OPC UA client and OPC UA server. The architecture is shown in Figure 4.

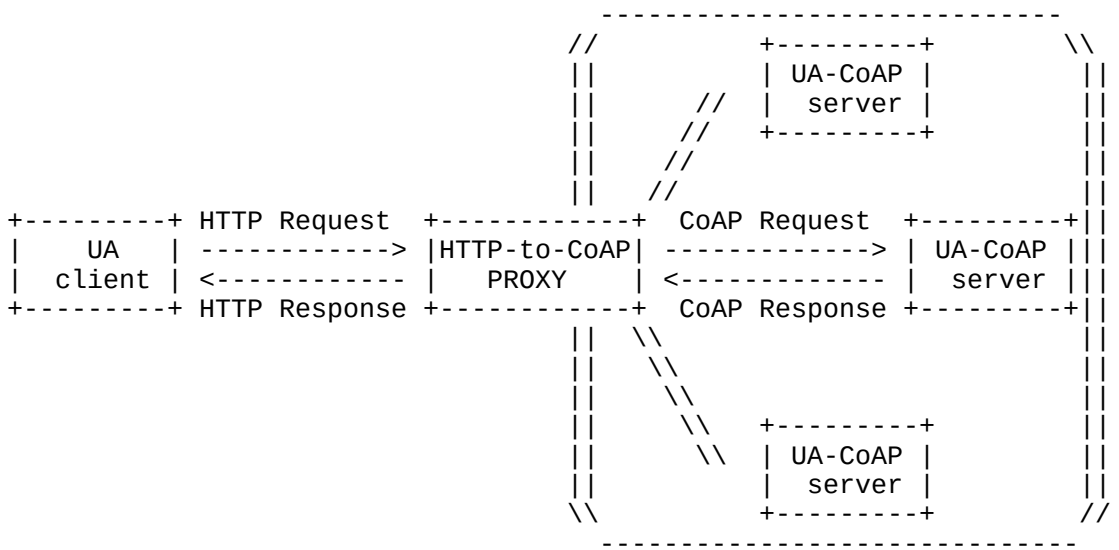


Figure 4: Proxy for OPC UA to CoAP

As shown in Figure 5, assume all OPC UA servers are based on CoAP [I-D.wang-core-opcua-transmition-requirements], and all OPC UA-CoAP server can be viewed as a network, introducing UA-to-CoAP proxy at the boundary of the network. When a traditional OPC UA client initiates an HTTP request to the UA-CoAP server in the network, the UA-to-CoAP proxy maps the http request to the corresponding CoAP request and sends it to the UA-CoAP server in the network. After receiving the request, the UA-CoAP server sends a response to the UA-CoAP proxy. The proxy maps the CoAP response to the HTTP response and returns it to the UA client. For the UA client, the network proxy and conversion is transparent, in this way, the transfer of OPC UA in CoAP does not need to make any changes to the UA Client.

4.2. Direct transmission

The transmission of OPC UA supports TCP protocol and HTTP protocol, when security is ignored, OPC UA can be considered to support HTTP transmission. On the other hand, CoAP is seen as a simplified HTTP protocol so that it can be applied to resource-constrained network. Therefore, this document considers the use of CoAP to directly

transfer OPC UA messages. OPC UA packets are encoded in either binary or xml format, and the optional fields in the CoAP header specify parameters that support these two formats, and the option field in the CoAP header can specify parameters that support both formats. Therefore, according to the format specified by the CoAP header, the entire packet of the OPC UA can be encapsulated in the payload of the CoAP message for direct transmission, as shown in Figure 5. Noted that this method of transmission needs to be modified on the server side and the client side of the OPC UA according to CoAP.

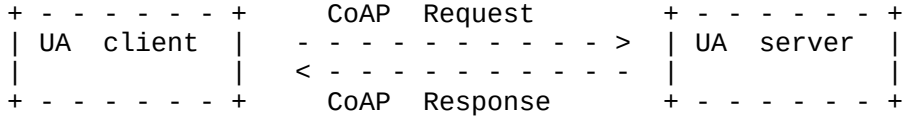


Figure 5: Direct transmission OPC UA based on CoAP

4.3. REST transmission for OPC UA

OPC UA is a set of data exchange specifications for industrial communication, the core of the OPC UA protocol is information modeling and transmission, which marks each node in the address space with a unique identifier. A series of state interactions are needed before performing normal reading and writing, including message handshaking, opening a secure channel, creating a session, activating a session, etc. Besides, some states also need to be maintained during read and write operations.

In OPC UA, each node has an independent identifier in the address space, and different types of nodes can establish contact with each other by reference. OPC UA defines a variety of services, and these services are fixed, the user cannot arbitrarily modify, each service is invoked through a single message, without relying on the previous message, the service response is also completed by a separate message and do not rely on other messages. The above features are in line with the REST architecture, due to CoAP is based on the REST architecture. Therefore, it is possible to simplify the interaction before the OPC UA performs the normal communication, and carry the OPC UA message by using the communication mode of the CoAP. Communication process is shown in Figure 6.

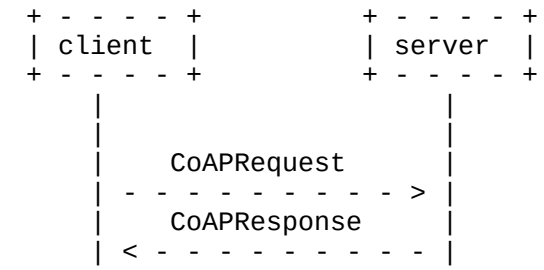


Figure 6: REST architecture communication of OPC UA

In Figure 2, the traditional OPC UA requires a series of interactions between normal read and write operations. Figure 6 shows that when using CoAP to carry OPC UA message, the interaction process is significantly reduced, which is conducive to the application of OPC UA in the restricted scene. The cost of simplifying the interaction process is that the secure channel number is set to 0 by default, how to conduct secure data interaction needs further discussion.

5. Publish subscription for OPC UA and CoAP

As an application sublayer, CoAP provides publish-subscribe functionality, primarily for resource or network-constrained scenarios. Introducing broker into the network [I-D.ietf-core-coap-pubsub], when a node needs to sleep, the node information is sent to the broker agent, when a node requests to obtain information of this node, the broker release function can provide information. OPC UA defines the publish-and-subscribe function as a service in the service set. The client initiates the subscription request directly to the server, and the server periodically sends the information to the client. Comparing the characteristics of the two protocols, it is found that each of them has its own advantages. Joint design can be conducted for constrained applications.

TODO.

6. Security Considerations

7. IANA Considerations

This memo includes no request to IANA.

8. References

8.1. Normative References

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol", RFC 7252, June 2014, <https://tools.ietf.org/html/rfc7252>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997, <https://tools.ietf.org/html/rfc2119>.

[RFC8075] Castellani, A., Loreto, S., and A. Rahman, "Guidelines for HTTP-to-CoAP Mapping Implementations", RFC 8075, November 2016, <https://tools.ietf.org/html/rfc8075>.

8.2. Informative References

[IEC TR 62541-1]
IEC, "OPC unified architecture-Part1:Overview and concepts-IEC 62541", 2016, <https://webstore.iec.ch/preview/info_iec62541-1%7Bed2.0%7Den.pdf>.

[I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-07 (work in progress), March 2017.

[I-D.wang-core-opcua-transmition-requirements]
Wang, H., Pu, C., Wang, P., Yang, Y., and D. Xiong, "Requirements Analysis for OPC UA over CoAP", draft-wang-core-opcua-transmition-requirements-00 (work in progress), December 2016.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol(CoAP)", draft-ietf-core-coap-pubsub-00 (work in progress), October 2016.

Authors' Addresses

Ping Wang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: wangping@cqupt.edu.cn

Chenggen Pu
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: mentospcg@163.com

Heng Wang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6248-7845
Email: wangheng@cqupt.edu.cn

Yi Yang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6248-7845
Email: 382991208@qq.com

Lun Shao
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: yjsslcqupt@163.com

Jianqiang Hou
Huawei Technologies CO.,LTD
101 Software Avenue,
Nanjing 210012
China

Phone: (86)-15852944235
Email: houjianqiang@huawei.com

Core
Internet Draft
Interned status: Standards Track
Expires: July 1, 2017

H. Wang
C. Pu
P. Wang
Y. Yang
D. Xiong
Chongqing University of
Posts and Telecommunications

December 28, 2016

Requirements Analysis for OPC UA over CoAP
draft-wang-core-opcua-transmition-requirements-00

Abstract

Industrial Internet of Things is an attractive application area for Constrained Application Protocol (CoAP). OPC Unified Architecture (OPC UA) defines a semantic-based information model for industrial control system that can satisfy the requirements of Industry 4.0 based on semantic information exchange. This document analyses requirements for OPC UA transmission over 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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on July 1, 2017.

Table of Contents

1. Introduction

CoAP is a web application protocol designed for resource constrained devices and limited networks that has been widely used in machine-to-machine (M2M) communications [RFC7252]. However, the purpose of applying CoAP to the Industrial Internet of Things (IIoT) is to provide connectivity for the devices. Whereas the communication of Industry 4.0 is not only based on data transmission, but also based on semantic information exchange. Driven by this, using CoAP in the IIoT, there is a need to provide good support for data transmission of the application layer in the automation field. According to the definition of Industry 4.0 for communication, CoAP needs to support the exchange of semantic information, namely the semantic information model. For the current protocols supporting semantic information model in the IIoT, the information model defined by OPC UA [IEC 62541-1] is very promising and its transmission mode is

similar to the transmission mode of CoAP, so it can be applied as a branch of the CoAP message payload.

2. Architecture of OPC UA over CoAP

With the vision of IIoT in mind, we believe that the architecture of OPC UA over CoAP can be mainly divided into the following two:

1) Figure 1 presents a logical layered structure of OPC UA Information Model over CoAP. In the transport layer, DTLS runs on top of UDP to secure transmission. Then, the middle layer utilizes the message mode defined in the CoAP protocol. Last, the information model of OPC UA [IEC TR 62541-5] is defined as an application of CoAP at the top. In such a hierarchical structure, the semantic-based data information in OPC UA can be transmitted in restricted scenarios, so that CoAP can meet the requirements of semantic information transmission.

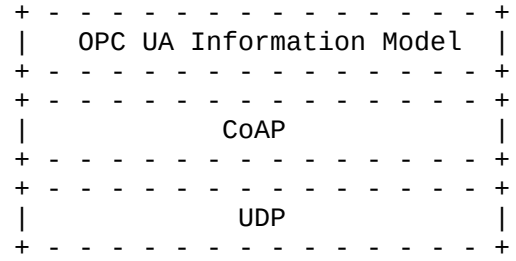


Figure 1: OPC UA Information Model over CoAP

2) In order to take full advantage of the service set defined by OPC UA, this document proposes the other architecture for OPC UA

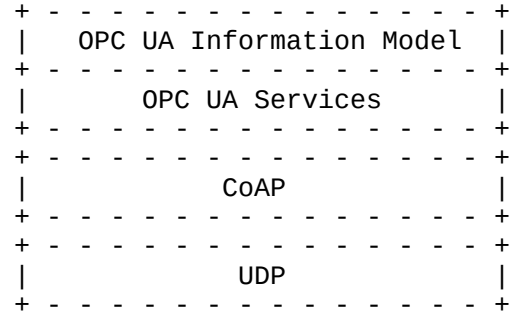


Figure 2: OPC UA Information Model and Services over CoAP

transmission over CoAP. As shown in Figure 2, the information model of OPC UA is defined as the application of CoAP, moreover, the connection establishment, creation session, publish/subscribe and other functions related to data information interaction are all implemented by the service set defined by OPC UA. CoAP is mainly responsible for the definition of message format and runs over UDP to keep the implementation lightweight.

3. Requirements for OPC UA over CoAP

3.1. Encoding

CoAP messages are encoded in a simple binary format that starts with a fixed-size 4-byte header. The header is followed by a variable-length Token value, which can be between 0 and 8 bytes long. Following the Token value comes a sequence of zero or more CoAP Options in Type-Length-Value (TLV) format, optionally followed by a payload that takes up the rest of the datagram. In addition, the OPC UA protocol coding mainly includes two ways that are binary and XML. Therefore, in order to transmit the information model of OPC UA over CoAP, specific frame formats of CoAP need to be designed to support two kinds of coding modes of OPC UA.

3.2. Application Sublayer Optimization

For information exchange, the document [draft-ietf-core-coap-pubsub-00] defines the corresponding application sublayer, OPC UA also defines a number of specific communication patterns. For example, in the publish/subscribe mode defined by OPC UA, when the client needs to obtain a data periodically, it will initiate a subscription request to the server. In addition, the server will send the data to the client periodically as it receives the request from the client successfully. Correspondingly, in the publish/subscribe specification of CoAP, it introduces Broker mechanism in which the client sends the state information to the Broker and the Broker provides storage and forwarding function to implement the publish/subscribe function. Comparing above two protocols, their achieving methods have a difference on communication mode of the publish/subscribe function. Therefore, it is necessary to optimize the application sublayer of CoAP to support some particular communication modes of OPC UA.

3.3. Consistency

The interactive model of CoAP is the client/server model. However, in M2M scenarios, CoAP entities often act as both servers and clients. Comparing to OPC UA, though the interactive model is also

the client/server model, there is a set of supported services in the OPC UA server. Consequently, for the great difference of the server definition of these two protocols, we need to tackle with the consistency and integration issues between the CoAP server and the OPC UA server.

3.4. Reliability

One of the main design goals of CoAP is to satisfy some special requirements such as communication in the constrained scenarios that address power consumption. Hence, in order to reduce network overhead and avoid network congestion, CoAP is designed to run over UDP, which is a good choice to achieve inter-network data transmissions in use of the IP architecture. However, UDP is a connectionless transport layer protocol that provides unreliable information transmission services. In the field of IIoT, we need to ensure the reliability of data transmission to avoid losing some important data information. Moreover, CoAP addresses transmission reliability by defining a message as requiring acknowledgment, obviously this is not enough to meet the high reliability requirements in the field of IIoT, so the reliability of COAP remains to be optimized.

4. Security Considerations

The security of CoAP includes four modes in which three modes implemented based on the Datagram Transport Layer Security (DTLS) except the non-security mode. However, the security architecture of OPC UA is built on the application layer and the communication layer above the transport layer. Specifically, the application layer adopts the authentication and authorization and the communication layer achieves the security of OPC UA [IEC TR 62541-2] through secure channel encryption. Though OPC UA has four modes, the security model of OPC UA is realized based on Transport Layer Security (TLS). Actually, DTLS is an addition to TLS to solve the unreliable transmission feature of UDP. Currently, some documents show that CoAP needs to support TLS. Therefore, the security of the two protocols can be implemented jointly.

5. IANA Considerations

This memo includes no request to IANA.

6. References

6.1. Normative References

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol", RFC 7252, June 2014, <https://tools.ietf.org/html/rfc7252>.

6.2. Informative References

[IEC TR 62541-1]
IEC, "OPC unified architecture-Part1: Overview and concepts-IEC 62541", 2016, <https://webstore.iec.ch/preview/info_iec62541-1%7Bed2.0%7Den.pdf>.

[IEC 62541-5]
IEC, "OPC unified architecture-Part5: Information Model-IEC 62541", 2015, <https://webstore.iec.ch/preview/info_iec62541-5%7Bed2.0%7Db.pdf>.

[I-D.koster-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-00 (work in progress), Qctober 2016.

[IEC TR 62541-2]
IEC, "OPC unified architecture-Part2: Security Model-IEC 62541", 2016, <https://webstore.iec.ch/preview/info_iec62541-2%7Bed2.0%7Db.pdf>.

Authors' Addresses

Heng Wang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6248-7845
Email: wangheng@cqupt.edu.cn

Chenggen Pu
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: mentospcg@163.com

Ping Wang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: wangping@cqupt.edu.cn

Yi Yang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: 15023705316@163.com

Daijing Xiong
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: 15111825021@163.com