Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 3, 2018

J. Arkko
Ericsson
C. Jennings
Cisco
Z. Shelby
Sensinode

October 30, 2017

Uniform Resource Names for Device Identifiers
draft-arkko-core-dev-urn-05

Abstract

This memo describes a new Uniform Resource Name (URN) namespace for hardware device identifiers. A general representation of device identity can be useful in many applications, such as in sensor data streams and storage, or equipment inventories. A URN-based representation can be easily passed along in any application that needs the information.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This memo describes a new Uniform Resource Name (URN) [RFC2141] [RFC3406] namespace for hardware device identifiers. A general representation of device identity can be useful in many applications, such as in sensor data streams and storage, or equipment inventories [RFC7252], [I-D.ietf-core-senml]. A URN-based representation can be easily passed along in any application that needs the information, as it fits in protocols mechanisms that are designed to carry URNs [RFC2616], [RFC3261], [RFC7252]. Finally, URNs can also be easily carried and stored in formats such as XML [W3C.REC-xml-19980210] or JSON [I-D.ietf-core-senml] [RFC4627]. Using URNs in these formats is often preferable as they are universally recognized, self-describing, and therefore avoid the need for agreeing to interpret an octet string as a specific form of a MAC address, for instance.

This memo defines identity URN types for situations where no such convenient type already exist. For instance, [RFC6920] defines cryptographic identifiers, [RFC7254] defines International Mobile station Equipment Identity (IMEI) identifiers for use with 3GPP cellular systems, and [I-D.atarius-dispatch-meid-urn] defines Mobile Equipment Identity (MEID) identifiers for use with 3GPP2 cellular systems. Those URN types should be employed when such identities are transported; this memo does not redefine these identifiers in any way.

Universally Unique IDentifier (UUID) URNs [RFC4122] are another alternative way for representing device identifiers, and already support MAC addresses as one of type of an identifier. However, UUIDs can be inconvenient in environments where it is important that the identifiers are as simple as possible and where additional requirements on stable storage, real-time clocks, and identifier length can be prohibitive. UUID-based identifiers are recommended for all general purpose uses when MAC addresses are available as identifiers. The device URN defined in this memo is recommended for constrained environments.

Future device identifier types can extend the device device URN type defined here, or define their own URNs.

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 rest of this memo is organized as follows. Section 3 defines the "DEV" URN type, and Section 4 defines subtypes for IEEE MAC-48, EUI-48 and EUI-64 addresses and 1-wire device identifiers. Section 5 gives examples. Section 6 discusses the security considerations of the new URN type. Finally, Section 7 specifies the IANA registration for the new URN type and sets requirements for subtype allocations within this type.

2. Requirements language

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

3. DEV URN Definition

Namespace ID: "dev" requested

Registration Information: This is the first registration of this namespace, 2011-08-27.

Registration version number: 1

Registration date: 2011-08-27

Declared registrant of the namespace: IETF and the CORE working group. Should the working group cease to exist, discussion should be directed to the general IETF discussion forums or the IESG.

Declaration of syntactic structure: The identifier is expressed in ASCII (UTF-8) characters and has a hierarchical structure as follows:

```
devurn = "urn:dev:" body componentpart
body = macbody / owbody / orgbody / otherbody
macbody = "mac:" hexstring
owbody = "ow:" hexstring
orgbody = "dn:" number ":" identifier
otherbody = subtype ":" identifier
subtype = ALPHA *(DIGIT / ALPHA)
identifier = 1*unreservednout
unreservednout = ALPHA / DIGIT / "-" / "."
componentpart = [ "_" component [ componentpart ]]
component = *1(DIGIT / ALPHA)
hexstring = hexbyte /
            hexbyte hexstring
hexbyte = hexdigit hexdigit
hexdigit = DIGIT / hexletter
hexletter = "a" / "b" / "c" / "d" / "e" / "f"
number = *1DIGIT
```

The above Augmented Backus-Naur Form (ABNF) uses the DIGIT and ALPHA rules defined in [RFC5234], which are not repeated here. The rule for unreserved is defined in Section 2.3 of [RFC3986].

The device identity namespace includes three subtypes, and more may be defined in the future as specified in Section 7.

The optional components following the hexstring are strings depicting individual aspects of a device. The specific strings and their semantics are up to the designers of the device, but could be used to refer to specific interfaces or functions within the device.

Relevant ancillary documentation: See Section 4.

Identifier uniqueness considerations: Device identifiers are generally expected to be unique, barring the accidental issue of multiple devices with the same identifiers.

Identifier persistence considerations: This URN type SHOULD only be used for persistent identifiers, such as hardware-based identifiers or cryptographic identifiers based on keys intended for long-term usage.

Process of identifier assignment: The process for identifier assignment is dependent on the used subtype, and documented in the specific subsection under Section 4.

Process for identifier resolution: The device identities are not expected to be globally resolvable. No identity resolution system is expected. Systems may perform local matching of identities to previously seen identities or configured information, however.

Rules for Lexical Equivalence: The lexical equivalence of the DEV URN is defined as an exact and case sensitive string match. Note that the two subtypes defined in this document use only lower case letters, however. Future types might use identifiers that require other encodings that require a more full-blown character set (such as BASE64), however.

Conformance with URN Syntax: The string representation of the device identity URN and of the MEID sub namespace is fully compatible with the URN syntax.

Validation Mechanism: Specific subtypes may be validated through mechanisms discussed in Section 4.

Scope: DEV URN is global in scope.

4. DEV URN Subtypes

4.1. MAC Addresses

DEV URNs of the "mac" subtype are based on the EUI-64 identifier [IEEE.EUI64] derived from a device with a built-in 64-bit EUI-64. The EUI-64 is formed from 24 or 36 bits of organization identifier followed by 40 or 28 bits of device-specific extension identifier assigned by that organization.

In the DEV URN "mac" subtype the hexstring is simply the full EUI-64 identifier represented as a hexadecimal string. It is always exactly 16 characters long.

MAC-48 and EUI-48 identifiers are also supported by the same DEV URN subtype. To convert a MAC-48 address to an EUI-64 identifier, The OUI of the Ethernet address (the first three octets) becomes the organization identifier of the EUI-64 (the first three octets). The fourth and fifth octets of the EUI are set to the fixed value FFFF hexadecimal. The last three octets of the Ethernet address become the last three octets of the EUI-64. The same process is used to convert an EUI-48 identifier, but the fixed value FFFE is used instead.

Identifier assignment for all of these identifiers rests within the IEEE.

4.2. 1-Wire Device Identifiers

The 1-Wire* system is a device communications bus system designed by Dallas Semiconductor Corporation. 1-Wire devices are identified by a 64-bit identifier that consists of 8 byte family code, 48 bit identifier unique within a family, and 8 bit CRC code [OW].

*) 1-Wire is a registered trademark.

In DEV URNs with the "ow" subtype the hexstring is a representation of the full 64 bit identifier as a hexadecimal string. It is always exactly 16 characters long. Note that the last two characters represent the 8-bit CRC code. Implementations MAY check the validity of this code.

Family code and identifier assignment for all 1-wire devices rests with the manufacturers.

4.3. Organization-Defined Identifiers

Device identifiers that have only a meaning within an organisation can also be used to represent vendor-specific or experimental identifiers or identifiers designed for use within the context of an organisation. Organisations are identified by the Private Enterprise Number [RFC2578].

5. Examples

The following three examples provide examples of MAC-based, 1-Wire, and Cryptographic identifiers:

```
urn:dev:mac:0024befffe804ff1           # The MAC address of
                                       # Jari's laptop

urn:dev:ow:10e2073a01080063            # The 1-Wire temperature
                                       # sensor in Jari's
                                       # kitchen

urn:dev:ow:264437f5000000ed_humidity    # The laundry sensor's
                                       # humidity part

urn:dev:ow:264437f5000000ed_temperature # The laundry sensor's
                                       # temperature part

urn:dev:org:32473:123456               # Device 123456 in
                                       # the RFC 5612 example
                                       # organisation
```

6. Security Considerations

On most devices, the user can display device identifiers. Depending on circumstances, device identifiers may or may not be modified or tampered by the user. An implementation of the DEV URN MUST NOT change these properties from what they were intended. In particular, a device identifier that is intended to be immutable should not become mutable as a part of implementing the DEV URN type. More generally, nothing in this memo should be construed to override what the relevant device specifications have already said about the identifiers.

Other devices in the same network may or may not be able to identify the device. For instance, on Ethernet network, the MAC address of a device is visible to all other devices.

The URNs generated according to the rules defined in this document result in long-term stable unique identifiers for the devices. Such identifiers may have privacy and security implications because they may enable correlating information about a specific device over a long period of time, location tracking, and device specific vulnerability exploitation [RFC7721]. Also, usually there is no easy way to change the identifier. Therefore these identifiers need to be used with care and especially care should be taken avoid leaking them outside of the system that is intended to use the identifiers.

7. IANA Considerations

This document requests the registration of a new URN namespace for "DEV", as described in Section 3.

Additional subtypes for DEV URNs can be defined through IETF Review or IESG Approval [RFC5226].

Such allocations are appropriate when there is a new namespace of some type of device identifiers, defined in stable fashion and with a publicly available specification that can be pointed to.

Note that the organisation (Section 4.3) device identifiers can also be used in some cases, at least as a temporary measure. It is preferrable, however, that long-term usage of a broadly employed device identifier be registered with IETF rather than used through the organisation device identifier type.

8. References

8.1. Normative References

[IEEE.EUI64]
IEEE, "Guidelines For 64-bit Global Identifier (EUI-64)", IEEE , unknown year, <http://standards.ieee.org/db/oui/tutorials/EUI64.html>.

[OW] IEEE, "Overview of 1-Wire(R) Technology and Its Use", MAXIM http://www.maxim-ic.com/app-notes/index.mvp/id/1796, June 2008, <http://www.maxim-ic.com/app-notes/index.mvp/id/1796>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, DOI 10.17487/RFC2141, May 1997, <https://www.rfc-editor.org/info/rfc2141>.

[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, <https://www.rfc-editor.org/info/rfc2578>.

[RFC3406] Daigle, L., van Gulik, D., Iannella, R., and P. Faltstrom, "Uniform Resource Names (URN) Namespace Definition Mechanisms", RFC 3406, DOI 10.17487/RFC3406, October 2002, <https://www.rfc-editor.org/info/rfc3406>.

[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, <https://www.rfc-editor.org/info/rfc3986>.

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

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

8.2. Informative References

[I-D.atarius-dispatch-meid-urn] Atarius, R., "A Uniform Resource Name Namespace for the Device Identity and the Mobile Equipment Identity (MEID)", draft-atarius-dispatch-meid-urn-13 (work in progress), October 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-10 (work in progress), July 2017.

[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, <https://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, <https://www.rfc-editor.org/info/rfc3261>.

[RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <https://www.rfc-editor.org/info/rfc3971>.

[RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, DOI 10.17487/RFC3972, March 2005, <https://www.rfc-editor.org/info/rfc3972>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006, <https://www.rfc-editor.org/info/rfc4627>.

[RFC5612] Eronen, P. and D. Harrington, "Enterprise Number for Documentation Use", RFC 5612, DOI 10.17487/RFC5612, August 2009, <https://www.rfc-editor.org/info/rfc5612>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

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

[RFC7254] Montemurro, M., Ed., Allen, A., McDonald, D., and P. Gosden, "A Uniform Resource Name Namespace for the Global System for Mobile Communications Association (GSMA) and the International Mobile station Equipment Identity (IMEI)", RFC 7254, DOI 10.17487/RFC7254, May 2014, <https://www.rfc-editor.org/info/rfc7254>.

[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, <https://www.rfc-editor.org/info/rfc7721>.

[W3C.REC-xml-19980210]
Sperberg-McQueen, C., Bray, T., and J. Paoli, "XML 1.0 Recommendation", World Wide Web Consortium FirstEdition REC-xml-19980210, February 1998, <http://www.w3.org/TR/1998/REC-xml-19980210>.

Appendix A. Changes from Previous Version

Version -05 made a change to the delimiter for parameters within a DEV URN. Given discussions on allowed character sets in SenML [I-D.ietf-core-senml], we would like to suggest that the "_" character be used instead of ";", to avoid the need to translate DEV URNs in SenML-formatted communications or files. However, this reverses the earlier decision to not use unreserved characters. This also means that device IDs cannot use "_" characters, and have to employ other characters instead. Feedback on this decision is sought.

Version -05 also introduced local or organisation-specific device identifiers. Organisations are identified by their PEN number (although we considered FQDNs as a potential alternative. The authors belive an organisation-specific device identifier type will make experiments and local use easier, but feedback on this point and the choice of PEN numbers vs. other possible organisation identifiers would be very welcome.

Version -05 also added some discussion of privacy concerns around long-term stable identifiers.

Finally, version -05 clarified the situations when new allocations within the registry of possible device identifier subtypes is appropriate.

Version -04 is a refresh, as the need and interest for this specification has re-emerged. And the editing author has emerged back to actual engineering from the depths of IETF administration.

Version -02 introduced several changes. The biggest change is that with the NI URNs [RFC6920], it was no longer necessary to define cryptographic identifiers in this specification. Another change was that we incorporated a more generic syntax for future extensions; non-hexstring identifiers can now also be supported, if some future device identifiers for some reason would, for instance, use BASE64. As a part of this change, we also changed the component part separator character from '-' to ';' so that the general format of the rest of the URN can employ the unreserved characters [RFC3986].

Appendix B. Acknowledgments

The authors would like to thank Ari Keranen, Stephen Farrell, Christer Holmberg, Peter Saint-Andre, Wouter Cloetens, and Ahmad Muhanna for interesting discussions in this problem space. We would also like to note prior documents that focused on specific device identifiers, such as [RFC7254] or [I-D.atarius-dispatch-meid-urn].

Authors' Addresses

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach@sensinode.com

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.

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).

```
             CoAP-REQ           CoAP-REQ
+------+   (SMS)   +-------+   (SMS)  +------+
|  A   | --------> | SMS-C | -------> |  B   |
|(cell)| <-------- |       | <------- |(cell)|
+------+ CoAP-RES  +-------+ CoAP-RES +------+
           (SMS)              (SMS)
```

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).

```
                        CIMD               CoAP-REQ
             +------+   SMPP    +-------+    (SMS)   +------+
             |  A   | --------> | SMS-C | ---------> |  B   |
             | (IP) | <-------- |       | <--------- |(cell)|
             +------+           +-------+  CoAP-RES  +------+
                                             (SMS)
```

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).

```
             HTTP-REQ                CIMD            CoAP-REQ
   +------+ (CoAP-DATA) +----------+  SMPP   +-----+    (SMS)   +------+
   |      |             | SMS      |  SS7    |SMS-C|            |      |
   |  A   | ----------> | Service  | ------> |  /  | ---------> |  B   |
   | (IP) | <---------- | Provider | <------ | HLR | <--------- |(cell)|
   +------+  HTTP-RES   +----------+         +-----+  CoAP-RES  +------+
            (CoAP-DATA)                                 (SMS)
```

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).

```
                       CoAP-REQ              CIMD
             +------+   (SMS)   +-------+    SMPP    +------+
             |  A   | --------> | SMS-C | ---------> |  B   |
             |(cell)| <-------- |       | <--------- | (IP) |
             +------+  CoAP-RES +-------+            +------+
                       (SMS)
```

Figure 4: Cellular and IP Communication

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

```
           CoAP-REQ              CIMD               HTTP-REQ
+------+   (SMS)    +--------+  SMPP  +----------+ (CoAP-DATA) +-----+
|      |            | SMS-C  |  SS7   | SMS      |             |     |
|  A   | ---------> |   /    | -----> | Service  | ----------> | B   |
|(cell)| <--------- |  HLR   | <----- | Provider | <---------- |(IP)|
+------+  CoAP-RES  +--------+        +----------+  HTTP-RES   +-----+
           (SMS)                                   (CoAP-DATA)
```

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:

```
 MS/CoAP CLIENT        Service Center         MS/CoAP SERVER
      |                        |                        |
      |   ---SMS-SUBMIT--->    |                        |
      | <-SMS-SUBMIT-REPORT--  |                        |
      |                        |                        |
      |                        |   --SMS-DELIVER--->    |
      |                        | <-SMS-DELIVER-REPORT-- |
      |                        |                        |
      | <-SMS-STATUS-REPORT--  |                        |
      |                        |                        |
```

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

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

Acknowledgements

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

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

Contributors

Appendix A has been contributed by Carsten Bormann.

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

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

Authors' Addresses

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

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

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

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

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

Email: kepeng.lkp@alibaba-inc.com

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

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

SACM Working Group H. Birkholz
Internet-Draft Fraunhofer SIT
Intended status: Informational T. Zhou
Expires: April 21, 2018 Huawei
X. Liu
Jabil
E. Voit
Cisco Systems
October 18, 2017

YANG Push Operations for CoMI
draft-birkholz-yang-push-coap-problemstatement-00

Abstract

This document provides a problem statement, derives an initial gap analysis and illustrates a first set of solution approaches in regard to augmenting YANG data stores based on the CoAP Management Interface with YANG Push capabilities. A binary transfer mechanism for YANG Subscribed Notifications addresses both the requirements of constrained-node networks and the need for semantic interoperability via self-descriptiveness of the corresponding data in motion.

Status of This Memo

Copyright Notice

Table of Contents

1. Context of the Problem

A binary transfer capability for YANG Subscribed Notifications [I-D.ietf-netconf-subscribed-notifications] based on YANG Push [I-D.ietf-netconf-yang-push] can be realized by using existing RFC and I-D work as building blocks. This section is intended to provide a corresponding overview of the existing ecosystem in order to identify gaps and therefore provide a problem statement.

1.1. Binary YANG transfer protocol

The CoAP Management Interface I-D (CoMI [I-D.ietf-core-comi]) defines operations for a YANG data store based on the Constrained Application Protocol (CoAP [RFC7252]). CoAP uses a request/response interaction model that is based on HTTP (similar to RESTCONF [RFC8040]) and allows for multiple transports, including UDP or TCP (see [I-D.ietf-core-coap-tcp-tls]). The Concise Binary Object Representation (CBOR [RFC7049]) is used for the serialization of data in motion in respect to CoAP operations and the data modeled with YANG [I-D.ietf-core-yang-cbor].

1.2. Device-Type Scope

[I-D.ietf-core-comi] states that CoAP "is designed for Machine to Machine (M2M) applications such as smart energy, smart city 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. 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."

In addition, [I-D.ietf-core-comi] highlights that "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 messages, 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."

In essence, via CoMI, a small sensor can emit a set of measurements as binary encoded YANG notifications, which would only add a minimal overhead to the data in motion, but would increase interoperability significantly due to the powerful and widely used semantics enabled by YANG (in contrast to a set of raw values that always require additional context information and imperative guidance to be managed and post-processed appropriately).

1.3. Subscriptions via CoAP

The CoAP pub/sub I-D defines a CoAP Subscribe operation [I-D.ietf-core-coap-pubsub] that is based on observing resources via the Observe option for the GET operation as defined in [RFC7641]. The CoAP pub/sub draft is intended to provide the capabilities and characteristics of MQTT via a CoAP based protocol. The only other CoAP operation that supports the Observe option is the FETCH operation defined in [RFC8132].

The Observe option creates a small corresponding state on the server side that eliminates the need for continuous polling of a resource via subsequent requests. Instead, subsequent responses including both the Observe option and using the token of the request that initiated the observation are returned when the observed resource changes. A subscription (i.e. the observe state retained on the server) can be discarded by the client by sending a correspond CoAP GET with Observe using an Observe parameter of 1 or simply by "forgeting" the observation and return a CoAP Reset after receiving a notification in the context of the subscription. A subscription can also be discarded by the server by sending a corresponding response that does not contain an Observe option.

The subscription used in CoAP pub/sub are used to subscribe to a topic provided by a CoAP broker REST API. YANG Push [I-D.ietf-netconf-yang-push] and corresponding YANG Subscribed Notifications are used to subscribe to data node updates provided by a YANG management interface. YANG subscriptions can include a filter expression (either a subtree expression or an XPATH expression). The encoding rules of XPATH expressions in CBOR are covered by [I-D.ietf-core-yang-cbor].

1.4. Configured Subscriptions and Call-Home

Configured subscriptions are basically static configuration that creates subscription state on the YANG data store when it is started and persists between boot-cycles without the need of a client to create that subscription state. In consequence, a configured subscription can result in unsolicited pushed notifications in respect to a YANG client.

A popular variant of the configured subscription as defined in [I-D.ietf-netconf-yang-push] is the Call Home procedure defined in [RFC8071]. In this approach, a Transport Layer application association with the YANG client is initiated by the YANG data store. After this "initial phase, in which the YANG server is acting like a client", the existing Transport Layer connection (or session, in case of, for example, TLS) is then used to the YANG client to initiate a subscription (i.e. the YANG client is initiating a dynamic subscription based on a pre-configured request retained and issued by the YANG data store).

1.5. Bootstrapping of Drop-Shipped Pledges

[I-D.ietf-anima-bootstrapping-keyinfra] highlights that effectively "to literally ’pull yourself up by the bootstraps’ is an impossible action. Similarly, the secure establishment of a key infrastructure without external help is also an impossibility."

According to [I-D.ietf-anima-bootstrapping-keyinfra] the bootstrapping approach Call-Home has problems and limitations, which (amongst others) the draft itself is trying to address:

- o the pledge requires realtime connectivity to the vendor service
- o the domain identity is exposed to the vendor service (this is a privacy concern)
- o the vendor is responsible for making the authorization decisions (this is a liability concern)

A Pledge in the context of [I-D.ietf-anima-bootstrapping-keyinfra] is "the prospective device, which has an identity installed by a third-party (e.g., vendor, manufacturer or integrator)."

A Pledge can be "drop-shipped", which refers to "the physical distribution of equipment containing the 'factory default' configuration to a final destination. In zero-touch scenarios there is no staging or pre-configuration during drop-ship."

In the scope of Call-Home as a part of YANG Push, either the factory default configuration of a drop-shipped Pledge that is a YANG data store would require to include the "home to Call Home" configuration or it has to be configured locally.

[I-D.ietf-netconf-zerotouch] is intended to provide more flexibility to the Call-Home procedure already - by allowing to stage connection attempts to a locally administered network and if that fails fall back to connecting to a remotely administered network. Alas, [I-D.ietf-netconf-zerotouch] is either prone to the same limitations as cited above or requires local configuration in order to find the home to Call-Home.

The "Join Registrar" defined by [I-D.ietf-anima-bootstrapping-keyinfra] mitigates the cited problems and limitation by introducing "a representative of the domain that is configured, perhaps autonomically, to decide whether a new device is allowed to join the domain. The administrator of the domain interfaces with a Join Registrar (and Coordinator) to control this process. Typically a Join Registrar is "inside" its domain."

2. Summary of the Problem Statement

Currently, the following gaps are identified:

- o no CoAP Subscribe procedure for dynamic YANG subscriptions is standardized that is able to convey a filter expression and potentially other metadata required in the context of a YANG Subscribed Notifications application association. Analogously, new payload types (e.g. a FETCH payload media-type) have to be defined.

- no CoAP Call Home feature is standardized to support a popular variant of configured YANG subscriptions.

- no general Call Home mechanism is standardized that enables the discovery of "a home to Call Home" or that would be able to deal with "changing homes" in a dynamic but secure manner.

In addition to the identified gaps, the semantics of metadata - if there are any - that have to be conveyed to or from a YANG data store in order to subscribe to a (filtered) YANG module or data node are not identified.

The problem statement could be summarized as follows:

"There is no complete solution based on CoAP to enable a freshly unpacked YANG data store ("drop-shipped pledge", e.g. the cliche light bulb) to discover an appropriate home it can than Call-Home to in a secure and trusted manner in order to push (un-)solicited subscribed notifications."

3. Potential Approaches and Solutions

There are multiple approaches that could lead to viable solutions that address the identified gaps. The following sections illustrate the general solution context and some of the most promising approaches.

3.1. YANG subscription variants

A YANG Push update subscription service both provides support for dynamic subscription (i.e. subscription state created by a client request, allowing for solicited push notifications in the context of an up-time cycle of the server) and configured subscription (i.e. subscription configuration retained on the server, allowing for unsolicited push notifications across up-time cycles of the server).

3.2. YANG Push via CoAP

The two CoAP operations that enable a subscription mechanism are GET and FETCH (i.e. by supporting the Observe option). Both operations are viable candidates for creating a CoAP-based YANG Push mechanism for CoMI.

3.3. Dynamic Subscriptions

Using CoAP, the client issuing the initial subscription request creates the subscription state. Examples are the GET or FETCH operation including an Observe option using an Observe parameter of 0 (zero).

3.3.1. YANG Push via GET

This usage scenario requires two consecutive operations. It is not possible to transfer a filter expression included in a GET operation. In consequence, a POST operation on a collection resource has to be conducted in order to convey a filter expression to the YANG data store, allowing it to return an URI that contains the data node information filtered in respect to the posted filter expression (encoded in CBOR).

This variant allows for multiple clients to observe a specific filtered data node without conducting a POST operation, if the corresponding URI is made known to other clients that did not conduct the POST operation or, for example, is canonically linked to/ derivable from a filter expression.

3.3.2. YANG Push via FETCH

This usage scenario requires only one operation. A FETCH operation can include a body that is capable to contain a filter expression and potentially other metadata that might be required to establish a suitable subscription state on the YANG data store.

It might be possible that this variant could introduce a slight delay in respect to response time if providing a filtered resource requires a lot of computation time on a constrained device. I.e. the resource cannot be prepared "beforehand".

3.4. Configured Subscriptions

Using CoAP, the server retains configuration that creates subscription state when the YANG data store is started. The client has to have or gain knowledge of the CoAP tokens that are included in the responses created in the context of the subscription state create from server configuration.

3.4.1. Retaining the Content of a GET Operation as Configuration

This usage scenario "mimics" the receiving of a subscription request by storing the corresponding information that are relevant for creating a subscription state as configuration on the YANG data store. I.e. the configuration would be including the YANG client IP address and the CoAP token to be used in the responses that convey the subscribed notifications.

This variant requires that the client also knows or gains knowledge of the corresponding CoAP token in order to not discard the incoming responses.

3.4.2. Call Home via CoAP

This usage scenario defines the Call Home procedure standardized in [RFC8071] as an additional capability of CoAP. DTLS or TLS state is initiated by the YANG data store and triggers a dynamic subscription procedure of the YANG client using the session initiated by the YANG data store.

3.4.3. Dynamic Home Discovery

This usage scenario is based on the Bootstrapping Remote Secure Key Infrastructures I-D [I-D.ietf-anima-bootstrapping-keyinfra] and EST over secure CoAP I-D [I-D.vanderstok-ace-coap-est] and requires the standardization of a general use of Join Registrars in the context of YANG data stores that support YANG Push via static subscriptions.

4. IANA considerations

This document includes no requests to IANA, but solutions drafts incubated via this document might.

5. Security Considerations

This document includes no security considerations, but solution drafts incubated via this document will.

6. Acknowledgements

Carsten Bormann, Klaus Hartke, Michel Veillette

7. Change Log

First version -00

8. Normative References

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-08 (work in progress), October 2017.

[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-02 (work in progress), July 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-09 (work in progress), May 2017.

[I-D.ietf-core-comi]
Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-01 (work in progress), July 2017.

[I-D.ietf-core-sid]
Veillette, M., Pelov, A., Turner, R., Minaburo, A., and A. Somaraju, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-01 (work in progress), May 2017.

[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-05 (work in progress), August 2017.

[I-D.ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Custom Subscription to Event Notifications", draft-ietf-netconf-subscribed-notifications-05 (work in progress), October 2017.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-10 (work in progress), October 2017.

[I-D.ietf-netconf-zerotouch]
Watsen, K., Abrahamsson, M., and I. Farrer, "Zero Touch Provisioning for NETCONF or RESTCONF based Management", draft-ietf-netconf-zerotouch-17 (work in progress), September 2017.

[I-D.vanderstok-ace-coap-est]
Kumar, S., Stok, P., Kampanakis, P., Furuhed, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-vanderstok-ace-coap-est-02 (work in progress), June 2017.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7641>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <https://www.rfc-editor.org/info/rfc8071>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Tianran Zhou
Huawei
156 Beiqing Rd.
Beijing, Haidian District
China

Email: zhoutianran@huawei.com

Xufeng Liu
Jabil
8281 Greensboro Drive, Suite 200
McLean VA 22102
USA

Email: Xufeng_Liu@jabil.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: February 23, 2018

P. van der Stok
consultant
K. Hartke
Universitaet Bremen TZI

August 22, 2017

The 'Pending' Response Code for the Constrained Application Protocol (CoAP)

draft-hartke-core-pending-01

Abstract

This document proposes a new CoAP response code, 2.__ 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.__ response can include status information and indicate a location where the result will become available.

Note

The string "2.__" is a placeholder for the CoAP response code that will be assigned by IANA on completion of this document.

Status of This Memo

Copyright Notice

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. For example, a CoAP-over-UDP 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 the time 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 actually 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 for this is the "possibly long" authorization request specified in EST-coaps [I-D.vanderstok-ace-coap-est].

This document proposes a new CoAP response code, 2.__ 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.__ (Pending) response code adapts this status to CoAP. The 2.__ (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.__ Pending

A 2.__ (Pending) response in reply to a GET request indicates that the target resource exists but a representation of the resource is not 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 as defined in [RFC7641] to be notified when the representation becomes available (see Section 2.1).

A 2.__ (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 the GET request method. The Max-Age Option indicates how long the client should wait before making the GET request.

A 2.__ (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.__ (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.__ (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.__ (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 a resource [RFC7641] for which no representation is available yet, the server MAY send one or more 2.__ (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.

```
            ___________       ___________       ___________
           |           |     |           |     |           |
      ---->|   2.__    |---->|  2.05 /   |---->|  4.xx /   |
           | Pending   |     |   2.03    |     |   5.xx    |
           |___________|     |___________|     |___________|
              ^   \ \           ^    \            ^
               \__/  \           \___/           /
                      \_________________________/
```

Figure 1: Sequence of Notifications

Unless the server is unwilling to add the client to the list of observers, each 2.__ (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.__ (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.__ (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.__ (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.__ (Pending) response code to the "CoAP Response Codes" registry.

Code	Description	Reference
2.__	Pending	[RFCXXXX]

Table 1: New CoAP Response Codes

[[IANA: Please assign a code point in the range 2.06-2.30.]] [[RFC Editor: Please replace every occurrence of "2.__" in this document with the assigned code point and remove this paragraph before publication.]]

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., Stok, P., Kampanakis, P., Furuhed, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-vanderstok-ace-coap-est-02 (work in progress), June 2017.

[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 A. Hope-Bailie
Internet-Draft Ripple
Intended status: Standards Track October 31, 2017
Expires: May 4, 2018

HTTP-Payments
draft-hope-bailie-http-payments-00

Abstract

HTTP-Payments describes a mechanism for passing a standardized payment request in the headers of an HTTP 402 response and the expected behaviour of HTTP clients that receive such a response.

Feedback

This specification is an early experiment in bringing the work of the W3C Web Payments working group to the HTTP protocol. It is maintained at https://github.com/adrianhopebailie/http-payments [1].

The work is inspired by work in the Interledger community on [HTTP-ILP]

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The W3C Web Payments working group has defined a Web Platform API that is being widely deployed to browsers for requesting a payment. The PaymentRequest API [W3C.CR-payment-request-20170921], defines an interface for a website to pass a payment request to the user agent via this API.

The user agent will then, through interaction with the user, complete or reject the requested payment.

HTTP-Payments describes a manner in which an HTTP server can request payment from a client in the same manner as a website would from a user agent using the W3C APIs.

The critical portion of the payment request is the set of, one or more, supported payment methods and associated payment-method-specific data. HTTP-Payments defines a mechaism by which these are expressed in the response headers of an HTTP request for which the server requires a payment.

In the website and user-agent scenario, when handling the payment request, the user-agent will prompt the user to pick one of the supported payment methods and will then handle the payment in a manner that is appropriate for that payment method. In an HTTP-Payment, the HTTP client will perform this function, likely with no user interaction.

2. Terminology

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

3. Payment Methods

A payment method is a way that the payee can be paid. Examples include, via credit card, bank wire transfer, or Bitcoin.

A payment method is identified by a payment method identifier as specified in the Payment Method Identifiers specification [2]. This is either a standardized short-string, identified in a registry maintained by the W3C Web Payments WG, or a URL.

The most common case will be for the URL form to be used. In cases where there is no authority responsible for the payment method that can host the payment method URL, the WG will consider adding a new identifier for the payment method to the registry.

Payment methods define the data that the payer and payee need to exchange, to complete a payment, and the process by which this occurs.

4. HTTP Status Code 402

The HTTP Status Code, 402 (Payment Required) is currently defined in [RFC7231] as "reserved for future use". Using HTTP-Payment a service MAY respond to any request with the 402 response code and use the "Pay" header to specify the payment request details.

4.1. The "Pay" Header

The body of the "Pay" header is defined as follows:

```
Pay: <payment-method-identifier> <amount> <address> <payment-method-data>
```

Multiple "Pay" headers MAY be present in an HTTP 402 response.

The fields in the header are:

- o payment-method: The payment method identifier for the accepted payment method. Either a standardized short-string or a URL.

- o amount: The amount that must be paid, expressed as an integer. The currency, scale and precision of the destination account are expected to be expressed in the account address.

- o address: A payment-method specific payee address. For example, if the payment method is Bitcoin this would be a Bitcoin address.

- o payment-method-data: Payment method specific data. This is either a URI identifying the data or, if it is small enough, is the data itself, BASE64URL encoded as described in [RFC4648], Section 5.

4.2. The "Pay-Token" Header

An HTTP client that makes a paid-HTTP request, after paying for the request to be processed, MAY attach a "Pay-Token" header with a token referencing the payment.

This mechanism can be employed by services wishing to accept payments without binding these to an HTTP session.

4.3. The "Pay-Balance" Header

Ann HTTP Service that accepts payments may respond to any request with a "Pay-Balance" header. This contains an integer indicating the current balance of paid credit the client has with the HTTP service.

4.4. Flow

Upon receipt of a 402 response, an HTTP client MUST look for any "Pay" headers and parse these. The client can discard all headers for which it is not equipped to make a payment (i.e. filter on payment-method-identifier)

The client MUST then select the header that is preferred for processing based upon external interactions (such as with a human user) or pre-configured rules. The client MUST attempt to make a payment using the payment method identified in the header, for the amount specified, and to the destination address specified.

The payment-method specific data SHOULD be sufficient for the system processing the payment to reconcile the payment with the original HTTP request.

The client SHOULD receive a token in return for completeing the payment. If the payment method used does return a token to the payer, it MUST pass this token in subsequent HTTP requests.

The token MUST be passed in the "Pay-Token" header, BASE64URL encoded as described in [RFC4648], Section 5.

The HTTP service MUST process the "Pay-Token" header and use this to reconcile this HTTP request with the payment received prior.

4.5. Example

Client requests access to a paid resource:

```
POST /upload HTTP/1.1
Host: myservice.example
```

Server responds with payment request (and optionally indicates that the client has a zero balance):

```
HTTP/1.1 402 Payment Required
Pay: http://interledger.org 10 us.nexus.ankita.˜recv.filepay SkTcFTZCBKgP6A6QOUV
cwWCCgYIP4rJPHlIzreavHdU
Pay-Balance: 0
```

Client makes the payment through an appropriate payment side-channel and then attempts the request again:

```
POST /upload HTTP/1.1
Host: myservice.example
Pay-Token: 7y0SfeN7lCuq0GFF5UsMYZofIjJ7LrvPvsePVWSv450
```

Server responds:

```
HTTP/1.1 200 Success
Pay-Balance: 0
```

5. References

5.1. Normative References

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[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, <https://www.rfc-editor.org/info/rfc7231>.

[W3C.CR-payment-method-id-20170914]
Bateman, A., Koch, Z., McElmurry, R., and M. Caceres, "Payment Method Identifiers", World Wide Web Consortium CR CR-payment-method-id-20170914, September 2017, <https://www.w3.org/TR/2017/CR-payment-method-id-20170914>.

[W3C.CR-payment-request-20170921]
Bateman, A., Koch, Z., McElmurry, R., Denicola, D., and M. Caceres, "Payment Request API", World Wide Web Consortium CR CR-payment-request-20170921, September 2017, <https://www.w3.org/TR/2017/CR-payment-request-20170921>.

5.2. Informative References

[HTTP-ILP]
Interledger Community Group, "HTTP-ILP", October 2017, <https://github.com/interledger/rfcs/blob/58d8dcb015b160a381313126fa3065c64406db05/0014-http-ilp/0014-http-ilp.md>.

5.3. URIs

[1] https://github.com/adrianhopebailie/http-payments

[2] W3C.CR-payment-method-id-20170914

Appendix A. Security Considerations

TBD

Appendix B. IANA Considerations

B.1. Payment Method Identifier Short-string Registry

The W3C maintains a registry of standardized short-string payment method identifiers as part of the [Payment Method Identifier] specification. If standardized short-string identifiers are to be used for HTTP-Payments this may be better served as an IANA registry.

Author’s Address

Adrian Hope-Bailie
Ripple
315 Montgomery Street
San Francisco, CA 94104
US

Phone: -----------------
Email: adrian@ripple.com
URI: https://www.ripple.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2018

M. Koster
SmartThings
A. Keranen
J. Jimenez
Ericsson
July 4, 2017

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-ietf-core-coap-pubsub-02

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

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 required features of this specification.

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 and pending notifications.

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 REST 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.

```
Clients       pubsub         Broker
+-------+         |
| CoAP  |         |
|pubsub |---------|------+
|Client |         |      |    +-------+
+-------+         |      +----| CoAP  |
                  |           |pubsub |
+-------+         |      +----|Broker |
| CoAP  |         |      |    +-------+
|pubsub |---------|------+
|Client |         |
+-------+         |
```

Figure 1: CoAP pubsub Architecture

3.2. CoAP pubsub Broker

A CoAP pubsub Broker is a CoAP Server that exposes a REST 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 REST API defined in this document. 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 data 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.

```
  Peer        pubsub         Peer
+-------+        |         +-------+
| CoAP  |        |         | CoAP  |
|pubsub |---------|---------|pubsub |
|Client |        |         |Broker |
+-------+        |         +-------+
| CoAP  |        |         | CoAP  |
|pubsub |---------|---------|pubsub |
|Broker |        |         |Client |
+-------+        |         +-------+
```

Figure 2: Brokerless pubsub

4. CoAP pubsub REST API

This section defines the REST 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 SHOULD support the DISCOVERY, CREATE, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, READ, and REMOVE operations defined in this section. Optimized implementations MAY support a subset of the operations as required by particular constrained use cases.

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 the entry point of its pubsub API at its .well-known/core location [RFC6690]. A CoAP pubsub broker MAY register its pubsub REST API entry point 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 entry point 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: {+ps}/{+topic}{?q*}

URI Template Variables: ps := pubsub REST API entry point (optional). The entry point of the pubsub REST 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.

```
Client                                          Broker
  |                                               |
  | ------ GET /.well-known/core?rt=core.ps ---->>|
  | -- Content-Format: application/link-format ---|
  |                                               |
  | <<--- 2.05 Content                            |
  | </ps/>;rt=core.ps;rt=core.ps.discover;ct=40 --|
  |                                               |
```

Figure 3: Example of DISCOVER pubsub function

```
          Client                                          Broker
            |                                               |
            | ---------- GET /ps/?rt="temperature" ------->>|
            |    Content-Format: application/link-format    |
            |                                               |
            | <<-- 2.05 Content                             |
            |   </ps/currentTemp>;rt="temperature";ct=50 ---|
            |                                               |
```

Figure 4: Example of DISCOVER topic

```
          Client                                          Broker
            |                                               |
            | -------- GET /.well-known/core?ct=50 ------->>|
            |    Content-Format: application/link-format    |
            |                                               |
            | <<-- 2.05 Content                             |
            |   </ps/currentTemp>;rt="temperature";ct=50 ---|
            |                                               |
```

Figure 5: Example of DISCOVER topic

4.2. CREATE

A CoAP pubsub broker MAY allow Clients to 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}{?q*}

URI Template Variables: ps := pubsub REST API entry point (optional). The entry point of the pubsub REST 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

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.

```
          Client                                          Broker
            |                                               |
            | ---------- POST /ps/ "<topic1>;ct=50" -------->|
            |                                               |
            | <---------------- 2.01 Created ---------------|
            |                Location: /ps/topic1           |
            |                                               |
```

Figure 6: Example of CREATE topic

```
          Client                                          Broker
            |                                               |
            | ------- POST /ps/ "<mainTopic>;ct=40" ------->|
            |                                               |
            | <---------------- 2.01 Created ---------------|
            |              Location: /ps/mainTopic/         |
            |                                               |
            | --- POST /ps/mainTopic/ "<subTopic>;ct=50" -->|
            |                                               |
            | <---------------- 2.01 Created ---------------|
            |         Location: /ps/mainTopic/subTopic      |
            |                                               |
            |                                               |
```

Figure 7: Example of CREATE sub-topic

4.3. PUBLISH

A CoAP pubsub broker MAY allow clients to PUBLISH to 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 shall respond with the 2.04 Changed status code.

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}{?q*}

URI Template Variables: ps := pubsub REST API entry point (optional). The entry point of the pubsub REST 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: 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.

```
Client                                          Broker
  |                                               |
  | ---------- PUT /ps/topic1 "1033.3"  --------> |
  |                                               |
  |                                               |
  | <--------------- 2.04 Changed---------------- |
  |                                               |
```

Figure 8: Example of PUBLISH

```
Client                                          Broker
  |                                               |
  | -------- PUT /ps/exa/mpl/e "1033.3"  -------> |
  |                                               |
  |                                               |
  | <--------------- 2.01 Created---------------- |
  |             Location: /ps/exa/mpl/e           |
  |                                               |
```

Figure 9: Example of CREATE on PUBLISH

4.4. SUBSCRIBE

A CoAP pubsub broker MAY allow Clients to 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 the Observe option set to 0 (zero). 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 "2.07 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}{?q*}

URI Template Variables: ps := pubsub REST API entry point (optional). The entry point of the pubsub REST 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.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful subscribe, current value included

Success: 2.07 "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.

```
 Client1   Client2                                          Broker
    |          |                   Subscribe                   |
    |          | ----- GET /ps/topic1 Observe:0 Token:XX ----> |
    |          |                                               |
    |          | <---------- 2.05 Content Observe:10---------- |
    |          |                                               |
    |          |                                               |
    |          |                    Publish                    |
    | ---------|----------- PUT /ps/topic1 "1033.3"  --------> |
    |          |                   Notify                      |
    |          | <---------- 2.05 Content Observe:11 --------- |
    |          |                                               |
```

Figure 10: Example of SUBSCRIBE

4.5. UNSUBSCRIBE

If a CoAP pubsub broker allows clients to SUBSCRIBE to topics on the broker, it MUST allow Clients to 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}{?q*}

URI Template Variables: ps := pubsub REST API entry point (optional). The entry point of the pubsub REST 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.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful unsubscribe, current value included

Success: 2.07 "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.

```
Client                                          Broker
  |                                               |
  | ----- GET /ps/topic1 Observe:1 Token:XX ----> |
  |                                               |
  | <------------- 2.05 Content ----------------- |
  |                                               |
```

Figure 11: Example of UNSUBSCRIBE

4.6. READ

A CoAP pubsub broker MAY accept Read requests on a topic using the the CoAP GET method 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 "2.07 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}{?q*}

URI Template Variables: ps := pubsub REST API entry point (optional). The entry point of the pubsub REST 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.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful READ, current value included

Success: 2.07 "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.

```
  Client1   Client2                                          Broker
     |          |                     Read                     |
     |          | --------------- GET /ps/topic1 -------------> |
     |          |                                               |
     |          | <---------- 2.05 Content "1007.1"------------ |
     |          |                                               |
     |          |                                               |
     |          |                    Publish                    |
     | ---------|----------- PUT /ps/topic1 "1033.3"  --------> |
     |          |                                               |
     |          |                                               |
     |          |                     Read                      |
     |          | --------------- GET /ps/topic1 -------------> |
     |          |                                               |
     |          | <----------- 2.05 Content "1033.3" ---------- |
     |          |                                               |
```

Figure 12: Example of READ

4.7. REMOVE

A CoAP pubsub broker MAY allow clientsremove a topics from the broker using the CoAP Delete method on the URI of the topic. The CoAP pubsub Broker MUST return "2.02 Deleted" if the removal 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}{?q*}

URI Template Variables: ps := pubsub REST API entry point (optional). The entry point of the pubsub REST 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: 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.

```
        Client                                           Broker
          |                                                 |
          | ------------- DELETE /ps/topic1 ------------> |
          |                                                 |
          |                                                 |
          | <-------------- 2.02 Deleted ---------------- |
          |                                                 |
```

Figure 13: Example of REMOVE

5. CoAP pubsub Operation with Resource Directory

A CoAP pubsub Broker may register the base URI, which is the REST API entry point for a pubsub service, 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 as described in [I-D.palombini-ace-coap-pubsub-profile]. 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 '2.07'

- Response Code: 2.07
- Description: Add No Content response to GET to the existing definition of the 2.07 response code.
- Reference: [[This document]]
- Notes: The server sends this code to the client to indicate that the request was valid and accepted, but the responce may contain an empty payload. It is comparable to and may be proxied with the http 204 No Content status code.

9.4. Response Code value '4.29'

- Response Code: 4.29

- o Description: This error code is used by a server to indicate that a client is making too many requests on a resource.

- o Reference: [[This document]]

- o 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

[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

[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.palombini-ace-coap-pubsub-profile]
Palombini, F., "CoAP Pub-Sub Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-palombini-ace-coap-pubsub-profile-00 (work in progress), March 2017.

[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.

[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: 7641, 7959 (if approved)
Intended status: Standards Track
Expires: May 3, 2018

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
October 30, 2017

CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets
draft-ietf-core-coap-tcp-tls-10

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 7641 for use with these transports and RFC 7959 to enable the use of larger messages over a reliable transport.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP [RFC0768] can be used unimpeded, as can the Datagram Transport Layer Security protocol (DTLS [RFC6347]) over UDP. The use of CoAP over UDP is focused on simplicity, has a low code footprint, and a small over-the-wire message size.

The primary reason for introducing CoAP over TCP [RFC0793] and TLS [RFC5246] is that some networks do not forward UDP packets. Complete blocking of UDP happens in between about 2% and 4% of terrestrial access networks, according to [EK2016]. UDP impairment is especially concentrated in enterprise networks and networks in geographic regions with otherwise challenged connectivity. Some networks also rate-limit UDP traffic, as reported in [BK2015] and deployment investigations related to the standardization of QUIC revealed numbers around 0.3 % [SW2016].

The introduction of CoAP over TCP also leads to some additional effects that may be desirable in a specific deployment:

- Where NATs are present along the communication path, CoAP over TCP leads to different NAT traversal behavior than CoAP over UDP. 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 lifecycle. UDP, on the other hand, does not provide such information to a NAT and timeouts tend to be much shorter [HomeGateway]. According to [HomeGateway] the mean for TCP and UDP NAT binding timeouts is 386 minutes (TCP) and 160 seconds (UDP). Shorter timeout values require keepalive messages to be sent more frequently. Hence, the use of CoAP over TCP requires less frequent transmission of keep-alive messages.

- TCP utilizes more sophisticated congestion and flow control mechanisms than the default mechanisms provided by CoAP over UDP, which is useful for the transfer of larger payloads. (Work is, however, ongoing to add advanced congestion control to CoAP over UDP as well, see [I-D.ietf-core-cocoa].)

Note that the use of CoAP over UDP (and CoAP over DTLS over UDP) is still the recommended transport for use in constrained node networks, particularly when used in concert with blockwise transfer. CoAP over TCP is applicable for those cases where the networking infrastructure leaves no other choice. The use of CoAP over TCP leads to a larger code size, more roundtrips, increased RAM requirements and larger packet sizes. Developers implementing CoAP over TCP are encouraged to consult [I-D.gomez-lwig-tcp-constrained-node-networks] for guidance on low-footprint TCP implementations for IoT devices.

Standards based on CoAP such as Lightweight Machine to Machine [LWM2M] currently use CoAP over UDP as a transport; adding support for CoAP over TCP enables them to address the issues above for specific deployments and to protect investments in existing CoAP implementations and deployments.

Although HTTP/2 could also potentially address the need for enterprise firewall traversal, there would be additional costs and delays introduced by such a transition from CoAP to HTTP/2. Currently, there are also fewer HTTP/2 implementations available for constrained devices in comparison to CoAP. Since CoAP also support group communication using IP layer multicast and unreliable communication IoT devices would have to support HTTP/2 in addition to CoAP.

Furthermore, 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.

Finally, CoAP applications running inside a web browser may be without access to connectivity other than HTTP. In this case, the WebSocket protocol [RFC6455] may be used to transport CoAP requests and responses, as opposed to cross-proxying them via HTTP to an HTTP-to-CoAP cross-proxy. This preserves the functionality of CoAP without translation, in particular the Observe mechanism [RFC7641].

To address the above-mentioned deployment 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:

```
+--------------------------------+
|          Application           |
+--------------------------------+
+--------------------------------+
|  Requests/Responses/Signaling  |  CoAP (RFC 7252) / This Document
|--------------------------------|
|        Message Framing         |  This Document
+--------------------------------+
|      Reliable Transport        |
+--------------------------------+
```

Figure 1: Layering of CoAP over Reliable Transports

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.

Section 7 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.)

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.

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.

The management of the connections is left to the application, i.e., the present specification does not describe how an application decides to open a connection or to re-open another one in the presence of failures (or what it would deem to be a failure, see also Section 5.4). In particular, the Connection Initiator need not be the client of the first request placed on the connection.

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.

```
        CoAP Client       CoAP Server  CoAP Client       CoAP Server
            |                    |         |                    |
            |   CON [0xbc90]     |         | (-------) [------] |
            | GET /temperature   |         | GET /temperature   |
            |   (Token 0x71)     |         |   (Token 0x71)     |
            +------------------->|         +------------------->|
            |                    |         |                    |
            |   ACK [0xbc90]     |         | (-------) [------] |
            |   2.05 Content     |         |   2.05 Content     |
            |   (Token 0x71)     |         |   (Token 0x71)     |
            |     "22.5 C"       |         |     "22.5 C"       |
            |<-------------------+         |<-------------------+
            |                    |         |                    |

                CoAP over UDP                 CoAP over reliable
                                                  transport
```

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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 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:

- o 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Len  |  TKL  | Extended Length (if any, as chosen by Len) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Code     | Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: CoAP frame for reliable transports

Length (Len): 4-bit unsigned integer. A value between 0 and 12 inclusive 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]).

For simplicity, a Payload Marker (0xFF) is shown in Figure 4; the Payload Marker indicates the start of the optional payload and is absent for zero-length payloads (see Section 3 of [RFC7252]). (If present, the Payload Marker is included in the message length, which counts from the start of the Options field to the end of the Payload field.)

For example: A CoAP message just containing a 2.03 code with the token 7f and no options or payload is encoded as shown in Figure 5.

```
 0                   1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      0x01     |      0x43     |      0x7f     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 Len   =    0 ------>  0x01
 TKL   =    1 ___/
 Code  =  2.03     --> 0x43
 Token =               0x7f
```

Figure 5: CoAP message with no options or payload

The semantics of the other CoAP header fields are left unchanged.

3.3. Message Transmission

Once a connection is established, each endpoint 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 after its initial CSM without waiting to receive the Connection Acceptor’s CSM; however, it is important to note that the Connection Acceptor’s CSM might indicate capabilities that impact how the initiator is expected to communicate with the acceptor. For example, the acceptor CSM could indicate a Max-Message-Size option (see Section 5.3.1) that is smaller than the base value (1152) in order to limit both buffering requirements and head-of-line blocking.

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 Section 7.4) 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 6). 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.

```
            ___________                            ___________
           |           |                          |           |
           |          _|___      requests      ___|_          |
           |   CoAP  /  \  \  -------------->  /  /  \  CoAP   |
           |  Client \__/__/  <--------------  \__\__/ Server  |
           |           |         responses        |           |
           |___________|                          |___________|
                   WebSocket  =============>  WebSocket
                     Client     Connection     Server
```

Figure 6: 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 8.3 and Section 8.4 define new URI schemes that 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 7), 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.

```
     ___________                ___________                ___________
    |           |              |           |              |           |
    |          _|___        ___|_         _|___        ___|_          |
    |   CoAP  /  \  \ ---> /  /  \ CoAP  /  \  \ ---> /  /  \  CoAP   |
    |  Client \__/__/ <--- \__\__/ Proxy \__/__/ <--- \__\__/ Server  |
    |           |              |           |              |           |
    |___________|              |___________|              |___________|
            WebSocket ===> WebSocket      UDP            UDP
              Client        Server      Client          Server
```

Figure 7: 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 8). 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.

```
     ____________              ____________              ____________
    |            |            |            |            |            |
    |          __|___      ___|_        __|___      ___|_           |
    |   CoAP  /  \  \ ---> /  /  \ CoAP  /  /  \ ---> /  \  \  CoAP   |
    |  Client \__/__/ <--- \__\__/ Proxy \__\__/ <--- \__/__/ Server  |
    |            |            |            |            |            |
    |____________|            |____________|            |____________|
              UDP             UDP      WebSocket <=== WebSocket
            Client          Server      Server        Client
```

Figure 8: 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 9 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.

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 9: 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 10 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.

```
  0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Len=0 |  TKL  |      Code     |    Token (TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 10: 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, each endpoint 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 specifically introduced only for CoAP over reliable transports to allow peers to:

- Learn related characteristics, such as maximum message size for the connection

- Shut down the connection in an orderly fashion

- o 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 11.1).

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 11.2).

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:

- o Each capability option indicates one capability of the sender to the recipient.

- o Each setting option indicates a setting that will be applied by the sender.

One CSM MUST be sent by each endpoint 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]. Default values apply on a per-message basis and thus reset when the value is not present in a given Capabilities and Settings message.

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 size of a message in bytes that it can receive. The message size indicated includes the entire message, starting from the first byte of the message header and ending at the end of the message payload (there is no relationship of the message size to the overall request or response body size that may be achievable in block-wise transfer.)

#	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 wishing to offer block-wise transfers to its peer therefore needs to 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. (Note that indication of BERT support obliges neither peer to actually choose to make use of BERT.)

Implementation note: When indicating a value of the Max-Message-Size option with an intention to enable BERT, the indicating implementation may want to choose a BERT size message it wants to encourage and add a delta for the header and any options that also need to be included in the message. Section 4.6 of [RFC7252] adds 128 bytes to a maximum block size of 1024 to arrive at a default message size of 1152. A BERT-enabled implementation may want to indicate a BERT block size of 2048 or a higher multiple of 1024, and at the same time be more generous for the size of header and options added (say, 256 or 512). Adding 1024 or more however to the base BERT block size may encourage the peer implementation to vary the BERT block size based on the size of the options included, which can be harder to establish interoperability for.

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 the Ping carries 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).

Note that, as with similar mechanisms defined in [RFC6455] and [RFC7540], the present specification does not define any specific maximum time that the sender of a Ping message has to allow waiting for a Pong reply. Any limitations on the patience for this reply are a matter of the application making use of these messages, as is any approach to recover from a failure to respond in time.

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 or responses outstanding when the sender decides to send a Release message. The peer responding to the Release message SHOULD delay the closing of the connection until it has responded to all requests received by it before the Release message. It also MAY wait for the responses to its own requests.

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]. (Existing state related to the connection is not transferred from the present connection to the new connection.)

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 or responses outstanding 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 (not processing) 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 11.

```
 0                   1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      0x01     |      0xe2     |      0x42     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 Len   =    0 -------> 0x01
 TKL   =    1 ___/
 Code  = 7.02 Ping --> 0xe2
 Token =               0x42
```

Figure 11: Ping Message Example

An encoded example of the corresponding Pong message is shown in Figure 12.

```
 0                   1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      0x01     |      0xe3     |      0x42     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 Len   =    0 -------> 0x01
 TKL   =    1 ___/
 Code  = 7.03 Pong --> 0xe3
 Token =               0x42
```

Figure 12: 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:

- o large messages, such as firmware downloads, may cause undesired head-of-line blocking when a single TCP connection is used

- o 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 multiple blocks. For non-final BERT blocks, the payload is always a multiple of 1024 bytes. For final BERT blocks, the payload is a multiple (possibly 0) of 1024 bytes plus a partial block of less than 1024 bytes.

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 13 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.

```
CoAP Client                             CoAP Server
  |                                          |
  | GET, /status                      ------> |
  |                                          |
  | <------   2.05 Content, 2:0/1/BERT(3072)  |
  |                                          |
  | GET, /status, 2:3/0/BERT          ------> |
  |                                          |
  | <------   2.05 Content, 2:3/1/BERT(5120)  |
  |                                          |
  | GET, /status, 2:8/0/BERT         ------>  |
  |                                          |
  | <------   2.05 Content, 2:8/0/BERT(4711)  |
```

Figure 13: GET with BERT blocks

6.2. Example: PUT with BERT Blocks

Figure 14 demonstrates a PUT exchange with BERT blocks.

```
CoAP Client                             CoAP Server
  |                                           |
  | PUT, /options, 1:0/1/BERT(8192)    ------> |
  |                                           |
  | <------   2.31 Continue, 1:0/1/BERT        |
  |                                           |
  | PUT, /options, 1:8/1/BERT(16384)   ------> |
  |                                           |
  | <------   2.31 Continue, 1:8/1/BERT        |
  |                                           |
  | PUT, /options, 1:24/0/BERT(5683)   ------> |
  |                                           |
  | <------   2.04 Changed, 1:24/0/BERT        |
  |                                           |
```

Figure 14: PUT with BERT blocks

7. Observing Resources over Reliable Transports

This section describes how the procedures defined in [RFC7641] for observing resources over CoAP are applied (and modified, as needed) for reliable transports. In this section, "client" and "server" refer to the CoAP client and CoAP server.

7.1. 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.

Implementation note: This means that a proxy from a reordering transport to a reliable (in-order) transport (such as a UDP-to-TCP proxy) needs to process the Observe Option in notifications according to the rules in Section 3.4 of [RFC7641].

7.2. 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).

7.3. 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 single CoAP Ping Signaling message (Section 5.4) rather than individual requests to confirm each active observation. (Note that such a Ping/Pong only confirms a single hop: there is no obligation, and no expectation, of a proxy to react to a Ping by checking all its onward observations or all the connections, if any, underlying them. A proxy MAY maintain its own schedule for confirming the onward observations it relies on; it is however generally inadvisable for a proxy to generate a large number of outgoing checks based on a single incoming check.)

7.4. 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.

8. CoAP over Reliable Transport URIs

CoAP over UDP [RFC7252] defines the "coap" and "coaps" URI schemes. This document introduces four additional URI schemes for identifying CoAP resources and providing a means of locating the resource:

- the "coap+tcp" URI scheme for CoAP over TCP
- the "coaps+tcp" URI scheme for CoAP over TCP secured by TLS
- the "coap+ws" URI scheme for CoAP over WebSockets
- the "coaps+ws" URI scheme for CoAP over WebSockets secured by TLS

Resources made available via these schemes have no shared identity even if their resource identifiers indicate the same authority (the same host listening to the same TCP port). They are hosted in distinct namespaces because each URI scheme implies a distinct origin server.

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", and "query" are adopted from [RFC3986].

Section 8 (Multicast CoAP) in [RFC7252] is not applicable to these schemes.

As with the "coap" and "coaps" schemes defined in [RFC7252], all URI schemes defined in this section also support the path prefix "/.well-known/" defined by [RFC5785] for "well-known locations" in the namespace of a host. This enables discovery as per Section 7 of [RFC7252].

8.1. coap+tcp URI scheme

The "coap+tcp" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over TCP.

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

The syntax defined in Section 6.1 of [RFC7252] applies to this URI scheme with the following changes:

- o The port subcomponent indicates the TCP port at which the CoAP Connection Acceptor is located. (If it is empty or not given, then the default port 5683 is assumed, as with UDP.)

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.2. coaps+tcp URI scheme

The "coaps+tcp" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over TCP secured with TLS.

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

The syntax defined in Section 6.2 of [RFC7252] applies to this URI scheme, 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+tcp endpoint on TCP port 5684. This endpoint MAY also be ALPN enabled. A TLS server MAY offer coaps+tcp endpoints on ports other than TCP port 5684, which 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 11.7) 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+tcp is implicitly selected.

- For TCP port 5684, if the TLS client does not use the ALPN extension to negotiate the protocol, then coaps+tcp is implicitly selected.

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.3. coap+ws URI scheme

The "coap+ws" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over WebSockets.

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

The port subcomponent is OPTIONAL. The default is port 80.

The WebSocket endpoint is identified by a "ws" URI that is composed of the authority part of the "coap+ws" URI and the well-known path "/.well-known/coap" [RFC5785] [I-D.bormann-hybi-ws-wk]. The path and query parts of a "coap+ws" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP:

```
          coap+ws://example.org/sensors/temperature?u=Cel
               \______  ______/\___________  ___________/
                      \/                   \/
                                         Uri-Path: "sensors"
    ws://example.org/.well-known/coap    Uri-Path: "temperature"
                                         Uri-Query: "u=Cel"
```

Figure 15: The "coap+ws" URI Scheme

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.4. coaps+ws URI scheme

The "coaps+ws" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over WebSockets secured by TLS.

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

The port subcomponent is OPTIONAL. The default is port 443.

The WebSocket endpoint is identified by a "wss" URI that is composed of the authority part of the "coaps+ws" URI and the well-known path "/.well-known/coap" [RFC5785] [I-D.bormann-hybi-ws-wk]. The path and query parts of a "coaps+ws" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP.

```
             coaps+ws://example.org/sensors/temperature?u=Cel
                   \______  ______/\___________  ___________/
                          \/                   \/
                                            Uri-Path: "sensors"
       wss://example.org/.well-known/coap   Uri-Path: "temperature"
                                            Uri-Query: "u=Cel"
```

Figure 16: The "coaps+ws" URI Scheme

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.5. Uri-Host and Uri-Port Options

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.

8.6. Decomposing URIs into Options

The steps are the same as specified in Section 6.4 of [RFC7252] with minor changes.

This step from [RFC7252]:

3. If |url| does not have a <scheme> component whose value, when converted to ASCII lowercase, is "coap" or "coaps", then fail this algorithm.

is updated to:

3. If |url| does not have a <scheme> component whose value, when converted to ASCII lowercase, is "coap+tcp", "coaps+tcp", "coap+ws", or "coaps+ws", then fail this algorithm.

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 TCP port, include a Uri-Port Option and let that option's value be |port|.

8.7. 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. For CoAP over TCP, if the request is secured using TLS, let |url| be the string "coaps+tcp://". Otherwise, let |url| be the string "coap+tcp://". For CoAP over WebSockets, if the request is secured using TLS, let |url| be the string "coaps+ws://". Otherwise, let |url| be the string "coap+ws://".

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 TCP port.

9. 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]).

9.1. 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].

This guidance assumes implementation in a constrained device or for communication with a constrained device. CoAP over TCP/TLS has, however, a wider applicability. It may, for example, be implemented on a gateway or on a device that is less constrained (such as a smart phone or a tablet), for communication with a peer that is likewise less constrained, or within a backend environment that only communicates with constrained devices via proxies. As an exception to the previous paragraph, in this case, the recommendations in [RFC7525] are more appropriate.

Since the guidance offered in [RFC7925] and [RFC7525] differs in terms of algorithms and credential types, it is assumed that a CoAP over TCP/TLS implementation that needs to support both cases implements the recommendations offered by both specifications.

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+tcp" 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+tcp" scheme and TLS-secured CoAP default port.

9.2. TLS usage for CoAP over WebSockets

A CoAP client requesting a resource identified by a "coaps+ws" URI negotiates a secure WebSocket connection to a WebSocket server endpoint with a "wss" URI. This is described in Section 8.4.

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+ws" 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, usually modern 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].

10. 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.

10.1. 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.

11. IANA Considerations

11.1. 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 [RFC8126].

11.2. 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 11.1), 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 policy is analogous rather than identical because the structure of the subregistry includes an additional column; however, the value of this column has no influence on the policy.)

The documentation for a Signaling Option Number should specify the semantics of an option with that number, including the following properties:

o Whether the option is critical or elective, as determined by the Option Number.

o Whether the option is repeatable.

o The format and length of the option’s value.

o The base value for the option, if any.

11.3. Service Name and Port Number Registration

IANA is requested to assign the port number 5683 and the service name "coap+tcp", in accordance with [RFC6335].

Service Name.
  coap+tcp

Transport Protocol.
  tcp

Assignee.
  IESG <iesg@ietf.org>

Contact.
  IETF Chair <chair@ietf.org>

Description.
  Constrained Application Protocol (CoAP)

Reference.
  [RFCthis]

Port Number.
  5683

11.4. 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 to address the exceptional case of TLS implementations that do not support the "Application-Layer Protocol Negotiation Extension" [RFC7301].

Service Name.
  coaps+tcp

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

11.5. URI Scheme Registration

URI schemes are registered within the "Uniform Resource Identifier (URI) Schemes" registry maintained at [IANA.uri-schemes].

11.5.1. coap+tcp

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coap+tcp". This registration request complies with [RFC7595].

Scheme name:
coap+tcp

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using TCP.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.1 in [RFCthis]

11.5.2. coaps+tcp

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coaps+tcp". This registration request complies with [RFC7595].

Scheme name:
coaps+tcp

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using TLS.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.2 in [RFCthis]

11.5.3. coap+ws

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coap+ws". This registration request complies with [RFC7595].

Scheme name:
coap+ws

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.3 in [RFCthis]

11.5.4. coaps+ws

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coaps+ws". This registration request complies with [RFC7595].

Scheme name:
coaps+ws

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol secured with TLS.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

References:
Section 8.4 in [RFCthis]

11.6. 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.

11.7. 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]

11.8. 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]

11.9. 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

12. References

12.1. 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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6066>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7525>.

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

[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, <https://www.rfc-editor.org/info/rfc7595>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7959>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

12.2. Informative References

[BK2015] Byrne, C. and J. Kleberg, "Advisory Guidelines for UDP Deployment", Proceedings draft-byrne-opsec-udp-advisory-00 (expired), 2015.

[EK2016] Edeline, K., Kuehlewind, M., Trammell, B., Aben, E., and B. Donnet, "Using UDP for Internet Transport Evolution", Proceedings arXiv preprint 1612.07816, 2016.

[HomeGateway] Eggert, L., "An experimental study of home gateway characteristics", Proceedings of the 10th annual conference on Internet measurement , 2010.

[I-D.gomez-lwig-tcp-constrained-node-networks] Gomez, C., Crowcroft, J., and M. Scharf, "TCP over Constrained-Node Networks", draft-gomez-lwig-tcp-constrained-node-networks-03 (work in progress), June 2017.

[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 for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-06 (work in progress), October 2017.

[IANA.uri-schemes]
IANA, "Uniform Resource Identifier (URI) Schemes", <http://www.iana.org/assignments/uri-schemes>.

[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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6347>.

[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>.

[SW2016]   Swett, I., "QUIC Deployment Experience @Google", Proceedings https://www.ietf.org/proceedings/96/slides/slides-96-quic-3.pdf, 2016.

Appendix A. 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+ws" URI might be as follows. Figure 17 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI <coap+ws://example.org/sensors/temperature?u=Cel>, for example, from a resource representation that it retrieved previously.

2. It establishes a WebSocket connection to the endpoint URI composed of the authority "example.org" and the well-known path "/.well-known/coap", <ws://example.org/.well-known/coap>.

3. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") options.

4. It waits for the server to return a response.

5. The CoAP client uses the connection for further requests, or the connection is closed.

```
      CoAP        CoAP
     Client      Server
   (WebSocket  (WebSocket
     Client)     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
        |           |
        |           |
        +--------->|  Binary frame (opcode=%x2, FIN=1, MASK=1)
        |           |    +-------------------------+
        |           |    | GET                     |
        |           |    | Token: 0x53             |
        |           |    | Uri-Path: "sensors"     |
        |           |    | Uri-Path: "temperature" |
        |           |    | Uri-Query: "u=Cel"      |
        |           |    +-------------------------+
        |           |
        |<---------+  Binary frame (opcode=%x2, FIN=1, MASK=0)
        |           |    +-------------------------+
        |           |    | 2.05 Content            |
        |           |    | Token: 0x53             |
        |           |    | Payload: "22.3 Cel"     |
        |           |    +-------------------------+
        :           :
        :           :
        |           |
        +--------->|  Close frame (opcode=%x8, FIN=1, MASK=1)
        |           |
        |<---------+  Close frame (opcode=%x8, FIN=1, MASK=0)
        |           |
```

Figure 17: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 18 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.

```
  CoAP         CoAP       CoAP
 Client       Proxy      Server
(WebSocket  (WebSocket    (UDP
 Client)     Server)   Endpoint)

   |          |           |
   +--------->|           |  Binary frame (opcode=%x2, FIN=1, MASK=1)
   |          |           |    +-------------------------------------+
   |          |           |    | GET                                 |
   |          |           |    | Token: 0x7d                         |
   |          |           |    | Proxy-Uri: "coap://[2001:db8::1]/"  |
   |          |           |    +-------------------------------------+
   |          |           |
   |          +---------->|  CoAP message (Ver=1, T=Con, MID=0x8f54)
   |          |           |    +-------------------------------------+
   |          |           |    | GET                                 |
   |          |           |    | Token: 0x0a15                       |
   |          |           |    +-------------------------------------+
   |          |           |
   |          |<----------+  CoAP message (Ver=1, T=Ack, MID=0x8f54)
   |          |           |    +-------------------------------------+
   |          |           |    | 2.05 Content                        |
   |          |           |    | Token: 0x0a15                       |
   |          |           |    | Payload: "ready"                    |
   |          |           |    +-------------------------------------+
   |          |           |
   |<---------+           |  Binary frame (opcode=%x2, FIN=1, MASK=0)
   |          |           |    +-------------------------------------+
   |          |           |    | 2.05 Content                        |
   |          |           |    | Token: 0x7d                         |
   |          |           |    | Payload: "ready"                    |
   |          |           |    +-------------------------------------+
   |          |           |
```

Figure 18: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSocket-enabled CoAP proxy

Appendix B. Change Log

The RFC Editor is requested to remove this section at publication.

B.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

B.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

B.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

B.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

B.5. Since draft-ietf-core-coap-tcp-tls-06

Addressed feedback from second Working Group Last Call

B.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

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 Yoshifumi Nishida, Mark Nottingham, and Meral Shirazipour as well as several IESG reviewers provided extensive comments; from the IESG, we would like to specifically call out Ben Campbell, Mirja Kuehlewind, Eric Rescorla, Adam Roach, 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 C. Bormann
Internet-Draft Universitaet Bremen TZI
Intended status: Informational A. Betzler
Expires: May 3, 2018 Fundacio i2CAT
C. Gomez
I. Demirkol
Universitat Politecnica de Catalunya/Fundacio i2CAT
October 30, 2017

CoAP Simple Congestion Control/Advanced
draft-ietf-core-cocoa-02

Abstract

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

This specification defines more advanced, but still simple CoRE Congestion Control mechanisms, called CoCoA. The core of these mechanisms is a Retransmission TimeOut (RTO) algorithm that makes use of Round-Trip Time (RTT) estimates, in contrast with how the RTO is determined as per the base CoAP specification (RFC 7252). The mechanisms defined in this document have relatively low complexity, yet they improve the default CoAP RTO algorithm. The design of the mechanisms in this specification has made use of input from simulations and experiments in real networks.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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.

The present specification defines such an advanced CoRE Congestion Control mechanism, with the goal of improving performance while retaining safety as well as the simplicity that is appropriate for constrained devices. Hence, we are calling this mechanism Simple CoCoA (Congestion Control/Advanced).

In the Internet, congestion control is typically implemented in a way that it can be introduced or upgraded unilaterally. Still, a new congestion control scheme must not be introduced lightly. To ensure that the new scheme is not posing a danger to the network, considerable work has been done on simulations and experiments in real networks. Some of this work will be mentioned in "Discussion" subsections in the following sections; an overview is given in Appendix A. Extended rationale for this specification can also be found in the historical Internet-Drafts [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 (see Section 4.3 of [RFC7252]) conveying a request.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

(Note that the present document is itself informational, but it is discussing normative statements about behavior that makes the congestion control scheme work in a safe manner.)

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

2. Context

In the definition of the CoAP protocol [RFC7252], an approach was taken that includes a very simple basic scheme (lock-step with the number of parallel exchanges usually limited to 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. The algorithm defined in this document is intended to adapt to the current characteristics of any underlying network, and therefore is well suited for a wide range of network conditions, in terms of bandwidth, latency, load, loss rate, topology, etc. 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.

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. In particular, a wide spectrum of RTT values is expected in different types of networks where CoAP is used. Those RTTs range from several orders of magnitude below the default initial timeout to values larger than the default. The algorithm defined in this document is based on the algorithm for RTO estimation defined in [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 default RTO estimate back to the default RTO estimate, until fresh measurements become available again, as proposed in Section 4.3.

RTT variability challenges RTO estimation. In TCP, delayed ACKs contribute to RTT variability, since this option adds a delay of up to 500 ms (typically, 200 ms) before an ACK is sent by a receiving TCP endpoint. However, 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 (MAX_RETRANSMIT, see Section 4.2 of [RFC7252], normally 4) in the default interval of 2 to 3 s. The approach chosen for a mechanism with variable backoff factors is presented in Section 4.2.1.

It may also be worthwhile to perform RTT estimation 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], with the differences introduced 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 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 (either the weak or strong estimator) that made the most recent contribution:

```
RTO := w_weak * E_weak_ + (1 - w_weak) * RTO (1)

RTO := w_strong * E_strong_ + (1 - w_strong) * RTO (2)
```

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

The default values for the corresponding weights, w_weak and w_strong, are 0.25 and 0.5, respectively. Pseudocode and examples for the overall RTO estimate presented are available in Appendix B.1 and Appendix C.1.

4.2.1. Differences with the algorithm of RFC 6298

This subsection presents three differences of the algorithm defined in this document with the one defined in [RFC6298]. 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.

In order to avoid that exchanges with small initial RTOs (i.e. RTO estimate lower than 1 s) use up all retransmissions in a short interval of time, the RTO for a retransmission is multiplied by 3 for each retransmission as long as the RTO is less than 1 s.

On the other hand, to avoid exchanges with large initial RTOs (i.e., RTO estimate greater than 3 s) not being able to carry out all retransmissions within MAX_TRANSMIT_WAIT (normally 93 s), the RTO is multiplied only by 1.5 when RTO is greater than 3 s.

Pseudocode for the variable backoff factor is in Appendix B.3.

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. In fact, the weak estimator allows to better update the RTO estimator when mostly weak RTTs are available, either due to the lossy nature of links or due to congestion-induced losses. In presence of the latter, spurious timeouts are avoided and the rate of retries is reduced, which allows to decrease congestion. 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 A).

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.)

Pseudocode and examples for the aging mechanism presented are available in Appendix B.2 and in Appendix C.2.

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. An RTO as specified in Section 4 must be used for confirmable messages.

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

The mechanism defined above for non-confirmables 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 Non-Confirmables SHOULD NOT exceed 1/RTT on average, if the server can maintain an RTT estimate for a client. CoCoA limits the packet rate of Non-Confirmables 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

The security considerations of, e.g., [RFC5681], [RFC2914], and [RFC8085] apply. Some issues are already discussed in the security considerations of [RFC7252].

If a malicious node manages to prevent the delivery of some packets, a consequence will be an RTO increase, which will further reduce network performance. Note that this type of attack is not specific for CoCoA (and not even specific for CoAP), and many congestion control algorithms increase the RTO upon packet loss detection. While it is hard to prevent radio jamming, some mitigation for other forms of this type of attack is provided by network access control techniques. Also, the weak estimator in CoCoA increases the chances of obtaining RTT measurements in the presence of heavy packet losses, allowing to keep the RTO updated, which in turn allows recovery from a jamming attack in reasonable time.

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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc8085>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7641>.

Appendix A. Supporting evidence

(Editor’s note: The references local to this appendix may need to be merged with those from the specification proper, depending on the discretion of the RFC editor.)

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 testbeds 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 WiFi 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].

A.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].

A.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.

Appendix B. Pseudocode

B.1. Updating the RTO estimator

```
// Default values
ALPHA = 0.125 // RFC 6298
BETA = 0.25 // RFC 6298
W_STRONG = 0.5
W_WEAK = 0.25

updateRTO(retransmissions, RTT) {
  if (retransmissions == 0) {
    RTTVAR_strong = (1 - BETA) * RTTVAR_strong
                  + BETA * (RTT_strong - RTT);
    RTT_strong  = (1 - ALPHA) * RTT_strong + ALPHA * RTT;
    E_strong = RTT_strong  + 4 * RTTVAR_strong;
    RTO = W_STRONG * E_strong + (1 - W_STRONG) * RTO;
  } else if (retransmissions <= 2) {
    RTTVAR_weak = (1 - BETA) * RTTVAR_weak
                + BETA * (RTT_weak - RTT);
    RTT_weak  = (1 - ALPHA) * RTT_weak + ALPHA * RTT;
    E_weak = RTT_weak  + 1 * RTTVAR_weak;
    RTO = W_WEAK * E_weak + (1 - W_WEAK) * RTO
  }
}
```

B.2. RTO aging

```
checkAging() {
  clock_time difference = getCurrentTime() - lastUpdatedTime;

  if ((RTO < 1s) && (difference > (16 * RTO))) {
    RTO = 2 * RTO;
    lastUpdatedTime = getCurrentTime();
  } else if ((RTO > 3s) && (difference > (4 * RTO))) {
    RTO = 1s + 0.5 * RTO;
    lastUpdatedTime = getCurrentTime();
  }
}
```

B.3. Variable Backoff Factor

```
backOffRTO() {
  if (RTO < 1s) {
    RTO = RTO * 3;
  } else if (RTO > 3s) {
    RTO = RTO * 1.5;
  } else {
    RTO = RTO * 2;
  }
}
```

Appendix C. Examples

C.1. Example A.1: weak RTTs

A large network of sensor nodes that report periodical measurements is operating normally, without congestion. The nodes transmit their sensor readings via CON messages every 20 s in an asynchronous way towards a server located behind a gateway, obtaining strong RTT measurements (RTT 1.1 s, RTTVAR 0.1 s) that lead to the calculation of an RTO of 1.5 s (in average) in each node. In this mode of operation, no aging is applied, since the RTO is refreshed before the aging mechanism applies.

Suddenly, upon detection of a global event, the majority of sensor nodes start transmitting at a higher rate (every 5 s) to increase the resolution of the acquired data, which creates heavy congestion that leads to packet losses and an important increase of real RTT between the nodes and the server (RTT 2 s, RTTVAR 1 s). Due to the packet losses and spurious retransmissions (which can fuel congestion even more), many nodes are not able to update their RTO via strong RTT measurements, but they are able to obtain weak RTT measurements. A node with an initial RTO of 1.5 s would run into a retransmission, before obtaining an ACK (given the RTT of 2 s and that the ACK is not lost).

This weak RTT measurement would increase the overall RTO of the node to 1.875 s (RTO = 0.25 * 3 s + 0.75 * 1.5 s). Following the same calculus (and RTT/RTTVAR values), after obtaining another weak RTT, the RTO would increase to 2.156 s. At this point, the benefits of the weak RTT measurements are twofold:

1. Further spurious retransmissions are avoided as the RTO has increased above the real RTT.

2. The increase of RTOs across the whole network reduces the rate with which retransmissions are generated, decreasing the network congestion (which leads to an RTT and packet loss decrease).

C.2. Example A.2: VBF and aging

Assuming that the frequency of message generation is even higher (every 3 s) and the real RTT would further increase due to congestion, the RTO at some point would increase to 4 s. Since now the RTO is above 3 s, no longer a binary backoff is used to avoid the RTO growing too much in case of retransmissions. As the generation of data from the nodes ceases at some point (the network returns to a normal state), the aging mechanism would reduce the RTO automatically (with an RTO of 4 s, after 16 s the RTO would be shifted to 3 s before a new RTT is measured).

C.3. Example B: VBF and aging

A network of nodes connected over 4G with an Internet service is calculating very small RTO values (0.3 s) and the nodes are transmitting CON messages every 1 s. Suddenly, the connection quality gets worse and the nodes switch to a more stable, yet slower connection via GPRS. As a result of this change, the nodes run into retransmissions, as the real RTT has increased above the calculated RTO.

Since the RTO is below 1 s, the Variable Backoff Factor increases the backoff values quickly to avoid spurious retransmissions (0.9 s first retry, 2.7 s second retry, etc.). Further, if due to the packet losses and increased delays in the network no new RTT measurements are obtained, the aging mechanism automatically increases the RTO (doubling it) after 3.8 s (16 * 0.3 s) to adapt better to the sudden changes of network conditions. Without the Variable Backoff Factor and the aging mechanism, the number of spurious retransmissions would be much higher and the RTO would be corrected more slowly.

Appendix D. Analysis: difference between strong and weak estimators

This section analyzes the difference between the strong and weak RTO estimators. If there is no congestion, assume a static RTT of R'. Then, E_strong_can be expressed as:

E_strong_ = R' + G,

since RTTVAR is reduced constantly by RTTVAR = RTTVAR * 3/4 (according to [RFC6298], and SRTT=R'), G would be dominant term in the max(G, K * RTTVAR) expression in the long run.

For the weak estimator: assume that the RTO setting converges to E_strong_ calculated above in the long run. If there is a packet loss, and an RTT is obtained for the first retransmission, then the weak RTT sample obtained by the weak estimator is:

RW' = R'+ G + R'

Therefore, E_weak_ can be expressed as:

E_weak_ = RW' + max(G, RW'/2) = 3 * R'

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. Further reviews by Michael Scharf and Ingemar Johansson led to further improvements, including some more discussion in the appendices.

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, TEC2012-32531, TEC2016-79988-P 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@i2cat.net

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: January 18, 2018

M. Veillette, Ed.
Trilliant Networks Inc.
P. van der Stok, Ed.
consultant
A. Pelov
Acklio
A. Bierman
YumaWorks

July 17, 2017

CoAP Management Interface
draft-ietf-core-comi-01

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 datastore and data node 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

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy, smart city 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. 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 [RFC8040] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data models specified in a standardized language, such as YANG, promotes interoperability between devices and applications from different manufacturers.

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 messges, 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].

The following terms are defined in the YANG data modelling language [RFC7950]: action, anydata, anyxml, client, configuration data, container, data model, data node, datastore, identity, instance identifier, key, key leaf, leaf, leaf-list, list, module, RPC, schema node, server, state data, submodule.

The following term is defined in [I-D.ietf-core-yang-cbor]: YANG schema item identifier (SID).

The following terms are defined in the CoAP protocol [RFC7252]: Confirmable Message, Content-Format.

The following terms are defined in this document:

data node resource: a CoAP resource that models a YANG data node.

datastore resource: a CoAP resource that models a YANG datastore.

event stream resource: a CoAP resource used by clients to observe YANG notifications.

target resource: the resource that is associated with a particular CoAP request, identified by the request URI.

data node instance: An instance of a data node specified in a YANG module and stored in the server.

notification instance: An instance of a schema node of type notification, specified in a YANG module implemented by the server. The instance is generated in the server at the occurrence of the corresponding event and reported by an event stream.

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 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: The value assigned to a data node instance. Data node values are serialized into the payload according to the rules defined in section 4 of [I-D.ietf-core-yang-cbor].

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for reading and modifying the content of datastore(s) used for the management of the instrumented node.

```
+------------------------------------------------------------+
|                     SMIv2 specification (2)                |
+------------------------------------------------------------+
                               |
                               V
+------------------------------------------------------------+
|                     YANG specification  (1)                |
+------------------------------------------------------------+
         |                                          |
Client   V                               Server     V
+----------------+                       +-------------------+
| Request        |--> CoAP request(3) -->| Indication        |
| Confirm        |<-- CoAP response(3)<--| Response      (4) |
|                |                       |                   |
|                |<==== Security (7) ===>|+-----------------+|
+----------------+                       || Datastore (5)   ||
                                         |+-----------------+|
                                         |+-----------------+|
                                         || Event stream (6)||
                                         |+-----------------+|
                                         +-------------------+
```

Figure 1: Abstract CoMI architecture

Figure 1 is a high-level representation of the main elements of the CoMI management architecture. 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) CoAP request/response messages: The CoMI client sends request messages to and receives response messages from the CoMI server.

(4) Request, Indication, Response, Confirm: The processes performed by the CoMI clients and servers.

(5) Datastore: A resource used to access configuration data, state data, RPCs and actions. A CoMI server may support multiple datastores to support more complex operations such as configuration rollback, scheduled update.

(6) Event stream: An observable resource used to get real time notifications. A CoMI server may support multiple Event streams serving different purposes such as normal monitoring, diagnostic, syslog, security monitoring.

(7) Security: The server MUST prevent unauthorized users from reading or writing any CoMI resources. CoMI relies on security protocols such as DTLS [RFC6347] to secure CoAP communication.

2.1. Major differences between RESTCONF and CoMI

CoMI is a RESTful protocol for small devices where saving bytes to transport counts. Contrary to RESTCONF, many design decisions are motivated by the saving of bytes. Consequently, CoMI is not a RESTCONF over CoAP protocol, but differs more significantly from RESTCONF. Some major differences are cited below:

- 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 does not support "insert" query parameter (first, last, before, after) and the "point" query parameter which are supported by RESTCONF.

- o CoMI does not support the "start-time" and "stop-time" query parameters to retrieve past notifications.

- o CoMI and RESTCONF also differ in the handling of:

  * notifications.

  * default values.

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 identifiers are used instead of these strings. YANG Schema Item iDentifier (SID) is defined in [I-D.ietf-core-yang-cbor] section 2.1.

When used in a URI, SIDs are encoded in based64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5. The last 6 bits encoded is always aligned with the least significant 6 bits of the SID represented using an unsigned integer. 'A' characters (value 0) at the start of the resulting string are removed.

```
SID in basae64 = URLsafeChar[SID >> 60 & 0x3F] |
                 URLsafeChar[SID >> 54 & 0x3F] |
                 URLsafeChar[SID >> 48 & 0x3F] |
                 URLsafeChar[SID >> 42 & 0x3F] |
                 URLsafeChar[SID >> 36 & 0x3F] |
                 URLsafeChar[SID >> 30 & 0x3F] |
                 URLsafeChar[SID >> 24 & 0x3F] |
                 URLsafeChar[SID >> 18 & 0x3F] |
                 URLsafeChar[SID >> 12 & 0x3F] |
                 URLsafeChar[SID >> 6 & 0x3F] |
                 URLsafeChar[SID & 0x3F]
```

For example, SID 1717 is encoded as follow.

```
URLsafeChar[1717 >> 60 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 54 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 48 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 42 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 36 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 30 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 24 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 18 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 12 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 6 & 0x3F]  = URLsafeChar[26] = 'a'
URLsafeChar[1717 & 0x3F]       = URLsafeChar[53] = '1'
```

The resulting base64 representation of SID 1717 is "a1"

2.3. Instance identifier

Instance identifiers are used to uniquely identify data node instances within a datastore. This YANG built-in type is defined in [RFC7950] section 9.13. An instance identifier is composed of the data node identifier (i.e. a SID) and for data nodes within list(s) the keys used to index within these list(s).

When part of a payload, instance identifiers are encoded in CBOR based on the rules defined in [I-D.ietf-core-yang-cbor] section 5.13.1. When part of a URI, the SID is appended to the URI of the targeted datastore, the keys are specified using the 'k' URI-Query as defined in Section 5.1.

2.4. CBOR ordered map schematic

An ordered map is used as a root container of the application/yang-tree+cbor Content-Format. This datatype share the same functionalities as a CBOR map without the following limitations:

- o The ordering of the pairs of data items is preserved from serialization to deserialization.
- o Duplicate keys are allowed

This schematic is constructed using a CBOR array comprising pairs of data items, each pair consisting of a key that is immediately followed by a value. Unlike a CBOR map for which the length denotes the number of pairs, the length of the ordered map denotes the number of items (i.e. number of keys plus number of values).

The use of this schematic can be inferred from its context or by the presence of a preceding tag. The tag assigned to the Ordered map is defined in Section 11.4.

In the case of CoMI, the use of the ordered map as the root container of the application/yang-tree+cbor Content-Format is inferred, the Ordered map tag is not used.

2.5. Content-Formats

ComI uses Content-Formats based on the YANG to CBOR mapping specified in [I-D.ietf-core-yang-cbor]. All Content-Formats defined hereafter are constructed using one or both of these constructs:

- o YANG data node value, encoded based on the rules defined in [I-D.ietf-core-yang-cbor] section 4.
- o YANG instance identifier, encoded based on the rules defined in [I-D.ietf-core-yang-cbor] section 5.13.1.

The following Content-formats are defined:

application/yang-value+cbor: represents a CBOR YANG document containing one YANG data node value. The YANG data node instance can be a leaf, a container, a list, a list instance, a RPC input, a RPC output, an action input, an action output, a leaf-list, an anydata or an anyxml. The CBOR encoding for each of these YANG data node instances are defined in [I-D.ietf-core-yang-cbor] section 4.

FORMAT: data-node-value

DELTA ENCODING: SIDs included in a YANG container, a list instance, a RPC input, a RPC output, an action input, an actions output and an anydata are encoded using a delta value equal to the SID of the current schema node minus the SID of the parent. The parent SID of root data nodes is defined by the URI carried in the associated request (i.e. GET, PUT, POST).

application/yang-values+cbor: represents a YANG document containing a list of data node values.

FORMAT: CBOR array of data-node-value

DELTA ENCODING: SIDs included in a YANG container, a list instance and an anydata are encoded using a delta value equal to the SID of the current schema node minus the SID of the parent. The parent SID of root data nodes is defined by the corresponding instance-identifier carried in the FETCH request.

application/yang-tree+cbor: represents a CBOR YANG document containing a YANG data tree.

FORMAT: ordered map of single-instance-identifier, data-node-value

DELTA ENCODING: The SID part of the first instance-identifier within the ordered map is encoded using its absolute value. Subsequent instance-identifiers are encoded using a delta value equal to the SID of the current instance-identifiers minus the SID of the previous instance-identifier.

application/yang-selectors+cbor: represents a CBOR YANG document containing a list of data node selectors (i.e. instance identifier).

FORMAT: CBOR array of instance-identifier

DELTA ENCODING: The SID part of the first instance-identifier within the CBOR array is encoded using its absolute value. Subsequent instance-identifiers are encoded using a delta value equal to the SID of the current instance-identifiers minus the SID of the previous instance-identifier.

application/yang-patch+cbor: represents a CBOR YANG document containing a list of data nodes to be replaced, created, or deleted.

For each data node instance, D, for which the instance identifier is the same as for a data node instance, I, in the targeted resource: the data node value of D replaces the data node value of I. When the data node value of D is null, the data node instance I is removed. When the targeted resource does not contain a data node instance with the same instance identifier as D, a new data node instance is created in the targeted resource with the same instance identifier and data node value as D.

FORMAT: CBOR array of instance-identifier, data-node-value

DELTA ENCODING: Same as Content-Format application/yang-tree+cbor

The different Content-formats usage is summarized in the table below:

Method	Resource	Content-Format
GET response	data node	/application/yang-value+cbor
PUT request	data node	/application/yang-value+cbor
POST request	data node	/application/yang-value+cbor
DELETE	data node	n/a
GET response	datastore	/application/yang-tree+cbor
PUT request	datastore	/application/yang-tree+cbor
POST request	datastore	/application/yang-tree+cbor
FETCH request	datastore	/application/yang-selectors+cbor
FETCH response	datastore	/application/yang-values+cbor
iPATCH request	datastore	/application/yang-patch+cbor
GET response	event stream	/application/yang-tree+cbor
POST request	rpc, action	/application/yang-value+cbor
POST response	rpc, action	/application/yang-value+cbor

3. Example syntax

This section presents the notation used for the examples. The YANG modules that are used throughout this document are shown in Appendix C. The example modules are copied 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.

SIDs in URIs are represented as a base64 number, SIDs in the payload are represented as decimal numbers.

4. CoAP Interface

The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Collection Interface. CoMI end-points that implement the CoMI management protocol, support at least one discoverable management resource of resource type (rt): core.c.datastore, with example path: /c, where c is short-hand for CoMI. The path /c is recommended but not compulsory (see Section 8).

Three CoMI resources are accessible with the following three example paths:

/c: Datastore resource with path "/c" and using CBOR content encoding format. Sub-resouces of format /c/instance-identifier may be available to access directly each data node resource for this datastore.

/mod.uri: URI identifying the location of the YANG module library used by this server, with path "/mod.uri" and Content-Format "text/plain; charset=utf-8". An ETag 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.

/s: Event stream resource to which YANG notification instances are reported. 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. Every data node of the YANG modules loaded in the CoMI server represents a sub-resource of the datastore resource (e.g. /c/instance-identifier).

When multiple instances of a list exist, instance selection is possible as described in Section 5.1, Section 5.2.4, and Section 5.2.3.1.

The description of the management collection interface, with if=core.c, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

Function	Recommended path	rt
Datastore	/c	core.c.datastore
Data node	/c/instance-identifier	core.c.datanode
YANG module library	/mod.uri	core.c.moduri
Event steam	/s	core.c.eventstream

The path values are example values. On discovery, the server makes the actual path values known for these four resources.

5. CoMI Collection Interface

The CoMI Collection Interface provides a CoAP interface to manage YANG servers.

The methods used by CoMI are:

Operation	Description
GET	Retrieve the datastore resource or a data node resource
FETCH	Retrieve specific data nodes within a datastore resource
POST	Create a datastore resource or a data node resource, invoke an RPC or action
PUT	Create or replace a datastore resource or a data node resource
iPATCH	Idem-potently create, replace, and delete data node resource(s) within a datastore resource
DELETE	Delete a datastore resource or a data node resource

There is one Uri-Query option for the GET, PUT, POST, and DELETE methods.

Uri-Query option	Description
k	Select an instance within YANG list(s)

This parameter is not used for FETCH and iPATCH, because their request payloads support list instance selection.

5.1. Using the 'k' Uri-Query option

The "k" (key) parameter specifies a specific instance of a data node. The SID in the URI is followed by the (?k=key1, key2,..). Where SID identifies a data node, and key1, key2 are the values of the key leaves that specify an instance. Lists can have multiple keys, and lists can be part of lists. The order of key value generation is given recursively by:

- o For a given list, if a parent data node is a list, generate the keys for the parent list first.
- o For a given list, generate key values in the order specified in the YANG module.

Key values are encoded using the rules defined in the following table.

YANG datatype	Uri-Query text content
uint8,uint16,unit32, uint64	int2str(key)
int8, int16,int32, int64	urlSafeBase64(CBORencode(key))
decimal64	urlSafeBase64(CBOR key)
string	key
boolean	"0" or "1"
enumeration	int2str(key)
bits	urlSafeBase64(CBORencode(key))
binary	urlSafeBase64(key)
identityref	int2str(key)
union	urlSafeBase64(CBORencode(key))
instance-identifier	urlSafeBase64(CBORencode(key))

In this table:

- o The method int2str() is used to convert an integer value to a string. For example, int2str(0x0123) return the string "291".
- o The method urlSafeBase64() is used to convert a binary string to base64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5. For example, urlSafeBase64(\xF9\x56\xA1\x3C) return the string "-VahPA".
- o The method CBORencode() is used to convert a YANG value to CBOR as specified in [I-D.ietf-core-yang-cbor] section 5, item 8.

The resulting key string is encoded in a Uri-Query as specified in [RFC7252] section 6.5.

5.2. Data Retrieval

One or more data nodes 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 [RFC8132].

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] may be used, as explained in more detail in Section 7.

There are two additional Uri-Query options for the GET and FETCH methods.

Uri-Query option	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' Uri-Query option

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 node resources. A 4.02 (Bad Option) error is returned if used for other methods or resource types.

If this Uri-Query option is not present, the default value is "a".

5.2.2. Using the 'd' Uri-Query option

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 by any client yet, the server MUST return the default value of the leaf.

If the target of a GET method is a data node that represents a container or list that has child resources with default values, and these have not been given value yet,

The server MUST not return the child resource if d= 't'

The server MUST return the child resource if d= 'a'.

If this Uri-Query option 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. An instance identifier is specified in the URI path prefixed with the example path /c.

```
FORMAT:
  GET /c/instance-identifier

  2.05 Content (Content-Format: application/yang-value+cbor)
  data-node-value
```

The returned payload contains the CBOR encoding of the specified data node instance value.

5.2.3.1. GET Examples

Using for example the current-datetime leaf from Appendix C.1, a request is sent to retrieve the value of system-state/clock/current-datetime specified in container system-state. The SID of system-state/clock/current-datetime is 1719, encoded in octal 3267, yields two 6 bit decimal numbers 26 and 55, encoded in base64, (according to table 2 of [RFC4648]) yields a3. The response to the request returns the CBOR encoding of this leaf of type 'string' as defined in [I-D.ietf-core-yang-cbor] section 5.4.

```
REQ: GET example.com/c/a3

RES: 2.05 Content (Content-Format: application/yang-value+cbor)
"2014-10-26T12:16:31Z"
```

The next example represents the retrieval of a YANG container. In this case, the CoMI client performs a GET request on the clock container (SID = 1717; base64: a1). The container returned is encoded using a CBOR map as specified by [I-D.ietf-core-yang-cbor] section 4.2.

```
REQ: GET example.com/c/a1

RES: 2.05 Content (Content-Format: application/yang-value+cbor)
{
  +2 : "2014-10-26T12:16:51Z",   / SID 1719 /
  +1 : "2014-10-21T03:00:00Z"    / SID 1718 /
}
```

This example shows the retrieval of the /interfaces/interface YANG list accessed using SID 1533 (base64: X9). The return payload is encoded using a CBOR array as specified by [I-D.ietf-core-yang-cbor] section 4.4.1 containing 2 instances.

```
REQ: GET example.com/c/X9

RES: 2.05 Content (Content-Format: application/yang-value+cbor)
[
  {
    +4 : "eth0",                / name  (SID 1537) /
    +1 : "Ethernet adaptor",    / description (SID 1534) /
    +5 : 1179,                  / type, (SID 1538) identity /
                                / ethernetCsmacd (SID 1179) /
    +2 : true                   / enabled ( SID 1535) /
  },
  {
    +4 : "eth1",                / name (SID 1537) /
    +1 : "Ethernet adaptor",    / description (SID 1534) /
    +5 : 1179,                  / type, (SID 1538) identity /
                                / ethernetCsmacd (SID 1179) /
    +2 : false                  / enabled /
  }
]
```

It is equally possible to select a leaf of a specific instance of a list. The example below requests the description leaf (SID=1534, base64: X-) within the interface list corresponding to the list key "eth0". The returned value is encoded in CBOR based on the rules specified by [I-D.ietf-core-yang-cbor] section 5.4.

```
REQ: GET example.com/c/X-?k="eth0"

RES: 2.05 Content (Content-Format: application/yang-value+cbor)
"Ethernet adaptor"
```

5.2.4. FETCH

The FETCH is used to retrieve multiple data node values. The FETCH request payload contains a list of instance-identifier encoded based on the rules defined by Content-Format application/yang-selectors+cbor in Section 2.5. The return response payload contains a list of values encoded based on the rules defined by Content-Format application/yang-values+cbor in Section 2.5. A value MUST be returned for each instance-identifier specified in the request. A CBOR null is returned for each data node requested by the client, not supported by the server or not currently instantiated.

```
FORMAT:
  FETCH /c (Content-Format :application/yang-selectors+cbor)
  CBOR array of instance-identifier

  2.05 Content (Content-Format: application/yang-values+cbor)
  CBOR array of data-node-value
```

5.2.4.1. FETCH examples

The example uses the current-datetime leaf and the interface list from Appendix C.1. In the following example the value of current-datetime (SID 1719 and the interface list (SID 1533) instance identified with name="eth0" are queried.

```
REQ:  FETCH /c (Content-Format :application/yang-selectors+cbor)
[
  1719,             / SID 1719 /
  [-186, "eth0"]    / SID 1533 with name = "eth0" /
]

RES:  2.05 Content (Content-Format :application/yang-value+cbor)
[
  "2014-10-26T12:16:31Z",
  {
    +4 : "eth0",                / name (SID 1537) /
    +1 : "Ethernet adaptor",    / description (SID 1534) /
    +5 : 1179,                  / type (SID 1538), identity /
                                / ethernetCsmacd (SID 1179) /
    +2 : true                   / enabled (SID 1535) /
  }
]
```

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 CBOR arrays so messages will preserve their order.

5.3.2. POST

The CoAP POST operation is used in CoMI for creation of data node 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 a data node resource is sent with a confirmable CoAP POST message. The URI specifies the data node to be instantiated at the exception of list intances. In this case, for compactness, the URI specifies the list for which an instance is created.

```
FORMAT:
  POST /c/<instance identifier>
  (Content-Format :application/yang-value+cbor)
  data-node-value

  2.01 Created
```

If the data node resource already exists, then the POST request MUST fail and a "4.09 Conflict" response code MUST be returned

5.3.2.1. Post example

The example uses the interface list from Appendix C.1. Example is creating a new list instance within the interface list (SID = 1533):

```
REQ: POST /c/X9 (Content-Format :application/yang-value+cbor)
{
  +4 : "eth5",               / name (SID 1537) /
  +1 : "Ethernet adaptor",  / description (SID 1534) /
  +5 : 1179,                 / type (SID 1538), identity /
                             / ethernetCsmacd (SID 1179) /
  +2 : true                  / enabled (SID 1535) /
}


RES: 2.01 Created
```

5.3.3. PUT

A data node resource instance is created or replaced with the PUT method. A request to set the value of a data node instance is sent with a confirmable CoAP PUT message.

```
FORMAT:
  PUT /c/<instanceidentifier>
           (Content-Format :application/yang-value+cbor)
  data-node-value

  2.01 Created
```

5.3.3.1. PUT example

The example uses the interface list from Appendix C.1. Example is renewing an instance of the list interface (SID = 1533) with key name="eth0":

```
REQ: PUT /c/X9?k="eth0"
(Content-Format :application/yang-value+cbor)
{
  +4 : "eth0",             / name (SID 1537) /
  +1 : "Ethernet adaptor", / description (SID 1534) /
  +5 : 1179,               / type (SID 1538), identity /
                           / ethernetCsmacd ( SID 1179) /
  +2 : true                / enabled (SID 1535) /
}

RES:  2.04 Changed
```

5.3.4. iPATCH

One or multiple data node instances are replaced with the idempotent iPATCH method [RFC8132]. A request is sent with a confirmable CoAP iPATCH message.

There are no Uri-Query options for the iPATCH method.

The processing of the iPATCH command is specified by Content-Format application/yang-patch+cbor. In summary, if the CBOR patch payload contains a data node instance that is not present in the target, this instance is added. If the target contains the specified instance, the content of this instance is replaced with the value of the payload. A null value indicates the removal of an existing data node instance.

```
FORMAT:
  iPATCH /c (Content-Format :application/yang-patch+cbor)
  ordered map of instance-identifier, data-node-value

  2.04 Changed
```

5.3.4.1. iPATCH example

In this example, a CoMI client requests the following operations:

- o Set "/system/ntp/enabled" (SID 1751) to true.
- o Remove the server "tac.nrc.ca" from the"/system/ntp/server" (SID 1752) list.
- o Add the server "NTP Pool server 2" to the list "/system/ntp/server" (SID 1752).

```
REQ: iPATCH /c (Content-Format :application/yang-patch+cbor)
[
  1751 , true,                          / enabled (1751) /
  [+1, "tac.nrc.ca"], null,             / server (SID 1752) /
  +0,                                   / server (SID 1752) /
    {
      +3 : "tic.nrc.ca",                / name (SID 1755) /
      +4 : true,                        / prefer (SID 1756) /
      +5 : {                            / udp (SID 1757) /
        +1 : "132.246.11.231"           / address (SID 1758) /
      }
    }
]

RES: 2.04 Changed
```

5.3.5. DELETE

A data node resource is deleted with the DELETE method.

```
FORMAT:
  Delete /c/<instance identifier>

  2.02 Deleted
```

5.3.5.1. DELETE example

The example uses the interface list from Appendix C.3. Example is deleting an instance of the interface list (SID = 1533):

```
REQ:   DELETE /c/X9?k="eth0"

RES:   2.02 Deleted
```

5.4. Full datastore access

The methods GET, PUT, POST, and DELETE can be used to request, replace, create, and delete a whole datastore respectively.

```
FORMAT:
  GET /c

  2.05 Content (Content-Format: application/yang-tree+cbor)
  ordered map of single-instance-identifier, data-node-value
```

FORMAT:
  PUT /c (Content-Format: application/yang-tree+cbor)
  ordered map of single-instance-identifier, data-node-value

  2.04 Changed

FORMAT:
  POST /c (Content-Format: application/yang-tree+cbor)
  ordered map of single-instance-identifier, data-node-value

  2.01 Created

FORMAT:
   DELETE /c

  2.02 Deleted

The content of the ordered map represents the complete datastore of the server at the GET indication of after a successful processing of a PUT or POST request. When an Ordered map is used to carry a whole datastore, all data nodes MUST be identified using single instance identifiers (i.e. a SID), list instance identifiers are not allowed.

5.4.1. Full datastore examples

The example uses the interface list and the clock container from Appendix C.3. Assume that the datastore contains two modules ietf-system (SID 1700) and ietf-interfaces (SID 1500); they contain the list interface (SID 1533) with one instance and the container Clock (SID 1717). After invocation of GET, a map with these two modules is returned:

```
REQ:  GET /c

RES: 2.05 Content (Content-Format :application/yang-tree+cbor)
[
  1717,                            / Clock (SID 1717) /
    {
      +2: "2016-10-26T12:16:31Z", / current-datetime (SID 1719) /
      +1: "2014-10-05T09:00:00Z"  / boot-datetime (SID 1718) /
    },
  -186,                             / clock (SID 1533) /
    {
      +4 : "eth0",                 / name (SID 1537) /
      +1 : "Ethernet adaptor",     / description (SID 1534) /
      +5 : 1179,                   / type (SID 1538), identity: /
                                   / ethernetCsmacd (SID 1179) /
      +2 : true                    / enabled (SID 1535) /
    }
]
```

5.5. Event stream

Event notification is an essential function for the management of servers. CoMI allows notifications specified in YANG [RFC5277] to be reported to a list of clients. The recommended path of the default event stream is /s. The server MAY support additional event stream resources to address different notification needs.

Reception of notification instances is enabled with the CoAP Observe [RFC7641] function. Clients subscribe to the notifications by sending a GET request with an "Observe" option, specifying the /s resource when the default stream is selected.

Each response payload carries one or multiple notifications. The number of notification reported and the conditions used to remove notifications from the reported list is left to the implementers. When multiple notifications are reported, they MUST be ordered starting from the newest notification at index zero.

An example implementation is:

> Every time an event is generated, the generated notification instance is appended to the chosen stream(s). After appending the instance, the content of the instance is sent to all clients observing the modified stream.
>
> Dependending on the storage space allocated to the notification stream, the oldest notifications that do not fit inside the notification stream storage space are removed.

```
FORMAT:
  Get /<stream-resource> Observe(0)

  2.05 Content (Content-Format :application/yang-tree+cbor)
  ordered map of instance-identifier, data-node-value
```

The array of data node instances may contain identical entries which have been generated at different times.

5.5.1. Notify Examples

Suppose the server generates the event specified in Appendix C.4. By executing a GET on the /s resource the client receives the following response:

```
REQ:  GET /s Observe(0) Token(0x93)

RES:  2.05 Content (Content-Format :application/yang-tree+cbor)
                        Observe(12) Token(0x93)
[
  60010,                  / example-port-fault (SID 60010) /
    {
      +1 : "0/4/21",      / port-name (SID 60011) /
      +2 : "Open pin 2"   / port-fault (SID 60012) /
    },
  +0,                     / example-port-fault (SID 60010) /
    {
      +1 : "1/4/21",      / port-name (SID 60011) /
      +2 : "Open pin 5"   / port-fault (SID 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 periodically. 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. The response payload contains the values of the output container when specified. Both the input and output containers are encoded in CBOR using the rules defined in [I-D.ietf-core-yang-cbor] section 4.2.1. Root data nodes are encoded using the delta between the current SID and the SID of the invoked instance identifier a specified by the URI.

The returned success response code is 2.05 Content.

```
FORMAT:
  POST /c/<instance identifier>
           (Content-Format :application/yang-value+cbor)
  data-node-value

  2.05 Content (Content-Format :application/yang-value+cbor)
  data-node-value
```

5.6.1. RPC Example

The example is based on the YANG action specification of Appendix C.2. A server list is specified and the action "reset" (SID 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-value+cbor)
{
  +1 : "2016-02-08T14:10:08Z09:00" / reset-at (SID 60003) /
}

RES:  2.05 Content (Content-Format :application/yang-value+cbor)
{
  +2 : "2016-02-08T14:10:08Z09:18" / reset-finished-at (SID 60004)/
}
```

6. Access to MIB Data

Appendix C.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 /
       60024 : h'0A000033',    / ipNetToPhysicalNetAddress /
       60025 : h'00000A01172D',/ ipNetToPhysicalPhysAddress /
       60026 : 2333943,        / ipNetToPhysicalLastUpdated /
       60027 : 4,              / ipNetToPhysicalType /
       60028 : 1,              / ipNetToPhysicalState /
       60029 : 1               / ipNetToPhysicalRowStatus /
     },
     {
       60022 : 1,              / ipNetToPhysicalIfIndex /
       60023 : 1,              / ipNetToPhysicalNetAddressType /
       60024 : h'09020304',    / ipNetToPhysicalNetAddress  /
       60025 : h'00000A36200A',/ ipNetToPhysicalPhysAddress /
       60026 : 2329836,        / ipNetToPhysicalLastUpdated /
       60027 : 3,              / ipNetToPhysicalType /
       60028 : 6,              / ipNetToPhysicalState /
       60029 : 1               / ipNetToPhysicalRowStatus /
     }
   ]
}
```

In this example one instance of /ip/ipNetToPhysicalEntry (SID 60021, base64: Oz1) that matches the keys ipNetToPhysicalIfIndex = 1, ipNetToPhysicalNetAddressType = ipv4 and ipNetToPhysicalNetAddress = 9.2.3.4 (h'09020304', base64: CQIDBA) is requested.

REQ: GET example.com/c/Oz1?k="1,1,CQIDBA"

RES: 2.05 Content (Content-Format: application/yang-value+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 data need to be transported, 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. On the server, values are changed, lists are re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the resource, 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 the 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.datastore" [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, the value "/c" is used as an example. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.c.datastore

RES: 2.05 Content
</c>; rt="core.c.datastore"
```

Implemented data nodes MAY be discovered using the standard CoAP resource discovery. The implementation can add the data node identifiers (SID) supported to /.well-known/core with rt="core.c.datanode". The available SIDs can be discovered by sending a GET request to "/.well-known/core" including a resource type (rt) parameter with the value "core.c.datanode". Upon success, the return payload will contain the registered SIDs and their location.

The example below shows the discovery of the presence and location of data nodes.

```
REQ: GET /.well-known/core?rt=core.c.datanode

RES: 2.05 Content
</c/BaAiN>; rt="core.c.datanode",
</c/CF_fA>; rt="core.c.datanode"
```

The list of data nodes may become prohibitively long. Therefore, it is recommended to discover the details about the YANG modules implemented by reading a YANG module library (e.g. "ietf-comi-yang-library" ad defined by [I-D.veillette-core-yang-library]).

The resource "/mod.uri" is used to retrieve the location of the YANG module library. This library 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.

The following example shows the URI of a local instance of container modules-state (SID=1802) as defined in [I-D.veillette-core-yang-library].

```
REQ: GET example.com/mod.uri

RES: 2.05 Content (Content-Format: text/plain; charset=utf-8)
example.com/c/cK
```

The following example shows the URI of a remote instance of same container.

```
REQ: GET example.com/mod.uri

RES: 2.05 Content (Content-Format: text/plain; charset=utf-8)
example-remote-server.com/group17/cK
```

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.

Errors returned by a CoMI server can be broken into two categories, those associated to the CoAP protocol itself and those generated during the validation of the YANG data model constrains as described in [RFC7950] section 8.

The following list of common CoAP errors should be implemented by CoMI servers. This list is not exhaustive, other errors defined by CoAP and associated RFCs may be applicable.

- o Error 4.01 (Unauthorized) is returned by the CoMI server when the CoMI client is not authorized to perform the requested action on the targeted resource (i.e. data node, datastore, rpc, action or event stream).

- o Error 4.02 (Bad Option) is returned by the CoMI server when one or more CoAP options are unknown or malformed.

- o Error 4.04 (Not Found) is returned by the CoMI server when the CoMI client is requesting a non-instantiated resource (i.e. data node, datastore, rpc, action or event stream).

- o Error 4.05 (Method Not Allowed) is returned by the CoMI server when the CoMI client is requesting a method not supported on the targeted resource. (e.g. GET on an rpc, PUT or POST on a data node with "config" set to false).

- o Error 4.08 (Request Entity Incomplete) is returned by the CoMI server if one or multiple blocks of a block transfer request is missing, see [RFC7959] for more details.

- o Error 4.13 (Request Entity Too Large) may be returned by the CoMI server during a block transfer request, see [RFC7959] for more details.

- o Error 4.15 (Unsupported Content-Format) is returned by the CoMI server when the Content-Format used in the request don't match those specified in section 2.3.

CoMI server MUST also enforce the different constraints associated to the YANG data models implemented. These constraints are described in [RFC7950] section 8. These errors are reported using the CoAP error code 4.00 (Bad Request) and may have the following error container as payload. The YANG definition and associated .sid file are available in Appendix A and Appendix B. The error container is encoded using delta value equal to the SID of the current schema node minus the SID of the parent container (i.e 1024).

```
+--rw error!
   +--rw error-tag             identityref
   +--rw error-app-tag?        identityref
   +--rw data-node-in-error?   instance-identifier
   +--rw error-message?        string
```

The following error-tag and error-app-tag are defined by the ietf-comi YANG module, these tags are implemented as YANG identity and can be extended as needed.

- o error-tag operation-failed is returned by the CoMI server when the operation request cannot be processed successfully.

  * error-app-tag malformed-message is returned by the CoMI server when the payload received from the CoMI client don't contain a well-formed CBOR content as defined in [RFC7049] section 3.3 or don't comply with the CBOR structure defined within this document.

  * error-app-tag data-not-unique is returned by the CoMI server when the validation of the 'unique' constraint of a list or leaf-list fails.

  * error-app-tag too-many-elements is returned by the CoMI server when the validation of the 'max-elements' constraint of a list or leaf-list fails.

  * error-app-tag too-few-elements is returned by the CoMI server when the validation of the 'min-elements' constraint of a list or leaf-list fails.

  * error-app-tag must-violation is returned by the CoMI server when the restrictions imposed by a 'must' statement are violated.

  * error-app-tag duplicate is returned by the CoMI server when a client tries to create a duplicate list or leaf-list entry.

- o error-tag invalid-value is returned by the CoMI server when the CoMI client tries to update or create a leaf with a value encoded using an invalid CBOR datatype or if the 'range', 'length', 'pattern' or 'require-instance' constrain is not fulfilled.

   * error-app-tag invalid-datatype is returned by the CoMI server when CBOR encoding don't follow the rules set by or when the value is incompatible with the YANG Built-In type. (e.g. a value greater than 127 for an int8, undefined enumeration)

   * error-app-tag not-in-range is returned by the CoMI server when the validation of the 'range' property fails.

   * error-app-tag invalid-length is returned by the CoMI server when the validation of the 'length' property fails.

   * error-app-tag pattern-test-failed is returned by the CoMI server when the validation of the 'pattern' property fails.

o error-tag missing-element is returned by the CoMI server when the operation requested by a CoMI client fail to comply with the 'mandatory' constraint defined. The 'mandatory' constraint is enforced for leafs and choices, unless the node or any of its ancestors have a 'when' condition or 'if-feature' expression that evaluates to 'false'.

   * error-app-tag missing-key is returned by the CoMI server to further qualify an missing-element error. This error is returned when the CoMI client tries to create or list instance, without all the 'key' specified or when the CoMI client tries to delete a leaf listed as a 'key'.

   * error-app-tag missing-input-parameter is returned by the CoMI server when the input parameters of an RPC or action are incomplete.

o error-tag unknown-element is returned by the CoMI server when the CoMI client tries to access a data node of a YANG module not supported, of a data node associated to an 'if-feature' expression evaluated to 'false' or to a 'when' condition evaluated to 'false'.

o error-tag bad-element is returned by the CoMI server when the CoMI client tries to create data nodes for more than one case in a choice.

o error-tag data-missing is returned by the CoMI server when a data node required to accept the request is not present.

   * error-app-tag instance-required is returned by the CoMI server when a leaf of type 'instance-identifier' or 'leafref' marked with require-instance set to 'true' refers to an instance that does not exist.

* error-app-tag missing-choice is returned by the CoMI server when no nodes exist in a mandatory choice.

o error-tag error is returned by the CoMI server when an unspecified error has occurred.

For example, the CoMI server might return the following error.

```
RES:  4.00 Bad Request (Content-Format :application/yang-value+cbor)
{
 +4 : 1020,        / error-tag = invalid-value /
 +2 : 1012,        / error-app-tag = not-in-range /
 +1 : 1736,        / data-node-in-error = timezone-utc-offset /
 +3 : "maximum value exceeded" / error-message /
}
```

10. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. CoMI re-uses the security mechanisms already available to CoAP, 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

11.1. Resource Type (rt=) Link Target Attribute Values Registry

This document adds the following resource type to the "Resource Type (rt=) Link Target Attribute Values", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Value	Description	Reference
core.c.datastore	YANG datastore	RFC XXXX
core.c.datanode	YANG data node	RFC XXXX
core.c.liburi	YANG module library	RFC XXXX
core.c.eventstream	YANG event stream	RFC XXXX

// RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

11.2. CoAP Content-Formats Registry

This document adds the following Content-Format to the "CoAP Content-Formats", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Media Type	Excoding ID	Reference
application/yang-value+cbor	XXX	RFC XXXX
application/yang-values+cbor	XXX	RFC XXXX
application/yang-selectors+cbor	XXX	RFC XXXX
application/yang-tree+cbor	XXX	RFC XXXX
application/yang-ipatch+cbor	XXX	RFC XXXX

// RFC Ed.: replace XXX with assigned IDs and remove this note. // RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

11.3. Media Types Registry

This document adds the following media types to the "Media Types" registry.

Name	Template	Reference
yang-value+cbor	application/yang-value+cbor	RFC XXXX
yang-values+cbor	application/yang-values+cbor	RFC XXXX
yang-selectors+cbor	application/yang-selectors+cbor	RFC XXXX
yang-tree+cbor	application/yang-tree+cbor	RFC XXXX
yang-ipatch+cbor	application/yang-ipatch+cbor	RFC XXXX

Each of these media types share the following information:

o Subtype name: <as listed in table>

o Required parameters: N/A

o Optional parameters: N/A

o Encoding considerations: binary

o Security considerations: See the Security Considerations section of RFC XXXX

o Interoperability considerations: N/A

o Published specification: RFC XXXX

o Applications that use this media type: CoMI

o Fragment identifier considerations: N/A

o Additional information:

* Deprecated alias names for this type: N/A

* Magic number(s): N/A

* File extension(s): N/A

* Macintosh file type code(s): N/A

o Person & email address to contact for further information: iesg&ietf.org

- o Intended usage: COMMON
- o Restrictions on usage: N/A
- o Author: Michel Veillette, ietf&augustcellars.com
- o Change Controller: IESG
- o Provisional registration? No

// RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

11.4. Concise Binary Object Representation (CBOR) Tags Registry

This document adds the following tags to the "Concise Binary Object Representation (CBOR) Tags" registry.

Tag	Data Item	Semantics	Reference
xxx	array	Oedered map	RFC XXXX

// RFC Ed.: replace xxx by the assigned Tag and remove this note. // RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

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 D.

The draft has benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Tanguy Ropitault, Juergen Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

13. References

13.1. Normative References

[I-D.ietf-core-sid]
Veillette, M., Pelov, A., Turner, R., Minaburo, A., and A. Somaraju, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-01 (work in progress), May 2017.

[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-04 (work in progress), February 2017.

[I-D.veillette-core-yang-library]
Veillette, M., "Constrained YANG Module Library", draft-veillette-core-yang-library-00 (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>.

[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>.

[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>.

[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>.

[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>.

[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>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <http://www.rfc-editor.org/info/rfc8132>.

13.2. Informative References

[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-09 (work in progress), March 2017.

[netconfcentral]
YUMAworks, "NETCONF Central: library of YANG modules", Web http://www.netconfcentral.org/modulelist.

[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>.

[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>.

[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>.

[XML] W3C, "Extensible Markup Language (XML)", Web http://www.w3.org/xml.

[yang-cbor] Veillette, M., "yang-cbor Registry", Web https://github.com/core-wg/yang-cbor/tree/master/registry/.

Appendix A. ietf-comi YANG module

```
<CODE BEGINS> file "ietf-comi@2017-07-01.yang"
module ietf-comi {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-comi";
  prefix comi;

  organization
    "IETF Core Working Group";

  contact
    "Michel Veillette
     <mailto:michel.veillette@trilliantinc.com>

     Alexander Pelov
     <mailto:alexander@ackl.io>

     Peter van der Stok
     <mailto:consultancy@vanderstok.org>

     Andy Bierman
     <mailto:andy@yumaworks.com>";

  description
    "This module contains the different definitions required
     by the CoMI protocol.";
```

```
   revision 2017-07-01 {
      description
       "Initial revision.";
     reference
       "draft-ietf-core-comi";
   }

   identity error-tag {
     description
       "Base identity for error-tag.";
   }

   identity operation-failed {
     base error-tag;
     description
       "Returned by the CoMI server when the operation request
        can't be processed successfully.";
   }

   identity invalid-value {
     base error-tag;
     description
       "Returned by the CoMI server when the CoMI client tries to
        update or create a leaf with a value encoded using an
        invalid CBOR datatype or if the 'range', 'length',
        'pattern' or 'require-instance' constrain is not
        fulfilled.";
   }

   identity missing-element {
     base error-tag;
     description
       "Returned by the CoMI server when the operation requested
        by a CoMI client fails to comply with the 'mandatory'
        constraint defined. The 'mandatory' constraint is
        enforced for leafs and choices, unless the node or any of
        its ancestors have a 'when' condition or 'if-feature'
        expression that evaluates to 'false'.";
   }

   identity unknown-element {
     base error-tag;
     description
       "Returned by the CoMI server when the CoMI client tries to
        access a data node of a YANG module not supported, of a
        data node associated with an 'if-feature' expression
        evaluated to 'false' or to a 'when' condition evaluated
        to 'false'.";
```

```
     }

     identity bad-element {
       base error-tag;
       description
         "Returned by the CoMI server when the CoMI client tries to
          create data nodes for more than one case in a choice.";
     }

     identity data-missing {
       base error-tag;
       description
         "Returned by the CoMI server when a data node required to
          accept the request is not present.";
     }

     identity error {
       base error-tag;
       description
         "Returned by the CoMI server when an unspecified error has
         occurred.";
     }

     identity error-app-tag {
       description
         "Base identity for error-app-tag.";
     }


     identity malformed-message {
       base error-app-tag;
       description
         "Returned by the CoMI server when the payload received
          from the CoMI client don't contain a well-formed CBOR
          content as defined in [RFC7049] section 3.3 or don't
          comply with the CBOR structure defined within this
          document.";
     }

     identity data-not-unique {
       base error-app-tag;
       description
         "Returned by the CoMI server when the validation of the
          'unique' constraint of a list or leaf-list fails.";
     }

     identity too-many-elements {
       base error-app-tag;
```

```
       description
         "Returned by the CoMI server when the validation of the
          'max-elements' constraint of a list or leaf-list fails.";
     }

     identity too-few-elements {
       base error-app-tag;
       description
         "Returned by the CoMI server when the validation of the
          'min-elements' constraint of a list or leaf-list fails.";
     }

     identity must-violation {
       base error-app-tag;
       description
         "Returned by the CoMI server when the restrictions
          imposed by a 'must' statement are violated.";
     }

     identity duplicate {
       base error-app-tag;
       description
         "Returned by the CoMI server when a client tries to create
          a duplicate list or leaf-list entry.";
     }

     identity invalid-datatype {
       base error-app-tag;
       description
         "Returned by the CoMI server when CBOR encoding is
          incorect or when the value encoded is incompatible with
          the YANG Built-In type. (e.g. value greater than 127
          for an int8, undefined enumeration).";
     }

     identity not-in-range {
       base error-app-tag;
       description
         "Returned by the CoMI server when the validation of the
          'range' property fails.";
     }

     identity invalid-length {
       base error-app-tag;
       description
         "Returned by the CoMI server when the validation of the
          'length' property fails.";
     }
```

```
     identity pattern-test-failed {
       base error-app-tag;
       description
         "Returned by the CoMI server when the validation of the
          'pattern' property fails.";
     }

     identity missing-key {
       base error-app-tag;
       description
         "Returned by the CoMI server to further qualify a
          missing-element error. This error is returned when the
          CoMI client tries to create or list instance, without all
          the 'key' specified or when the CoMI client tries to
          delete a leaf listed as a 'key'.";
     }

     identity missing-input-parameter {
       base error-app-tag;
       description
         "Returned by the CoMI server when the input parameters
          of a RPC or action are incomplete.";
     }

     identity instance-required {
       base error-app-tag;
       description
         "Returned by the CoMI server when a leaf of type
          'instance-identifier' or 'leafref' marked with
          require-instance set to 'true' refers to an instance
          that does not exist.";
     }

     identity missing-choice {
       base error-app-tag;
       description
         "Returned by the CoMI server when no nodes exist in a
          mandatory choice.";
     }

     container error {
       presence "Error paylaod";

       description
         "Optional payload of a 4.00 Bad Request CoAP error.";

       leaf error-tag {
         type identityref {
```

```
          base error-tag;
        }
        mandatory true;
        description
          "The enumerated error-tag.";
      }

      leaf error-app-tag {
        type identityref {
          base error-app-tag;
        }
        description
          "The application-specific error-tag.";
      }

      leaf data-node-in-error {
        type instance-identifier;
        description
          "When the error reported is caused by a specific data node,
           this leaf identifies the data node in error.";
      }

      leaf error-message {
        type string;
        description
          "A message describing the error.";
      }
    }
  }
<CODE ENDS>
```

Appendix B. ietf-comi .sid file


```
{
  "assignment-ranges": [
    {
      "entry-point": 1000,
      "size": 100
    }
  ],
  "module-name": "ietf-comi",
  "module-revision": "2017-07-01",
  "items": [
    {
      "type": "Module",
      "label": "ietf-comi",
      "sid": 1000
    },
```

```
        {
          "type": "identity",
          "label": "/error-app-tag",
          "sid": 1001
        },
        {
          "type": "identity",
          "label": "/error-app-tag/data-not-unique",
          "sid": 1002
        },
        {
          "type": "identity",
          "label": "/error-app-tag/duplicate",
          "sid": 1003
        },
        {
          "type": "identity",
          "label": "/error-app-tag/instance-required",
          "sid": 1004
        },
        {
          "type": "identity",
          "label": "/error-app-tag/invalid-datatype",
          "sid": 1005
        },
        {
          "type": "identity",
          "label": "/error-app-tag/invalid-length",
          "sid": 1006
        },
        {
          "type": "identity",
          "label": "/error-app-tag/malformed-message",
          "sid": 1007
        },
        {
          "type": "identity",
          "label": "/error-app-tag/missing-choice",
          "sid": 1008
        },
        {
          "type": "identity",
          "label": "/error-app-tag/missing-input-parameter",
          "sid": 1009
        },
        {
          "type": "identity",
          "label": "/error-app-tag/missing-key",
```

```
    "sid": 1010
  },
  {
    "type": "identity",
    "label": "/error-app-tag/must-violation",
    "sid": 1011
  },
  {
    "type": "identity",
    "label": "/error-app-tag/not-in-range",
    "sid": 1012
  },
  {
    "type": "identity",
    "label": "/error-app-tag/pattern-test-failed",
    "sid": 1013
  },
  {
    "type": "identity",
    "label": "/error-app-tag/too-few-elements",
    "sid": 1014
  },
  {
    "type": "identity",
    "label": "/error-app-tag/too-many-elements",
    "sid": 1015
  },
  {
    "type": "identity",
    "label": "/error-tag",
    "sid": 1016
  },
  {
    "type": "identity",
    "label": "/error-tag/bad-element",
    "sid": 1017
  },
  {
    "type": "identity",
    "label": "/error-tag/data-missing",
    "sid": 1018
  },
  {
    "type": "identity",
    "label": "/error-tag/error",
    "sid": 1019
  },
  {
```

```
          "type": "identity",
          "label": "/error-tag/invalid-value",
          "sid": 1020
        },
        {
          "type": "identity",
          "label": "/error-tag/missing-element",
          "sid": 1021
        },
        {
          "type": "identity",
          "label": "/error-tag/operation-failed",
          "sid": 1022
        },
        {
          "type": "identity",
          "label": "/error-tag/unknown-element",
          "sid": 1023
        },
        {
          "type": "node",
          "label": "/error",
          "sid": 1024
        },
        {
          "type": "node",
          "label": "/error/data-node-in-error",
          "sid": 1025
        },
        {
          "type": "node",
          "label": "/error/error-app-tag",
          "sid": 1026
        },
        {
          "type": "node",
          "label": "/error/error-message",
          "sid": 1027
        },
        {
          "type": "node",
          "label": "/error/error-tag",
          "sid": 1028
        }
      ]
    }
```

Appendix C. YANG example specifications

This appendix shows five 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.

C.1. ietf-system

Excerpt of the YANG module ietf-system [RFC7317].

```
module ietf-system {                     // SID 1700
  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;
        }
      }
    }
  }
```

C.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;
         }
       }
     }
   }
}
```

C.3. interfaces

Excerpt of the YANG module ietf-interfaces [RFC7223].

```
module ietf-interfaces {                // SID 1500
  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;
          }
        }
      }
    }
  }
}
```

C.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";
             }
           }
         }
```

C.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
              ipNetToPhysicalNetAddress";
         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 D. Comparison with LWM2M

TO DO Need updated text based on the current version of CoMI. Multiple assumptions used in the original text are no more valid.

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Email: michel.veillette@trilliantinc.com

Peter van der Stok (editor)
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

Email: andy@yumaworks.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: March 18, 2018

Z. Shelby
ARM
Z. Vial
Schneider-Electric
M. Koster
SmartThings
C. Groves
J. Zhu
Huawei
B. Silverajan, Ed.
Tampere University of Technology

September 14, 2017

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-04

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

Status of This Memo

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	gt	xsd:decimal
Less Than	lt	xsd:decimal

Table 2: Binding Attributes Summary

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 (gt)

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 gt value. If the value continues to rise, no new state synchronizations are generated as a result of gt. 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 (lt)

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 lt value. If the value continues to fall no new state synchronizations are generated as a result of lt. 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, gt and lt 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, gt or lt are met, if pmin has not been exceeded then no state synchronization occurs. Likewise if st, gt or lt 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, gt or lt 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 gt and lt are included gt MUST be greater than lt otherwise an error CoAP error code 4.00 "Bad Request" (or equivalent) MUST be returned.

If st is included in a query with a gt or lt attribute then state synchronizations occur only when the conditions described by st AND gt 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 deleted 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/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"
Res: 2.04 Changed

Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/s/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"

Req: DELETE /bnd/a/light
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.

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}?lt	xsd:decimal
Greater Than	/{resource}?gt	xsd:decimal

Table 4: Resource Observation Attribute Summary

inimum Period: As per Section 3.3.2

aximum 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 Relation Type

This specification registers the new "boundto" 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

- General: Reverted to using "gt" and "lt" from "gth" and "lth" for this draft owing to concerns raised that the attributes are already used in LwM2M with the original names "gt" and "lt".

- New author and editor added.

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, <https://www.rfc-editor.org/info/rfc2119>.

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

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://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, <https://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, <https://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 (gt) example

```
        Observed   CLIENT  SERVER     Actual
    t   State         |      |         State
        ____________  |      |  ____________
    1                 |      |
    2    unknown      |      |     18.5 Cel
    3                 +----->|                  Header: GET
    4                 | GET  |                   Token: 0x4a
    5                 |      |                Uri-Path: temperature
    6                 |      |               Uri-Query: gt="25"
    7                 |      |                 Observe: 0 (register)
    8                 |      |
    9   ____________  |<-----+                  Header: 2.05
   10                 | 2.05 |                   Token: 0x4a
   11    18.5 Cel     |      |                 Observe: 9
   12                 |      |                 Payload: "18.5 Cel"
   13                 |      |
   14                 |      |
   15                 |      |  ____________
   16   ____________  |<-----+                  Header: 2.05
   17                 | 2.05 |     26 Cel        Token: 0x4a
   18    26 Cel       |      |                 Observe: 16
   29                 |      |                 Payload: "26 Cel"
   20                 |      |
   21                 |      |
```

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 (gt) and Period Max (pmax) example

```
        Observed   CLIENT  SERVER     Actual
    t   State         |      |         State
        ____________  |      |  ____________
    1                 |      |
    2    unknown      |      |     18.5 Cel
    3                 +----->|                  Header: GET
    4                 | GET  |                   Token: 0x4a
    5                 |      |                Uri-Path: temperature
    6                 |      |         Uri-Query: pmax="20";gt="25"
    7                 |      |                 Observe: 0 (register)
    8                 |      |
    9   ____________  |<-----+                  Header: 2.05
   10                 | 2.05 |                   Token: 0x4a
   11    18.5 Cel     |      |                 Observe: 9
   12                 |      |                 Payload: "18.5 Cel"
```

```
   13                 |      |
   14                 |      |
   15                 |      |
   16                 |      |
   17                 |      |
   18                 |      |
   19                 |      |
   20                 |      |
   21                 |      |
   22                 |      |
   23                 |      |
   24                 |      |
   25                 |      |
   26                 |      |
   27                 |      |
   28                 |      |
   29                 |      |  _____________
   30   _____________ |<-----+                  Header: 2.05
   31                 | 2.05 |     23 Cel        Token: 0x4a
   32    23 Cel       |      |                 Observe: 30
   33                 |      |                 Payload: "23 Cel"
   34                 |      |
   35                 |      |
   36                 |      |  _____________
   37   _____________ |<-----+                  Header: 2.05
   38                 | 2.05 |     26 Cel        Token: 0x4a
   39    26 Cel       |      |                 Observe: 37
   40                 |      |                 Payload: "26 Cel"
   41                 |      |
   42                 |      |
```

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

Julian Zhu
Huawei
No.127 Jinye Road, Huawei Base, High-Tech Development District
Xi’an, Shaanxi Province
China

Email: jintao.zhu@huawei.com

Bilhanan Silverajan (editor)
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

CoRE Working Group
Internet-Draft
Updates: 7959 (if approved)
Intended status: Standards Track
Expires: May 3, 2018

C. Amsuess
Energy Harvesting Solutions
J. Mattsson
G. Selander
Ericsson AB

October 30, 2017

Echo and Request-Tag
draft-ietf-core-echo-request-tag-00

Abstract

This document defines two optional extensions to the Constrained Application Protocol (CoAP): the Echo option and the Request-Tag option. Each of these options when integrity protected, such as with DTLS or OSCORE, protects against certain attacks on CoAP message exchanges.

The Echo option enables a CoAP server to verify the freshness of a request by requiring the CoAP client to make another request and include a server-provided challenge. The Request-Tag option allows the CoAP server to match message fragments belonging to the same request message, fragmented using the CoAP Block-Wise Transfer mechanism. This document also specifies additional processing requirements on Block1 and Block2 options.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The initial CoAP suite of specifications ([RFC7252], [RFC7641], [RFC7959]) was designed with the assumption that security could be provided on a separate layer, in particular by using DTLS

([RFC6347]). However, for some use cases, additional functionality or extra processing is needed to support secure CoAP operations.

This document specifies two server-oriented CoAP options, the Echo option and the Request-Tag option, addressing the security features request freshness and fragmented message body integrity, respectively. These options in themselves do not replace the need for a security protocol; they specify the format and processing of data which, when integrity protected in a message, e.g. using DTLS ([RFC6347]) or OSCORE ([I-D.ietf-core-object-security]), provide those security features. The Request-Tag option and also the ETag option are mandatory to use with Block1 and Block2, respectively, to secure blockwise operations with multiple representations of a particular resource as is specified in this document.

Additional applications of the options are introduced. For example, Echo can be used to mitigate amplification attacks.

1.1. Request Freshness

A CoAP server receiving a request may not be able to verify when the request was sent by the CoAP client. This remains true even if the request was protected with a security protocol, such as DTLS. This makes CoAP requests vulnerable to certain delay attacks which are particularly incriminating in the case of actuators ([I-D.mattsson-core-coap-actuators]). Some attacks are possible to mitigate by establishing fresh session keys (e.g. performing the DTLS handshake) for each actuation, but in general this is not a solution suitable for constrained environments.

A straightforward mitigation of potential delayed requests is that the CoAP server rejects a request the first time it appears and asks the CoAP client to prove that it intended to make the request at this point in time. The Echo option, defined in this document, specifies such a mechanism which thereby enables the CoAP server to verify the freshness of a request. This mechanism is not only important in the case of actuators, or other use cases where the CoAP operations require freshness of requests, but also in general for synchronizing state between CoAP client and server.

1.2. Fragmented Message Body Integrity

CoAP was designed to work over unreliable transports, such as UDP, and include a lightweight reliability feature to handle messages which are lost or arrive out of order. In order for a security protocol to support CoAP operations over unreliable transports, it must allow out-of-order delivery of messages using e.g. a sliding replay window such as described in Section 4.1.2.6 of DTLS ([RFC6347]).

The Block-Wise Transfer mechanism [RFC7959] extends CoAP by defining the transfer of a large resource representation (CoAP message body) as a sequence of blocks (CoAP message payloads). The mechanism uses a pair of CoAP options, Block1 and Block2, pertaining to the request and response payload, respectively. The blockwise functionality does not support the detection of interchanged blocks between different message bodies to the same endpoint having the same block number. This remains true even when CoAP is used together with a security protocol such as DTLS or OSCORE, within the replay window ([I-D.amsuess-core-request-tag]), which is a vulnerability of CoAP when using RFC7959.

A straightforward mitigation of mixing up blocks from different messages is to use unique identifiers for different message bodies, which would provide equivalent protection to the case where the complete body fits into a single payload. The ETag option [RFC7252], set by the CoAP server, identifies a response body fragmented using the Block2 option. This document defines the Request-Tag option for identifying the request body fragmented using the Block1 option, similar to ETag, but ephemeral and set by the CoAP client.

1.3. Terminology

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

Unless otherwise specified, the terms "client" and "server" refers to "CoAP client" and "CoAP server", respectively, as defined in [RFC7252].

The terms "payload" and "body" of a message are used as in [RFC7959]. The complete interchange of a request and a response body is called a (REST) "operation". An operation fragmented using [RFC7959] is called a "blockwise operation". A blockwise operation which is fragmenting the request body is called a "blockwise request operation". A blockwise operation which is fragmenting the response body is called a "blockwise response operation".

Two blockwise operations between the same endpoint pair on the same resource are said to be "concurrent" if a block of the second request is exchanged even though the client still intends to exchange further blocks in the first operation. (Concurrent blockwise request operations are impossible with the options of [RFC7959] because the second operation’s block overwrites any state of the first exchange.).

The Echo and Request-Tag options are defined in this document. The concept of two messages being "Request-Tag-matchable" is defined in Section 3.1.

2. The Echo Option

The Echo option is a server-driven challenge-response mechanism for CoAP. The Echo option value is a challenge from the server to the client included in a CoAP response and echoed in a CoAP request.

2.1. Option Format

The Echo Option is elective, safe-to-forward, not part of the cache-key, and not repeatable, see Figure 1.

No.	C	U	N	R	Name	Format	Length	Default	E
TBD					Echo	opaque	8-40	(none)	x

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt and Integrity Protect (when using OSCORE)

Figure 1: Echo Option Summary

The value of the Echo option MUST be a (pseudo-)random bit string of a length of at least 64 bits. A new (pseudo-)random bit string MUST be generated by the server for each use of the Echo option.

2.2. Echo Processing

It is important to identify under what conditions a CoAP request to a resource is required to be fresh. These conditions can for example include what resource is requested, the request method and other data in the request, and conditions in the environment such as the state of the server or the time of the day.

A server MAY include the Echo option in a response. The Echo option MUST NOT be used with empty CoAP requests (i.e. Code=0.00). If the server receives a request which has freshness requirements, and the request does not contain the Echo option, the server SHOULD send a 4.01 Unauthorized response with a Echo option. The server SHOULD cache the transmitted Echo option value and the response transmit time (here denoted t0).

Upon receiving a response with the Echo option within the EXCHANGE_LIFETIME ([RFC7252]) of the original request, the client SHOULD echo the Echo option with the same value in a new request to the server. Upon receiving a 4.01 Unauthorized response with the Echo option in response to a request within the EXCHANGE_LIFETIME of the original request, the client SHOULD resend the original request. The client MAY send a different request compared to the original request.

If the server receives a request which has freshness requirements, and the request contains the Echo option, the server MUST verify that the option value equals a cached value; otherwise the request is not processed further. The server MUST calculate 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 sent. The threshold T is application specific, its value depends e.g. on the freshness requirements of the request. An example message flow is illustrated in Figure 2.

When used to serve freshness requirements, CoAP messages containing the Echo option MUST be integrity protected, e.g. using DTLS or OSCORE ([I-D.ietf-core-object-security]).

If the server loses time synchronization, e.g. due to reboot, it MUST delete all cached Echo option values and response transmission times.

```
                   Client  Server
                      |       |
                      +------>|        Code: 0.03 (PUT)
                      | PUT   |       Token: 0x41
                      |       |    Uri-Path: lock
                      |       |     Payload: 0 (Unlock)
                      |       |
                      |<------+ t0     Code: 4.01 (Unauthorized)
                      | 4.03  |       Token: 0x41
                      |       |        Echo: 0x6c880d41167ba807
                      |       |
                      +------>| t1     Code: 0.03 (PUT)
                      | PUT   |       Token: 0x42
                      |       |    Uri-Path: lock
                      |       |        Echo: 0x6c880d41167ba807
                      |       |     Payload: 0 (Unlock)
                      |       |
                      |<------+        Code: 2.04 (Changed)
                      | 2.04  |       Token: 0x42
                      |       |
```

Figure 2: Echo option message flow

Constrained server implementations can use the mechanisms outlined in Appendix A to minimize the memory impact of having many unanswered Echo responses.

CoAP-CoAP proxies MUST relay the Echo option unmodified, and SHOULD NOT cache responses when a Echo option is present in request or response for more than the exchange. CoAP-HTTP proxies MAY request freshness, especially if they have reason to assume that access may require it (eg. because it is a PUT or POST); how this is determined is out of scope for this document. HTTP-CoAP-Proxies SHOULD respond to Echo challenges themselves if they know from the recent establishing of the connection that the HTTP request is fresh. Otherwise, they SHOULD respond with 503 Service Unavailable, Retry-After: 0 and terminate any underlying Keep-Alive connection. It MAY also use other mechanisms to establish freshness of the HTTP request that are not specified here.

2.3. Applications

1. Actuation requests often require freshness guarantees to avoid accidental or malicious delayed actuator actions.

2. To avoid additional roundtrips for applications with multiple actuator requests in rapid sequence between the same client and

server, the server may use the Echo option (with a new value) in response to a request containing the Echo option. The client then uses the Echo option with the new value in the next actuation request, and the server compares the receive time accordingly.

3. If a server reboots during operation it may need to synchronize state with requesting clients before continuing the interaction. For example, with OSCORE it is possible to reuse a persistently stored security context by synchronizing the Partial IV (sequence number) using the Echo option.

4. When a device joins a multicast/broadcast group the device may need to synchronize state or time with the sender to ensure that the received message is fresh. By synchronizing time with the broadcaster, time can be used for synchronizing subsequent broadcast messages. A server MUST NOT synchronize state or time with clients which are not the authority of the property being synchronized. E.g. if access to a server resource is dependent on time, then the client MUST NOT set the time of the server.

5. A server that sends large responses to unauthenticated peers SHOULD mitigate amplification attacks such as described in Section 11.3 of [RFC7252] (where an attacker would put a victim's address in the source address of a CoAP request). For this purpose, the server MAY ask a client to Echo its request to verify its source address. This needs to be done only once per peer, and limits the range of potential victims from the general Internet to endpoints that have been previously in contact with the server. For this application, the Echo option can be used in messages that are not integrity protected, for example during discovery.

3. The Request-Tag Option

The Request-Tag is intended for use as a short-lived identifier for keeping apart distinct blockwise request operations on one resource from one client. It enables the receiving server to reliably assemble request payloads (blocks) to their message bodies, and, if it chooses to support it, to reliably process simultaneous blockwise request operations on a single resource. The requests must be integrity protected in order to protect against interchange of blocks between different message bodies.

3.1. Option Format

The Request-Tag option has the same properties as the Block1 option: it is critical, unsafe, not part of the cache-key, and not repeatable, see Figure 3.

No.	C	U	N	R	Name	Format	Length	Default	E
TBD	x	x	-		Request-Tag	opaque	0-8	(none)	*

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt and Integrity Protect (when using OSCORE)

Figure 3: Request-Tag Option Summary

[Note to RFC editor: If this document is not released together with OSCORE but before it, the following paragraph and the "E" column above need to move into OSCORE.]

Request-Tag, like the Block1 option, is a special class E option in terms of OSCORE processing (see Section 4.3.1.2 of [I-D.ietf-core-object-security]): The Request-Tag MAY be an inner or outer option. The inner option is encrypted and integrity protected between client and server, and provides message body identification in case of end-to-end fragmentation of requests. The outer option is visible to proxies and labels message bodies in case of hop-by-hop fragmentation of requests.

The Request-Tag option is only used in request messages, and only in conjunction with the Block1 option.

Two messages are defined to be Request-Tag-matchable if and only if they are sent from and to the same end points (including security associations), and target the same URI, and if either neither carries a Request-Tag option, or both carry exactly one Request-Tag option and the option values are of same length and content.

The Request-Tag mechanism is applied independently on the server and client sides of CoAP-CoAP proxies. CoAP-HTTP proxies and HTTP-CoAP proxies can use Request-Tag on their CoAP sides; it is not applicable to HTTP requests.

For each separate blockwise request operation, the client can choose a Request-Tag value, or choose not to set a Request-Tag. Creating a new request operation whose messages are Request-Tag-matchable to a previous operation is called request tag recycling. Clients MUST NOT recycle a request tag unless the first operation has concluded. What constitutes a concluded operation depends on the application, and is outlined individually in Section 3.3.

Clients are encouraged to generate compact messages. This means sending messages without Request-Tag options whenever possible, and using short values when the absent option can not be recycled.

3.2. Request-Tag Processing

A server MUST NOT act on any two blocks in the same blockwise request operation that are not Request-Tag-matchable. This rule applies independent of whether the request actually carries a Request-Tag option (in this case, the request can only be acted on together with other messages not carrying the option, as per matchability definition).

As not all messages from the same source can be combined any more, a block not matchable to the first Block1 cannot overwrite context kept for an operation under a different tag (cf. [RFC7959] Section 2.5). The server is still under no obligation to keep state of more than one transaction. When an operation is in progress and a second one cannot be served at the same time, the server MUST respond to the second request with a 5.03 (Service Unavailable) response code and SHOULD indicate the time it is willing to wait for additional blocks in the first operation using the Max-Age option, as specified in Section 5.9.3.4 of [RFC7252].

A server receiving a Request-Tag MUST treat it as opaque and make no assumptions about its content or structure.

Two messages being Request-Tag-matchable is a necessary but not sufficient condition for being part of the same operation. They can still be treated as independent messages by the server (e.g. when it sends 2.01/2.04 responses for every block), or initiate a new operation (overwriting kept context) when the later message carries Block1 number 0.

If a request that uses Request-Tag is rejected with 4.02 Bad Option, the client MAY retry the operation without it, but then it MUST serialize all operations that affect the same resource. Security requirements can forbid dropping the use of Request-Tag mechanism.

3.3. Applications

3.3.1. Body Integrity Based on Payload Integrity

When a client fragments a request body into multiple message payloads, even if the individual messages are integrity protected, it is still possible for a man-in-the-middle to maliciously replace later operation's blocks with earlier operation's blocks (see Section 3.2 of [I-D.amsuess-core-request-tag]). Therefore, the integrity protection of each block does not extend to the operation's request body.

In order to gain that protection, use the Request-Tag mechanism as follows:

- o The message payloads MUST be integrity protected end-to-end between client and server.

- o The client MUST NOT recycle a request tag unless the previous blockwise request operation that used matchable Request-Tags has concluded.

- o The client MUST NOT regard a blockwise request operation as concluded unless all of the messages the client previously sent in the operation have been confirmed by the message integrity protection mechanism, or are considered invalid by the server if replayed.

  Typically, in OSCORE, these confirmations can result either from the client receiving an OSCORE response message matching the request (an empty ACK is insufficient), or because the message's sequence number is old enough to be outside the server's receive window.

  In DTLS, this can only be confirmed if the request message was not retransmitted, and was responded to.

- o The client MUST NOT fall back to not using the Request-Tag mechanisms when receiving a 4.02 Bad Option response.

Authors of other documents (e.g. [I-D.ietf-core-object-security]) are invited to mandate this behavior for clients that execute blockwise interactions over secured transports. In this way, the server can rely on a conforming client to set the Request-Tag option when required, and thereby conclude on the integrity of the assembled body.

Note that this mechanism is implicitly implemented when the security layer guarantees ordered delivery (e.g. CoAP over TLS [I-D.tschofenig-core-coap-tcp-tls]). This is because with each message, any earlier operation can be regarded as concluded by the client, so it never needs to set the Request-Tag option unless it wants to perform concurrent operations.

3.3.2. Multiple Concurrent Blockwise Operations

CoAP clients, especially CoAP proxies, may initiate a blockwise request operation to a resource, to which a previous one is already in progress, and which the new request should not cancel. One example is when a CoAP proxy fragments an OSCORE messages using blockwise (so-called "outer" blockwise, see Section 4.3.1. of [I-D.ietf-core-object-security])), where the Uri-Path is hidden inside the encrypted message, and all the proxy sees is the server’s "/" path.

When a client fragments a message as part of a blockwise request operation, it can do so without a Request-Tag option set. For this application, an operation can be regarded as concluded when a final Block1 option has been sent and acknowledged, or when the client chose not to continue with the operation (e.g. by user choice, or in the case of a proxy when it decides not to take any further messages in the operation due to a timeout). When another concurrent blockwise request operation is made (i.e. before the operation is concluded), the client can not recycle the request tag, and has to pick a new one. The possible outcomes are:

- The server responds with a successful code.

  The concurrent blockwise operations can then continue.

- The server responds 4.02 Bad Option.

  This can indicate that the server does not support Request-Tag. The client should wait for the first operation to conclude, and then try the same request without a Request-Tag option.

- The server responds 5.03 Service Unavailable with a Max-Age option to indicate when it is likely to be available again.

  This can indicate that the server supports Request-Tag, but still is not prepared to handle concurrent requests. The client should wait for as long as the response is valid, and then retry the operation, which may not need to carry a Request-Tag option by then any more.

In the cases where a CoAP proxy receives an error code, it can indicate the anticipated delay by sending a 5.03 Service Unavailable response itself. Note that this behavior is no different from what a CoAP proxy would need to do were it unaware of the Request-Tag option.

4. Block2 / ETag Processing

The same security properties as in Section 3.3.1 can be obtained for blockwise response operations. The threat model here is not an attacker (because the response is made sure to belong to the current request by the security layer), but blocks in the client's cache.

Analogous rules to Section 3.2 are already in place for assembling a response body in Section 2.4 of [RFC7959].

To gain equivalent protection to Section 3.3.1, a server MUST use the Block2 option in conjunction with the ETag option ([RFC7252], Section 5.10.6), and MUST NOT use the same ETag value for different representations of a resource.

5. IANA Considerations

[TBD: Fill out the option templates for Echo and Request-Tag]

6. Security Considerations

Servers that store a Echo challenge per client can be attacked for resource exhaustion, and should consider minimizing the state kept per client, e.g. using a mechanism as described in Appendix A.

7. References

7.1. 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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7959>.

7.2. Informative References

[I-D.amsuess-core-request-tag] Amsuess, C., "Request-Tag option", draft-amsuess-core-request-tag-00 (work in progress), March 2017.

[I-D.ietf-core-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-06 (work in progress), October 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.tschofenig-core-coap-tcp-tls] Bormann, C., Lemay, S., Technologies, Z., and H. Tschofenig, "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", draft-tschofenig-core-coap-tcp-tls-05 (work in progress), November 2015.

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

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

Appendix A. Performance Impact When Using the Echo Option

The Echo option requires the server to keep some state in order to later verify the echoed request.

Instead of caching Echo option values and response transmission times, the server MAY use the encryption of the response transmit time t0 as Echo option value. Such a scheme needs to ensure that the server can detect a replay of a previous encrypted response transmit time.

For example, the server MAY encrypt t0 with AES-CCM-128-64-64 using a (pseudo-)random secret key k generated and cached by the server. A unique IV MUST be used with each encryption, e.g. using a sequence number. If the server loses time synchronization, e.g. due to reboot, then k MUST be deleted and replaced by a new random secret key. When using encrypted response transmit times, the Echo processing is modified in the following way: The verification of cached option value in the server processing is replaced by the verification of the integrity of the encrypted option value using the cached key and IV (e.g. sequence number).

The two methods - (a) the list of cached values, and (b) the encryption of transmit time - have different impact on the implementation:

- o size of cached data (list of cached values vs. key and IV)
- o size of message (typically larger with encrypted time)
- o computation (encryption + decryption vs. generation new nonce + cache + lookup)

In general, the encryption of transmission times is most useful if the number of concurrent requests is high.

A hybrid scheme is also possible: the first Echo option values are cached, and if the number of concurrent requests reach a certain threshold, then encrypted times are used until there is space for storing new values in the list. In that case, the server may need to make both verifications - either that the Echo value is in the list, or that it verifies in decryption - and in either case that the transmission time is valid.

Appendix B. Request-Tag Message Size Impact

In absence of concurrent operations, the Request-Tag mechanism for body integrity (Section 3.3.1) incurs no overhead if no messages are lost (more precisely: in OSCORE, if no operations are aborted due to repeated transmission failure; in DTLS, if no packages are lost), or when blockwise request operations happen rarely (in OSCORE, if only one request operation with losses within the replay window).

In those situations, no message has any Request-Tag option set, and that can be recycled indefinitely.

When the absence of a Request-Tag option can not be recycled any more within a security context, the messages with a present but empty Request-Tag option can be used (1 Byte overhead), and when that is used-up, 256 values from one byte long options (2 Bytes overhead) are available.

In situations where those overheads are unacceptable (e.g. because the payloads are known to be at a fragmentation threshold), the absent Request-Tag value can be made usable again:

- In DTLS, a new session can be established.

- In OSCORE, the sequence number can be artificially increased so that all lost messages are outside of the replay window by the time the first request of the new operation gets processed, and all earlier operations can therefore be regarded as concluded.

Appendix C. Change Log

[The editor is asked to remove this section before publication.]

- Major changes since draft-amsuess-core-repeat-request-tag-00:

  * The option used for establishing freshness was renamed from "Repeat" to "Echo" to reduce confusion about repeatable options.

  * The response code that goes with Echo was changed from 4.03 to 4.01 because the client needs to provide better credentials.

  * The interaction between the new option and (cross) proxies is now covered.

  * Two messages being "Request-Tag matchable" was introduced to replace the older concept of having a request tag value with its slightly awkward equivalence definition.

Authors' Addresses

Christian Amsuess
Energy Harvesting Solutions

Email: c.amsuess@energyharvesting.at

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

CoRE Working Group Z. Shelby
Internet-Draft ARM
Intended status: Informational M. Vial
Expires: March 18, 2018 Schneider-Electric
M. Koster
SmartThings
C. Groves

J. Zhu
Huawei
B. Silverajan, Ed.
Tampere University of Technology
September 14, 2017

Reusable Interface Definitions for Constrained RESTful Environments
draft-ietf-core-interfaces-10

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, Parameter, 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 notes:

o The git repository for the draft is found at https://github.com/core-wg/interfaces

Status of This Memo

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 its associated content formats, data types, URI templates, REST methods, parameters, and responses. Basic interface descriptions are defined for sensors, and actuators.

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 collection 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 -10 to 09:

- Section 1: Amendments to remove discussing properties.

-

- New author and editor added.

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.

- o General: Changed interface type to interface description as that is the term defined in RFC6690.

- o Removed section on future interfaces.

- o Section 8: Updated IANA considerations.

- o Added Appendix A to discuss current state of the art wrt to collections, function sets etc.

Changes from -04 to -05:

- o Removed Link Bindings and Observe attributes. This functionality is now contained in I-D.ietf-core-dynlink.

- o Hypermedia collections have been removed. This is covered in a new T2TRG draft.

- o The WADL description has been removed.

- o Fixed minor typos.

- o Updated references.

Changes from -03 to -04:

- o Fixed tickets #385 and #386.

- o Changed abstract and into to better describe content.

- o Focus on Interface and not function set/profiles in intro.

- o Changed references from draft-core-observe to RFC7641.

- o Moved Function sets and Profiles to section after Interfaces.

- o Moved Observe Attributes to the Link Binding section.

- o Add a Collection section to describe the collection types.

- o Add the Hypermedia Collection Interface Description.

Changes from -02 to -03:

- o Added lt and gt to binding format section.

- o 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.

o Improved the WADL interface definition.

o 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, <https://www.rfc-editor.org/info/rfc2119>.

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

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://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-03 (work in progress), March 2017.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-11 (work in progress), July 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-10 (work in progress), July 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>.

[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, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6573] Amundsen, M., "The Item and Collection Link Relations", RFC 6573, DOI 10.17487/RFC6573, April 2012, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <https://www.rfc-editor.org/info/rfc7396>.

[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>.

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 consistency 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 interface 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
Australia

Email: cngroves.std@gmail.com

Julian Zhu
Huawei
No.127 Jinye Road, Huawei Base, High-Tech Development District
Xi’an, Shaanxi Province
China

Email: jintao.zhu@huawei.com

Bilhanan Silverajan (editor)
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

CoRE Working Group G. Selander
Internet-Draft J. Mattsson
Intended status: Standards Track F. Palombini
Expires: April 28, 2018 Ericsson AB
L. Seitz
SICS Swedish ICT
October 25, 2017

Object Security for Constrained RESTful Environments (OSCORE)
draft-ietf-core-object-security-06

Abstract

This document defines Object Security for Constrained RESTful Environments (OSCORE), a method for application-layer protection of the Constrained Application Protocol (CoAP), using CBOR Object Signing and Encryption (COSE). OSCORE provides end-to-end encryption, integrity and replay protection, as well as a secure message binding. OSCORE is designed for constrained nodes and networks and can be used over any layer and across intermediaries, and also with HTTP. OSCORE may be used to protect group communications as is specified in a separate draft.

Status of This Memo

Copyright Notice

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, and a mapping to HTTP is also specified [RFC8075]. CoAP [RFC7252] references DTLS [RFC6347] for security. CoAP and HTTP proxies require (D)TLS 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 message payload and metadata, in transit between the endpoints. The proxy can also inject, delete, or reorder packets since they are no longer protected by (D)TLS.

This document defines the Object Security for Constrained RESTful Environments (OSCORE) security protocol, protecting CoAP and CoAP-mappable HTTP requests and responses end-to-end across intermediary nodes such as CoAP forward proxies and cross-protocol translators including HTTP-to-CoAP proxies [RFC8075]. In addition to the core CoAP features defined in [RFC7252], OSCORE supports Observe [RFC7641] and Blockwise [RFC7959]. An analysis of end-to-end security for CoAP messages through some types of intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs]. OSCORE protects the Request/Response layer only, and not the CoAP Messaging Layer (Section 2 of [RFC7252]). Therefore, all the CoAP messages mentioned in this document refer to non-Empty CON, NON, and ACK messages [RFC7252]. Additionally, since the message formats for CoAP over unreliable transport [RFC7252] and for CoAP over reliable transport [I-D.ietf-core-coap-tcp-tls] differ only in terms of Messaging Layer, OSCORE can be applied to both unreliable and reliable transports.

OSCORE is designed for constrained nodes and networks and provides an in-layer security protocol that does not depend on underlying layers. OSCORE can be used anywhere where CoAP or HTTP can be used, including non-IP transports (e.g., [I-D.bormann-6lo-coap-802-15-ie]). An extension of OSCORE may also be used to protect group communication for CoAP [I-D.tiloca-core-multicast-oscoap]. The use of OSCORE does not affect the URI scheme and OSCORE can therefore be used with any URI scheme defined for CoAP or HTTP. The application decides the conditions for which OSCORE is required.

OSCORE builds on CBOR Object Signing and Encryption (COSE) [RFC8152], providing end-to-end encryption, integrity, replay protection, and secure message binding. A compressed version of COSE is used, as discussed in Section 8. The use of OSCORE is signaled with the Object-Security CoAP option or HTTP header, defined in Section 2 and Section 10.2. OSCORE is designed to protect as much information as possible, while still allowing proxy operations (Section 10). OSCORE provides protection of message payload, almost all CoAP options, and the RESTful method. The solution transforms a message into an "OSCORE message" before sending, and vice versa after receiving. The OSCORE message is related to the original message in the following way: the original message is translated to CoAP (if not already in CoAP) and the resulting message payload (if present), options not processed by a proxy, and the request/response method (CoAP Code) are protected in a COSE object. The message fields of the original message that are encrypted are transported in the payload of the OSCORE message, and the Object-Security option is included, see Figure 1.

```
Client                                           Server
   |      OSCORE request - POST example.com:      |
   |        Header, Token,                        |
   |        Options: {Object-Security, ...},      |
   |        Payload: COSE ciphertext              |
   +--------------------------------------------->|
   |                                              |
   |<---------------------------------------------+
   |      OSCORE response - 2.04 (Changed):       |
   |        Header, Token,                        |
   |        Options: {Object-Security, ...},      |
   |        Payload: COSE ciphertext              |
   |                                              |
```

Figure 1: Sketch of CoAP with OSCORE

OSCORE may be used in very constrained settings, thanks to its small message size and the restricted code and memory requirements in addition to what is required by CoAP. OSCORE can be combined with transport layer security such as DTLS or TLS, thereby enabling end-to-end security of e.g. CoAP Payload, Options and Code, in combination with hop-by-hop protection of the Messaging Layer, during transport between end-point and intermediary node. Examples of the use of OSCORE are given in Appendix B.

An implementation supporting this specification MAY only implement the client part, MAY only implement the server part, or MAY only implement one of the proxy parts. OSCORE is designed to work with legacy CoAP-to-CoAP forward proxies [RFC7252], but an OSCORE-aware proxy will be more efficient. HTTP-to-CoAP proxies [RFC8075] and CoAP-to-HTTP proxies need to implement respective parts of this specification to work with OSCORE (see Section 10).

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 [RFC8152], 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, Recipient ID/Key, and Common IV are defined in Section 3.1.

2. The CoAP Object-Security Option

The CoAP Object-Security option (see Figure 2) indicates that the CoAP message is an OSCORE message and that it contains a compressed COSE object (see Section 5 and Section 8). The Object-Security option is critical, safe to forward, part of the cache key, and not repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD	x				Object-Security	(*)	0-255	(none)

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable
(*) See below.

Figure 2: The Object-Security Option

The Object-Security option contains the OSCORE flag byte (Section 8), the Sender Sequence Number and the Sender ID when present

(Section 3). The detailed format is specified in Section 8). If the OSCORE flag byte is all zero (0x00) the Option value SHALL be empty (Option Length = 0). An endpoint receiving a CoAP message without payload, that also contains an Object-Security option SHALL treat it as malformed and reject it.

A successful response to a request with the Object-Security option SHALL contain the Object-Security option. Whether error responses contain the Object-Security option depends on the error type (see Section 7).

Since the payload and most options are encrypted Section 4, and the corresponding plain text message fields of the original are not included in the OSCORE message, the processing of these fields does not expand the total message size.

A CoAP proxy 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, see Section 10.1. As the compressed COSE Object is included in the cache key, messages with the Object-Security option will never generate cache hits. For Max-Age processing, see Section 4.2.3.1.

3. The Security Context

OSCORE requires that client and server establish a shared security context used to process the COSE objects. OSCORE uses COSE with an Authenticated Encryption with Additional Data (AEAD) algorithm for protecting message data between a client and a server. In this section, we define the security context and how it is derived in client and server based on a common shared master 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 OSCORE. 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 and Servers 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.

```
                  .-------------.           .-------------.
                  |  Common,    |           |  Common,    |
                  |  Sender,    |           |  Recipient, |
                  |  Recipient  |           |  Sender     |
                  '-------------'           '-------------'
                       Client                   Server
                          |                        |
Retrieve context for      | OSCORE request:        |
 target resource          |   Token = Token1,      |
Protect request with      |   kid = SID, ...       |
  Sender Context          +----------------------->| Retrieve context with
                          |                        |  RID = kid
                          |                        | Verify request with
                          |                        |  Recipient Context
                          | OSCORE response:       | Protect response with
                          |   Token = Token1, ...  |  Sender Context
Retrieve context with     |<-----------------------+
 Token = Token1           |                        |
Verify request with       |                        |
 Recipient Context        |                        |
```

Figure 3: Retrieval and use of the Security Context

The Common Context contains the following parameters:

o AEAD Algorithm (alg). The COSE AEAD algorithm to use for encryption. Its value is immutable once the security context is established.

o Key Derivation Function. The HMAC based HKDF [RFC5869] used to derive Sender Key, Recipient Key, and Common IV.

o 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.

o 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.

- Common IV. Byte string derived from Master Secret and Master Salt. Length is determined by the AEAD Algorithm. Its value is immutable once the security context is established.

The Sender Context contains the following parameters:

- Sender ID. Byte string used to identify the Sender Context and to assure unique nonces. Maximum length is determined by the AEAD Algorithm. 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 the AEAD Algorithm. Its value is immutable once the security context is established.

- Sender Sequence Number. Non-negative integer used by the sender to protect requests and Observe notifications. Used as "Partial IV" [RFC8152] to generate unique nonces for the AEAD. Maximum value is determined by the AEAD Algorithm.

The Recipient Context contains the following parameters:

- Recipient ID. Byte string used to identify the Recipient Context and to assure unique nonces. Maximum length is determined by the AEAD Algorithm. 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 AEAD Algorithm. Its value is immutable once the security context is established.

- Replay Window (Server only). The replay window to verify requests received.

An endpoint may free up memory by not storing the Common IV, Sender Key, and Recipient Key, deriving them from the Master Key and Master Salt 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.

Endpoints MAY operate in either or both roles as client and server and use the same security context for those roles. Indpendent of being client or server, the endpoint protects messages to send using its Sender Context, and verifies messages received using its Recipient Context. The endpoints MUST NOT change the Sender/

Recipient ID when changing roles. In other words, changing the roles does not change the set of keys to be used.

3.2. Establishment 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:

- o Master Secret
- o Sender ID
- o 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:

- o AEAD Algorithm (alg)
  - * Default is AES-CCM-16-64-128 (COSE algorithm encoding: 10)
- o Master Salt
  - * Default is the empty string
- o Key Derivation Function (KDF)
  - * Default is HKDF SHA-256
- o Replay Window Type and Size
  - * Default is DTLS-type replay protection with a window size of 32 ([RFC6347])

All input parameters need to be known to and agreed on by both endpoints, but the replay window may be different in the two endpoints. The replay window type and size is used by the client in the processing of the Request-Tag [I-D.amsuess-core-repeat-request-tag]. How the input parameters are pre-established, is application specific. The ACE framework may be used to establish the necessary input parameters [I-D.ietf-ace-oauth-authz].

3.2.1. Derivation of Sender Key, Recipient Key, and Common 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, Recipient Key, and Common 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 / nil,
    alg : int / tstr,
    type : tstr,
    L : uint
]
```

where:

- id is the Sender ID or Recipient ID when deriving keys and nil when deriving the Common IV. The encoding is described in Section 5

- 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-16-64-128 (see Section 10.2 in [RFC8152]) is used, the value for L is 16 for keys and 13 for the Common IV.

3.2.2. Initial Sequence Numbers and Replay Window

The Sender 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. The 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 and Master Salt. When a trusted third party assigns identifiers (e.g., using [I-D.ietf-ace-oauth-authz]) or by using a protocol that allows the parties to negotiate locally unique identifiers in each endpoint, the Sender IDs can be very short. The maximum length of Sender ID is length of nonce - 6 bytes. For AES-CCM-16-64-128 the maximum length of Sender ID is 7 bytes. If Sender ID uniqueness cannot be guaranteed by construction, Sender IDs MUST be long uniformly random distributed byte strings such 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 Client MAY provide a Context Hint Section 8.3 to help the Server find the right context.

While the triple (Master Secret, Master Salt, Sender ID) MUST be unique, the same Master Salt MAY be used with several Master Secrets and the same Master Secret MAY be used with several Master Salts.

4. Protected Message Fields

OSCORE transforms a CoAP message (which may have been generated from an HTTP message) into an OSCORE message, and vice versa. OSCORE protects as much of the original message as possible while still allowing certain proxy operations (see Section 10). This section defines how OSCORE protects the message fields and transfers them end-to-end between client and server (in any direction).

The remainder of this section and later sections discuss the behavior in terms of CoAP messages. If HTTP is used for a particular leg in the end-to-end path, then this section applies to the conceptual CoAP message that is mappable to/from the original HTTP message as discussed in Section 10. That is, an HTTP message is conceptually transformed to a CoAP message and then to an OSCORE message, and similarly in the reverse direction. An actual implementation might translate directly from HTTP to OSCORE without the intervening CoAP representation.

Message fields of the CoAP message may be protected end-to-end between CoAP client and CoAP server in different ways:

- Class E: encrypted and integrity protected,

- o Class I: integrity protected only, or
- o Class U: unprotected.

The sending endpoint SHALL transfer Class E message fields in the ciphertext of the COSE object in the OSCORE message. The sending endpoint SHALL include Class I message fields in the Additional Authenticated Data (AAD) of the AEAD algorithm, allowing the receiving endpoint to detect if the value has changed in transfer. Class U message fields SHALL NOT be protected in transfer. Class I and Class U message field values are transferred in the header or options part of the OSCORE message which is visible to proxies.

Message fields not visible to proxies, i.e., transported in the ciphertext of the COSE object, are called "Inner" (Class E). Message fields transferred in the header or options part of the OSCORE message, which is visible to proxies, are called "Outer" (Class I or U).

An OSCORE message may contain both an Inner and an Outer message field of certain CoAP message fields. Inner if the value is intended for the destination endpoint, Outer if the value is intended for a proxy. Inner and Outer message fields are processed independently.

4.1. CoAP Payload

The CoAP Payload, if present in the original CoAP message, SHALL be encrypted and integrity protected and is thus an Inner message field. The sending endpoint writes the payload of the original CoAP message into the plaintext (Section 5.2) input to the COSE object. The receiving endpoint verifies and decrypts the COSE object, and recreates the payload of the original CoAP message.

4.2. CoAP Options

A summary of how options are protected is shown in Figure 4. Options which require special processing, in particular those which may have both Inner and Outer message fields, are labelled with asterisks.

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 (Inner)
I = Integrity Protect only (Outer)
U = Unprotected (Outer)
* = Special

Figure 4: Protection of CoAP Options

Options that are unknown or for which OSCORE processing is not defined SHALL be processed as class E (and no special processing). Specifications of new CoAP options SHOULD define how they are processed with OSCORE. A new COAP option SHOULD be of class E unless it requires proxy processing. New CoAP options which are repeatable and of class I MUST specify that proxies MUST NOT change the order of the option’s occurences.

4.2.1. Inner Options

When using OSCORE, Inner option message fields (marked in column E of Figure 4) are sent in a way analogous to communicating in a protected manner directly with the other endpoint.

The sending endpoint SHALL write the Inner option message fields present in the original CoAP message into the plaintext of the COSE object Section 5.2, and then remove the Inner option message fields from the OSCORE message.

The processing of Inner option message fields by the receiving endpoint is specified in Section 7.2 and Section 7.4.

4.2.2. Outer Options

Outer option message fields (marked in column U or I of Figure 4) are used to support proxy operations.

The sending endpoint SHALL include the Outer option message field present in the original message in the options part of the OSCORE message. All Outer option message fields, including Object-Security, SHALL be encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included Outer option message field.

The processing of Outer options by the receiving endpoint is specified in Section 7.2 and Section 7.4.

A procedure for integrity-protection-only of Class I option message fields is specified in Section 5.3.

Note: There are currently no Class I option message fields defined.

4.2.3. Special Options

Some options require special processing, marked with an asterisk '*' in Figure 4. An asterisk in the columns E and U indicate that the option may be added as an Inner and/or Outer message by the sending endpoint; the processing is specified in this section.

4.2.3.1. Max-Age

The Inner Max-Age option is used to specify the freshness (as defined in [RFC7252]) of the resource, end-to-end from the server to the client, taking into account that the option is not accessible to proxies. The Inner Max-Age SHALL be processed by OSCORE as specified in Section 4.2.1.

The Outer Max-Age option is used to avoid unnecessary caching of OSCORE responses at OSCORE unaware intermediary nodes. A server MAY set a Class U Max-Age option with value zero to Observe responses (see Section 5.6.1 of [RFC7252]) which is then processed according to Section 4.2.2. The Outer Max-Age option value SHALL be discarded by the OSCORE client.

Non-Observe OSCORE responses do not need to include a Max-Age option since the responses are non-cacheable by construction (see Section 4.3).

4.2.3.2. The Block Options

Blockwise [RFC7959] is an optional feature. An implementation MAY support [RFC7252] and the Object-Security option without supporting [RFC7959]. The Block options are used to secure message fragmentation end-to-end (Inner options) or for proxies to fragment the OSCORE message for the next hop (Outer options). Inner and Outer block processing may have different performance properties depending on the underlying transport. The integrity of the message can be verified end-to-end both in case of Inner and Outer Blockwise provided all blocks are received (see Section 4.2.3.2.2).

4.2.3.2.1. Inner Block Options

The sending CoAP endpoint MAY fragment a CoAP message as defined in [RFC7959] before the message is processed by OSCORE. In this case the Block options SHALL be processed by OSCORE as Inner options (Section 4.2.1). The receiving CoAP endpoint SHALL process the OSCORE message according to Section 4.2.1 before processing blockwise as defined in [RFC7959].

For blockwise request operations using Block1, an endpoint MUST comply with the Request-Tag processing defined in Section 3 of [I-D.amsuess-core-repeat-request-tag]. In particular, the rules in section 3.3.1 of [I-D.amsuess-core-repeat-request-tag] MUST be followed, which guarantee that a specific request body is assembled only from the corresponding request blocks.

For blockwise response operations using Block2, an endpoint MUST comply with the ETag processing defined in Section 4 of [I-D.amsuess-core-repeat-request-tag].

4.2.3.2.2. Outer Block Options

Proxies MAY fragment an OSCORE message using [RFC7959], which then introduces Outer Block options not generated by the sending endpoint. Note that the Outer Block options are neither encrypted nor integrity protected. As a consequence, a proxy can maliciously inject block fragments indefinitely, since the receiving endpoint needs to receive the last block (see [RFC7959]) to be able to compose the OSCORE message and verify its integrity. Therefore, applications supporting OSCORE and [RFC7959] MUST specify a security policy defining a maximum unfragmented message size (MAX_UNFRAGMENTED_SIZE) considering the maximum size of message which can be handled by the endpoints.

Messages exceeding this size SHOULD be fragmented by the sending endpoint using Inner Block options (Section 4.2.3.2.1).

An endpoint receiving an OSCORE message with an Outer Block option SHALL first process this option according to [RFC7959], until all blocks of the OSCORE message have been received, or the cumulated message size of the blocks exceeds MAX_UNFRAGMENTED_SIZE. In the former case, the processing of the OSCORE message continues as defined in this document. In the latter case the message SHALL be discarded.

To allow multiple concurrent request operations to the same server (not only same resource), a CoAP proxy SHOULD follow the Request-Tag processing specified in section 3.3.2 of [I-D.amsuess-core-repeat-request-tag].

4.2.3.3. Proxy-Uri

Proxy-Uri, when present, is split by OSCORE into class U options and class E options, which are processed accordingly. When Proxy-Uri is used in the original CoAP message, Uri-* are not present [RFC7252].

The sending endpoint SHALL first decompose the Proxy-Uri value of the original 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 SHALL be protected and processed as Inner options (Section 4.2.1).

The Proxy-Uri option of the OSCORE message SHALL be set to the composition of Proxy-Scheme, Uri-Host and Uri-Port options (if present) as specified in section 6.5 of [RFC7252], and processed as an Outer option of Class U (Section 4.2.2).

Note that replacing the Proxy-Uri value with the Proxy-Scheme and Uri-* options works by design for all CoAP URIs (see Section 6 of [RFC7252]. OSCORE-aware HTTP servers should not use the userinfo component of the HTTP URI (as defined in section 3.2.1. of [RFC3986]), so that this type of replacement is possible in the presence of CoAP-to-HTTP proxies. In other documents specifying cross-protocol proxying behavior using different URI structures, it is expected that the authors will create Uri-* options that allow decomposing the Proxy-Uri, and specify in which OSCORE class they belong.

An example of how Proxy-Uri is processed is given here. Assume that the original CoAP message contains:

- o Proxy-Uri = "coap://example.com/resource?q=1"

During OSCORE processing, Proxy-Uri is split into:

- o Proxy-Scheme = "coap"
- o Uri-Host = "example.com"
- o Uri-Port = "5683"
- o Uri-Path = "resource"
- o Uri-Query = "q=1"

Uri-Path and Uri-Query follow the processing defined in Section 4.2.1, 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 OSCORE message, which has value:

- o Proxy-Uri = "coap://example.com"

See Section 6.1 and 12.6 of [RFC7252] for more information.

4.2.3.4. 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 an OSCORE-unaware proxy to support forwarding of Observe messages ([RFC7641]), there SHALL be an Outer Observe option, i.e., present in the options part of the OSCORE message. The processing of the CoAP Code for Observe messages is described in Section 4.3.

To secure the order of notifications, the client SHALL maintain a Notification Number for each Observation it registers. The Notification Number is a non-negative integer containing the largest Partial IV of the successfully received notifications for the associated Observe registration, see Section 6.4. The Notification Number is initialized to the Partial IV of the first successfully received notification response to the registration request. In contrast to [RFC7641], the received Partial IV MUST always be compared with the Notification Number, which thus MUST NOT be forgotten after 128 seconds.

If the verification fails, the client SHALL stop processing the response, and in the case of CON respond with an empty ACK. The client MAY ignore the Observe option value.

The Observe option in the CoAP request 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 OSCORE client receives a response to an Observe request without an outer Observe value, then it MUST verify the response as a non-Observe response. (The reverse case is covered in the verification of the response, see Section 7.)

4.3. CoAP Header

Most CoAP header fields (i.e. the message fields in the fixed 4-byte header) are required to be read and/or changed by CoAP proxies and thus cannot in general be protected end-to-end between the endpoints. As mentioned in Section 1, OSCORE protects the CoAP Request/Response layer only, and not the Messaging Layer (Section 2 of [RFC7252]), so fields such as Type and Message ID are not protected with OSCORE.

The CoAP header field Code is protected by OSCORE. Code SHALL be encrypted and integrity protected (Class E) to prevent an intermediary from eavesdropping or manipulating the Code (e.g., changing from GET to DELETE).

The sending endpoint SHALL write the Code of the original CoAP message into the plaintext of the COSE object Section 5.2. After that, the Outer Code of the OSCORE message SHALL be set to 0.02 (POST) for requests and to 2.04 (Changed) for responses, except for Observe messages. For Observe messages, the Outer Code of the OSCORE message SHALL be set to 0.05 (FETCH) for requests and to 2.05 (Content) for responses. This exception allows OSCORE to be compliant with the Observe processing in OSCORE-unaware proxies. The choice of POST and FETCH ([RFC8132]) allows all OSCORE messages to have payload.

The receiving endpoint SHALL discard the Code in the OSCORE message and write the Code of the Plaintext in the COSE object (Section 5.2) into the decrypted CoAP message.

The other CoAP header fields are Unprotected (Class U). The sending endpoint SHALL write all other header fields of the original message into the header of the OSCORE message. The receiving endpoint SHALL write the header fields from the received OSCORE message into the header of the decrypted CoAP message.

5. The COSE Object

This section defines how to use COSE [RFC8152] to wrap and protect data in the original message. OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Additional Data (AEAD) algorithm. The key lengths, IV length, nonce length, and maximum Sender Sequence Number are algorithm dependent.

The AEAD algorithm AES-CCM-16-64-128 defined in Section 10.2 of [RFC8152] is mandatory to implement. For AES-CCM-16-64-128 the length of Sender Key and Recipient Key is 128 bits, the length of nonce and Common IV is 13 bytes. The maximum Sender Sequence Number is specified in Section 11.

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 Sender Sequence Number. All leading zeroes SHALL be removed when encoding the Partial IV. The value 0 encodes to the byte string 0x00. This parameter SHALL be present in requests. In case of Observe (Section 4.2.3.4) the Partial IV SHALL be present in responses, and otherwise the Partial IV SHOULD NOT be present in responses. (A non-Observe example where the Partial IV is included in a response is provided in Section 6.5.2.)

  * The "kid" parameter. The value is set to the Sender ID. This parameter SHALL be present in requests and SHOULD NOT be present in responses. (An example where the Sender ID is included in a response is the extension of OSCORE to group communication [I-D.tiloca-core-multicast-oscoap].)

- The "ciphertext" field is computed from the secret key (Sender Key or Recipient Key), Nonce (see Section 5.1), Plaintext (see Section 5.2), and the Additional Authenticated Data (AAD) (see Section 5.3) following Section 5.2 of [RFC8152].

The encryption process is described in Section 5.3 of [RFC8152].

5.1. Nonce

The nonce is constructed in the following way (see Figure 5):

1. left-padding the Partial IV (in network byte order) with zeroes to exactly 5 bytes,

2. left-padding the (Sender) ID of the endpoint that generated the Partial IV (in network byte order) with zeroes to exactly nonce length - 6 bytes,

3. concatenating the size of the ID (S) with the padded ID and the padded Partial IV,

4. and then XORing with the Common IV.

Note that in this specification only algorithms that use nonces equal or greater than 7 bytes are supported.

When observe is not used, the request and the response may use the same nonce. In this way, the Partial IV does not have to be sent in responses, which reduces the size. For processing instructions, see Section 7.

```
+---+-----------------------+--+--+--+--+--+
| S | ID of PIV generator   |  Partial IV  |----+
+---+-----------------------+--+--+--+--+--+    |
                                                |
+------------------------------------------+    |
|                Common IV                 |->(XOR)
+------------------------------------------+    |
                                                |
+------------------------------------------+    |
|                  Nonce                   |<---+
+------------------------------------------+
```

Figure 5: AEAD Nonce Formation

5.2. Plaintext

The Plaintext is formatted as a CoAP message without Header (see Figure 6) consisting of:

- the Code of the original CoAP message as defined in Section 3 of [RFC7252]; and

- all Inner option message fields (see Section 4.2.1) present in the original CoAP message (see Section 4.2). 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

- o the Payload of original CoAP message, if present, and in that case prefixed by the one-byte Payload Marker (0xFF).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |    Class E options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 (only if there
   is payload)
```

Figure 6: Plaintext

5.3. Additional Authenticated Data

The external_aad SHALL be a CBOR array as defined below:

```
external_aad = [
   version : uint,
   alg : int / tstr,
   request_kid : bstr,
   request_piv : bstr,
   options : bstr
]
```

where:

- o version: contains the OSCORE version number. Implementations of this specification MUST set this field to 1. Other values are reserved for future versions.
- o alg: contains the AEAD Algorithm from the security context used for the exchange (see Section 3.1).
- o request_kid: contains the value of the ’kid’ in the COSE object of the request (see Section 5).
- o request_piv: contains the value of the ’Partial IV’ in the COSE object of the request (see Section 5).
- o options: contains the Class I options (see Section 4.2.2) present in the original CoAP message encoded as described in Section 3.1

of [RFC7252], where the delta is the difference to the previously included class I option.

NOTE: The format of the external_aad is for simplicity the same for requests and responses, although some parameters, e.g. request_kid need not be integrity protected in the requests.

6. Sequence Numbers, Replay, Message Binding, and Freshness

6.1. Message Binding

In order to prevent response delay and mismatch attacks [I-D.mattsson-core-coap-actuators] from on-path attackers and compromised proxies, OSCORE binds responses to the requests by including the kid and Partial IV of the request in the AAD of the response. The server therefore needs to store the kid and Partial IV of the request until all responses have been sent.

6.2. 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 Sender Sequence Number used to protect requests, and - in case of Observe - responses. If messages are processed concurrently, the operation of reading and increasing the Sender Sequence Number MUST be atomic.

The maximum Sender Sequence Number is algorithm dependent, see Section 11, and no greater than 2^40 - 1. If the Sender Sequence Number exceeds the maximum, 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.

6.3. Freshness

For requests, OSCORE provides weak absolute freshness as the only guarantee is that the request is not older than the security context. For applications having stronger demands on request freshness (e.g., control of actuators), OSCORE needs to be augmented with mechanisms providing freshness [I-D.amsuess-core-repeat-request-tag].

For responses, the message binding guarantees that a response is not older than its request. For responses without Observe, this gives strong absolute freshness. For responses with Observe, the absolute freshness gets weaker with time, and it is RECOMMENDED that the client regularly restart the observation.

For requests, and responses with Observe, OSCORE also provides relative freshness in the sense that the received Partial IV allows a recipient to determine the relative order of responses.

6.4. Replay Protection

In order to protect from replay of requests, the server's Recipient Context includes a Replay Window. A server SHALL verify that a Partial IV received in the COSE object has not been received before. If this verification fails and the message received is a CON message, the server SHALL respond with a 5.03 Service Unavailable error message with the inner Max-Age option set to 0. The diagnostic payload MAY contain the "Replay protection failed" string. 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 server MAY just store the last received Partial IV and require that newly received Partial IVs equals the last received Partial IV + 1.

Responses to non-Observe requests are protected against replay as they are cryptographically bound to the request.

In the case of Observe, a client receiving a notification SHALL verify that the Partial IV of a received notification is greater than the Notification Number bound to that Observe registration. If the verification fails, the client SHALL stop processing the response, and in the case of CON respond with an empty ACK. If the verification succeeds, the client SHALL overwrite the corresponding Notification Number with the received Partial IV.

If messages are processed concurrently, the Partial IV needs to be validated a second time after decryption and before updating the replay protection data. The operation of validating the Partial IV and updating the replay protection data MUST be atomic.

6.5. Losing Part of the Context State

To prevent reuse of the Nonce with the same key, or from accepting replayed messages, a node needs to handle the situation of losing rapidly changing parts of the context, such as the request Token, Sender Sequence Number, Replay Window, and Nofititfcation Numbers. These are typically stored in RAM and therefore lost in the case of an unplanned 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. However, establishing a fresh security context may have a non-negligible cost in terms of, e.g., power consumption.

After boot, a node MAY use a partly persistently stored security context, but then the node MUST NOT reuse a previous Sender Sequence Number and MUST NOT accept previously accepted messages. Some ways to achieve this is described below:

6.5.1. Sequence Number

To prevent reuse of Sender Sequence Numbers, a node MAY perform the following procedure during normal operations:

- o Each time the Sender Sequence Number is evenly divisible by K, where K is a positive integer, store the Sender Sequence Number in persistent memory. After boot, the node initiates the Sender Sequence Number to the value stored in persistent memory + K - 1. Storing to persistent memory can be costly. The value K gives a trade-off between the number of storage operations and efficient use of Sender Sequence Numbers.

6.5.2. Replay Window

To prevent accepting replay of previously received requests, the server MAY perform the following procedure after boot:

- o For each stored security context, the first time after boot the server receives an OSCORE request, the server responds with the Repeat option [I-D.amsuess-core-repeat-request-tag] to get a request with verifiable freshness. The server MUST use its Partial IV when generating the nonce and MUST include the Partial IV in the response.

If the server using the Repeat option can verify a second request as fresh, then the Partial IV of the second request is set as the lower limit of the replay window.

6.5.3. Replay Protection of Observe Notifications

To prevent accepting replay of previously received notification responses, the client MAY perform the following procedure after boot:

- o The client rejects notifications bound to the earlier registration, removes all Notification Numbers and re-register using Observe.

7. Processing

This section describes the OSCORE message processing.

7.1. Protecting the Request

Given a CoAP request, the client SHALL perform the following steps to create an OSCORE request:

1. Retrieve the Sender Context associated with the target resource.

2. Compose the Additional Authenticated Data, as described in Section 5.

3. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV (Sender Sequence Number in network byte order) as described in Section 5.1. Then (in one atomic operation, see Section 6.2) increment the Sender Sequence Number by one.

4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 8.

5. Format the OSCORE message according to Section 4. The Object-Security option is added, see Section 4.2.2.

6. Store the association Token - Security Context. The client SHALL be able to find the Recipient Context from the Token in the response.

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 4.2.3.2.

2. Discard the message Code and all non-special Inner option message fields (marked with 'x' in column E of Figure 4) present in the received message. For example, an If-Match Outer option is discarded, but an Uri-Host Outer option is not discarded.

3. 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 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. 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".

4. Verify the 'Partial IV' parameter using the Replay Window, as described in Section 6.

5. Compose the Additional Authenticated Data, as described in Section 5.

6. Compute the AEAD nonce from the Recipient ID, Common IV, and the 'Partial IV' parameter, received in the COSE Object.

7. 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 Replay Window, as described in Section 6.

8. For each decrypted option, check if the option is also present as an Outer option: if it is, discard the Outer. For example: the message contains a Max-Age Inner and a Max-Age Outer option. The Outer Max-Age is discarded.

9. Add decrypted code, options and payload to the decrypted request. The Object-Security option is removed.

10. The decrypted CoAP request is processed according to [RFC7252]

7.3. Protecting the Response

Given a CoAP response, the server SHALL perform the following steps to create an OSCORE response. Note that CoAP error responses derived from CoAP processing (point 10. in Section 7.2) are protected, as well as successful CoAP responses, while the OSCORE errors (point 3., 4., 7. in Section 7.2) do not follow the processing below, but are sent as simple CoAP responses, without OSCORE processing.

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. Compute the AEAD nonce

   * If Observe is used, Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV (Sender Sequence Number in network byte order). Then (in one atomic operation, see Section 6.2) increment the Sender Sequence Number by one.

   * If Observe is not used, either the nonce from the request is used or a new Partial IV is used.

4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 8. If in 3. the nonce was constructed from a new Partial IV, this Partial IV MUST be included in the message. If the nonce from the request was used, the Partial IV MUST NOT be included in the message.

5. Format the OSCORE message according to Section 4. The Object-Security option is added, see Section 4.2.2.

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 OSCORE message have been received, see Section 4.2.3.2.

2. Discard the message Code and all non-special Class E options from the message. For example, ETag Outer option is discarded, Max-Age Outer option is not discarded.

3. Retrieve the Recipient Context associated with the Token. Decompress the COSE Object (Section 8). If either the decompression or the COSE message fails to decode, then go to 11.

4. For Observe notifications, verify the received 'Partial IV' parameter against the corresponding Notification Number as described in Section 6. If the client receives a notification for which no Observe request was sent, then go to 11.

5. Compose the Additional Authenticated Data, as described in Section 5.

6. Compute the AEAD nonce

    1. If the Observe option and the Partial IV are not present in the response, the nonce from the request is used.

    2. If the Observe option is present in the response, and the Partial IV is not present in the response, then go to 11.

    3. If the Partial IV is present in the response, compute the AEAD nonce from the Recipient ID, Common IV, and the 'Partial IV' parameter, received in the COSE Object.

7. Decrypt the COSE object using the Recipient Key.

    * If decryption fails, then go to 11.

    * If decryption succeeds and Observe is used, update the corresponding Notification Number, as described in Section 6.

8. For each decrypted option, check if the option is also present as an Outer option: if it is, discard the Outer. For example: the message contains a Max-Age Inner and a Max-Age Outer option. The Outer Max-Age is discarded.

9. Add decrypted code, options and payload to the decrypted request. The Object-Security option is removed.

10. The decrypted CoAP response is processed according to [RFC7252]

11. (Optional) In case any of the previous erroneous conditions apply: if the response is a CON message, then the client SHALL send an empty ACK back and stop processing the response; if the response is a ACK or a NON message, then the client SHALL simply stop processing the response.

8. OSCORE Compression

The Concise Binary Object Representation (CBOR) [RFC7049] combines very small message sizes with extensibility. The CBOR Object Signing and Encryption (COSE) [RFC8152] 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 OSCORE called the "compressed COSE object", which significantly reduces the per-packet overhead.

8.1. Encoding of the Object-Security Value

The value of the Object-Security option SHALL contain the OSCORE flag byte, the Partial IV parameter, the Context Hint parameter (length and value), and the kid parameter as follows:

```
 0 1 2 3 4 5 6 7 <--------- n bytes ------------->
+-+-+-+-+-+-+-+-+---------------------------------
|0 0 0|h|k|  n  |       Partial IV (if any)
+-+-+-+-+-+-+-+-+---------------------------------

<-- 1 byte --> <------ s bytes ------>
+------------+----------------------+-----------------+
| s (if any) | Context Hint (if any) | kid (if any) ... |
+------------+----------------------+-----------------+
```

Figure 7: Object-Security Value

- o The first byte (= the OSCORE flag byte) encodes a set of flags and the length of the Partial IV parameter.

  - * The three least significant bits encode the Partial IV length n. If n = 0 then the Partial IV is not present in the compressed COSE object. The values n = 6 and n = 7 is reserved.

  - * The fourth least significant bit is the kid flag, k: it is set to 1 if the kid is present in the compressed COSE object.

  - * The fifth least significant bit is the Context Hint flag, h: it is set to 1 if the compressed COSE object contains a Context Hint, see Section 8.3.

  - * The sixth-eighth least significant bits are reserved and SHALL be set to zero when not in use.

- o The following n bytes encode the value of the Partial IV, if the Partial IV is present (n > 0).

- o The following 1 byte encode the length of the Context Hint (Section 8.3) s, if the Context Hint flag is set (h = 1).

- o The following s bytes encode the Context Hint, if the Context Hint flag is set (h = 1).

- o The remaining bytes encode the value of the kid, if the kid is present (k = 1)

Note that the kid MUST be the last field of the object-security value, even in case reserved bits are used and additional fields are added to it.

8.2. Encoding of the OSCORE Payload

The payload of the OSCORE message SHALL encode the ciphertext of the COSE object.

8.3. Context Hint

For certain use cases, e.g. deployments where the same Recipient ID is used with multiple contexts, it is necessary or favorable for the client to provide a Context Hint in order for the server to retrieve the Recipient Context. The Context Hint is implicitly integrity protected, as manipulation leads to the wrong or no context being retrieved resulting in a verification error, as described in Section 7.2. This parameter MAY be present in requests and SHALL NOT be present in responses.

Examples:

- o If the client has an identifier in some other namespace which can be used by the server to retrieve or establish the security context, then that identifier can be used as Context Hint.

- o In case of a group communication scenario [I-D.tiloca-core-multicast-oscoap], if the server belongs to multiple groups, then a group identifier can be used as Context Hint to enable the server to find the right security context.

8.4. Examples of Compressed COSE Objects

8.4.1. Example: Requests

Request with kid = 25 and Partial IV = 5

Before compression (24 bytes):

```
[
h'',
{ 4:h'25', 6:h'05' },
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (17 bytes):

Flag byte: 0b00001001 = 0x09

Option Value: 09 05 25 (3 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

Request with kid = empty string and Partial IV = 0

After compression (16 bytes):

Flag byte: 0b00001001 = 0x09

Option Value: 09 00 (2 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

Request with kid = empty string, Partial IV = 5, and Context Hint = 0x44616c656b

After compression (22 bytes):

Flag byte: 0b00011001 = 0x19

Option Value: 19 05 01 44 61 6c 65 6b (8 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

8.4.2. Example: Response (without Observe)

Before compression (18 bytes):

```
[
h'',
{},
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (14 bytes):

Flag byte: 0b00000000 = 0x00

Option Value: (0 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

8.4.3. Example: Response (with Observe)

Before compression (21 bytes):

```
[
h'',
{ 6:h'07' },
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (16 bytes):

Flag byte: 0b00000001 = 0x01

Option Value: 01 07 (2 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

9. Web Linking

The use of OSCORE MAY be indicated by a target attribute "osc" in a web link [RFC8288] to a resource. This attribute is a hint indicating that the destination of that link is to be accessed using OSCORE. Note that this is simply a hint, it does not include any security context material or any other information required to run OSCORE.

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. Proxy Operations

RFC 7252 defines operations for a CoAP-to-CoAP proxy (see Section 5.7 of [RFC7252]) and for proxying between CoAP and HTTP (Section 10 of [RFC7252]). A more detailed description of the HTTP-to-CoAP mapping is provided by [RFC8075]. This section describes the operations of OSCORE-aware proxies.

10.1. CoAP-to-CoAP Forwarding Proxy

OSCORE is designed to work with legacy CoAP-to-CoAP forward proxies [RFC7252], but OSCORE-aware proxies provide certain simplifications as specified in this section.

The targeted proxy operations are specified in Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs]. In particular caching is disabled since the CoAP response is only applicable to the original client's CoAP request. An OSCORE-aware proxy SHALL NOT cache a response to a request with an Object-Security option. As a consequence, the search for cache hits and CoAP freshness/Max-Age processing can be omitted.

Proxy processing of the (Outer) Proxy-Uri option is as defined in [RFC7252].

Proxy processing of the (Outer) Block options is as defined in [RFC7959] and [I-D.amsuess-core-repeat-request-tag].

Proxy processing of the (Outer) Observe option is as defined in [RFC7641]. OSCORE-aware proxies MAY look at the Partial IV value instead of the Outer Observe option.

10.2. HTTP-to-CoAP Translation Proxy

Section 10.2 of [RFC7252] and [RFC8075] specify the behavior of an HTTP-to-CoAP proxy. As requested in Section 1 of [RFC8075], this section describes the HTTP mapping for the OSCORE protocol extension of CoAP.

The presence of the Object-Security option, both in requests and responses, is expressed in an HTTP header field named Object-Security in the mapped request or response. The value of the field is the value of the Object-Security option Section 8.1 in base64url encoding (Section 5 of [RFC4648]) without padding (see [RFC7515] Appendix C for implementation notes for this encoding). The value of the payload is the OSCORE payload Section 8.2, also base64url-encoded without padding.

Example:

Mapping and notation here is based on "Simple Form" (Section 5.4.1.1 of [RFC8075]).

```
[HTTP request -- Before object security processing]

  GET http://proxy.url/hc/?target_uri=coap://server.url/orders HTTP/1.1

[HTTP request -- HTTP Client to Proxy]

  POST http://proxy.url/hc/?target_uri=coap://server.url/ HTTP/1.1
  Object-Security: 0b 25
  Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

[CoAP request -- Proxy to CoAP Server]

  POST coap://server.url/
  Object-Security: 0b 25
  Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

[CoAP response -- CoAP Server to Proxy]

  2.04 Changed
  Object-Security: [empty]
  Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[HTTP response -- Proxy to HTTP Client]

  HTTP/1.1 200 OK
  Object-Security: [empty]
  Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[HTTP response -- After object security processing]

  HTTP/1.1 200 OK
  Body: Exterminate! Exterminate!
```

Note that the HTTP Status Code 200 in the next-to-last message is the mapping of CoAP Code 2.04 (Changed), whereas the HTTP Status Code 200 in the last message is the mapping of the CoAP Code 2.05 (Content), which was encrypted within the compressed COSE object carried in the Body of the HTTP response.

10.3. CoAP-to-HTTP Translation Proxy

Section 10.1 of [RFC7252] describes the behavior of a CoAP-to-HTTP proxy. RFC 8075 [RFC8075] does not cover this direction in any more detail and so an example instantiation of Section 10.1 of [RFC7252] is used below.

Example:

```
[CoAP request -- Before object security processing]

  GET coap://proxy.url/
  Proxy-Uri=http://server.url/orders

[CoAP request -- CoAP Client to Proxy]

  POST coap://proxy.url/
  Proxy-Uri=http://server.url/
  Object-Security: 0b 25
  Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

[HTTP request -- Proxy to HTTP Server]

  POST http://server.url/ HTTP/1.1
  Object-Security: 0b 25
  Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

[HTTP response -- HTTP Server to Proxy]

  HTTP/1.1 200 OK
  Object-Security: [empty]
  Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response -- CoAP Server to Proxy]

  2.04 Changed
  Object-Security: [empty]
  Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response -- After object security processing]

  2.05 Content
  Payload: Exterminate! Exterminate!
```

Note that the HTTP Code 2.04 (Changed) in the next-to-last message is the mapping of HTTP Status Code 200, whereas the CoAP Code 2.05 (Content) in the last message is the value that was encrypted within the compressed COSE object carried in the Body of the HTTP response.

11. Security Considerations

In scenarios with intermediary nodes such as proxies or brokers, transport layer security such as (D)TLS 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.

(D)TLS protects hop-by-hop the entire message, including header, options, and payload. OSCORE protects end-to-end the payload, and all information in the options and header, that is not required for proxy operations (see Section 4). (D)TLS and OSCORE can be combined, thereby enabling end-to-end security of the message payload, in combination with hop-by-hop protection of the entire message, during transport between end-point and intermediary node. The message layer, however, cannot be protected end-to-end through intermediary devices since, even if the protocol itself isn't translated, the parameters Type, Message ID, Token, and Token Length may be changed by a proxy.

The use of COSE to protect 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 the ACE framework [I-D.ietf-ace-oauth-authz]. An OSCORE profile of ACE is described in [I-D.seitz-ace-oscoap-profile].

Most AEAD algorithms require a unique nonce for each message, for which the sender 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 sender sequence number is dependent on the AEAD algorithm. The maximum sender sequence number SHALL be 2^40 - 1, or any algorithm specific lower limit, after which a new security context must be generated. The mechanism to build the nonce (Section 5.1) assumes that the nonce is at least 56 bit-long, and the Partial IV is at most 40 bit-long. The mandatory-to-implement AEAD algorithm AES-CCM-16-64-128 is selected for compatibility with CCM*.

The inner block options enable the sender to split large messages into OSCORE-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.

12. Privacy Considerations

Privacy threats executed through intermediate nodes are considerably reduced by means of OSCORE. End-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations, 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.

Using the mechanisms described in Section 6.5 may reveal when a device goes through a reboot. This can be mitigated by the device storing the precise state of sender sequence number and replay window on a clean shutdown.

The length of message fields can reveal information about the message. Applications may use a padding scheme to protect against traffic analysis. 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. Similar arguments apply to other message fields such as resource names.

13. IANA Considerations

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

13.1. CoAP Option Numbers Registry

The Object-Security option is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD	Object-Security	[[this document]]

13.2. Header Field Registrations

The HTTP header field Object-Security is added to the Message Headers registry:

Header Field Name	Protocol	Status	Reference
Object-Security	http	standard	[[this document]]

14. Acknowledgments

The following individuals provided input to this document: Christian Amsuess, Tobias Andersson, Carsten Bormann, Joakim Brorsson, Thomas Fossati, Martin Gunnarsson, Klaus Hartke, Jim Schaad, Dave Thaler, 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.

15. References

15.1. Normative References

[I-D.amsuess-core-repeat-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Repeat And Request-Tag", draft-amsuess-core-repeat-request-tag-00 (work in progress), July 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://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, <https://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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc8075>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

15.2. 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, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-11 (work in progress), July 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-03 (work in progress), July 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-07 (work in progress), August 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-09 (work in progress), May 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., Palombini, F., and M. Gunnarsson, "OSCOAP profile of the Authentication and Authorization for Constrained Environments Framework", draft-seitz-ace-oscoap-profile-05 (work in progress), October 2017.

[I-D.tiloca-core-multicast-oscoap]
Tiloca, M., Selander, G., and F. Palombini, "Secure group communication for CoAP", draft-tiloca-core-multicast-oscoap-03 (work in progress), July 2017.

[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, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

Appendix A. Test Vectors

TODO: This section needs to be updated.

Appendix B. Examples

This section gives examples of OSCORE. 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 (compressed) 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.

```
   Client  Proxy  Server
     |       |       |
     +------>|       |            Code: 0.02 (POST)
     | POST  |       |           Token: 0x8c
     |       |       | Object-Security: [kid:5f,Partial IV:42]
     |       |       |         Payload: {Code:0.01,
     |       |       |                   Uri-Path:"alarm_status"}
     |       |       |
     |       +------>|            Code: 0.02 (POST)
     |       | POST  |           Token: 0x7b
     |       |       | Object-Security: [kid:5f,Partial IV:42]
     |       |       |         Payload: {Code:0.01,
     |       |       |                   Uri-Path:"alarm_status"}
     |       |       |
     |       |<------+            Code: 2.04 (Changed)
     |       |  2.04 |           Token: 0x7b
     |       |       | Object-Security: -
     |       |       |         Payload: {Code:2.05, "OFF"}
     |       |       |
     |<------+       |            Code: 2.04 (Changed)
     |  2.04 |       |           Token: 0x8c
     |       |       | Object-Security: -
     |       |       |         Payload: {Code:2.05, "OFF"}
     |       |       |
```

Figure 8: Secure Access to Sensor. Square brackets [...] indicate content of compressed COSE object. Curly brackets { ... } indicate encrypted data.

The request/response Codes are encrypted by OSCORE and only dummy Codes (POST/Changed) are visible in the header of the OSCORE message. The option Uri-Path ("alarm_status") and payload ("OFF") are encrypted.

The COSE header of the request contains an identifier (5f), indicating which security context was used to protect the message and a Partial IV (42).

The server verifies that the Partial IV 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.

```
   Client  Proxy  Server
     |       |       |
     +------>|       |            Code: 0.05 (FETCH)
     | FETCH |       |           Token: 0x83
     |       |       |         Observe: 0
     |       |       | Object-Security: [kid:ca,Partial IV:15]
     |       |       |         Payload: {Code:0.01,
     |       |       |                   Uri-Path:"glucose"}
     |       |       |
     |       +------>|            Code: 0.05 (FETCH)
     |       | FETCH |           Token: 0xbe
     |       |       |         Observe: 0
     |       |       | Object-Security: [kid:ca,Partial IV:15]
     |       |       |         Payload: {Code:0.01,
     |       |       |                   Uri-Path:"glucose"}
     |       |       |
     |       |<------+            Code: 2.05 (Content)
     |       |  2.05 |           Token: 0xbe
     |       |       |         Observe: 7
     |       |       | Object-Security: [Partial IV:32]
     |       |       |         Payload: {Code:2.05,
     |       |       |                   Content-Format:0, "220"}
     |       |       |
     |<------+       |            Code: 2.05 (Content)
     |  2.05 |       |           Token: 0x83
     |       |       |         Observe: 7
     |       |       | Object-Security: [Partial IV:32]
     |       |       |         Payload: {Code:2.05,
     |       |       |                   Content-Format:0, "220"}
    ...     ...     ...
     |       |       |
     |       |<------+            Code: 2.05 (Content)
     |       |  2.05 |           Token: 0xbe
     |       |       |         Observe: 8
     |       |       | Object-Security: [Partial IV:36]
     |       |       |         Payload: {Code:2.05,
     |       |       |                   Content-Format:0, "180"}
     |       |       |
     |<------+       |            Code: 2.05 (Content)
     |  2.05 |       |           Token: 0x83
```

```
        |       |       |         Observe: 8
        |       |       | Object-Security: [Partial IV:36]
        |       |       |         Payload: {Code:2.05,
        |       |       |                   Content-Format:0, "180"}
        |       |       |
```

Figure 9: Secure Subscribe to Sensor. Square brackets [...] indicate content of compressed COSE header. Curly brackets { ... } indicate encrypted data.

The request/response Codes are encrypted by OSCORE and only dummy Codes (FETCH/Content) are visible in the header of the OSCORE message. The options Content-Format (0) and the payload ("220" and "180"), are encrypted.

The COSE header of the request contains an identifier (ca), indicating the security context used to protect the message and a Partial IV (15). The COSE headers of the responses contains Partial IVs (32 and 36).

The server verifies that the Partial IV has not been received before. The client verifies that the responses are bound to the request and that the Partial IVs are greater than any Partial IV previously received in a response bound to the request.

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   K. Lynn
Internet-Draft   Verizon Labs
Intended status: Standards Track   P. van der Stok
Expires: January 4, 2018   consultant
                                                                M. Koster
                                                              SmartThings
                                                        C. Amsuess, Ed.
                                             Energy Harvesting Solutions
                                                            July 03, 2017

CoRE Resource Directory: DNS-SD mapping
draft-ietf-core-rd-dns-sd-00

Abstract

TBD

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

TBD ... [RFC7252] ... [I-D.ietf-core-resource-directory] ... DNS-SD

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [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 terminology of [I-D.ietf-core-resource-directory].

This specification makes use of the following additional terminology:

TBD: TBD

TBD: TBD

2. New Link-Format Attributes

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

```
link-extension    = ( "ins" "=" (ptoken | quoted-string) )
                      ; The token or string is max 63 bytes
link-extension    = ( "exp" )
```

2.1. Resource Instance attribute ’ins’

The Resource Instance "ins" attribute is an identifier for this resource, which makes it possible to distinguish it from other similar resources. This attribute is similar in use to the <Instance> portion of a DNS-SD record (see Section 3.1, and SHOULD be unique across resources with the same Resource Type attribute in the domain it is used. A Resource Instance might be a descriptive string like "Ceiling Light, Room 3", a short ID like "AF39" or a unique UUID or iNumber. This attribute is used by a Resource Directory to distinguish between multiple instances of the same resource type within the directory.

This attribute MUST be no more than 63 bytes in length. The resource identifier attribute MUST NOT appear more than once in a link description. This attribute MAY be used as a query parameter in the RD Lookup Function Set defined in Section 7.

2.2. Export attribute ’exp’

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

The CoRE Link Format is used for many purposes between CoAP endpoints. Some are useful mainly locally, for example checking the observability of a resource before accessing it, determining the size of a resource, or traversing dynamic resource structures. However, other links are very useful to be exported to other directories, for example the entry point resource to a functional service. This attribute MAY be used as a query parameter in the RD Lookup Function Set defined in Section 7.

3. DNS-SD Mapping

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

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

3.1. DNS-based Service discovery

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

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

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

The service name is the label of SRV/TXT resource records. The SRV RR specifies the host and the port of the endpoint. The TXT RR provides additional information in the form of key/value pairs.

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

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

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

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

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

3.2. mapping ins to <Instance>

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

The <Instance> part of the name of a service being offered on the network SHOULD be configurable by the user setting up the service, so that he or she may give it an informative name. However, the device or service SHOULD NOT require the user to configure a name before it can be used. A sensible choice of default name can allow the device or service to be accessed in many cases without any manual configuration at all. The default name should be short and descriptive, and MAY include a collision-resistant substring such as the lower bits of the device’s MAC address, serial number, fingerprint, or other identifier in an attempt to make the name relatively unique.

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

3.3. Mapping rt to <ServiceType>

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

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

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

3.4. Domain mapping

DNS domains may be derived from the "d" attribute. The domain attribute may be suffixed with the zone name of the authoritative DNS server to generate the domain name. The "ep" attribute is prefixed to the domain name to generate the FQDN to be stored into DNS with an AAAA RR.

3.5. TXT Record key=value strings

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

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

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

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

3.6. Importing resource links into DNS-SD

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

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

Res: 2.05 Content
<coap://[FDFD::1234]:5683/light/1>;
  exp;rt="dali.light";ins="Spot";
            d="office";ep="node1"
```

The agent subsequently registers the following DNS-SD RRs, assuming a zone name "example.com" prefixed with "office":

```
node1.office.example.com.          IN AAAA        FDFD::1234
_dali._udp.office.example.com      IN PTR
                          Spot._dali._udp.office.example.com
light._sub._dali._udp.example.com  IN PTR
                          Spot._dali._udp.office.example.com
Spot._dali._udp.office.example.com IN SRV  0 0 5683
                          node1.office.example.com.
Spot._dali._udp.office.example.com IN TXT
                          txtver=1;path=/light/1
```

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

4. Examples

4.1. DNS entries

It may be profitable to discover the light groups for applications, which are unaware ot the existence of the RD. An agent needs to query the RD to return all groups which are exported to be inserted into DNS.

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

   Res: 2.05 Content
   <coap://[FF05::1]/>;exp;gp="grp_R2-4-015;ins="grp1234";
ep="lm_R2-4-015_wndw";
ep="lm_R2-4-015_door
```

The group with FQDN grp_R2-4-015.bc.example.com can be entered into the DNS by the agent. The accompanying instance name is grp1234. The <ServiceType> is chosen to be _group._udp. The agent enters the following RRs into the DNS.

```
grp_R2-4-015.bc.example.com.        IN AAAA            FF05::1
_group._udp.bc.example.com          IN PTR
                            grp1234._group._udp.bc.example.com
grp1234._group._udp.bc.example.com  IN SRV  0 0 5683
                             grp_R2-4-015_door.bc.example.com.
grp1234._group._udp.bc.example.com  IN TXT
                                     txtver=1;path=/light/grp1
```

From then on applications, not familiar with the existence of the RD, can use DNS to access the lighting group.

5. IANA considerations

TBD

6. Security considerations

TBD

7. References

7.1. 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.

[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>.

[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>.

[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>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <http://www.rfc-editor.org/info/rfc6763>.

7.2. Informative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <http://www.rfc-editor.org/info/rfc1034>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <http://www.rfc-editor.org/info/rfc1123>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <http://www.rfc-editor.org/info/rfc5198>.

[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>.

Acknowledgements

This document was split off from [I-D.ietf-core-resource-directory]. TODO: Copy over relevant acknowledgements.

Authors' Addresses

Kerry Lynn
Verizon Labs
50 Sylvan Rd
Waltham, MA 02451
USA

Phone: +1 781 296 9722
Email: kerlyn@ieee.org

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Phone: +1-707-502-5136
Email: Michael.Koster@smartthings.com

Christian Amsuess (editor)
Energy Harvesting Solutions
Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639
Email: c.amsuess@energyharvesting.at

CoRE
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
C. Amsuess, Ed.
Energy Harvesting Solutions

October 30, 2017

CoRE Resource Directory
draft-ietf-core-resource-directory-12

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

Copyright Notice

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 querying "/.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.

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 have unique names.

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 has a unique name within the associated domain of the registration.

Context
A Context is a base URL that gives scheme and (typically) authority information about an Endpoint. The Context of an Endpoint is provided at registration time, and is used by the Resource Directory to resolve relative references inside the registration into absolute URIs.

Directory Resource
A resource in the Resource Directory (RD) containing registration resources.

Group Resource
A resource in the RD containing registration resources of the Endpoints that form a group.

Registration Resource
A resource in the RD that contains information about an Endpoint and its links.

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

3.1. Principles

The Resource Directory is primarily a tool to make discovery operations more efficient than querying /.well-known/core on all connected device, or across boundaries that would be limiting those operations.

It provides a cache (in the high-level sense, not as defined in [RFC7252]/[RFC2616]) of data that could otherwise only be obtained by directly querying the /.well-known/core resource on the target device, or by accessing those resources with a multicast request.

From that, it follows that no information should be stored in the resource directory that cannot be discovered from querying the described device's /.well-known/core resource directly.

It also follows that data in the resource directory can only be provided by the device whose descriptions are cached or a dedicated Commissioning Tool (CT). These CTs are thought to act on behalf agents too constrained, or generally unable, to present that information themselves. No other client can modify data in the resource directory or even expect those changes to propagate back to its source.

3.2. Architecture

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, 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 registration 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.

```
                Registration     Lookup, Group
                 Interface        Interfaces
     +----+          |                 |
     | EP |----      |                 |
     +----+    ----  |                 |
                   --|-    +------+    |
     +----+          | ----|      |    |     +--------+
     | EP | ---------|-----|  RD  |----|-----| Client |
     +----+          | ----|      |    |     +--------+
                   --|-    +------+    |
     +----+    ----  |                 |
     | EP |----      |                 |
     +----+
```

Figure 1: The resource directory architecture.

```
                  +------------+
                  |   Domain   | <-- Name
                  +------------+
                       |     |
                       |   +------------+
                       |   |   Group    | <-- Name, Scheme, IP, Port
                       |   +------------+
                       |     |
                  +------------+
                  |  Endpoint  |  <-- Name, Scheme, IP, Port
                  +------------+
                        |
                        |
                  +------------+
                  |  Resource  |  <-- Target, Parameters
                  +------------+
```

Figure 2: The resource directory information hierarchy.

3.3. Content model

The Entity-Relationship (ER) models shown in Figure 3 and Figure 4 model the contents of /.well-known/core and the resource directory respectively, with entity-relationship diagrams [ER]. Entities (rectangles) are used for concepts that exist independently. Attributes (ovals) are used for concepts that exist only in connection with a related entity. Relations (diamonds) give a semantic meaning to the relation between entities. Numbers specify the cardinality of the relations.

Some of the attribute values are URIs. Those values are always full URIs and never relative references in the information model. They can, however, be expressed as relative references in serializations, and often are.

These models provide an abstract view of the information expressed in link-format documents and a Resource Directory. They cover the concepts, but not necessarily all details of an RD's operation; they are meant to give an overview, and not be a template for implementations.

```
                    +----------------------+
                    |   /.well-known/core  |
                    +----------------------+
                               |
                               | 1
                       ////////\\\\\\\
                      <    contains   >
                       \\\\\\\\///////
                               |
                               | 0+
                     +--------------------+
                     |      link          |
                     +--------------------+
                               |
                               |  1   oooooooo
                               +-----o target o
                            0+ |      oooooooo
          oooooooooooo         |
         o    target  o--------+
         o  attribute o        | 0+   oooooo
          oooooooooooo         +-----o rel  o
                               |      oooooo
                               |
                               | 1    ooooooooo
                               +-----o context o
                                      ooooooooo
```

Figure 3: E-R Model of the content of /.well-known/core

The model shown in Figure 3 models the contents of /.well-known/core which contains:

o a set of links belonging to the host

The host is free to choose links it deems appropriate to be exposed in its ".well-known/core". Typically, the links describe resources that are served by the host, but the set can also contain links to resources on other servers (see examples in [RFC6690] page 14). The set does not necessarily contain links to all resources served by the host.

A link has the following attributes:

o Zero or more link relations: They describe a relations between the link context and the link target.

  In link-format serialization, they are expressed as space-separated values in the "rel" attribute, and default to "hosts".

o A link context URI: It defines the source of the relation, eg. _who_ "hosts" something.

  In link-format serialization, it is expressed in the "anchor" attribute. There, it can be a relative reference, in which case it gets resolved against the URI of the ".well-known/core" document it was obtained from . It defaults to that document's URI.

  In the serialization, the context also serves as the Base URI for resolving the target reference.

o A link target URI: It defines the destination of the relation (eg. _what_ is hosted), and is the topic of all target attributes.

In link-format serialization, it is expressed between angular brackets, and sometimes called the "href". If it is a relative URI, it gets resolved against the link context URI.

o Other target attributes (eg. resource type (rt), interface (if), cor content-type (ct)). These provide additional information about the target URI.

```
                +----------------------+
                |  resource-directory  |
                +----------------------+
                           |
                           |         oooooooooooo  0-1
                           |        o MC address o---+
                           |         oooooooooooo    |
                           |                         |
                      //////\\\\             0+  +--------+
                     < contains >----------------| group  |
                      \\\\\/////                 +--------+
                           |                         |
                       0-n |                         | 1+
    ooooooo     1  +---------------+          ///////\\\\\\
   o  con  o-------|  registration |---------< composed of >
    ooooooo        +---------------+          \\\\\\///////
                       |       |
                       |       +--------------+
          oooooooo   1 |                      |
         o  loc   o----+                 /////\\\\
          oooooooo     |                < contains >
                       |                 \\\\\/////
          oooooooo   1 |                      |
         o   ep   o----+                      | 0+
          oooooooo     |             +------------------+
                       |             |      link        |
          oooooooo 0-1 |             +------------------+
         o    d   o----+                      |
          oooooooo     |                      |  1   oooooooo
                       |                      +-----o target o
          oooooooo 0-1 |                      |      oooooooo
         o   lt   o----+     ooooooooooo   0+ |
          oooooooo     |    o  target   o-----+
                       |    o attribute o     | 0+   oooooo
       ooooooooooo 0+  |     ooooooooooo      +-----o rel  o
      o  endpoint o----+                      |      oooooo
      o attribute o                           |
       ooooooooooo                            | 1   ooooooooo
                                              +----o context o
                                                    ooooooooo
```

Figure 4: E-R Model of the content of the Resource Directory

The model shown in Figure 4 models the contents of the resource directory which contains in addition to /.well-known/core:

o 0 to n Registration (entries),

o 0 or more Groups

A Group has no or one Multicast address attribute and is composed of 0 or more endpoints. A registration is associated with one endpoint (ep). An endpoint can be part of 0 or more Groups . A registration defines a set of links as defined for /.well-known/core. A Registration has six attributes:

o one ep (endpoint with a unique name)

o one con (a string describing the scheme://authority part)

o one lt (lifetime),

o one loc (location in the RD)

o optional one d (domain for query filtering),

o optional additional endpoint attributes (from Section 9.3)

The cardinality of con is currently 1. Its value is used as a Base URI when resolving URIs in the links contained in the endpoint.

Links are modelled as they are in Figure 3.

3.4. 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 wireless 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.

Imagine a scenario where endpoints installed on vehicles enable tracking of the position of these vehicles for fleet management purposes and allow monitoring of environment parameters. During the boot-up process endpoints register with a Resource Directory, which is hosted by the mobile operator or somewhere in the cloud.

Periodically, these endpoints update their registration and may modify resources they offer.

When endpoints are not always connected, for example because they enter a sleep mode, a remote server is usually used to provide proxy access to the endpoints. Mobile apps or web 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.

3.5. 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.

3.6. 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

A device coming up may want to find one or more resource directories to make itself known with.

The device may be pre-configured to exercise specific mechanisms for finding the resource directory:

- It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close; 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 appropriate RD. (Instead of using an anycast address, a multicast address can also be preconfigured. The RD directory servers then need to configure one of their interfaces with this multicast address.)

- It may be configured with a DNS name for the RD and a resource-record type to look up under this name; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.

- It may be configured to use a service discovery mechanism such as DNS-SD [RFC6763]. The present specification suggests configuring the service with name rd._sub._coap._udp, preferably within the domain of the querying nodes.

For cases where the device is not specifically configured with a way to find a resource directory, the network may want to provide a suitable default.

- If the address configuration of the network is performed via SLAAC, this is provided by the RDAO option Section 4.1.

- If the address configuration of the network is performed via DHCP, this could be provided via a DHCP option (no such option is defined at the time of writing).

Finally, if neither the device nor the network offer any specific configuration, the device may want to employ heuristics to find a suitable resource directory.

The present specification does not fully define these heuristics, but suggests a number of candidates:

- In a 6LoWPAN, just assume the Edge Router (6LBR) can act as a resource directory (using the ABRO option to find that [RFC6775]). Confirmation can be obtained by sending a Unicast to "coap://[6LBR]/.well-known/core?rt=core.rd*".

- In a network that supports multicast well, discovering the RD using a multicast query for /.well-known/core as specified in CoRE Link Format [RFC6690]: Sending a Multicast GET to "coap://[ff02::1]/.well-known/core?rt=core.rd*". RDs within the multicast scope will answer the query.

As some of the RD addresses obtained by the methods listed here 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:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      | Length = 3    |        Valid Lifetime         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Reserved                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                          RD Address                           +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 5: 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.

All operations on the contents of the Resource Directory MUST be atomic and idempotent.

A resource directory MAY make the information submitted to it 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.

5.1. 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. URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. This section defines discovery of the RD and its URIs using the well-known interface of the CoRE Link Format [RFC6690]. A complete set of RD discovery methods is described in Section 4.

Discovery of the RD registration URI path 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 URIs for RD Lookup operations, and "core.rd-group" is used to discover the URI path for RD Group operations. Upon success, the response will contain a payload with a link format entry for each RD function discovered, indicating the URI path 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-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" or 200 "OK" 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: 4.00 "Bad Request" or 400 "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 directory 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 RD resource paths.

```
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-lookup/gp>;rt="core.rd-lookup-gp";ct=40,
</rd-group>;rt="core.rd-group";ct=40
```

Figure 6: Example discovery exchange

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 the the CBOR and JSON representation of link format. The RD resource paths /rd, /rd-lookup, and /rd-group are example values.

[The RFC editor is asked to replace these and later occurrences of TBD64 and TBD504 with the numeric ID values assigned by IANA to application/link-format+cbor and application/link-format+json, respectively, as they are defined in I-D.ietf-core-links-json.]

```
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 TBD64 TBD504",
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct="40 TBD64 TBD504",
</rd-lookup/gp>;rt="core.rd-lookup-gp";ct=40 TBD64 TBD504",
</rd-group>;rt="core.rd-group";ct="40 TBD64 TBD504"
```

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. Registration resources in the RD are kept active for the period indicated by the lifetime parameter. The endpoint is responsible for refreshing the registration resource 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 registration resources. A new registration resource may be created at any time to supersede an existing registration, replacing the registration parameters and links.

An empty payload is considered a malformed request.

The posted link-format document can (and typically does) contain relative references both in its link targets and in its anchors, or contain empty anchors. The RD server needs to resolve these references in order to faithfully represent them in lookups. The Base URI against which they are resolved is the context of the registration, which is provided either explicitly in the "con" parameter or constructed implicitly from the requester’s network address. When resolving relative target references, the server first resolves the context of that link, and then interprets the target as a reference relative to that context (see Appendix A.4).

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: {+rd}{?ep,d,lt,con,extra-attrs*}

URI Template Variables:

rd := RD registration URI (mandatory). This is the location of the RD, as obtained from discovery.

ep := Endpoint name (mostly mandatory). The endpoint name is an identifier that MUST be unique within a domain. The maximum length of this parameter is 63 bytes. If the RD is configured to recognize the endpoint (eg. based on its security context), the endpoint can elide the endpoint name, and assign one based on the configuration.

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.

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.

con := Context (optional). This parameter sets the Default Base URI under which the request's links are to be interpreted. The URI MUST NOT have a path component of its own, but MUST be suitable as a base URI to resolve any relative references given in the registration. The parameter is therefore of the shape "scheme://authority" for HTTP and CoAP URIs. In the absence of this parameter the scheme of the protocol, source address and source port of the registration request are assumed. This parameter is mandatory when the directory is filled by a third party such as an commissioning tool. 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.

extra-attrs := Additional registration attributes (optional). The endpoint can pass any parameter registered at Section 9.3 to the directory. If the RD is aware of the parameter's specified semantics, it processes it accordingly. Otherwise, it MUST store the unknown key and its value(s) as an endpoint attribute for further lookup.

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. A registration with an already registered ep and d value pair responds with the same success code and Location as the original registration; the set of links registered with the endpoint is replaced with the links from the payload.

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 following example shows an endpoint with the name "node1" registering two resources to an RD using this interface. The location "/rd" is an example RD location discovered in a request similar to Figure 6.

```
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 almost the same registration operation above, when done using HTTP and the JSON Link Format.

```
Req: POST /rd?ep=node1&con=http://[2001:db8:1::1] HTTP/1.1
Host : example.com
Content-Type: application/link-format+json
Payload:
[
{"href": "/sensors/temp", "ct": "41", "rt": "temperature-c", "if": "sensor"},
{"href": "/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 finally asks the directory server to probe it for resources and publish them as follows:

It sends (and regularly refreshes with) a POST request to the "/.well-known/core" URI of the directory server of choice. 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 includes the same registration parameters in the POST request as it would per Section 5.3. The context of the registration is taken from the requesting server's URI.

The endpoints MUST be deleted after the expiration of their lifetime. Additional operations cannot be executed because no registration location is returned.

The following example shows an endpoint using Simple Registration, by simply sending an empty POST to a resource directory.

```
Req:(to RD server from [2001:db8:2::1])
POST /.well-known/core?lt=6000&ep=node1
Content-Format: 40
No payload

Res: 2.04 Changed

(later)

Req: (from RD server to [2001:db8:2::1])
GET /.well-known/core
Accept: 40

Res: 2.05 Content
Payload:
</sen/temp>
```

5.3.2. Third-party registration

For some applications, even Simple Registration 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.

5.4. 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 lifetime and "keep-alive" the registration. When the lifetime of the registration has expired, the RD SHOULD NOT respond to discovery queries concerning this 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, or cancel the registration using DELETE.

These operations are described in this section.

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 initial registration operation.

An update MAY update the lifetime- or the context- registration parameters "lt", "con" as in Section 5.3. Parameters that are not being 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.

A registration update resets the timeout of the registration to the (possibly updated) lifetime of the registration, independent of whether a "lt" parameter was given.

If the context of the registration is changed in an update explicitly or implicitly, relative references submitted in the original registration or later updates are resolved anew against the new context (like in the original registration).

This operation only describes the use of POST with an empty payload. As with modification of individual using iPATCH or PATCH as proposed in Section 5.4.4, future standards might describe the semantics of using content formats and payloads with the POST method to update the links of a registration.

The update registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: {+location}{?lt,con,extra-attrs*}

URI Template Variables:

location := This is the Location 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 updates the context established in the original registration to a new value. If the parameter is set in an update, it is stored by the RD as the new Base URI under which to interpret the links of the registration, following the same restrictions as in the registration. If the parameter is not set and was set explicitly before, the previous context value is kept unmodified. If the parameter is not set and was not set explicitly before either, the source address and source port of the update request are stored as the context.

extra-attrs := Additional registration attributes (optional). As with the registration, the RD processes them if it knows their semantics. Otherwise, unknown attributes are stored as endpoint attributes, overriding any previously stored endpoint attributes of the same key.

Content-Format: none (no payload)

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: 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 resource 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 resource at an RD using this interface with the example location value: /rd/4521. The initial registration by the client set the following values:

- o endpoint name (ep)=endpoint1
- o lifetime (lt)=500
- o context (con)=coap://local-proxy-old.example.com:5683

The initial state of the Resource Directory is reflected in the following request:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content
Payload:
</sensors/temp>;ct=41;rt="temperature";anchor="coap://local-proxy-old.example.co
m:5683",
</sensors/light>;ct=41;rt="light-lux";if="sensor";anchor="coap://local-proxy-old
.example.com:5683"
```

The following example shows an EP changing the context to "coaps://new.example.com:5684":

```
Req: POST /rd/4521?con=coaps://new.example.com:5684

Res: 2.04 Changed
```

The consecutive query returns:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content
Payload:
</sensors/temp>;ct=41;rt="temperature";anchor="coaps://new.example.com:5684",
</sensors/light>;ct=41;rt="light-lux";if="sensor";anchor="coaps://new.example.co
m:5684",
```

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.

Removed endpoints are implicitly removed from the groups to which they belong.

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables:

location := This is the Location 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 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

An iPATCH (or PATCH) update [RFC8132] adds, removes or changes links of a registration by including link update information in the payload of the update with a media type that still needs to be defined.

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 is a list of links to registration resources of the endpoints that belong to that group.

The commissioning tool SHOULD not send any target attributes with the links to the registration resources, and the resource directory SHOULD ignore any attributes that are set.

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 URI (mandatory). This is the location of the RD Group REST API.

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 of the multicast address associated with the group. When con is used, scheme and host are mandatory and port parameter is 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 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. The RD group path /rd-group is an example RD location discovered in a request similar to Figure 6.

```
Req: POST coap://rd.example.com/rd-group?gp=lights
                                  &con=coap://[ff35:30:2001:db8::1]
Content-Format: 40
Payload:
</rd/4521>,
</rd/4522>

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

The href value is the path to the registration resource of the Endpoint.

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 initially 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 path of the group resource 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

To discover the resources registered with the RD, a lookup interface must 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 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 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.

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
Group	core.rd-lookup-gp	Optional

Table 1: Lookup Types

7.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD if they were accessed on the endpoint itself. The links and link parameters returned are equal to the submitted ones except for anchor, which was resolved by the server against the endpoint's context.

Links that did not have an anchor attribute are therefore returned with the (explicitly or implicitly set) context URI of the registration as the anchor. Links whose anchor was submitted as an absolute URI are returned as they were registered. The hrefs of links can always be served as they were submitted; the server MAY return relative references in absolute form in to resource lookups, but that results in needlessly verbose responses.

Above rules allow the client to interpret the response as links without any further knowledge of what the RD does. The Resource Directory MAY replace the contexts with a configured intermediate proxy, e.g. in the case of an HTTP lookup interface for CoAP endpoints.

7.2. Endpoint and group lookup

Endpoint and group lookups result in links to registration resources and group resources, respectively. Endpoint registration resources are annotated with their endpoint names (ep), domains (d, if present), context (con) and lifetime (lt, if present). Additional endpoint attributes are added as link attributes to their endpoint link unless their specification says otherwise. Group resources are annotated with their group names (gp), domain (d, if present) and multicast address (con, if present).

While Endpoint Lookup does expose the registration resources, the RD does not need to make them accessible to clients. Clients SHOULD NOT attempt to dereference or manipulate them.

7.3. Lookup filtering

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 search criteria MAY be included in a lookup. All included criteria MUST match for a link to be returned.

A link matches a search criterion if it has an attribute of the same name and the same value, allowing for a trailing "*" wildcard operator as in Section 4.1 of [RFC6690]. Attributes that are defined as "link-type" match if the search value matches any of their values (see Section 4.1 of [RFC6690]; eg. "?if=core.s" matches ";if="abc core.s";"). A link also matches a search criterion if the link that would be produced for any of its containing entities would match the criterion: A search criterion matches an endpoint if it matches the endpoint itself or any of the groups it is contained in, and one on a resource if it matches the resource, the resource's endpoint, or any of the endpoint's groups.

Note that "href" is also a valid search criterion and matches target references. Like all search criteria, on a resource lookup it can match the target reference of the resource link itself, but also the registration resource of the endpoint that registered it, or any group resource that endpoint is contained in.

Clients that are interested in a lookup result repeatedly or continuously can use mechanisms like ETag caching, resource observation ([RFC7641]), or any future mechanism that might allow more efficient observations of collections. These are advertised, detected and used according to their own specifications and can be used with the lookup interface as with any other resource.

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: {+type-lookup-location}{?page,count,search*}

URI Template Variables:

type-lookup-location := RD Lookup URI for a given lookup type (mandatory). The address is discovered as described in Section 5.2.

search := Search criteria for limiting the number of results (optional).

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.

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. The payload can contain zero links (which is an empty payload, "80" (hex) or "[]" in the respective content format), indicating that no entities matched the 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

7.4. Lookup examples

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 resource look-up locations discovered in Figure 6:

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

Res: 2.05 Content
</temp>;rt="temperature";anchor="coap://[2001:db8:3::123]:61616"
```

The same lookup using the CBOR Link Format media type:

```
Req: GET /rd-lookup/res?rt=temperature
Accept: TBD64

Res: 2.05 Content
Content-Format: TBD64
Payload in Hex notation:
81A301652F74656D70096B74656D706572617475726503781E636F61703A2F2F5B323030
313A6462383A333A3A3132335D3A3631363136
Decoded payload:
[{1: "/temp", 9: "temperature", 3: "coap://[2001:db8:3::123]:61616"}]
```

A client that wants to be notified of new resources as they show up can use observation:

```
Req: GET /rd-lookup/res?rt=light
Observe: 0

Res: 2.05 Content
Observe: 23
Payload: empty

(at a later point in time)

Res: 2.05 Content
Observe: 24
Payload:
</west>;rt="light";anchor="coap://[2001:db8:3::124]",
</south>;rt="light";anchor="coap://[2001:db8:3::124]",
</east>;rt="light";anchor="coap://[2001:db8:3::124]"
```

The following example shows a client performing an endpoint type lookup:

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

Res: 2.05 Content
</rd/1234>;con="coap://[2001:db8:3::127]:61616";ep="node5";
et="power-node";ct="40";lt="600",
</rd/4521>;con="coap://[2001:db8:3::129]:61616";ep="node7";
et="power-node";ct="40";lt="600";d="floor-3"
```

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

```
Req: GET /rd-lookup/gp

Res: 2.05 Content
</rd-group/1>;gp="lights1";d="example.com";con="coap://[ff35:30:2001:db8::1]",
</rd-group/2>;gp="lights2";d="example.com";con="coap://[ff35:30:2001:db8::2]"
```

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
</rd/abcd>;con="coap://[2001:db8:3::123]:61616";ep="node1";et="power-node";ct="4
0";lt="600",
</rd/efgh>;con="coap://[2001:db8:3::124]:61616";ep="node2";et="power-node";ct="4
0";lt="600"
```

The following example shows a client performing a lookup for all groups the endpoint "node1" belongs to:

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

Res: 2.05 Content
</rd-group/1>;gp="lights1"
```

The following example shows a client performing a paginated resource lookup

```
Req: GET /rd-lookup/res?page=0&count=5

Res: 2.05 Content
</res/0>;rt=sensor;ct=60;anchor="coap://[2001:db8:3::123]:61616",
</res/1>;rt=sensor;ct=60;anchor="coap://[2001:db8:3::123]:61616",
</res/2>;rt=sensor;ct=60;anchor="coap://[2001:db8:3::123]:61616",
</res/3>;rt=sensor;ct=60;anchor="coap://[2001:db8:3::123]:61616",
</res/4>;rt=sensor;ct=60;anchor="coap://[2001:db8:3::123]:61616"

Req: GET /rd-lookup/res?page=1&count=5

Res: 2.05 Content
</res/5>;rt=sensor;ct=60;anchor="coap://[2001:db8:3::123]:61616",
</res/6>;rt=sensor;ct=60;anchor="coap://[2001:db8:3::123]:61616",
</res/7>;rt=sensor;ct=60;anchor="coap://[2001:db8:3::123]:61616",
</res/8>;rt=sensor;ct=60;anchor="coap://[2001:db8:3::123]:61616",
</res/9>;rt=sensor;ct=60;anchor="coap://[2001:db8:3::123]:61616"
```

The following example shows a client performing a lookup of all resources from endpoints of a given endpoint type. It assumes that two endpoints (with endpoint names "sensor1" and "sensor2") have previously registered with their respective addresses "coap://sensor1.example.com" and "coap://sensor2.example.com", and posted the very payload of the 6th request of section 5 of [RFC6690].

It demonstrates how the link targets stay unmodified, but the anchors get constructed by the resource directory:

Req: GET /rd-lookup/res?et=sensor-node

```
</sensors>;ct=40;title="Sensor Index";
    anchor="coap://sensor1.example.com",
</sensors/temp>;rt="temperature-c";if="sensor";
    anchor="coap://sensor1.example.com",
</sensors/light>;rt="light-lux";if="sensor";
    anchor="coap://sensor1.example.com",
<http://www.example.com/sensors/t123>;rel="describedby";
    anchor="coap://sensor1.example.com/sensors/temp",
</t>;rel="alternate";anchor="coap://sensor1.example.com/sensors/temp",
</sensors>;ct=40;title="Sensor Index";
    anchor="coap://sensor2.example.com",
</sensors/temp>;rt="temperature-c";if="sensor";
    anchor="coap://sensor2.example.com",
</sensors/light>;rt="light-lux";if="sensor";
    anchor="coap://sensor2.example.com",
<http://www.example.com/sensors/t123>;rel="describedby";
    ;anchor="coap://sensor2.example.com/sensors/temp",
</t>;rel="alternate";anchor="coap://sensor2.example.com/sensors/temp"
```

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 within (the domain of) 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.

Consider te following threat: two devices A and B are managed by a single server. Both devices have unique, per-device credentials for use with DTLS to make sure that only parties with authorization to access A or B can do so.

Now, imagine that a malicious device A wants to sabotage the device B. It uses its credentials during the TLS exchange. Then, it puts the endpoint name of device B. If the server does not check whether the identifier provided in the DTLS handshake matches the identifier used at the CoAP layer then it may be inclined to use the endpoint name for looking up what information to provision to the malicious device.

Therfore, Endpoints MUST include the Endpoint identifier in the message, 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 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", 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 short name as used in query parameters or link attributes, * indication of whether it can be passed as a query parameter at registration of endpoints or groups, as a query parameter in lookups, or be expressed as a link attribute, * validity requirements if any, and * a description.

The query parameter MUST be both a valid URI query key [RFC3986] and a parmname as used in [RFC5988].

The description must give details on which registrations they apply to (Endpoint, group registrations or both? Can they be updated?), and how they are to be processed in lookups.

The mechanisms around new RD parameters should be designed in such a way that they tolerate RD implementations that are unaware of the parameter and expose any parameter passed at registration or updates on in endpoint lookups. (For example, if a parameter used at registration were to be confidential, the registering endpoint should be instructed to only set that parameter if the RD advertises support for keeping it confidential at the discovery step.)

Initial entries in this sub-registry are as follows:

Full name	Short	Validity	Use	Description
Endpoint Name	ep		RLA	Name of the endpoint, max 63 bytes
Lifetime	lt	60-4294967295	RLA	Lifetime of the registration in seconds
Domain	d		RLA	Domain to which this endpoint belongs
Context	con	URI	RLA	The scheme, address and port and path at which this server is available
Group Name	gp		RLA	Name of a group in the RD
Page	page	Integer	L	Used for pagination
Count	count	Integer	L	Used for pagination
Endpoint Type	et		RLA	Semantic name of the endpoint (see Section 9.4)

Table 2: RD Parameters

(Short: Short name used in query parameters or link attributes. Use: R = used at registration, L = used at lookup, A = expressed in link attribute

The descriptions for the options defined in this document are only summarized here. To which registrations they apply and when they are to be shown is described in the respective sections of this document.

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC8126]. The evaluation should consider formal criteria, duplication of functionality (Is the new entry redundant with an existing one?), topical suitability (Eg. is the described property actually a property of the endpoint and not a property of a particular resource, in which case it should go into the payload of the registration and need not be registered?), and the potential for conflict with commonly used link attributes (For example, "if" could be used as a parameter for conditional registration if it were not to be used in lookup or attributes, but would make a bad parameter for lookup, because a resource lookup with an "if" query parameter could ambiguously filter by the registered endpoint property or the [RFC6690] link attribute). It is expected that the registry will receive between 5 and 50 registrations in total over the next years.

9.3.1. Full description of the "Endpoint Type" Registration Parameter

An endpoint registering at an RD can describe itself with endpoint types, similar to how resources are described with Resource Types in [RFC6690]. An endpoint type is expressed as a string, which can be either a URI or one of the values defined in the Endpoint Type subregistry. Endpoint types can be passed in the "et" query parameter as part of extra-attrs at the Registration step, are shown on endpoint lookups using the "et" target attribute, and can be filtered for using "et" as a search criterion in resource and endpoint lookup. Multiple endpoint types are given as separate query parameters or link attributes.

Note that Endpoint Type differs from Resource Type in that it uses multiple attributes rather than space separated values. As a result, Resource Directory implementations automatically support correct filtering in the lookup interfaces from the rules for unknown endpoint attributes.

9.4. "Endpoint Type" (et=) RD Parameter values

This specification establishes a new sub-registry under "CoRE Parameters" called '"Endpoint Type" (et=) RD Parameter values'. The registry properties (required policy, requirements, template) are identical to those of the Resource Type parameters in [RFC6690], in short:

The review policy is IETF Review for values starting with "core", and Specification Required for others.

The requirements to be enforced are:

- o The values MUST be related to the purpose described in Section 9.3.1.
- o The registered values MUST conform to the ABNF reg-rel-type definition of [RFC6690] and MUST NOT be a URI.
- o It is recommended to use the period "." character for segmentation.

The registry is initially empty.

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.

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	2001:db8:4::1
luminary2	2001:db8:4::2
Presence sensor	2001:db8:4::3
Resource directory	2001:db8:4::ff

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

It is assumed that the CT knows of the RD’s address, and has performed URI discovery on it that gave a response like the one in the Section 5.2 example.

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://[2001:db8:4::ff]/rd
  ?ep=lm_R2-4-015_wndw&con=coap://[2001:db8:4::1]&d=R2-4-015
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"

Res: 2.01 Created
Location: /rd/4521

Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=lm_R2-4-015_door&con=coap://[2001:db8:4::2]&d=R2-4-015
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"

Res: 2.01 Created
Location: /rd/4522

Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=ps_R2-4-015_door&con=coap://[2001:db8:4::3]d&d=R2-4-015
Payload:
</ps>;rt="p-sensor"

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://[2001:db8:4::ff]/rd-group
?gp=grp_R2-4-015&con=coap://[ff05::1]
Payload:
</rd/4521>,
</rd/4522>,
</rd/4523>

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://[2001:db8:4::ff]/rd-lookup/ep
  ?d=R2-4-015;rt=light

Res: 2.05 Content
</rd/4521>;con="coap://[2001:db8:4::1]",
  ep="lm_R2-4-015_wndw",
</rd/4522>;con="coap://[2001:db8:4::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://[2001:db8:4::ff]/rd-lookup/gp
  ?ep=lm_R2-4-015_wndw

Res: 2.05 Content
</rd-group/501>;gp="grp_R2-4-015";con="coap://[ff05::1]"
```

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 //[2001:db8:4::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 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 RD registration URI path of the LWM2M Resource Directory is specified to be "/rd".

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, Lifetime, and LWM2M Version are mandatory parameters for the register operation, all other registration parameters are optional.

Additional optional LWM2M registration parameters are defined:

Name	Query	Validity	Description
Binding Mode	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

The LwM2M update is really very similar to the registration update as described in Section 5.4.1, with the only difference that there are more parameters defined and available. All the parameters listed in that section are also available with the initial registration but are all optional:

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, Jim Schaad, Mohit Sethi, Hauke Petersen, Hannes Tschofenig, Sampo Ukkola, Linyi Tian, 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 -11 to -12

- o added Content Model section, including ER diagram
- o removed domain lookup interface; domains are now plain attributes of groups and endpoints
- o updated chapter "Finding a Resource Directory"; now distinguishes configuration-provided, network-provided and heuristic sources
- o improved text on: atomicity, idempotency, lookup with multiple parameters, endpoint removal, simple registration
- o updated LWM2M description
- o clarified where relative references are resolved, and how context and anchor interact
- o new appendix on the interaction with RFCs 6690, 5988 and 3986
- o lookup interface: group and endpoint lookup return group and registration resources as link targets
- o lookup interface: search parameters work the same across all entities

- removed all methods that modify links in an existing registration (POST with payload, PATCH and iPATCH)

- removed plurality definition (was only needed for link modification)

- enhanced IANA registry text

- More examples and improved text

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 o clarify group configuration section 6.1 that endpoints must be registered before including them in a group

o removed all superfluous client-server diagrams

o simplified lighting example

o introduced Commissioning Tool

o RD-Look-up text is extended.

changes from -06 to -07

o added text in the discovery section to allow content format hints to be exposed in the discovery link attributes

o editorial updates to section 9

o update author information

o minor text corrections

Changes from -05 to -06

o added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

o added Update Endpoint Links using PATCH

o http access made explicit in interface specification

o Added http examples

Changes from -03 to -04:

o Added http response codes

o Clarified endpoint name usage

o Add application/link-format+cbor content-format

Changes from -02 to -03:

o Added an example for lighting and DNS integration

o Added an example for RD use in OMA LWM2M

- o Added Read Links operation for link inspection by endpoints

- o Expanded DNS-SD section

- o Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

- o Added a catalogue use case.

- o Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.

- o Additional examples section added for more complex use cases.

- o New DNS-SD mapping section.

- o Added text on endpoint identification and authentication.

- o Error code 4.04 added to Registration Update and Delete requests.

- o Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

- o Removed the ETag validation feature.

- o Place holder for the DNS-SD mapping section.

- o Explicitly disabled GET or POST on returned Location.

- o New registry for RD parameters.

- o Added support for the JSON Link Format.

- o Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

- o Updated the version and date.

Changes from -04 to -05:

- o Restricted Update to parameter updates.

- o Added pagination support for the Lookup interface.

- 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.

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

13. References

13.1. Normative References

[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-09 (work in progress), July 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://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, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <https://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, <https://www.rfc-editor.org/info/rfc6570>.

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

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

13.2. Informative References

[ER] Chen, P., "The entity-relationship model---toward a unified view of data", ACM Transactions on Database Systems Vol. 1, pp. 9-36, DOI 10.1145/320434.320440, March 1976.

[I-D.arkko-core-dev-urn] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-04 (work in progress), July 2017.

[I-D.nottingham-rfc5988bis] Nottingham, M., "Web Linking", draft-nottingham-rfc5988bis-08 (work in progress), August 2017.

[I-D.silverajan-core-coap-protocol-negotiation] Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", draft-silverajan-core-coap-protocol-negotiation-07 (work in progress), October 2017.

[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, <https://www.rfc-editor.org/info/rfc2616>.

[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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7390>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

Appendix A. Web links and the Resource Directory

Understanding the semantics of a link-format document and its URI references is a journey through different documents ([RFC3986] defining URIs, [RFC6690] defining link-format documents based on [RFC5988] which defines link headers, and [RFC7252] providing the transport). This appendix summarizes the mechanisms and semantics at play from an entry in ".well-known/core" to a resource lookup.

This text is primarily aimed at people entering the field of Constrained Restful Environments from applications that previously did not use web mechanisms.

A.1. A simple example

Let’s start this example with a very simple host, "2001:db8:f0::1". A client that follows classical CoAP Discovery ([RFC7252] Section 7), sends the following multicast request to learn about neighbours supporting resources with resource-type "temperature".

The client sends a link-local multicast:

```
GET coap://[ff02::fd]:5683/.well-known/core?rt=temperature

RES 2.05 Content
</temp>;rt=temperature;ct=0
```

where the response is sent by the server, "[2001:db8:f0::1]:5683".

While the client - on the practical or implementation side - can just go ahead and create a new request to "[2001:db8:f0::1]:5683" with Uri-Path: "temp", the full resolution steps without any shortcuts are:

A.1.1. Resolving the URIs

The client parses the single returned record. The link's target (sometimes called "href") is ""/temp"", which is a relative URI that needs resolving. The Base URI to resolve that against is, in absence of an "anchor" parameter, the URI of the requested resource as described in [RFC6690] Section 2.1.

The URI of the requested resource can be composed by following the steps of [RFC7252] section 6.5 (with an addition at the end of 8.2) into ""coap://[2001:db8:f0::1]/.well-known/core"".

The record's target is resolved by replacing the path ""/.well-known/core"" from the Base URI (section 5.2 [RFC3986]) with the relative target URI ""/temp"" into ""coap://[2001:db8:f0::1]/temp"".

A.1.2. Interpreting attributes and relations

Some more information but the record's target can be obtained from the payload: the resource type of the target is "temperature", and its content type is text/plain (ct=0).

A relation in a web link is a three-part statement that the context resource has a named relation to the target resource, like "_This page_ has _its table of contents_ at _/toc.html_". In [RFC6690] link-format documents, there is an implicit "host relation" specified with default parameter: rel="hosts".

In our example, the context of the link is the URI of the requested document itself. A full English expression of the "host relation" is:

'"coap://[2001:db8:f0::1]/.well-known/core" is hosting the resource "coap://[2001:db8:f0::1]/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'

A.2. A slightly more complex example

Omitting the "rt=temperature" filter, the discovery query would have given some more records in the payload:

```
</temp>;rt=temperature;ct=0,
</light>;rt=light-lux;ct=0,
</t>;anchor="/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;anchor="/sensors/temp";
    rel=describedby,
<t123.pdf>;rel=alternate;ct=65001;
    anchor="http://www.example.com/sensors/t123"
```

Parsing the third record, the client encounters the "anchor" parameter. It is a URI relative to the document's Base URI and is thus resolved to ""coap://[2001:db8:f0::1]/sensors/temp"". That is the context resource of the link, so the "rel" statement is not about the target and the document Base URI any more, but about the target and that address.

Thus, the third record could be read as ""coap://[2001:db8:f0::1]/sensors/temp" has an alternate representation at "coap://[2001:db8:f0::1]/t"".

The fourth record can be read as ""coap://[2001:db8:f0::1]/sensors/temp" is described by "http://www.example.com/sensors/t123""

In the last example the anchor is absolute, where a ""t123.pdf"" is resolved relative to ""http://www.example.com/sensors/t123"", which gives a statement that ""http://www.example.com/sensors/t123/t123.pdf" is an alternate representation to ""http://www.example.com/sensors/t123" of which the content type is PDF".

A.3. Enter the Resource Directory

The resource directory tries to carry the semantics obtainable by classical CoAP discovery over to the resource lookup interface as faithfully as possible.

For the following queries, we will assume that the simple host has used Simple Registration to register at the resource directory that was announced to it, sending this request from its UDP port "[2001:db8:f0::1]:6553":

```
POST coap://[2001:db8:f01::ff]/.well-known/core?ep-simple-host1
```

The resource directory would have accepted the registration, and queried the simple host's ".well-known/core" by itself. As a result, the host is registered as an endpoint in the RD with the name "simple-host1". The registration is active for 86400 seconds, and the endpoint registration Base URI is ""coap://[2001:db8:f0::1]/""

because that is the address the registration was sent from (and no explicit "con=" was given).

If the client now queries the RD as it would previously have issued a multicast request, it would go through the RD discovery steps by fetching "coap://[2001:db8:f0::ff]/.well-known/core?rt=core.rd-lookup-res", obtain "coap://[2001:db8:f0::ff]/rd-lookup/res" as the resource lookup endpoint, and issue a request to "coap://[2001:db8:f0::ff]/rd-lookup/res?rt=temperature" to receive the following data:

```
</temp>;rt=temperature;ct=0;anchor="coap://[2001:db8:f0::1]"
```

This is not _literally_ the same response that it would have received from a multicast request, but it would contain the (almost) same statement:

'"coap://[2001:db8:f0::1]" is hosting the resource "coap://[2001:db8:f0::1]/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'

(The difference is whether "/" or "/.well-known/core" hosts the resources, which is subject of ongoing discussion about RFC6690).

To complete the examples, the client could also query all resources hosted at the endpoint with the known endpoint name "simple-host1". A request to "coap://[2001:db8:f0::ff]/rd-lookup/res?ep=simple-host1" would return

```
</temp>;rt=temperature;ct=0;anchor="coap://[2001:db8:f0::1]",
</light>;rt=light-lux;ct=0;anchor="coap://[2001:db8:f0::1]",
</t>;anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;
    anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=describedby,
<t123.pdf>;rel=alternate;ct=65001;
    anchor="http://www.example.com/sensors/t123"
```

Note that the last link was not modified at all because its anchor was already an absolute reference.

Had the simple host registered with an explicit context (eg. "?ep=simple-host1&con=coap+tcp://simple-host1.example.com"), that context would have been used to resolve the relative anchor values instead, giving

```
</temp>;rt=temperature;ct=0;anchor="coap+tcp://simple-host1.example.com"
```

and analogous records.

A.4. A note on differences between link-format and Link headers

While link-format and Link headers look very similar and are based on the same model of typed links, there are some differences between [RFC6690] and [RFC5988] that should be kept in mind when using or implementing a Resource Directory:

- There is no percent encoding in link-format documents.

  A link-format document is a UTF-8 encoded string of Unicode characters and does not have percent encoding, while Link headers are practically ASCII strings that use percent encoding for non-ASCII characters, stating the encoding explictly when required.

  For example, while a Link header in a page about a Swedish city might read

  "Link: </temperature/Malm%C3%B6>;rel="live-environment-data""

  a link-format document from the same source might describe the link as

  "</temperature/Malmoe>;rel="live-environment-data""

- In a link-format document, if the anchor attribute is present, the link target reference is resolved by using the the (resolved) anchor value as Base URI for that link, while in Link headers, it is resolved against the URI of the requested document.

  This is explicit in [RFC6690] section 2.1 for link-format, and spelled out in section B.2 of [I-D.nottingham-rfc5988bis] , which obsoletes the older [RFC5988]. [RFC6690] is based on [RFC5988] and has not been updated with clarifications from [I-D.nottingham-rfc5988bis].

Appendix B. Syntax examples for Protocol Negotiation

[This appendix should not show up in a published version of this document.]

The protocol negotiation that is being worked on in [I-D.silverajan-core-coap-protocol-negotiation] makes use of the Resource Directory.

Until that document is update to use the latest resource-directory specification, here are some examples of protocol negotiation with the current Resource Directory:

An endpoint could register as follows:

```
Req: POST coap://rd.example.com/rd?ep=node1
    &at=coap+tcp://[2001:db8:f1::2]
    &at=coap://[2001:db8:f1::2]
Content-Format: 40
Payload:
</temperature>;ct=0;rt="temperature";if="core.s"

Res: 2.01 Created
Location: /rd/1234
```

A UDP client would then query:

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

Res: 2.05 Content
</temperature>;ct=0;rt="temperature";if="core.s";
    anchor="coap://[2001:db8:f1::2]"
```

while a TCP capable client could say:

```
Req: GET /rd-lookup/res?rt=temperature&tt=tcp

Res: 2.05 Content
</temperature>;ct=0;rt="temperature";if="core.s";
    anchor="coap+tcp://[2001:db8:f1::2]"
```

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

Christian Amsuess (editor)
Energy Harvesting Solutions
Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639
Email: c.amsuess@energyharvesting.at

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

C. Jennings
Cisco
Z. Shelby
ARM
J. Arkko
A. Keranen
Ericsson
C. Bormann
Universitaet Bremen TZI

October 30, 2017

Media Types for Sensor Measurement Lists (SenML)
draft-ietf-core-senml-11

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 will expire on May 3, 2018.

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 fields 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 from large numbers of constrained devices. Keeping the total size of payload small makes it easy to use SenML also in constrained networks, e.g., in a 6LoWPAN [RFC4944]. 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 given by the "v" field, the time of a measurement is in the "t" field, the "n" field has a unique sensor name, and the unit of the measurement is carried in the "u" field.

```
[
  {"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" field 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" field and the "n" fields 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.

SenML Label: A short name used in SenML Records to denote different SenML fields (e.g., "v" for "value").

SenML Field: A component of a record that associates a value to a SenML Label for this record.

This document uses the terms "attribute" and "tag" where they occur with the underlying technologies (XML, CBOR [RFC7049], and Link Format [RFC6690]), not for SenML concepts per se. Note that "attribute" has been widely used previously as a synonym for SenML "field", though.

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 fields described below. There are two kinds of fields: base and regular. The base fields can be included in any SenML Record and they apply to the entries in the Record. Each base field also applies to all Records after it up to, but not including, the next Record that has that same base field. All base fields are optional. Regular fields can be included in any SenML Record and apply only to that Record.

4.1. Base Fields

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 field 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 Fields

Name: Name of the sensor or parameter. When appended to the Base Name field, 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 field is in the units specified in the Unit value multiplied by seconds.

Time: Time when value was recorded. Optional.

Update Time: Period of time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement. Optional. 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 fields. Both new base and regular fields are allowed. See Section 12.2 for details. Implementations MUST ignore fields they don't recognize unless that field has a label name that ends with the '_' character in which case an error MUST be generated.

All SenML Records in a Pack MUST have the same version number. This is typically done by adding a Base Version field to only the first Record in the Pack.

Systems reading one of the objects MUST check for the Version field. 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 structure or semantics that is different from what is defined in the present document beyond just making use of the extension points provided here. 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 yield the name of the sensor. The resulting concatenated name needs to uniquely identify and differentiate the sensor from all others. The concatenated name MUST consist only of characters out of the set "A" to "Z", "a" to "z", "0" to "9", "-", ":", ".", "/", and "_"; furthermore, 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 concatenated names can be used directly within various URI schemes (including segments of an HTTP path with no special encoding) and can be used directly in many databases and analytic systems. [RFC5952] contains advice on encoding an IPv6 address in a name. See Section 14 for privacy considerations that apply to the use of long-term stable unique identifiers.

Although it is RECOMMENDED that concatenated names are represented as URIs [RFC3986] or URNs [RFC8141], the restricted character set specified above puts strict limits on the URI schemes and URN namespaces that can be used. As a result, implementers need to take care in choosing the naming scheme for concatenated names, because such names both need to be unique and need to conform to the restricted character set. One approach is to include a bit string that has guaranteed uniqueness (such as a 1-wire address). Some of the examples within this document use the device URN namespace as specified in [I-D.arkko-core-dev-urn]. UUIDs [RFC4122] are another way to generate a unique name. However, the restricted character set does not allow the use of many URI schemes in names as such. The use of URIs with characters incompatible with this set, and possible mapping rules between the two, are outside of the scope of the present document.

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 field 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 field 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 field(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 fields to provide better information about the statistical properties of the measurement.

In summary, the structure of a SenML record is laid out to support a single measurement per record. If multiple data values are measured at the same time (e.g., air pressure and altitude), they are best kept as separate records linked through their Time value; this is even true where one of the data values is more "meta" than others (e.g., describes a condition that influences other measurements at the same time).

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, i.e., labels starting with the character 'b', except for Version fields (see below), and has no relative times. To resolve the records, 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.

The Version field MUST NOT be present in resolved records if the SenML version defined in this document is used and MUST be present otherwise in all the resolved SenML Records.

Future specification that defines new base fields need to specify how the field 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=) link attribute (which is defined for the Link Format in Section 7.2.1 of [RFC7252]).

4.6. Configuration and Actuation usage

SenML can also be used for configuring parameters and controlling actuators. When a SenML Pack is sent (e.g., using a HTTP/CoAP POST or PUT method) and the semantics of the target are such that SenML is interpreted as configuration/actuation, SenML Records are interpreted as a request to change the values of given (sub)resources (given as names) to given values at the given time(s). The semantics of the target resource supporting this usage can be described, e.g., using [I-D.ietf-core-interfaces]. Examples of actuation usage are shown in Section 5.1.7.

5. JSON Representation (application/senml+json)

For the SenML fields shown in Table 1, the SenML labels are used as the JSON object member names within JSON objects representing the JSON SenML Records.

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

Table 1: JSON SenML Labels

The root JSON value 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 member values of 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], with padding omitted.

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 12.3.2). 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	161	184

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 network that can guarantee delivery of the messages to the two lights within 15 ms (e.g. a network using 802.1BA [IEEE802.1ba-2011] and 802.1AS [IEEE802.1as-2011] 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 labels, as 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

Table 3: CBOR representation: integers for map keys

- o For streaming SensML in CBOR representation, the array containing the records SHOULD be a 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
```

In CBOR diagnostic notation (Section 6 of [RFC7049]), this is:

```
[{-2: "urn:dev:ow:10e2073a0108006:",
  -3: 1276020076.001, -4: "A", -1: 5, 0: "voltage", 1: "V", 2: 120.1},
 {0: "current", 6: -5, 2: 1.2}, {0: "current", 6: -4, 2: 1.3},
 {0: "current", 6: -3, 2: 1.4}, {0: "current", 6: -2, 2: 1.5},
 {0: "current", 6: -1, 2: 1.6}, {0: "current", 6: 0, 2: 1.7}]
```

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 element that contains a series of senml elements for each SenML Record. The SenML fields are represented as XML attributes. For each field defined in this document, the following table shows the SenML labels, which are used for the XML attribute name, as well as the according restrictions on the XML attribute values ("type") as used in the XML senml elements.

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

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 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="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 f3 ab 93 71 d3 23 2b b1 d3 7b b9 |.0......q.#+..{.|
0010 d1 89 83 29 91 81 b9 9b 09 81 89 81 c1 81 81 b1 |...)............|
0020 99 d2 84 bb 37 b6 3a 30 b3 b2 90 1a b1 58 84 c0 |....7.:0.....X..|
0030 33 04 b1 ba b9 39 32 b7 3a 10 1a 09 06 40 38    |3....92.:....@8|
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 06 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 device 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 earlier records)

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" field of the measurement. It might optionally include the current power in the "v" field.

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. Implementers 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.ietf-cbor-cddl] specification in Figure 1.

```
SenML-Pack = [1* 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 = non-b-label / b-label
non-b-label = tstr .regexp  "[A-Zac-z0-9][-_:.A-Za-z0-9]*" / uint
b-label = tstr .regexp  "b[-_:.A-Za-z0-9]+" / nint

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 1)	float	RFC-AAAA
lon	degrees longitude (note 1)	float	RFC-AAAA
pH	pH value (acidity; logarithmic quantity)	float	RFC-AAAA
dB	decibel (logarithmic quantity)	float	RFC-AAAA
dBW	decibel relative to 1 W (power level)	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,	float	RFC-AAAA

	note 2)		
%	1 (Ratio e.g., value of a switch, note 2)*	float	RFC-AAAA
%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: Assumed to be in WGS84 unless another reference frame is known for the sensor.

- Note 2: 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!

New entries can be added to the registration by Expert Review as defined in [RFC8126]. 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. implementers 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.

   (Note that some amount of judgment will be required here, as even SI itself is not entirely consistent in this respect. For instance, for temperature [ISO-80000-5] defines a quantity, item 5-1 (thermodynamic temperature), and a corresponding unit 5-1.a (Kelvin), and then goes ahead to define another quantity right besides that, item 5-2 ("Celsius temperature"), and the corresponding unit 5-2.a (degree Celsius). The latter quantity is defined such that it gives the thermodynamic temperature as a delta from T0 = 275.15 K. ISO 80000-5 is defining both units side by side, and not really expressing a preference. This level of recognition of the alternative unit degree Celsius is the reason why Celsius temperatures exceptionally seem acceptable in the SenML units list alongside Kelvin.)

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	Type	EXI 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

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 EXI ID column 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 lose 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 [RFC8126]. Experts should exercise their own good judgment but need to consider that shorter labels should have more strict review. New entries should not be made that counteract the advice at the end of Section 4.3.

All new SenML labels that have "base" semantics (see Section 4.1) MUST start with the 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 and 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 XML tags or attributes that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" attribute in the senml XML 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 XML tags or attributes that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" attribute in the senml XML 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 XML tags or attributes that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" attribute in the senml XML 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 XML tags or attributes that they do not understand. This allows backwards compatibility extensions to this specification. The "bver" attribute in the senml XML 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.

The name fields need to uniquely identify the sources or destinations of the values in a SenML Pack. However, the use of long-term stable unique identifiers can be problematic for privacy reasons [RFC6973], depending on the application and the potential of these identifiers to be used in correlation with other information. They should be used with care or avoided as for example described for IPv6 addresses in [RFC7721].

15. Acknowledgement

We would like to thank Alexander Pelov, Andrew McClure, Andrew McGregor, Bjoern Hoehrmann, Christian Amsuess, Christian Groves, Daniel Peintner, Jan-Piet Mens, Jim Schaad, Joe Hildebrand, John Klensin, Karl Palsson, Lennart Duhrsen, Lisa Dusseault, Lyndsay Campbell, Martin Thomson, Michael Koster, Peter Saint-Andre, 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <https://www.rfc-editor.org/info/rfc7303>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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-05 (work in progress), October 2017.

[I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-00 (work in progress), July 2017.

[I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-10 (work in progress), September 2017.

[IEEE802.1as-2011]
IEEE, "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks", 2011.

[IEEE802.1ba-2011]
IEEE, "IEEE Standard for Local and metropolitan area networks--Audio Video Bridging (AVB) Systems", 2011.

[ISO-80000-5]
"Quantities and units - Part 5: Thermodynamics", ISO 80000-5, Edition 1.0, May 2007.

[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, <https://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, <https://www.rfc-editor.org/info/rfc4122>.

[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, <https://www.rfc-editor.org/info/rfc4944>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

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

[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, <https://www.rfc-editor.org/info/rfc6973>.

[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, <https://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, <https://www.rfc-editor.org/info/rfc7721>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

[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>.

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 M. Veillette, Ed.
Internet-Draft Trilliant Networks Inc.
Intended status: Standards Track A. Pelov, Ed.
Expires: May 3, 2018 Acklio
October 30, 2017

YANG Schema Item iDentifier (SID)
draft-ietf-core-sid-02

Abstract

YANG Schema Item iDentifiers (SID) are globally unique 64-bit unsigned numbers used to identify YANG items. 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

Copyright Notice

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".

Assignment of SIDs to YANG items can be automated, the recommended process to assign SIDs is as follows:

1. A tool extracts the different items defined for a specific YANG module.

2. 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.

3. 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.

4. 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, only unassigned YANG items are processed at step #3.

Section 3 provides 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.

```
          +---------------+
   O      | Creation of a |
  -|- ->| YANG module   |
  / \     +---------------+
                  |
                  V
           /-------------\
          / Standardized  \ yes
          \ YANG module ? /-------------+
           \-------------/              |
                  | no                  |
                  V                     V
           /-------------\      +---------------+
          / Constrained   \ yes | SID range     |
      +-->\ application ? /---->| registration  |
      |    \-------------/      +---------------+
      |           | no                  |
      |           V                     V
      |   +---------------+     +---------------+
      +---| YANG module   |     | SID sub-range |
          | update        |     | assignment    |
          +---------------+     +---------------+
                                        |
                                        V
                                +---------------+
                                | .sid file     |
                                | generation    |
                                +---------------+
                                        |
                                        V
                                 /-------------\      +---------------+
                                /  Publicly     \ yes | YANG module   |
                                \  available ?  /---->| registration  |
                                 \-------------/      +---------------+
                                        | no                  |
                                        +---------------------+
                                        |
                                      [DONE]
```

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.

```
          +---------------+
     O    | Update of the |
    -|- ->| YANG module   |
    / \   | or include(s) |
          | or import(s)  |
          +---------------+
                  |
                  V
           /-------------\      +----------------+
          /  More SIDs    \ yes | Extra sub-range|
          \  required ?   /---->| assignment     |
           \-------------/      +----------------+
                  | no                    |
                  +---------------------+
                  |
                  V
          +---------------+
          | .sid file     |
          | update based  |
          | on previous   |
          | .sid file     |
          +---------------+
                  |
                  V
           /-------------\      +---------------+
          /  Publicly     \ yes | YANG module   |
          \  available ?  /---->| registration  |
           \-------------/      +---------------+
                  | no                    |
                  +---------------------+
                  |
                [DONE]
```

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
    "Michel Veillette
     <mailto:michel.veillette@trilliantinc.com>

     Alexander Pelov
     <mailto:a@ackl.io>";

  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.";

  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 uint64;
           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 uint64;
         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:

- o The entry point (first entry) of the registered SID range.
- o The size of the registered SID range.
- o 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:

- o 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 at least 33% above 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 an existing YANG module if the initial range is 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	RFC number
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:

- o The YANG module name
- o The associated ".yang" file(s)
- o The associated ".sid" file

The validity of the ".yang" and ".sid" files added to this registry MUST be verified.

- o The syntax of the registered ".yang" and ".sid" files must be valid.
- o Each YANG item defined by the registered ".yang" file must have a corresponding SID assigned in the ".sid" file.
- o Each SID is assigned to a single YANG item, duplicate assignment is not allowed.
- o 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.
- o 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 Andy Bierman, Carsten Bormann, Abhinav Somaraju, Laurent Toutain and Randy Turner for their help during the development of this document and their 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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

8.2. Informative References

[I-D.ietf-core-comi] Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-01 (work in progress), July 2017.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

[RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <https://www.rfc-editor.org/info/rfc7224>.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <https://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, <https://www.rfc-editor.org/info/rfc7317>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://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:

- o ietf-system@2014-08-06.yang

- o ietf-yang-types@2013-07-15.yang

- o ietf-inet-types@2013-07-15.yang

- o ietf-netconf-acm@2012-02-22.yang

- o 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

Internet Engineering Task Force M. Veillette, Ed.
Internet-Draft Trilliant Networks Inc.
Intended status: Standards Track A. Pelov, Ed.
Expires: February 9, 2018 Acklio
A. Somaraju
Tridonic GmbH & Co KG
R. Turner
Landis+Gyr
A. Minaburo
Acklio
August 08, 2017

CBOR Encoding of Data Modeled with YANG
draft-ietf-core-yang-cbor-05

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].

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:

- o Any text within and including a pair of slashes is considered a comment.

- o 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-state' container instance with a single child, a clock container. The clock container container has two children, a 'current-datetime' leaf and a 'boot-datetime' leaf.

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 system-state {

  container clock {
    leaf current-datetime {
      type date-and-time;
    }

    leaf boot-datetime {
      type date-and-time;
    }
  }
}
```

For this first representation, we assume that parent SID of the root container (i.e. 'system-state') is not available to the serializer. In this case, root data nodes are encoded using absolute SIDs.

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
```

On the other hand, if the serializer is aware of the parent SID, 1716 in the case 'system-state' container, root data nodes are encoded using deltas.

CBOR diagnostic notation:

```
{
  +1 : {                               / 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)
   01                                     # unsigned(1)
   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)
         4e524320544943207365727665 72 # "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)
         4e5243205441432073657276657 2 # "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)
         4e524320544943207365727665722         # "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)
         4e524320544143207365727665722         # "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.

- o bits
- o enumeration
- o identityref
- o 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:

- o The first entry MUST be encoded as a CBOR unsigned integer data item (major type 0) and set to the targeted data node SID.

- o 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] Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-01 (work in progress), July 2017.

[I-D.ietf-core-sid] Veillette, M., Pelov, A., Turner, R., Minaburo, A., and A. Somaraju, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-01 (work in progress), May 2017.

[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

CoRE Working Group   Y. Liu
Internet-Draft   J. Zhu
Intended status: Standards Track   Huawei
Expires: May 3, 2018   October 30, 2017

Mitigating delay attacks on Constrained Application Protocol
draft-liu-core-coap-delay-attacks-01

Abstract

Various attacks including delay attack have become a topic in the security of Internet of Things (IoT), especially for the constrained nodes utilizing sensors and actuators which connect and interact with the physical world. [I-D.mattsson-core-coap-actuators] describes several serious delay attacks, discusses tougher requirements and then recommends mechanisms to mitigate the attacks. It also specifies some disadvantages with the mechanisms. This document proposes alternative mechanisms to address some of the disadvantages

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Various attacks including delay attack have become a topic in the security of Internet of Things (IoT), especially for the resource-constrained nodes [RFC7252] utilizing sensors and actuators which connect and interact with the physical world. It is recommended to use the Constrained Application Protocol (CoAP) [RFC7252], which is designed for resource-constrained nodes, and message exchange between them. Also, it is required to use security protocols such as TLS [RFC5246], DTLS [RFC6347], TLS/DTLS profiles for the IoT [RFC7925], or OSCORE [I-D.ietf-core-object-security] to protect CoAP messages due to security and privacy. The security protocols can provide confidentiality, authentication and integrity protection of CoAP messages at both the application layer and the transport layer.

There are still issues related to delay attacks as descirbed in [I-D.mattsson-core-coap-actuators]. For example, [I-D.mattsson-core-coap-actuators] describes several serious attacks, discusses tougher requirements and then recommends solution to mitigate the attacks. The draft also indicates the disadvantage that CoAP messages need two round trips for the solution. This document will show alternative mechanisms which take CoAP messages only one round trip by utilizing the sending messages containing valid time window(s), Sequence Number and Response Policy.

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 [I-D.mattsson-core-coap-actuators] and [RFC7252].

3. Attacks

It is assumed that the reader is familiar with the following attacks as specified in section 2 of [I-D.mattsson-core-coap-actuators]:

- The Block Attack

- The Request Delay Attack

- The Response Delay and Mismatch Attack

- Relay Attack

4. Solutions

In order to mitigate the attacks as above, [I-D.mattsson-core-coap-actuators] provides a challenge-response mechanism for CoAP using a new CoAP Option "Repeat". This option is described below in section 4.1, which is originally specified in section 3 of the [I-D.mattsson-core-coap-actuators]. An editor’s note indicates the disadvantages that the mechanism takes two round trips and provides two potential enhancements utilizing time.

Section 4.2 in this document describes another method which takes only one round trip CoAP messages which utilizes a "Valid Time Window" , "Sequence Number" and "Response Policy" on receiving messages to mitigate the delay attacks.

4.1. The Repeat Option

The Repeat Option is a challenge-response mechanism for CoAP, which is generated by the server and binded to an 4.03 forbidden response. The client bind the same Repeat Option value into a new request to echo the challenge. Then the server verifies the freshness of the original request. An example message flow is illustrated in Figure 1

```
                    Client  Server
                      |       |
                      +------>|        Code: 0.03 (PUT)
                      | PUT   |       Token: 0x41
                      |       |    Uri-Path: lock
                      |       |     Payload: 0 (Unlock)
                      |       |
                      |<------+ t0     Code: 4.03 (Forbidden)
                      | 4.03 |       Token: 0x41
                      |       |      Repeat: 0x6c880d41167ba807
                      |       |
                      +------>| t1     Code: 0.03 (PUT)
                      | PUT   |       Token: 0x42
                      |       |    Uri-Path: lock
                      |       |      Repeat: 0x6c880d41167ba807
                      |       |     Payload: 0 (Unlock)
                      |       |
                      |<------+        Code: 2.04 (Changed)
                      | 2.04 |       Token: 0x42
                      |       |
```

Figure 1: The Repeat Option

1) The client sends the original request without the Repeat Option to a resource on a server with freshness requirements. E.g. the client wants to unlock the door.

2) After receiving the original request, the server sends a 4.03 Forbidden response with a Repeat Option to challenge the client. The repeat Option value and the response transmit time 't0' are stored on the server.

3) The client SHOULD resend the original request with the same Repeat Option value contained in the previous response to echo the challenge. The server SHOULD store the request receive time 't1'.

4) The server firstly verifies that the Repeat Option value equals the previously sent one. Then the server calculates the round-trip time RTT = (t1 - t0). The server MUST only accept requests with a round-trip time below a certain threshold T, i.e. RTT < T. The threshold T is application specific.

4.2. The Enhanced Options

According to the method using a Repeat Option (see Section 4.1), there are still the following potential situations:

- o If the RTT of the second message to the third message (see Figure 1) is larger than the certain threshold T, the server can determine that the request message from the client is delayed and then discard it.

- o If the RTT of the second message to the third message (see Figure 1) is large but does not exceed the certain threshold T, the server treats these messages as valid and then processes them normally. But these messages may have become invalid especially in the situation where the request(s) containing actions relevant for actuators are required to be processed in a specific and limited period of time. For example, the actuator with the air conditioning may be required to keep it open in a specific time and temperature, which depends on some reasons such as user’s preference and current room temperature. In other words, the specific time may be varied, it is possible that the server determines the request is valid by RTT < T but the potential specific time associated with the request is actually past.

- o If the RTT of the third message to the fourth message (see Figure 1) is larger than the certain threshold T, it may cause that the client resends the request message but the actuator’s actions associated with the previous message has already been processed.

- o Regardless of whether the delay exists, the two round-trips increase the delay in overall processing of the original action (e.g. PUT)

In fact, the server cannot accurately know whether the messages are delayed in a reasonable period of time or not, because the reasons for the delay may be caused by the network itself and/or some attacks such as man-in-the-middle. In other words, how to set T value depends on many factors. Also, it is not enough to determine whether the delay happens.

Due to IoT covering different vertical domains actuators have different delay sensitivity requirements. Simple actuators (such as a smart switch) support a single action and may not be delay sensitive. There are others with complicated capabilities that are able to process multi-interrelated actions especially in Industrial Control and Production Systems. These actuators with multi-interrelated actions are usually associated with strict time requirements. Therefore, it is lack of a mechanism that assures them process multi-continuous actions addressed in different request(s) when the delay attack happens and even causes some mismatch/disorder.

4.2.1. Simple Single Action Actuators

For simple single action for the actuators, the Time Window Option is introduced as a new CoAP option and is to address the validity period of the request(s) from the client. The Time Window Option including T-start (i.e., a start valid time) and T-duration (i.e., a valid duration) of a request enables the server to accurately know the freshness of a request, determine how to process it, and thus achieve to mitigate the attacks described in Section 3.

```
      Client  Server
         |       |
         +----->|            Code: 0.03 (PUT)
         | PUT  |           Token: 0x41
         |       |        Uri-Path: lock
         |       |         Payload: 0 (Unlock)
         |       |    Valid-Window: T-start, T-duration
         |       |
         |<-----+ T-receive<T-start
         | 2.03 |            Code: 2.03 (Valid)
         |       |           Token: 0x41
         |       |         Payload: queueing
         |       |
         |<-----+ T-start
         | 2.05 |            Code: 2.05 (Content)
         |       |           Token: 0x41
         |       |         Payload: OK
         |       |
```

Figure 2: The Time Window Option(1)

Upon receiving a request containing the Time Window Option, the server extracts the T-start and T-duration from the first request from the client.

If T-receive (a reception time for the server receiving a request) < T-start as illustrated in Figure 2, it means that the server SHOULD not do the actions carried in the request until T-start is coming. The server SHOULD add this request to a waiting queue, and issues a temporarily response (e.g. 2.03) to the client with the payload indicating "queueing". When T-start is coming, the server gets the corresponding request from the processing buffer, executes the actions carried in the request, and sends a response 2.05 containing a payload indicating "OK".

```
          Client  Server
             |       |
             +------>|           Code: 0.03 (PUT)
             | PUT   |          Token: 0x41
             |       |       Uri-Path: lock
             |       |        Payload: 0 (Unlock)
             |       |    Valid-Window: T-start, T-duration
             |       |
             |<------+ T-receive in [T-start, T-start + T-duration]
             |       |           Code: 2.05 (Content)
             | 2.05  |          Token: 0x41
             |       |        Payload: OK
             |       |
```

Figure 3: The Time Window Option(2)

If T-receive (i.e., a reception time for the server receiving a request) >= T-start and T-receive < (T-start + T-duration) as illustrated in Figure 3, it means that the request is just in the valid period of time. The server SHOULD process this request immediately, stores a payload indicating "OK" in a normal response for the client and returns this response with Code = 2.05.

```
        Client  Server
           |       |
           +------>|           Code: 0.03 (PUT)
           | PUT   |          Token: 0x41
           |       |       Uri-Path: lock
           |       |        Payload: 0 (Unlock)
           |       |    Valid-Window: T-start, T-duration
           |       |
           |<------+ T-receive > (T-start + T-duration)
           | 4.03  |           Code: 4.03 (Forbidden)
           |       |          Token: 0x41
           |       |        Payload: Delay, Offset (T-receive - T-start)
           |       |
```

Figure 4: The Time Window Option(3)

If T-receive (i.e., a reception time for the server receiving a request) > (T-start + T-duration) as illustrated in Figure 4, it means that the request has become invalid. The server discards the request and sends a 4.03 error response to the client with a "Delay" payload indicating a time offset between T- receive and T-current. The offset helps the client to estimate the RTT between the client and the server, and thus set a more reasonable T-duration for the subsequent messages.

4.2.2. Multi-interrelated Actions

When some complicated actuators are able to support multi-interrelated actions with different request(s), it is desirable to be required give some indications to the server to make actions especially when there are delay caused by some attacks.

This document proposes the use of a Sequence Number CoAP Option to address the sending sequence of request(s) at the client side. It is used to provide some corresponding rules when the server recognizes that request(s) are disorder via the Sequence Number Options in these messages.

This document also proposes a new Response Policy CoAP Option which is valid with the Sequence Number Option. The Response Policy includes 3 modes - preemptive mode, sequential mode, and sequential with conditional discard mode. Also, the Response Policy may be pre-configured at the server side or may be specified in the requests at the client side. If the server cannot get the Response Policy, the server will select preemptive mode by default.

Upon receiving a request containing the Sequence Number Option, the server will do the following steps:

1) The server is aware that the Sequence Number value in current request (SNcur) is larger than the largest Sequence Number (SNmax) of all previous requests.

a. If the previous request with SNmax has already been normally responded and SNcur = (SNmax + 1) , the current request SHOULD be responded as specified in Section 4.2.1.

b. If the previous request with SNmax is still being queued, the server SHOULD respond the current request with SNcur according to the Response Policy and the validity period of the requests as below:

- o Preemptive mode: If T-start of the current request is expired, the server SHOULD process the current request immediately, and then discard all the previous requests in the queue since SNcur > SNmax.

- o Sequential mode: The server SHOULD respond to the requests orderly based on their Sequence Numbers. Although T-start of the current request is expired, the server SHOULD not respond to it until all the requests with Sequence Numbers less than SNcur have been responded, even if the request with a sequence number less than the SNcur has not been received by the server. Consequently, there is a possibility that the current request MAY not be responded due to its Valid Time Window (T-start + T-duration) expiration.

o Sequential with conditional discard mode: The server SHOULD respond to the requests based on their Sequence Numbers as well as the Valid Time Window (T-start + T-duration) of the requests. Once the Valid Time Window of the current request expires, the sever SHOULD respond to the current request immediately, then discard all the requests with Sequence Numbers less than SNcur.

2) The server is aware that the Sequence Number value in current request (SNcur) is smaller than the largest Sequence Number(SNmax) of all previous requests.

o Preemptive mode: If the request with SNmax has already been processed, the server SHOULD discard the current request and respond an error indicating the delay. Otherwise, if the request with SNmax is still being queued, the server SHOULD add the current request to the queue and respond these queued requests in order based on Section 4.2.1 till the T-start of the request with SNmax.

o Sequential mode: The server SHOULD add the current request to the queue till its T-start.

o Sequential with conditional discard mode: The server SHOULD add the current request to the queue till its T-start. If the Valid Time Window (T-start + T-duration) of the SNmax request expires earlier than the T-start of the current request, the server SHOULD process the request with SNmax and discard the current request.

When some complicated actuators are able to support multi-interrelated actions with different requests, it is desirable to be required give some indications to the server to process actions, especially when there are delays caused by attacks.

Note: It is to be added figures to illustrate the above examples in the future.

5. Security Considerations

The whole document can be seen as security considerations for CoAP.

6. IANA Considerations

This document requests the registration of the following Option Number, whose value have been assigned to the CoAP Option Numbers Registry defined by [RFC7252].

6.1. Tables

Number	Name
30	Time Window
31	Sequence Number
32	Response Policy

Table 1

7. Acknowledgements

TBD.

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>.

[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>.

[RFC7925] Tschofenig, H. and T. Fossati, "Transport Layer Security(TLS)/ Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC6347, December 2016, <http://www.rfc-editor.org/info/rfc7925>.

8.2. Informative References

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP", draft-ietf-core-object-security-06 (work in progress), October 2017.

[I-D.mattsson-core-coap-actuators]
Mattsson, J., Fornehed, J., Selander, G., and F. Palombini, "Controlling Actuators with CoAP", draft-mattsson-core-copa-actuators-02 (work in progress), November 2016.

Authors' Addresses

Yan Liu
Huawei
P.R.China

Email: scarlett.liuyan@huawei.com

Jintao Zhu
Huawei
P.R.China

Email: jintao.zhu@huawei.com

CoRE Working Group M. Tiloca
Internet-Draft RISE SICS AB
Intended status: Standards Track G. Selander
Expires: April 30, 2018 F. Palombini
Ericsson AB
J. Park
Universitaet Duisburg-Essen
October 27, 2017

Secure group communication for CoAP
draft-tiloca-core-multicast-oscoap-04

Abstract

This document describes a method for protecting group communication over the Constrained Application Protocol (CoAP). The proposed approach relies on Object Security for Constrained RESTful Environments (OSCORE) and the CBOR Object Signing and Encryption (COSE) format. All security requirements fulfilled by OSCORE are maintained for multicast OSCORE request messages and related OSCORE response messages. Source authentication of all messages exchanged within the group is ensured, by means of digital signatures produced through 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 [RFC7228].

Group communication for CoAP [RFC7390] addresses use cases where deployed devices benefit from a group communication model, for example to reduce 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 (see Appendix A). Furthermore, [RFC7390] recognizes the importance to introduce a secure mode for CoAP group communication. This specification defines such a mode.

Object Security for Constrained RESTful Environments (OSCORE)[I-D.ietf-core-object-security] describes a security protocol based on the exchange of protected CoAP messages. OSCORE builds on CBOR Object Signing and Encryption (COSE) [RFC8152] and provides end-to-end encryption, integrity, and replay protection between a sending endpoint and a receiving endpoint across intermediary nodes. 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 OSCORE, providing end-to-end security of CoAP messages exchanged between members of a multicast group. In particular, the described approach defines how OSCORE 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 CoAP responses sent as reply by multiple listener nodes. Multicast OSCORE provides source authentication of all CoAP messages exchanged within the group, by means of digital signatures produced through private keys of sender devices and embedded in the protected CoAP messages. As in OSCORE, 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP

14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252]; group communication for CoAP [RFC7390]; COSE and counter signatures [RFC8152].

Readers are also expected to be familiar with the terms and concepts for protection and processing of CoAP messages through OSCORE, such as "Security Context", "Master Secret" and "Master Salt", defined in [I-D.ietf-core-object-security].

Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

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 and IVs [RFC4949].

- Group Manager (GM): entity responsible for creating a multicast group, establishing and provisioning Security Contexts among authorized group members, as well as managing the joining of new group members and the leaving of current group members. A GM can be responsible for multiple multicast groups. Besides, a GM is not required to be an actual group member and to take part in the group communication. The GM is also responsible for renewing/ updating Security Contexts and related keying material in the multicast groups of its competence. Each endpoint in a multicast group securely communicates with the respective GM.

- Multicaster: member of a multicast group that sends multicast CoAP messages 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. 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 response messages associated to a given multicast request message that it has previously sent to the group.

- 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 response message to the multicaster which has sent the multicast message.

- o Pure listener: member of a multicast group that is configured as listener and never replies back to multicasters after receiving multicast messages.

- o Endpoint ID: identifier assigned by the Group Manager to an endpoint upon joining the group as a new member, unless configured exclusively as pure listener. The Group Manager generates and manages Endpoint IDs in order to ensure their uniqueness within a same multicast group. That is, within a single multicast group, the same Endpoint ID cannot be associated to more endpoints at the same time. Endpoint IDs are not necessarily related to any protocol-relevant identifiers, such as IP addresses.

- o Group request: multicast CoAP request message sent by a multicaster in the group to all listeners in the group through multicast IP, unless otherwise specified.

- o 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 either by a different group member or by a non-group member.

2. Assumptions and Security Objectives

This section presents a set of assumptions and security objectives for the approach described in this document.

2.1. Assumptions

The following assumptions are assumed to be already addressed and are out of the scope of this document.

- o 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 low-power and lossy network (LLN). For instance, in a typical lighting control use case, a single switch is the only entity 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.

- Establishment and management of Security Contexts: 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.

2.2. Security Objectives

The approach described in this document aims at fulfilling the following security objectives:

- Data replay protection: replayed group request messages or response messages MUST be detected.

- Group-level data confidentiality: messages sent within the multicast group SHALL be encrypted if privacy sensitive data is exchanged within the group. In fact, some control commands and/or associated responses could pose unforeseen security and privacy risks to the system users, when sent as plaintext. 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).

- Message integrity: messages sent within the multicast group SHALL 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 ordering: it MUST be possible to determine the ordering of messages coming from a single sender endpoint. In accordance with OSCORE [I-D.ietf-core-object-security], this results in providing relative freshness of group requests and absolute freshness of responses. It is not required to determine ordering of messages from different sender endpoints.

3. OSCORE Security Context

To support multicast communication secured with OSCORE, 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. All the endpoints in the group agree on the same COSE AEAD algorithm. In addition to what is defined in Section 3 of [I-D.ietf-core-object-security], the Common Context includes the following information.

   * Group Identifier (Gid). Variable length byte string identifying the Security Context and used as Master Salt parameter in the derivation of keying material. The Gid is used together with the multicast IP address of the group to retrieve the Security Context, upon receiving a secure multicast request message (see Section 5.2). The Gid associated to a multicast group is determined by the responsible Group Manager. The choice of the Gid for a given group's Security Context is application specific. However, a Gid MUST be random as well as long enough, in order to achieve a negligible probability of collisions between Group Identifiers from different Group Managers. It is the role of the application to specify how to handle possible collisions. An example of specific formatting of the Group Identifier that would follow this specification is given in Appendix B.

   * Counter signature algorithm. Value identifying the algorithm used for source authenticating messages sent within the group, by means of a counter signature (see Section 4.5 of [RFC8152]). Its value is immutable once the Security Context is established. All the endpoints in the group agree on the same counter signature algorithm. The Group Manager MUST define a list of supported signature algorithms as part of the group communication policy. Such a list MUST include the EdDSA signature algorithm ed25519 [RFC8032].

2. one Sender Context, unless the endpoint is configured exclusively as pure listener. The Sender Context is used to secure outgoing messages and is initialized according to Section 3 of [I-D.ietf-core-object-security], once the endpoint has joined the multicast group. In practice, the sender endpoint shares the same symmetric keying material stored in the Sender Context with all the recipient endpoints receiving its outgoing OSCORE messages. The Sender ID in the Sender Context coincides with the Endpoint ID received upon joining the group. It is responsibility of the Group Manager to assign Endpoint IDs to new joining endpoints in such a way that uniqess is ensured within 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 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. In practice, the recipient endpoint shares the symmetric keying material stored in the Recipient Context with the associated other endpoint from which secure messages are received. 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.

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.

It is RECOMMENDED that the Group Manager acts as trusted key repository, and hence is configured to store public keys of group members and provide them to other members of the same group upon request. Possible approaches to provision public keys upon joining the group and to retrieve public keys of group members are discussed in Appendix C.2.

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].

3.1. Management of Group Keying Material

The approach described in this specification should take into account 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. Nevertheless, the adoption of key management schemes for secure revocation and renewal of Security Contexts and group keying material should be considered.

Consistently with the security assumptions in Section 2, it is RECOMMENDED to adopt a group key management scheme, and securely distribute a new value for the Master Secret parameter of the group's Security Context, before a new joining endpoint is added to the group or after a currently present endpoint leaves the group. This is necessary in order to preserve backward security and forward security in the multicast group. The Group Manager responsible for the group is entrusted with such a task.

In particular, the Group Manager MUST distribute also a new Group Identifier (Gid) for that group, together with a new value for the Master Secret parameter. An example of how this can be done is provided in Appendix B. Then, each group member re-derives the keying material stored in its own Sender Context and Recipient Contexts as described in Section 3, using the updated Group Identifier.

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.

4. The COSE Object

When creating a protected CoAP message, an endpoint in the group computes the COSE object using the untagged COSE_Encrypt0 structure [RFC8152] as defined in Section 5 of [I-D.ietf-core-object-security], with the following modifications.

o The value of the "kid" parameter in the "unprotected" field of responses SHALL be set to the Sender ID of the endpoint transmitting the group message.

o The "unprotected" field of the "Headers" field SHALL additionally include the following parameters:

  * gid : its value is set to the Group Identifier (Gid) of the group’s Security Context. This parameter MAY be omitted if the message is a CoAP response.

  * countersign : its value is set to the counter signature of the COSE object (Appendix C.3.3 of [RFC8152]), computed by the endpoint by means of its own private key as described in Section 4.5 of [RFC8152].

In particular, "gid" is included as COSE header parameter as defined in Figure 1.

name	label	value type	value registry	description
gid	TBD	bstr		Identifies the OSCORE group Security Context

Figure 1: Additional common header parameter for the COSE object

o The Additional Authenticated Data (AAD) considered to compute the COSE object is extended, in order to include also the Group Identifier (Gid) of the Security Context used to protect the request message. In particular, the "external_aad" in Section 5.3 of [I-D.ietf-core-object-security] SHALL include also gid as follows:

```
external_aad = [
   version : uint,
   alg : int,
   request_kid : bstr,
   request_piv : bstr,
   gid : bstr,
   options : bstr
]
```

o The OSCORE compression defined in Section 8 of [I-D.ietf-core-object-security] is used, with the following additions for the encoding of the object-security option.

   * The fourth least significant bit of the first byte of the object-security option value SHALL be set to 1, to indicate the presence of the "kid" parameter for both multicast requests and responses.

   * The fifth least significant bit of the first byte MUST be set to 1 for multicast requests, to indicate the presence of the Context Hint in the OSCORE payload. The Context Hint flag MAY be set to 1 for responses.

   * The sixth least significant bit of the first byte is set to 1 if the "countersign" parameter is present, or to 0 otherwise. In order to ensure source authentication of group messages as described in this specification, this bit SHALL be set to 1.

   * The Context Hint value encodes the Group Identifier value (Gid) of the group's Security Context.

* The following q bytes (q given by the counter signature algorithm specified in the Security Context) encode the value of the "countersign" parameter including the counter signature of the COSE object.

* The remaining bytes in the Object-Security value encode the value of the "kid" parameter, which is always present both in multicast requests and in responses.

```
 0 1 2 3 4 5 6 7 <----------- n bytes -----------> <-- 1 byte -->
+-+-+-+-+-+-+-+-+---------------------------------+--------------+
|0 0|1|h|1|  n  |       Partial IV (if any)       |  s (if any)  |
+-+-+-+-+-+-+-+-+---------------------------------+--------------+

<------ s bytes ------> <--------- q bytes --------->
-----------------------+-----------------------------+-----------+
      Gid (if any)     |         countersign         |    kid    |
-----------------------+-----------------------------+-----------+
```

Figure 2: Object-Security Value

5. Message Processing

Each multicast request message and response message is protected and processed as specified in [I-D.ietf-core-object-security], with the modifications described in the following sections.

Furthermore, endpoints in the multicast group locally perform error handling and processing of invalid messages according to the same principles adopted in [I-D.ietf-core-object-security]. However, a receiver endpoint MUST stop processing and silently reject any message which is malformed and does not follow the format specified in Section 4, without sending back any error message. This prevents listener endpoints from sending multiple error messages to a multicaster endpoint, so avoiding the risk of flooding the multicast group.

5.1. 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 - Group Identifier. That is, it SHALL be able to find the correct Security Context used to protect the multicast request and verify the response(s) by using the CoAP Token used in the message exchange.

2. The multicaster computes the COSE object as defined in Section 4 of this specification.

5.2. 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 Group Identifier from the "gid" parameter of the received COSE object. Then, it uses the Group Identifier together with the destination IP address of the multicast request message to identify the correct group's Security Context.

2. The listener endpoint retrieves the Sender ID from the "kid" parameter of the received 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. Then, it verifies the counter signature and decrypts the request message.

5.3. Protecting the Response

A listener endpoint that has received a multicast request message may reply with a secure 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 computes the COSE object as defined in Section 4 of this specification.

5.4. Verifying the Response

Upon receiving a secure 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 retrieves the Security Context by using the Token of the received response message.

2. The multicaster endpoint retrieves the Sender ID from the "kid" parameter of the received 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. Then, it verifies the counter signature and decrypts the response message.

The mapping between response messages from listener endpoints and the associated multicast request message from a multicaster endpoint relies on the 3-tuple (Group ID, Sender ID, Partial IV) associated to the secure multicast request message. This is used by listener endpoints as part of the Additional Authenticated Data when protecting their own response message, as described in Section 4.

6. Synchronization of Sequence Numbers

Upon joining the multicast group, new listeners are not aware of the sequence number values currently used by different multicasters to transmit multicast request messages. This means that, when such listeners receive a secure multicast request from a given multicaster for the first time, they are not able to verify if that request is fresh and has not been replayed. The same applies when a listener endpoint loses synchronization with sequence numbers of multicasters, for instance after a device reboot.

The exact way to address this issue depends on the specific use case and its synchronization requirements. The Group Manager should define also how to handle synchronization of sequence numbers, as part of the policies enforced in the multicast group. In particular, the Group Manager can suggest to single specific listener endpoints how they can exceptionally behave in order to synchronize with sequence numbers of multicasters. Appendix D describes three possible approaches that can be considered.

7. Security Considerations

The same security considerations from OSCORE (Section 11 of [I-D.ietf-core-object-security]) apply to this specification. Additional 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.

8. IANA Considerations

TBD. Header parameter 'gid'.

9. Acknowledgments

The authors sincerely thank Stefan Beck, Rolf Blom, Carsten Bormann, Klaus Hartke, Richard Kelsey, John Mattsson, Jim Schaad and Ludwig Seitz 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 for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-06 (work in progress), October 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://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, <https://www.rfc-editor.org/info/rfc7252>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

10.2. Informative References

[I-D.amsuess-core-repeat-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Repeat And Request-Tag", draft-amsuess-core-repeat-request-tag-00 (work in progress), July 2017.

[I-D.aragon-ace-ipsec-profile]
Aragon, S., Tiloca, M., and S. Raza, "IPsec profile of ACE", draft-aragon-ace-ipsec-profile-00 (work in progress), July 2017.

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-01 (work in progress), July 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-08 (work in progress), October 2017.

[I-D.seitz-ace-oscoap-profile]
Seitz, L., Palombini, F., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-seitz-ace-oscoap-profile-06 (work in progress), October 2017.

[I-D.somaraju-ace-multicast]
Somaraju, A., Kumar, S., Tschofenig, H., and W. Werner, "Security for Low-Latency Group Communication", draft-somaraju-ace-multicast-02 (work in progress), October 2016.

[I-D.tiloca-ace-oscoap-joining]
Tiloca, M. and J. Park, "Joining of OSCOAP multicast groups in ACE", draft-tiloca-ace-oscoap-joining-00 (work in progress), July 2017.

[RFC2093] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification", RFC 2093, DOI 10.17487/RFC2093, July 1997, <https://www.rfc-editor.org/info/rfc2093>.

[RFC2094] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, DOI 10.17487/RFC2094, July 1997, <https://www.rfc-editor.org/info/rfc2094>.

[RFC2627] Wallner, D., Harder, E., and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, DOI 10.17487/RFC2627, June 1999, <https://www.rfc-editor.org/info/rfc2627>.

[RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, DOI 10.17487/RFC3376, October 2002, <https://www.rfc-editor.org/info/rfc3376>.

[RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, DOI 10.17487/RFC3740, March 2004, <https://www.rfc-editor.org/info/rfc3740>.

[RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <https://www.rfc-editor.org/info/rfc3810>.

[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, <https://www.rfc-editor.org/info/rfc4046>.

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

[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, <https://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, <https://www.rfc-editor.org/info/rfc4944>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7390>.

Appendix A. 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. This 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.

- 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.

Appendix B. Example of Group Identifier Format

This section provides an example of how the Group Identifier (Gid) can be specifically formatted. That is, the Gid can be composed of two parts, namely a Group Prefix and a Group Epoch.

The Group Prefix is uniquely defined in the set of all the multicast groups associated to the same Group Manager. The choice of the Group Prefix for a given group’s Security Context is application specific. Group Prefixes are random as well as long enough, in order to achieve a negligible probability of collisions between Group Identifiers from different Group Managers.

The Group Epoch is set to 0 upon the group’s initialization, and is incremented by 1 upon completing each renewal of the Security Context and keying material in the group (see Section 3.1). In particular, once a new Master Secret has been distributed to the group, all the group members increment by 1 the Group Epoch in the Group Identifier of that group (see Section 3).

Appendix C. Set-up of New Endpoints

An endpoint joins a multicast group by explicitly interacting with the responsible Group Manager. All communications between a joining endpoint and the Group Manager rely on the CoAP protocol and MUST be secured. Specific details on how to secure communications between joining endpoints and a Group Manager are out of the scope of this specification.

In order to receive multicast messages sent to the group, a joining endpoint has to register with a network router device [RFC3376][RFC3810], signaling its intent to receive packets sent to the multicast IP address of that group. As a particular case, the Group Manager can also act as such a network router device. Upon joining the group, endpoints are not required to know how many and what endpoints are active in the same group.

Furthermore, in order to participate in the secure group communication, an endpoint needs to maintain a number of information elements stored in its own Security Context (see Section 3). The following Appendix C.1 describes which of this information is provided to an endpoint upon joining a multicast group through the responsible Group Manager.

C.1. Join Process

An endpoint requests to join a multicast group by sending a confirmable CoAP POST request to the Group Manager responsible for that group. The join request is addressed to a CoAP resource associated to that group and carries the following information.

- Role: the exact role of the joining endpoint in the multicast group. Possible values are: "multicaster", "listener", "pure listener", "multicaster and listener", or "multicaster and pure listener".

- Identity credentials: information elements to enforce source authentication of group messages from the joining endpoint, such as its public key. The exact content depends on whether the Group Manager is configured to store the public keys of group members. If this is the case, this information is omitted if it has been provided to the same Group Manager upon previously joining the same or a different multicast group under its control. This information is also omitted if the joining endpoint is configured exclusively as pure listener for the joined group. Appendix C.2 discusses additional details on provisioning of public keys and other information to enforce source authentication of joining node's messages.

- o Retrieval flag: indication of interest to receive the public keys of the endpoints currently in the multicast group, as included in the following join response. This flag MUST be set to false if the Group Manager is not configured to store the public keys of group members, or if the joining endpoint is configured exclusively as pure listener for the joined group.

The Group Manager MUST be able to verify that the joining enpoint is authorized to become a member of the multicast group. To this end, the Group Manager can directly authorize the joining endpoint, or expect it to provide authorization evidence previously obtained from a trusted entity. Appendix C.3 describes how this can be achieved by leveraging the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

In case of successful authorization check, the Group Manager generates an Endpoint ID assigned to the joining node, before proceeding with the rest of the join process. Instead, in case the authorization check fails, the Group Manager MUST abort the join process. Further details about the authorization of joining endpoint are out of the scope of this specification.

As discussed in Section 3.1, it is then RECOMMENDED that the Security Context is renewed before the joining endpoint becomes a new active member of the multicast group. This is achieved by securely distributing a new Master Secret and a new Group Identifier to the endpoints currently present in the same group.

Once renewed the Security Context in the multicast group, the Group Manager replies to the joining endpoint with a CoAP response carrying the following information.

- o Security Common Context: the OSCORE Security Common Context associated to the joined multicast group (see Section 3).

- o Endpoint ID: the Endpoint ID associated to the joining node. This information is not included in case "Role" in the join request is equal to "pure listener".

- o Management keying material: the set of administrative keying material used to participate in the group rekeying process run by the Group Manager (see Section 3.1). The specific elements of this management keying material depend on the group rekeying protocol used in the group. For instance, this can simply consist in a group key encryption key and a pairwise symmetric key shared between the joining node and the Group Manager, in case GKMP [RFC2093][RFC2094] is used. Instead, if key-tree based rekeying protocols like LKH [RFC2627] are used, it can consist in the set of symmetric keys associated to the key-tree leaf representing the group member up to the key-tree root representing the group key encryption key.

- o Member public keys: the public keys of the endpoints currently present in the multicast group. This includes: the public keys of the non-pure listeners currently in the group, if the joining endpoint is configured (also) as multicaster; and the public keys of the multicasters currently in the group, if the joining endpoint is configured (also) as listener or pure listener. This information is omitted in case the Group Manager is not configured to store the public keys of group members or if the "Retrieval flag" was set to false in the join request. Appendix C.2 discusses additional details on provisioning public keys upon joining the group and on retrieving public keys of group members.

C.2. Provisioning and Retrieval of Public Keys

As mentioned in Section 3, it is RECOMMENDED that the Group Manager acts as trusted key repository, stores public keys of group members and provide them to other members of the same group upon request. In such a case, a joining endpoint provides its own public key to the Group Manager, as "Identity credentials" of the join request, when joining the multicast group (see Appendix C.1).

After that, the Group Manager MUST verify that the joining endpoint actually owns the associated private key, for instance by performing a proof-of-possession challenge-response. In case of success, the Group Manager stores the received public key as associated to the joining endpoint and its Endpoint ID, before sending the join response and continuing with the rest of the join process. From then on, that public key will be available for secure and trusted delivery to other endpoints in the multicast group.

The joining node does not have to provide its own public key if that already occurred upon previously joining the same or a different multicast group under the same Group Manager. However, separately for each multicast group under its control, the Group Manager maintains an updated list of active Endpoint IDs associated to a same endpoint’s public key.

Instead, in case the Group Manager does not act as trusted key repository, the following information is exchanged with the Group Manager during the join process.

1. The joining endpoint signs its own certificate by using its own private key. There is no restriction on the Certificate Subject included in the joining node's certificate.

2. The joining endpoint includes the following information as "Identity credentials" in the join request (Appendix C.1): the signed certificate; and the identifier of the Certification Authority that issued the certificate. The joining endpoint can optionally specify also a list of public key repositories storing its own certificate.

3. When processing the join request, the Group Manager first validates the certificate by verifying the signature of the issuer CA, and then verifies the signature of the joining node.

4. The Group Manager stores the association between the Certificate Subject of the joining node's certificate and the pair {Group ID, Endpoint ID of the joining node}. If received from the joining endpoint, the Group Manager also stores the list of public key repositories storing the certificate of the joining endpoint.

When a group member X wants to retrieve the public key of another group member Y in the same multicast group, the endpoint X proceeds as follows.

1. The endpoint X contacts the Group Manager, specifying the pair {Group ID, Endpoint ID of the endpoint Y}.

2. The Group Manager provides the endpoint X with the Certificate Subject CS from the certificate of endpoint Y. If available, the Group Manager provides the endpoint X also with the list of public key repositories storing the certificate of the endpoint Y.

3. The endpoint X retrieves the certificate of the endpoint X from a key repository storing it, by using the Certificate Subject CS.

C.3. Group Joining Based on the ACE Framework

The join process to register an endpoint as a new member of a multicast group can be based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz], built on re-use of OAuth 2.0 [RFC6749].

In particular, the approach described in [I-D.tiloca-ace-oscoap-joining] uses the ACE framework to delegate the authentication and authorization of joining endpoints to an Authorization Server in a trust relation with the Group Manager. At the same time, it allows a joining endpoint to establish a secure channel with the Group Manager, by leveraging protocol-specific profiles of ACE [I-D.seitz-ace-oscoap-profile][I-D.ietf-ace-dtls-authorize][I-D.aragon-ace-ipsec-profile] to achieve communication security, proof-of-possession and server authentication.

More specifically and with reference to the terminology defined in OAuth 2.0:

- The joining endpoint acts as Client;
- The Group Manager acts as Resource Server, with different CoAP resources for different multicast groups it is responsible for;
- An Authorization Server enables and enforces authorized access of the joining endpoint to the Group Manager and its CoAP resources paired with multicast groups to join.

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, such as OSCORE, DTLS [RFC6347] or IPsec [RFC4301].

Appendix D. Examples of Synchronization Approaches

This section describes three possible approaches that can be considered by listener endpoints to synchronize with sequence numbers of multicasters.

D.1. Best-Effort Synchronization

Upon receiving a multicast request from a multicaster, a listener endpoint does not take any action to synchonize with the sequence number of that multicaster. This provides no assurance at all as to message freshness, which can be acceptable in non-critical use cases.

D.2. Baseline Synchronization

Upon receiving a multicast request from a given multicaster for the first time, a listener endpoint initializes its last-seen sequence number in its Recipient Context associated to that multicaster. However, the listener drops the multicast request without delivering it to the application layer. This provides a reference point to identify if future multicast requests from the same multicaster are fresher than the last one received.

A replay time interval exists, between when a possibly replayed message is originally transmitted by a given multicaster and the first authentic fresh message from that same multicaster is received. This can be acceptable for use cases where listener endpoints admit such a trade-off between performance and assurance of message freshness.

D.3. Challenge-Response Synchronization

A listener endpoint performs a challenge-response exchange with a multicaster, by using the Repeat Option for CoAP described in Section 2 of [I-D.amsuess-core-repeat-request-tag].

That is, upon receiving a multicast request from a particular multicaster for the first time, the listener processes the message as described in Section 5.2 of this specification, but, even if valid, does not deliver it to the application. Instead, the listener replies to the multicaster with a 4.03 Forbidden response message including a Repeat Option, and stores the option value included therein.

Upon receiving a 4.03 Forbidden response that includes a Repeat Option and originates from a verified group member, a multicaster MUST send a group request as a unicast message addressed to the same listener, echoing the Repeat Option value. In particular, the multicaster does not necessarily resend the same group request, but can instead send a more recent one, if the application permits it. This makes it possible for the multicaster to not retain previously sent group requests for full retransmission, unless the application explicitly requires otherwise. In either case, the multicaster uses the sequence number value currently stored in its own Sender Context. If the multicaster stores group requests for possible retransmission with the Repeat Option, it should not store a given request for longer than a pre-configured time interval. Note that the unicast request echoing the Repeat Option is correctly treated and processed as a group message, since the "gid" field including the Group Identifier of the OSCORE group is still present in the Object-Security Option as part of the COSE object (see Section 4).

Upon receiving the unicast group request including the Repeat Option, the listener verifies that the option value equals the stored and previously sent value; otherwise, the request is silently discarded. Then, the listener verifies that the unicast group request has been received within a pre-configured time interval, as described in [I-D.amsuess-core-repeat-request-tag]. In such a case, the request is further processed and verified; otherwise, it is silently discarded. Finally, the listener updates the Recipient Context associated to that multicaster, by setting the Replay Window according to the Sequence Number from the unicast group request conveying the Repeat Option. The listener either delivers the request to the application if it is an actual retransmission of the original one, or discard it otherwise. Mechanisms to signal whether the resent request is a full retransmission of the original one are out of the scope of this specification.

In case it does not receive a valid group request including the Repeat Option within the configured time interval, the listener node SHOULD perform the same challenge-response upon receiving the next multicast request from that same multicaster.

A listener SHOULD NOT deliver group request messages from a given multicaster to the application until one valid group request from that same multicaster has been verified as fresh, as conveying an echoed Repeat Option [I-D.amsuess-core-repeat-request-tag]. Also, a listener MAY perform the challenge-response described above at any time, if synchronization with sequence numbers of multicasters is (believed to be) lost, for instance after a device reboot. It is the role of the application to define under what circumstances sequence numbers lose synchronization. This can include a minimum gap between the sequence number of the latest accepted group request from a multicaster and the sequence number of a group request just received from the same multicaster. A multicaster MUST always be ready to perform the challenge-response based on the Repeat Option in case a listener starts it.

Note that endpoints configured as pure listeners are not able to perform the challenge-response described above, as they do not store a Sender Context to secure the 4.03 Forbidden response to the multicaster. Therefore, pure listeners should adopt alternative approaches to achieve and maintain synchronization with sequence numbers of multicasters.

This approach provides an assurance of absolute message freshness. However, it can result in an impact on performance which is undesirable or unbearable, especially in large multicast groups where many nodes at the same time might join as new members or lose synchronization.

Appendix E. No Verification of Signatures

There are some application scenarios using group communications that have particularly strict requirements. One example of this is the requirement of low message latency in non-emergency lighting applications [I-D.somaraju-ace-multicast]. For those applications which have tight performance constraints and relaxed security requirements, it can be inconvenient for some endpoints to verify digital signatures in order to assert source authenticity of received group messages. In other cases, the signature verification can be deferred or only checked for specific actions. For instance, a command to turn a bulb on where the bulb is already on does not need the signature to be checked. In such situations, the counter signature needs to be included anyway as part of the group message, so that an endpoint that needs to validate the signature for any reason has the ability to do so.

In this specification, it is NOT RECOMMENDED that endpoints do not verify the counter signature of received group messages. However, it is recognized that there may be situations where it is not always required. The consequence of not doing the signature validation is that security in the group is based only on the group-authenticity of the shared keying material used for encryption. That is, endpoints in the multicast group have evidence that a received message has been originated by a group member, although not specifically identifiable in a secure way. This can violate a number of security requirements, as the compromise of any element in the group means that the attacker has the ability to control the entire group. Even worse, the group may not be limited in scope, and hence the same keying material might be used not only for light bulbs but for locks as well. Therefore, extreme care must be taken in situations where the security requirements are relaxed, so that deployment of the system will always be done safely.

Authors' Addresses

Marco Tiloca
RISE SICS AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Goeran Selander
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

Network Working Group A. Minaburo
Internet-Draft Acklio
Intended status: Informational L. Toutain
Expires: April 29, 2018 Institut Mines Telecom Atlantique
October 26, 2017

CoAP Time Scale Option
draft-toutain-core-time-scale-00

Abstract

SCHC compression mechanism for LPWAN network enables IPv6 on devices connected to a constrained network (LPWAN). They can communicate with a CoAP server located anywhere in the Internet. LPWAN network characteristics limits the number of exchanges and may impose a long RTT. The CoAP server must be aware of these properties to manage correctly requests. The Time Scale option allows a device to inform a CoAP server of the duration the message ID value should be kept in memory to manage correctly message duplication.

Status of This Memo

Copyright Notice

1. CoAP Message ID

Constraint Application Protocol (CoAP) [RFC7252] implements a simple reliable transport mechanism based on ARQ. Each CoAP message contains a 16 bit Message ID (noted afterward MID). A client selects a MID in a CON message and expects an ACK message containing the same MID value. A timer makes the client resend the request if no ACK is received during a pre-defined period.

To avoid a second process of duplicated requests by the server, a list of messages ID already acknowledged must be maintained for a period of time. If the message ID is already in the list, the message is just acknowledged and not processed by upper layer. Therefore, the client cannot use this MID value in another request during the same period of time.

```
               client                              server
                 |           CON MID = XXXX          |
                 |---------------------------------->|=> process
    EXCHANGE   ^ |           ACK MID = XXXX          |  ^
    LIFETIME   | |                  <----------------|  | EXCHANGE
               | |                                   |  | LIFETIME
     XXXX      | |           CON MID = XXXX          |  |
     cannot    | |---------------------------------->|  | XXXX in
     be reused | |           ACK MID = XXXX          |  | memory
               | |<----------------------------------|  |
               | |                                   |  |
               v |                                   |  v
    XXXX can be  .                                   .
    reused       .                                   .
                 .                                   .
                 |           CON MID = XXXX          |
                 |---------------------------------->|=> process
               ^ |           ACK MID = XXXX          |  ^
               | |<----------------------------------|  |
                 .                                   .
```

Figure 1: Delayed transmission.

[RFC7252] calls the period a MID is assigned to a request the EXCHANGE_LIFETIME. The value is based on the worst case scenario taking into account the propagation time, the number of retransmissions and the processing time. The default value for EXCHANGE_LIFETIME is set to 247 seconds for MAX_RTT of 202 seconds.

2. LPWAN networks

Low Power Wide Area Network (LPWAN) family regroups networks dedicated to the Internet of Things. They provide a large coverage with a limited energy consumption. They mostly use the license-free ISM band. The [I-D.ietf-lpwan-overview] gives an overview of the technology and the star oriented topology architecture. A Network Gateway (NGW) is at the interconnection between the LPWAN and the Internet network.

To ensure fairness among nodes, regulation imposes a duty cycle. In practice, with a 1% duty cycle, a node sending a message of s seconds must wait 99 x s seconds before sending another message. For instance, in some technologies sending a 50 bytes message takes 2 seconds, forcing a silence of 198 seconds.

The device sleeps most of the time to preserve energy. If a device can use the uplink channel at any time, downlink channel is generally available during a short receiving window following the message emission. Therefore a message sent to a device out of this receiving window will be lost. Network Gateways are aware of this restriction and buffers downlink messages until an uplink message is received which opens the receiving window.

Figure 2 illustrates this. A CoAP client sends a request every hour. Even if the server replies immediately, the answer may be buffered by the Network GW until an new uplink message is sent. In that case, the client will only receive the answer after one hour when the next request is sent. The RTT is influenced by the message periodicity and the EXCHANGE_LIFTEIME value can be computed locally by client to dimension its timers.

```
        client on LPWAN         NGW              server
               |                 |                 |
           ^   |---------------------------------->| CON MID = 1
           |   |                 |                 |
    1 hour |   |         delayed H<----------------| ACK MID = 1
           |   |                 H                 |
           |   |                 H                 |
           |   |                 H                 |
           v   |---------------------------------->| CON MID = 2
   ACK MID = 1 |<----------------|                 |
               |                 |                 |
               v                 v                 v
```

Figure 2: Delayed transmission.

The server should remain as generic as possible and EXCHANGE_LIFETIME parameter has to be adapted to the client behavior. If the period is too large, the server will have to memorize a longer list of MID for fast responding client. On the other hand, if the EXCHANGE_LIFETIME is too short, this leads to misbehaviors as shown in Figure 3, a retransmission will be viewed as a new request.

```
        client on LPWAN         NGW              server
               |   CON MID = 1   |                 |
    Timer  -   |---------------------------------->| => process
           |   |         delayed H<----------------| ^
           |   |                 H   ACK MID = 1   | | EXCHANGE
           |   |                 H                 | v LIFETIME
           |   |   CON MID = 2   H                 |
           |   |---------------------------------->|
           |   |       X---------|                 |
           |   |                 |                 |
           |   |   CON MID = 1   |                 |
   Expire  O   |---------------------------------->| => process
               |                 |<----------------|
               .                 .   ACK MID = 1   .
               .                 .                 .
```

Figure 3: Retransmission.

The Time Scale option, added into all the CoAP requests, informs the server of the duration a message ID should be memorized into the server and therefore the duration during which a client should not reuse the same message ID for a new request. This way, the server can adapt its behavior to different environments.

It is important to notice that this option will not contribute to an DoS attack. This option does not increase the number of message ID memorized by the server. In fact, the Time Scale option can be viewed as a contract between the client and the server, which means that the client will send a reasonable number of request during that period. The number of memorized message ID is independent of the duration of the exchange but linked to the number a simultaneous request a client can send. If a client is sending a number of request larger than expected, they can be easily discarded by the server.

3. Timescale Option

Timescale is a new CoAP option that tells the server how many seconds the MID should be memorized by the server. This option must be included in all the exchanges coming from a high latency device.

Number	C	U	N	R	Name	Format	Length	Default
259	X				Time Scale	uint	1-4	3600

Figure 4: Time Scale Option.

This option is critical, if a server does not recognize it, it must inform the client that EXCHANGE_LIFETIME cannot be modified. The option is Safe-to-forward so a proxy does not have to understand this option, since only the server is concerned with the MID management. The value (in seconds) contains the new EXCHANGE_LIFETIME set by the server for this request. If the value is smaller than the default value, this option is discarded and the client receives an error message.

4. Normative References

[I-D.ietf-lpwan-overview]
Farrell, S., "LPWAN Overview", draft-ietf-lpwan-overview-07 (work in progress), October 2017.

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

Authors' Addresses

Ana Minaburo
Acklio
2bis rue de la Chataigneraie
35510 Cesson-Sevigne Cedex
France

Email: ana@ackl.io

Laurent Toutain
Institut Mines Telecom Atlantique
2 rue de la Chataigneraie
CS 17607
35576 Cesson-Sevigne Cedex
France

Email: Laurent.Toutain@imt-atlantique.fr

ACE S. Kumar
Internet-Draft Philips Lighting Research
Intended status: Standards Track P. van der Stok
Expires: December 14, 2017 Consultant
P. Kampanakis
Cisco Systems
M. Furuhed
Nexus Group
S. Raza
RISE SICS
June 12, 2017

EST over secure CoAP (EST-coaps)
draft-vanderstok-ace-coap-est-02

Abstract

Low-resource devices in a Low-power and Lossy Network (LLN) can operate in a mesh network using the IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) and IEEE 802.15.4 link-layer standards. Provisioning these devices in a secure manner with keys (often called secure bootstrapping) used to encrypt and authenticate messages, is the subject of Bootstrapping of Remote Secure Key Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra] and 6tisch Secure Join [I-D.ietf-6tisch-dtsecurity-secure-join]. Enrollment over Secure Transport (EST) [RFC7030], based on TLS and HTTP, is used in BRSKI. Low-resource devices often use the lightweight Constrained Application Protocol (CoAP) [RFC7252] for message exchanges. This document defines how low-resource devices are expected to use EST over secure CoAP (EST-coaps) for secure bootstrapping and certificate enrollment. 6LoWPAN fragmentation management and extensions to CoAP registries are needed to enable EST-coaps.

Status of This Memo

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 14, 2017.

Copyright Notice

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 industry 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). [I-D.ietf-6tisch-minimal-security] and [I-D.ietf-6tisch-dtsecurity-secure-join] also address secure bootstrapping in the 6tisch context targeted to low-resource devices. BRSKI has not been developed specifically for low-resource devices in constrained networks. Constrained 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. EST-coaps provides a subset of EST functionality and extends EST with BRSKI functions. EST-coaps replaces the invocations of TLS and HTTP by DTLS and CoAP invocations thus enabling EST and BRSKI for CoAP-based low-resource devices.

Although EST-coaps paves the way for the utilization of EST for constrained devices on constrained networks, some devices will not have enough resources to handle the large payloads that come with

EST-coaps. The specification of EST-coaps is intended to ensure that bootstrapping works for less constrained devices that choose to limit their communications stack to UDP/CoAP. It is up to the network designer to decide which devices execute the EST protocol and which not.

EST-coaps is designed for use in professional control networks such as Building Control. 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 HTTP Basic 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. Therefore, full PKI request messages are not supported by EST-coaps.

Because the relatively large EST messages cannot be readily transported over constrained (6LoWPAN, LLN) wireless networks, this document specifies the use of CoAP Block-Wise Transfer ("Block") [RFC7959] to fragment EST messages at the application layer.

Support for Observe CoAP options [RFC7641] 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.

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].

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.

The following terms are defined in the BRSKI protocol [I-D.ietf-anima-bootstrapping-keyinfra]: pledge, Join proxy (or Circuit Proxy?), Join Registrar, and Manufacturer Authorized Signing Authorities (MASA).

2. EST operational differences

Only the differences to EST with respect to operational scenarios are described in this section. EST-coaps server differs from EST server as follows:

- o Replacement of TLS by DTLS and HTTP by CoAP, resulting in:

  * DTLS-secured CoAP sessions between EST-coaps client and EST-coaps server.

- o Only certificate-based client authentication is supported, which results in:

  * The EST-coaps client does not support HTTP Basic authentication (as described in Section 3.2.3 of [RFC7030]).

  * The EST-coaps client does not support authentication at the application layer (as described in Section 3.2.3 of [RFC7030]).

- o EST-coaps does not support full PKI request messages[RFC5272].

  * Consequently, the fullcmc request of section 4.3 of [RFC7030] and response MUST NOT be supported by EST-coaps].

- o EST-coaps specifies the BRSKI extensions over CoAP as specified in sections 3.2, 3.4, 3.5, and 3.8.4 of [I-D.ietf-anima-bootstrapping-keyinfra].

3. Conformance to RFC7925 profiles

This section shows how EST-coaps fits into the profiles of low-resource devices as described in [RFC7925]. Within the bootstrap context a Public Key Infrastructure (PKI) is used, where the client is called "pledge", the Registration Authority (RA) is called Join Registrar, which acts at the front-end for the Certificate Authority (CA) and receives voucher feedback from as many Manufacturer Authorized Signing Authorities (MASA) as there are manufacturers. A Join Proxy (Circuit Proxy?)is placed between client and RA to receive join requests over a 1-hop unsecured channel and transmitted over the secure network to the EST-server. The EST-server of EST-coaps is placed between Join-Proxy (Circuit Proxy) and RA or is part of RA.

EST-coaps can transport certificates and private keys. Private keys can be transported as response to a request to a server-side key generation as described in section 4.4 of [RFC7030]. In the bootstrapping context, EST-coaps transport is limited to the EST certificate transport conformant to section 4.4 of [RFC7925]. For

BRSKI, outside the profiles of [RFC7925], EST-coaps transports vouchers, which are YANG files specified in [I-D.ietf-anima-voucher].

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.

The EST-coaps client MUST be configured with an explicit TA database or at least an implicit TA database from its manufacturer. 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

- o DTLS with a previously issued client certificate (e.g., an existing certificate issued by the EST CA); this could be a common case for simple re-enrollment of clients;

- o DTLS with a previously installed certificate (e.g., manufacturer-installed certificate or a certificate issued by some other party);

4. Protocol Design and Layering

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" for the transfer of larger EST messages is specified in Section 4.5. The Figure 1 below shows the layered EST-coaps architecture.

```
+------------------------------------------------+
|    EST request/response messages               |
+------------------------------------------------+
|    CoAP for message transfer and signaling     |
+------------------------------------------------+
|    DTLS for transport security                 |
+------------------------------------------------+
|    UDP for transport                           |
+------------------------------------------------+
```

Figure 1: EST-coaps protocol layers

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 identified by their message types:

- o Simple enroll and reenroll, for CA to sign public client-identity key.
- o CA certificate retrieval, needed to receive the complete set of CA certificates.
- o CSR Attributes request messages, informs the pledge of the fields to include in generated CSR.
- o Server-side key generation messages, to provide a private client-identity key when the client is too restricted or because of lack of an entropy source. [Encrypting these keys is important. RFC7030 specifies how the private key can be encrypted with CMS using symmetric or asymmetric keys.]

4.1. Discovery and URI

EST-coaps is targeted to low-resource networks with small packets. Saving header space is important and the EST-coaps URI is shorter than the EST URI.

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 "ace.est" [RFC6690]. Upon success, the return payload will contain the root resource of the EST resources. It is up to the implementation to choose its root resource; throughout this document the example root resource /est is used. The example below shows the discovery of the presence and location of management data.

```
  REQ: GET /.well-known/core?rt=ace.est

  RES: 2.05 Content
</est>; rt="ace.est"
```

The EST-coaps server URIs differ from the EST URI by replacing the scheme https by coaps and by specifying shorter resource path names:

coaps://www.example.com/est/short-name

Figure 5 in section 3.2.2 of [RFC7030] enumerates the operations and corresponding paths which are supported by EST. Table 1 provides the mapping from the EST and BRSKI URI path to the EST-coaps URI path.

BRSKI	EST	EST-coaps
	/cacerts	/crts
	/simpleenroll	/sen
	/simplereenroll	/sren
	/csrattrs	/att
	/serverkeygen	/skg
/requestvoucher		/rv
/voucher_status		/vs
/enrollstatus		/es

Table 1

/requestvoucher and /enrollstatus are needed between pledge and Registrar.

When discovering the root path for the EST resources, the server MAY return the full resource paths and the used content types. This is useful when multiple content types are specified for EST-coaps server. For example, the following more complete response is possible.

```
  REQ: GET /.well-known/core?rt=ace.est

  RES: 2.05 Content
</est>; rt="ace.est"
</est/crts>; rt="ace.est";ct=TBD1
</est/sen>; rt="ace.est";ct=TBD1 TBD4
</est/sren>; rt="ace.est";ct=TBD1 TBD4
</est/att>; rt="ace.est";ct=TBD4
</est/skg>; rt="ace.est";ct=TBD1 TBD4 TBD2
</est/rv>; rt="ace.est";ct=TBD5 TBD6
</est/vs>; rt="ace.est";ct=50
</est/es>; rt="ace.est";ct=50
```

ct=50 stands for the Content-Format "application/json"

The return of the content-types allows the client to choose the most appropriate one from multiple content types.

4.2. Payload format

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 for coap MUST map to an allowed combination of path-suffix and media type as defined for EST. The required content-formats for these request and response messages are defined in Section 8. The CoAP response codes are defined in Section 4.4.

EST-coaps is designed for use between low-resource devices using CoAP and hence does not need to send base64-encoded data. Simple binary is more efficient (30% less payload compared to base64) and well supported by CoAP. Therefore, the content formats specification in Section 8 requires the use of binary for all EST-coaps Content-Formats.

4.3. Message Bindings

This section describes BRSKI to CoAP message mappings.

All /crts, /sen, /sren, /att, /skg, /rv, /vs, and /es EST-coaps messages expect a response, so they are all CoAP CON messages.

The Ver, TKL, Token, and Message ID values of the CoAP header are not influenced by EST.

CoAP options are used to convey Uri-Host, Uri-Path, Uri-Port, Content-Format and more in CoAP. The CoAP Options are used to communicate the HTTP fields specified in the BRSKI REST messages.

BRSKI URLs are HTTPS based (https://), in CoAP these will be assumed to be transformed to coaps (coaps://)

Appendix A includes some practical examples of EST messages translated to CoAP.

4.4. 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 to CoAP 2.06 as specified in [I-D.hartke-core-pending]. All 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.

4.5. Message fragmentation

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". 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.

That is not always possible. Even though ECC certificates are small in size, they can vary greatly based on signature algorithms, key sizes, and OID fields used. 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 cacerts response from the server can include multiple certs that amount to large payloads. CoAP [RFC7252]'s 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, even with ECC certs, EST-coaps messages can still exceed sizes in MTU of 1280 for IPv6 or 60-80 bytes for 6LoWPAN [RFC4919] as explained in section 2 of [RFC7959]. EST-coaps needs to be able to fragment EST messages into multiple DTLS datagrams. Fine-grained fragmentation of EST messages is essential.

To perform fragmentation in CoAP, [RFC7959] specifies the "Block1" option for fragmentation of the request payload and the "Block2" option for fragmentation of the return payload of a CoAP flow.

The BLOCK draft defines SZX in the Block1 and Block2 option fields. These are used to convey the size of the blocks in the requests or responses.

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. As explained in Section 1 of [RFC7959]), blockwise transfers SHOULD be used in Confirmable CoAP messages to avoid the exacerbation of lost blocks.

The Size1 response MAY be parsed by the client as a 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.

Examples of fragmented messages are shown in Appendix B.

5. Transport Protocol

EST-coaps depends on a secure transport mechanism over UDP that can secure (confidentiality, authenticity) the CoAP messages exchanged.

5.1. DTLS

DTLS is one such secure protocol. Within BRSKI and EST when "TLS" is referred to, it is understood that in EST-coaps, security is provided using DTLS instead. No other changes are necessary (all provisional modes etc. are the same as for TLS).

CoAP was designed to avoid fragmentation. DTLS is used to secure CoAP messages. However, fragmentation is still possible at the DTLS layer during the DTLS handshake when using ECC ciphersuites. 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].

CoAP and DTLS can provide proof of identity for EST-coaps clients and server with simple PKI messages 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].

Channel-binding information for linking proof-of-identity with connection-based proof-of-possession is optional for EST-coaps. When proof-of-possession is desired, a set of actions are required regarding the use of tls-unique, described in section 3.5 in [RFC7030]. The tls-unique information translates to the contents of the first "Finished" message in the TLS handshake between server and client [RFC5929]. The client is then supposed to add this "Finished" message as a ChallengePassword in the attributes section of the PKCS#10 Request Info to prove that the client is indeed in control of the private key at the time of the TLS session when performing a /simpleenroll, for example. In the case of EST-coaps, the same operations can be performed during the DTLS handshake. In the event of handshake message fragmentation, the Hash of the handshake messages used in the MAC calculation of the Finished message

```
PRF(master_secret, finished_label, Hash(handshake_messages))
   [0..verify_data_length-1];
```

MUST be computed as if each handshake message had been sent as a single fragment [RFC6347].

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 a simpleenroll request can use the same authenticated DTLS connection. Given that after a successful 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.

5.2. 6tisch approach

The 6tisch bootstrapping is targeted to the "imprinting" of the "pledge" with layer 2 keys. The content formats for the transport are being defined and may be expressed in a YANG module.

Instead of using transport security, the 6tisch approach relies on application security provided by OSCOAP [I-D.ietf-core-object-security] and EDHOC [I-D.selander-ace-cose-ecdhe]. [I-D.selander-ace-eals] uses OSCOAP to securely enroll certificates by using Certificate Management over CMS (CMC) (EST is profile of CMC).

It is suggested that the EST-coaps communication between pledge and registrar, specified in this document, can be freely exchanged with the same communication specified in [I-D.ietf-6tisch-dtsecurity-secure-join] and [I-D.ietf-6tisch-minimal-security].

[EDNOTE: The evolution of this section depends on the directions taken by 6tisch and anima and the possible commonality that will be provided.]

6. Proxying

In real-world deployments, entities like the EST server, CA or MASA will not always reside within the COAP boundary. The MASA or a CA can exist outside the constrained network in a non-constrained network that supports TLS/HTTP. In such environments EST-coaps is used by the pledge within the COAP boundary and TLS is used to transport the EST/BRSKI messages outside the CoAP boundary. A proxy entity at the edge is required to operate between the COAP environment and the external HTTP network. The ESTcoaps-to-HTTPS proxy SHOULD terminate EST-coaps downstream and initiate EST/BRSKI connections over TLS upstream.

Two separate use-cases, shown in one figure below, are expected to be deployed in practice:

- o A proxy between any EST-client and EST-server indepedent of BRSKI
- o A proxy between Registrar and MASA

```
                                        Constrained Network
                                    .----------------------------.
                                    |.--------------------------.|
                                    ||                          ||
   .------.  HTTP   .-----------------.  COAPS  .-----------.   ||
   | MASA |<------->|ESTcoaps-to-HTTPS|<------->| Registrar |   ||
   |Server|over TLS |      Proxy       |         '-----------'   ||
   '------'         '-----------------'              /|\        ||
                                    ||         COAPS  |         ||
                                    ||           .----'-----.   ||
                                    ||           |Join proxy|   ||
                                    ||           '----------'   ||
                                    ||                /|\       ||
                                    ||          COAPS  |        ||
                                    ||             .---'----.   ||
                                    ||             | Pledge |   ||
                                    ||             '--------'   ||
   .------.  HTTP   .-----------------.  COAPS   .-----------.  ||
   | EST  |<------->|ESTcoaps-to-HTTPS|<-------->| EST Client|  ||
   |Server|over TLS |      Proxy       |          '-----------'  ||
   '------'         '-----------------'                         ||
                                    ||                          ||
                                    |'--------------------------'|
                                    '----------------------------'
```

ESTcoaps-to-HTTPS proxy at the COAP boundary.

Table 1 contains the mapping between the EST-coaps and EST/BRSKI URIs the proxy SHOULD adhere to. Section 7 of [RFC8075] and Section 4.4 define the mapping between EST-coaps and HTTP response codes, that determines how a proxy translates COAP response codes from/to HTTP status codes. The mapping from Content-Type to media type is defined in Section 8. The conversion from binary to BSD64 needs to be done in the proxy. Conversion is possible because a TLS link exists between EST-coaps-to-HTTP proxy and HTTP MASA or EST server and a corresponding DTLS linked exists between EST-coaps-to-HTTP proxy and EST client or Registrar.

Due to fragmentation of large messages into blocks, an EST-coaps-to-HTTP proxy SHOULD reassemble the BLOCKs before translating the binary content to BSD64, and consecutively relay the message upstream into the HTTP environment.

For the discovery of the EST server by the EST client in the coap environment, the EST-coaps-to-HTTP proxy MUST announce itself according to the rules of Section 4.1. The available functions of the proxies MUST be announced with as many resource paths. The discovery of MASA and EST server in the http environment follow the rules specified in [I-D.ietf-anima-bootstrapping-keyinfra].

[EDNOTE: PoP will be addressed here.]

A proxy SHOULD authenticate the client downstream and it should be authenticated by the EST or BRSKI server or CA upstream. A trust relationship needs to be pre-established between the proxy and the TCP entities (EST, BRSKI servers) to be able to proxy these connections on behalf of various clients.

[EDNOTE: To add more details about trust relations in this section.]

7. Parameters

[EDNOTE: 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 a CoAP ACK every 2s while processing.]

8. IANA Considerations

8.1. Content-Format registry

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

*

  + application/voucherrequest

  + Type name: application

  + Subtype name: voucherrequest

  + ID: TBD5

  + Required parameters: None

  + Optional parameters: None

  + Encoding considerations: binary

  + Security considerations: As defined in this specification

  + Published specification: BRSKI??

  + Applications that use this media type: ANIMA bootstrap (BRSKI)

*

  + application/voucher+cms

  + Type name: application

  + Subtype name: voucher+cms

  + ID: TBD6

  + Required parameters: None

  + Optional parameters: None

  + Encoding considerations: binary

  + Security considerations: As defined in this specification

  + Published specification: BRSKI??

+ Applications that use this media type: ANIMA bootstrap (BRSKI)

8.2. Resource Type registry

Additions to the sub-registry "CoAP Resource Type", within the "CoRE Parameters" registry are needed for a new resource type.

- rt="ace.est" needs registration with IANA.

9. Security Considerations

9.1. proxy considerations

In the BRSKI bootstrap protocol, there is a direct TLS connection from pledge to EST-server. With the EST-coaps specification a direct DTLS connection from pledge to EST-server is possible thus avoiding the placement of a https/coaps proxy between pledge and http EST-server. Such a https/coaps proxy presents a security issue because the proxy needs to make a TLS connection with the EST-server and a DTLS connection with the pledge.

In [I-D.ietf-anima-bootstrapping-keyinfra] the EST-server and Registrar are co-located on the same host, thus avoiding security connections between Registrar and EST-server. It is RECOMMENDED that the links "Registrar/MASA" and "Registrar/CA" use a http TLS connection, identical to BRSKI protocol. The consequence is that the Registrar host provides both a coaps and a https stack.

The proxies proposed in Section 6 must be deployed with great care, and only when the recommended connections are impossible.

9.2. EST server considerations

The security considerations of section 6 of [RFC7030] are only partially valid for the purposes of this document. As HTTP Basic Authentication is not supported, the considerations expressed for using passwords do not apply.

Given that the client has only limited resources and may not be able to generate sufficiently random keys to encrypt its identity, it is possible that the client uses server generated private/public keys to encrypt its certificate. The transport of these keys is inherently risky. A full probability analysis MUST be done to establish whether server side key generation enhances or decreases the probability of identity stealing.

When a client uses the Implicit TA database for certificate validation, the client cannot verify that the implicit data base can act as an RA. It is RECOMMENDED that such clients include "Linking Identity and POP Information" Section 5.1 in requests (to prevent such requests from being forwarded to a real EST server by a man in the middle). It is RECOMMENDED that the Implicit Trust Anchor database used for EST server authentication be carefully managed to reduce the chance of a third-party CA with poor certification practices from being trusted. Disabling the Implicit Trust Anchor database after successfully receiving the Distribution of CA certificates response (Section 4.1.3 of [RFC7030]) limits any vulnerability to the first DTLS exchange.

In accordance with [RFC7030], TLS cipher suites that include "_EXPORT_" and "_DES_" in their names MUST NOT be used. More information about recommendations of TLS and DTLS are included in [RFC7525].

As described in CMC, Section 6.7 of [RFC5272], "For keys that can be used as signature keys, signing the certification request with the private key serves as a POP on that key pair". The inclusion of tls-unique in the certification request links the proof-of-possession to the TLS proof-of-identity. This implies but does not prove that the authenticated client currently has access to the private key.

Regarding the CSR attributes that the CA may list for inclusion in an enrollment request, an adversary could exclude attributes that a server may want, include attributes that a server may not want, and render meaningless other attributes that a server may want. The CA is expected to be able to enforce policies to recover from improper CSR requests.

Interpreters of ASN.1 structures should be aware of the use of invalid ASN.1 length fields and should take appropriate measures to guard against buffer overflows, stack overruns in particular, and malicious content in general.

10. 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. They would also like to thank Peter Panburana from Cisco for his feedback on technical details of the solution. Constructive comments were received from Eliot Lear and Julien Vermillard.

11. Change Log

-02:

binary instead of CBOR binary in mime types.

supported content types are discoverable.

DTLS POP text improved.

First version of Security considerations section written.

First version of Proxying section written.

Various text improvements.

-01:

Merging of draft-vanderstok-ace-coap-est-00 and draft-pritikin-coap-bootstrap-01

URI and discovery are modified

More text about 6tisch bootstrap including EDHOC and OSCOAP

mapping to DICE IoT profiles

adapted to BRSKI progress

12. References

12.1. Normative References

[I-D.hartke-core-pending]
Stok, P. and K. Hartke, "The 'Pending' Response Code for the Constrained Application Protocol (CoAP)", draft-hartke-core-pending-00 (work in progress), February 2017.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-06 (work in progress), May 2017.

[I-D.selander-ace-eals]
Selander, G., Raza, S., Vucinic, M., Furuhed, M., and M. Richardson, "Enrollment with Application Layer Security", draft-selander-ace-eals-00 (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>.

[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>.

[RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, DOI 10.17487/RFC5967, August 2010, <http://www.rfc-editor.org/info/rfc5967>.

[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>.

[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>.

[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>.

12.2. Informative References

[I-D.ietf-6tisch-dtsecurity-secure-join]
Richardson, M., "6tisch Secure Join protocol", draft-ietf-6tisch-dtsecurity-secure-join-01 (work in progress), February 2017.

[I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-02 (work in progress), March 2017.

[I-D.ietf-anima-voucher]
Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "Voucher Profile for Bootstrapping Protocols", draft-ietf-anima-voucher-03 (work in progress), June 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.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.

[ieee802.15.4]
Institute of Electrical and Electronics Engineers, "IEEE Standard 802.15.4-2006", 2006.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <http://www.rfc-editor.org/info/rfc4492>.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <http://www.rfc-editor.org/info/rfc4919>.

[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>.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <http://www.rfc-editor.org/info/rfc5929>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <http://www.rfc-editor.org/info/rfc5958>.

[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>.

[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>.

[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>.

[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>.

Appendix A. EST messages to EST-coaps

This section takes all examples from Appendix A of [RFC7030], changes the payload from Base64 to binary and replaces the http headers by their CoAP equivalents.

A.1. cacerts

In EST-coaps, a coaps cacerts message can be:

GET coaps://[192.0.2.1:8085]/est/crts

The corresponding CoAP header fields are shown below. The use of block and DTLS are worked out in Appendix B.

```
  Ver = 1
  T = 0 (CON)
  Code = 0x01 (0.01 is GET)
  Options
   Option1 (Uri-Host)
     Option Delta = 0x3  (option nr = 3)
     Option Length = 0x9
     Option Value = 192.0.2.1
   Option2 (Uri-Port)
     Option Delta = 0x4  (option nr = 4+3=7)
     Option Length = 0x4
     Option Value = 8085
   Option3 (Uri-Path)
     Option Delta = 0x4   (option nr = 7+4= 11)
     Option Length = 0x9
     Option Value = /est/crts
  Payload = [Empty]
```

A 2.05 Content response with a cert in EST-coaps will then be:

```
2.05 Content (Content-Format: application/pkcs7-mime)
   {payload}
```

with CoAP fields

```
  Ver = 1
  T = 2 (ACK)
  Code = 0x45 (2.05 Content)
  Options
    Option1 (Content-Format)
      Option Delta = 0xC  (option nr = 12)
      Option Length = 0x2
      Option Value = TBD1 (defined in this document)
```

Payload =

30233906092a6206734107028c2a302326020101310030b06092a6206734107018
c0c3020bb302063c20102020900a61e75193b7acc0d06092a62067341010505003O

```
4fe1c30b396424664ac426284a9f6022e026938436880adfcd95c98ca1dfc2e6d75
319b85d0458de28a9d13fb16d620fff7541f6a25d7daf004355020301000134b050
300e0603551d0f0101f104030204c1d0603551d0e04160414084d321ca0135e7721
7a486b686b334b01f0603551d23041830165311296e304761732fbfe6a2c823c30
0d06092a620673410105050003204100 2e106933a443070acf5594a3a584d08af7e
06c295059370a06639eff9bd418d13bc25a298223164a6cf1856b11a81617282e4a
410d82ef086839c6e235690322763065455351e4c596acc7c016b225dec094706c2
a10608f403b10821984c7c152343b18a768c2ad30238dc45dd653ee6092b0d5cd4c
2f7d236043269357f76d13f95fb5f00d0e19263c6833948e1ba612ce8197af650e2
5d882c12f4b6b9b67252c608ef064aca3f9bc867d71172349d510bb7651cd438837
73d927deb41c4673020bb302063c201020209009b9dda3324700d06092a62067341
01050500301b311930170603550403131065737445786 16d706c654341204e774e3
01e170d3133303530393033353333325a170d3134303530393033353333325a301b
311930170603550403131065737445786 16d706c654341204e774e302062300d060
92a6206734101010500032 04f0030204a02204100ef6b677a3247c1fc03d2b9baf1
13e5e7e11f49e0421120e6b8384160f2bf02630ef544d5fd0d5623b35713c79a722
9283a7908751a634aa420a3e2a4b1f10519d046f02f5a5dd6d760c2a842356e067b
7bd94338d1faa3b3ddd4813060a207b0a097067007e45b052b60fdbae4656e11562
c4f5abb7b0cf87a79d221f1127313c53371ce1245d63db45a1203a23340ba08042c
768d03b8076a028d3a51d87d2ef107bbd6f2305ce5e67668724002fb726df9c1447
6c37de0f55033f192a5ad21f9a2a71c20301000130b040300f0603551d130101f10
530030101fc1d0603551d0e04160414112966e304761732fbfe6a2c823c300e0603
551d0f0101f10403020106300d06092a6206734101050500032 04100423f06d4b76
0f4b42744a279035571696f272a0060f1325a40898509601ad14004f652db6312a1
475c4d7cd50f4b269035585d7856c5337765a66b38462d5bdaa7778aab24bbe2815
e37722cd10e7166c50e75ab75a1271324460211991e7445a2960f47351a1a629253
34119794b90e320bc730d6c1bee496e7ac125ce9a1eca595a3a4c54a865e6b623c9
247bfd0a7c19b56077392555c955e233642bec643ae37c166c5e221d797aea3748f
0391c8d692a5cf9bb71f6d0e37984d6fa673a30d0c006343116f58403100
```


A.2. csrattrs

In the following valid /csrattrs exchange, the EST-coaps client authenticates itself with a certificate issued by the connected CA.

The initial DTLS handhake is identical to the enrollment example. The CoAP GET request looks like:

GET coaps://[192.0.2.1:8085]/est/att

with CoAP header fields

```
  Ver = 1
  T = 0 (CON)
  Code = 0x01 (0.01 is GET)
  Options
   Option1 (Uri-Host)
     Option Delta = 0x3  (option nr = 3)
     Option Length = 0x9
     Option Value = 192.0.2.1
   Option2 (Uri-Port)
     Option Delta = 0x4  (option nr = 4+3=7)
     Option Length = 0x4
     Option Value = 8085
   Option3 (Uri-Path)
     Option Delta = 0x4   (option nr = 7+4= 11)
     Option Length = 0x8
     Option Value = /est/att
  Payload = [Empty]
```

A 2.05 Content response contains attributes which are relevant for the authenticated client. In this example, the EST-coaps server two attributes that the client can ignore when they are unknown to him.:

```
2.05 Content (Content-Format: application/crsattrs)
   {payload}
```

with CoAP fields

```
  Ver = 1
  T = 2 (ACK)
  Code = 0x45 (2.05 Content)
  Options
    Option1 (Content-Format)
      Option Delta = 0xC  (option nr = 12)
      Option Length = 0x2
      Option Value = TBD3 (defined in this document)

  Payload =
```

307c06072b060101010116302206038dc1311b131950617273652053455420617
320322e3939392e31206461746106092a620673410907302c06038dc231250603
8dc306038dc4131950617273652053455420617320322e3939392e32206461746
106092b240303020801010b06096062016503040202

A.3. enroll / reenroll

[EDNOTE: We might need a new Option for the Retry-After response message. We might need a new Option for the WWW-Authenticate response.]

During the Enroll/Reenroll exchange, the EST-coaps client uses a CSR (PKCS#10) request in the POST request payload.

```
POST coaps://[192.0.2.1:8085]/est/sen
(Content-Format: application/pkcs10)
```

with CoAP header fields

```
     Ver = 1
     T = 0 (CON)
     Code = 0x02 (0.02 is POST)
     Options
      Option1 (Uri-Host)
        Option Delta = 0x3  (option nr = 3)
        Option Length = 0x9
        Option Value = 192.0.2.1
      Option2 (Uri-Port)
        Option Delta = 0x4  (option nr = 4+3=7)
        Option Length = 0x4
        Option Value = 8085
      Option3 (Uri-Path)
        Option Delta = 0x4   (option nr = 7+4= 11)
        Option Length = 0x8
        Option Value = /est/sen
      Option4 (Content-Format)
         Option Delta = 0x1  (option nr = 11+1 = 12)
         Option Length = 0x2
         Option Value = TBD5 (defined in this document)

     Payload =
   [EDNOTE: If POP is used, make sure tls-unique in the CSR is a
    valid HMAC output. ]
```


```
   30208530206d020100301f311d301b0603550403131464656d6f7374657034203
   13336383134313335323020623000d06092a620673410101050003204f0030204a
   022041005d9f4dffd3c5949f646a9584367778560950b355c35b8e34726dd3764
   54231734795b4c09b9c6d75d408311307a81f7adef7f5d241f7d5be85620c5d44
   38bbb4242cf215c167f2ccf36c364ea2618a62f0536576369d6304e6a96877224
   7d86824f079faac7a6f694cfda5b84c42087dc062d462190c525813f210a036a7
   37b4f30d8891f4b75559fb72752453146332d51c937557716ccec624f5125c3a4
   447ad3115020048113fef54ad554ee88af09a2583aac9024075113db4990b1786
   b871691e0f02030100018701f06092a620673410907311213102b72724369722f
   372b45597535305434300d06092a620673410105050003204100441b40177a3a6
   5501487735a8ad5d3827a4eaa867013920e2afcda87aa81733c7c0353be47e1bf
   a7cda5176e7ccc6be22ae03498588d5f2de3b143f2b1a6175ec544e8e7625af6b
   836fd4416894c2e55ea99c6606f69075d6d53475d410729aa6d806afbb9986caf
   7b844b5b3e4545f19071865ada007060cad6db26a592d4a7bda7d586b68110962
   17071103407553155cddc75481e272b5ed553a8593fb7e25100a6f7605085dab4
   fc7e0731f0e7fe305703791362d5157e92e6b5c2e3edbcadb40
```


After verification of the certificate by the server, a 2.05 Content response with the issued certificate will be:

```
   2.05 Content (Content-Format: application/pkcs7-mime)
      {payload}
```

with CoAP fields

```
  Ver = 1
  T = 2 (ACK)
  Code = 0x45 (2.05 Content)
  Options
    Option1 (Content-Format)
      Option Delta = 0xC  (option nr = 12)
      Option Length = 0x2
      Option Value = TBD1 (defined in this document)

  Payload =
```

3020f806092a6206734107028329302 0e502010131003 00b06092a62067341070

A.4. serverkeygen

During this valid /serverkeygen exchange, the EST-coaps client authenticates itself using the certificate provided by the connected CA.

The initial DTLS handhake is identical to the enrollment example. The CoAP GET request looks like:

POST coaps://[192.0.2.1:8085]/est/skg

with CoAP header fields

```
     Ver = 1
     T = 0 (CON)
     Code = 0x02 (0.02 is POST)
     Options
      Option1 (Uri-Host)
        Option Delta = 0x3  (option nr = 3)
        Option Length = 0x9
        Option Value = 192.0.2.1
      Option2 (Uri-Port)
        Option Delta = 0x4  (option nr = 4+3=7)
        Option Length = 0x4
        Option Value = 8085
      Option3 (Uri-Path)
        Option Delta = 0x4   (option nr = 7+4= 11)
        Option Length = 0x8
        Option Value = /est/skg
      Option4 (Content-Format)
   [EDNOTE: the client incudes a CSR with a public key  that the
    server should ignore, so we need a content-format here. ]
         Option Delta = 0x1  (option nr = 12)
         Option Length = 0x2
         Option Value = TBD5 (defined in this document)
     Payload =
   [EDNote: If POP is used, make sure tls-unique in the CSR
    is a valid HMAC output. ]
```


```
   3020813020690201003305b313e303c0603550403133573657276657246b6579476
```


Without the DecryptKeyIdentifier atttribute, the response has no additional encryption beyond the DTLS one. The EST-coaps server response is:

```
2.05 Content (Content-Format: application/pkcs8)
   {payload}
```

The response contains first a preamble that can be ignored. The EST-coaps server can use the preamble to include additional explanations, like ownership or support information

```
  Ver = 1
  T = 2 (ACK)
  Code = 0x45 (2.05 Content)
  Options
    Option1 (Content-Format)
      Option Delta = 0xC  (option nr = 12)
      Option Length = 0x2
      Option Value = TBD2 (defined in this document)

  Payload =
```


```
30213e020100300d06092a6206734101010500042128302124020100022041003
c0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274
dd01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a1
1bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c
0c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d5
45e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704
efec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e31083010
001022041004e6b3f78b7791d6377f33117c17844531c81111fb8000282816264
915565bc7c3f3f643b537a2c69140a31c22550fa97e5132c61b74166b68626704
260620333050f510096b6570f5880e7e1c15dc0ca6ce2b5f187e2325da14ab705
ad004717f3b2f779127b5c535e0cee6a343b502722f2397a26126e0af606b5aa7
f96313511c0b7eb26354f91b82269de62757e3def807a6afdf83ddcbb0614bb7c
542e6975d6456554e7bd9988fbd1930cd44d0e01ee9182ca54539418653150254
1ad1a2a11e5021040bfce554b642c29131e7d65455e83c5406d76771912f758f5
ee3ee36af386f38ffa313c0f661880c5a2b0970485d36f528e7f77a2e55b4ad76
1242d1c2f75939c8061217d31491d305d3e07d6161c43e26f7de4477b1811de92
33dc75b426302104015bf48ac376f52887813461fc54635517bcb67293837053e
8ce1a33da7a35565a75a370dc14555b5316cb55742380350774d769d151ff0456
0214389a232a2258326163167504cfce44cd316f63bb8a52da53a4cb74fd87194
c0844881f791f23b0813ea0921325edd14459d41c8a1593f04316388e40b35fef
7d2a195a5930fa54774427ac821eee2c62790d2c17bd192af794c611011506557
83d4efe22185cbd83368786f2b1e68a5a27067e321066f0217b4b6d7971a3c21a
241366b7907187583b511102103369047e5cce0b65012200df5ec697b5827575c
db6821ff299d6a69574b31ddf0fbe9245ea2f74396c24b3a7565067e41366423b
5bdd2b2a78194094dbe333f493d159b8e07722f2280d48388db7f1c9f0633bb0e
173de2c3aa1f200af535411c7090210401421e2ea217e37312dcc606f453a6634
f3df4dc31a9e910614406412e70eec9247f10672a500947a64356c015a845a7d1
50e2e3911a2b3b61070a73247166da10bb45474cc97d1ec2bc392524307f35118
f917438f607f18181684376e13a39e07
```


```
--estServerExampleBoundary
Ver = 1
```

```
T = 2 (ACK)
Code = 0x45 (2.05 Content)
Options
  Option1 (Content-Format)
    Option Delta = 0xC  (option nr = 12)
    Option Length = 0x2
    Option Value = TBD1 (defined in this document)
Payload=
```


3020c506092a6206734107028336302Of20201013100300b06092a62067341070

A.5. enrollstatus

[EDNOTE: Include CoAP message examples.]

A.6. voucher_status

[EDNOTE: Include CoAP message examples.]

A.7. requestvoucher

[EDNOTE: Include CoAP message examples.]

Appendix B. EST-coaps Block message examples

This section provides a detailed example 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 over DTLS. The content length of the cacerts response in appendix A.1 of [RFC7030] is 4246 bytes using base64. This leads to a length of 2509 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 39 packets with a payload of 64 bytes each, followed by a packet of 13 bytes. The client sends an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP Request 40 times. 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.

```
GET [192.0.2.1:8085]/est/crts     -->
              <--   (2:0/1/39) 2.05 Content
    GET URI (2:1/1/39)                            -->
              <--   (2:1/1/39) 2.05 Content
                            |
                            |
                            |
     GET URI (2:65/1/39)                          -->
              <--   (2:65/0/39) 2.05 Content
```

For further detailing the CoAP headers of the first two blocks are written out.

The header of the first GET looks like:

```
     Ver = 1
     T = 0 (CON)
     Code = 0x01 (0.1 GET)
     Options
      Option1 (Uri-Host)
        Option Delta = 0x3  (option nr = 3)
        Option Length = 0x9
        Option Value = 192.0.2.1
      Option2 (Uri-Port)
        Option Delta = 0x4   (option nr = 3+4=7)
        Option Length = 0x4
        Option Value = 8085
      Option3 (Uri-Path)
        Option Delta = 0x4    (option nr = 7+4=11)
        Option Length = 0x9
        Option Value = /est/crts
     Payload = [Empty]
```

The header of the first response looks like:

```
     Ver = 1
     T = 2 (ACK)
     Code = 0x45 (2.05 Content.)
     Options
       Option1 (Content-Format)
         Option Delta = 0xC  (option 12)
         Option Length = 0x2
         Option Value = TBD1
       Option2 (Block2)
         Option Delta = 0xB  (option 23 = 12 + 11)
         Option Length = 0x1
         Option Value = 0x0A (block number = 0, M=1, SZX=2)
     Payload =
   30233906092a6206734107028c2a30232602010131 00300b06092a6206734107018
   c0c3020bb302063c20102020900a61e75193b7acc0d06092a6206734101
```

The second Block2:

```
  Ver = 1
  T = 2 (means ACK)
  Code = 0x45 (2.05 Content.)
  Options
    Option1 (Content-Format)
      Option Delta = 0xC   (option 12)
      Option Length = 0x2
      Option Value = TBD1
    Option2 (Block2)
      Option Delta = 0xB  (option 23 = 12 + 11)
      Option Length = 0x1
      Option Value = 0x1A (block number = 1, M=1, SZX=2)
  Payload =
05050030
1b3119301706035504031310657374445786 16d706c654341204f774f301e170d313
330353039303335333315a170d313430353039303335333315a
```

The 40th and final Block2:

```
  Ver = 1
  T = 2 (means ACK)
  Code = 0x21
  Options
    Option1 (Content-Format)
      Option Delta = 0xC   (option 12)
      Option Length = 0x2
      Option Value = TBD1
    Option2 (Block2)
      Option Delta = 0xB  (option 23 = 12 + 11)
      Option Length = 0x2
      Option Value = 0x272 (block number = 39, M=0, SZX=2)
  Payload = 73a30d0c006343116f58403100
```

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

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Martin Furuhed
Nexus Group

Email: martin.furuhed@nexusgroup.com

Shahid Raza
RISE SICS
Isafjordsgatan 22
Kista, Stockholm 16440
SE

Email: shahid@sics.se

Core
Internet Draft
Intended status: Standards Track
Expires: March 11, 2018

P. Wang
C. Pu
H. Wang
Y. Yang
L. Shao
C hongqing University of
Posts and Telecommunications
J. Hou
Huawei Technologies
September 7, 2017

OPC UA Message Transmission Method over CoAP
draft-wang-core-opcua-transmission-02

Abstract

OPC Unified Architecture (OPC UA) is a data exchange standard that provides interoperability in industrial automation. With the arrival of 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 are different. Driven by this, to implement 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. It is worth noting that the design of CoAP is inspired by HTTP, thus, there are some similarities in transmission method between them. This document provides some transmission methods for message of OPC UA over CoAP, so that the communication between OPC UA client and OPC UA server could be established.

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 safety, 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 layers 5, 6 or 7 protocol such as 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 implemented 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.

```
+--------------------------------------------------------+ ------
|                     UA Application                     |
+--------------------------------------------------------+
+--------------------------+ +--------------------------+
|        UA Binary         | |          UA XML          |
+--------------------------+ +--------------------------+
+---------------+                        +---------------+  App
|  UA Secure    |                        |  WS Secure    |
| Conversation  |                        | Conversation  | Layer
+---------------+                        +---------------+
+---------------+ +----------+ +----------+ +---------------+
|               | |          | |  SOAP    | |     SOAP      |
|    UA TCP     | |  HTTPS   | |----------| |---------------|
|               | |          | |  HTTPS   | |     HTTP      |
+---------------+ +----------+ +----------+ +---------------+ ------
+--------------------------------------------------------+
|                         TCP/IP                         |
+--------------------------------------------------------+
```

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.

```
            Client Secure    UA                    UA    Secure   Server
                  Channel   TCP                   TCP   Channel
             |       |       |     Open Socket     |       |       |
             |       |       | - - - - - - - - - > |       |       |
             |       |       |        Hello        |       |       |
             |       |       | - - - - - - - - - > |       |       |
             |       |       |      Acknowledge    |       |       |
             |       |       | < - - - - - - - - - |       |       |
             |       |       |                     |       |       |
             |       |       |OpenSecureChannelReq |       |       |
             |       | - - - - - - - - - - - - - - - - - > |       |
             |       |       |OpenSecureChannelRes |       |       |
             |       | < - - - - - - - - - - - - - - - - - |       |
             |       |       |                     |       |       |
             |       |       |   CreateSessionReq  |       |       |
             | - - - - - - - - - - - - - - - - - - - - - - - - - > |
             |       |       |   CreateSessionRes  |       |       |
             | < - - - - - - - - - - - - - - - - - - - - - - - - - |
             |       |       | ActivateSessionReq  |       |       |
             | - - - - - - - - - - - - - - - - - - - - - - - - - > |
             |       |       | ActivateSessionRes  |       |       |
             | < - - - - - - - - - - - - - - - - - - - - - - - - - |
             |       |       |                     |       |       |
             |       |       |       Request       |       |       |
             | = = = = = = = = = = = = = = = = = = = = = = = = = > |
             |       |       |       Response      |       |       |
             | < = = = = = = = = = = = = = = = = = = = = = = = = = |
```

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 conducted 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.

```
+-------------------------------+ ------
|         UA Application        |
+-------------------------------+
+--------------+ +--------------+
|  UA Binary   | |    UA XML    |  App
+--------------+ +--------------+
+-------------------------------+
|         Secure Channel        | Layer
+-------------------------------+
+-------------------------------+
|             CoAP              |
+-------------------------------+ ------
+-------------------------------+
|           UDP, DTLS           |
+-------------------------------+
```

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.

```
                                            - - - - - - - - - - - - -
                                          //         + - - - - +     \\
                                         ||          | UA-CoAP |     ||
                                         ||      //  |  server |     ||
                                         ||     //   + - - - - +     ||
                                         ||    //                    ||
                                         ||   //                     ||
 + - - - +   HTTP Request   + - - - - - - + CoAP Request  + - - - - +||
 |   UA  | - - - - - - -  > | HTTP-to-CoAP| - - - - - - > | UA-CoAP |||
 | client| < - - - - - - -  |    PROXY    | < - - - - - - |  server |||
 + - - - +   HTTP Response  + - - - - - - + CoAP Response + - - - - +||
                                         ||  \\                      ||
                                         ||   \\                     ||
                                         ||    \\                    ||
                                         ||     \\   + - - - - +     ||
                                         ||      \\  | UA-CoAP |     ||
                                         ||          |  server |     ||
                                          \\         + - - - - +     //
                                            - - - - - - - - - - - - -
```

Figure 4: Proxy for OPC UA to CoAP

As shown in Figure 5, assuming all OPC UA servers are based on CoAP [draft-wang-core-opcua-transmition-requirements], and all OPC UA-CoAP server can be treated as a network, then 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 are 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 to 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. According to CoAP, 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.

```
+ - - - - - - +     CoAP  Request       + - - - - - - +
| UA  client  |   - - - - - - - - - - > | UA  server  |
|             |   < - - - - - - - - - - |             |
+ - - - - - - +     CoAP  Response      + - - - - - - +
```

Figure 5: Direct transmission OPC UA based on CoAP

4.3. REST transmission for OPC UA

OPC UA is a set of data which exchange specifications for industrial communication, the core of the OPC UA protocol are 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 referencing. 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.

```
          + - - - - - +          + - - - - - +
          | client  |          | server  |
          + - - - - - +          + - - - - - +
               |                      |
               |                      |
               |    CoAPRequest       |
               | - - - - - - - - - - > |
               |    CoAPResponse      |
               | < - - - - - - - - - - |
```

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 scenes. 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 proxy into the network [I-D.ietf-core-coap-pubsub], when a node needs to sleep, the node information is sent to the proxy agent, when another 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

This document does not add any new security considerations beyond what the referenced technologies already have.

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-09 (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-01 (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-02 (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