

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 3, 2012

J. Arkko
Ericsson
C. Jennings
Cisco
Z. Shelby
Sensinode
October 31, 2011

Uniform Resource Names for Device Identifiers
draft-arkko-core-dev-urn-01

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

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

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

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

This Internet-Draft will expire on May 3, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
2. Requirements language	4
3. DEV URN Definition	4
4. DEV URN Subtypes	5
4.1. MAC Addresses	5
4.2. 1-Wire Device Identifiers	6
4.3. Cryptographically Generated Identifiers	6
5. Examples	7
6. Security Considerations	7
7. IANA Considerations	8
8. References	8
8.1. Normative References	8
8.2. Informative References	8
Appendix A. Acknowledgments	10
Authors' Addresses	10

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 [I-D.ietf-core-coap], [I-D.jennings-senml], [I-D.arkko-core-sleepy-sensors] [I-D.arkko-core-security-arch]. 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], [I-D.ietf-core-coap]. Finally, URNs can also be easily carried and stored in formats such as XML [W3C.REC-xml-19980210] or JSON [I-D.jennings-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, [I-D.montemurro-gsma-imei-urn] defines International Mobile station Equipment Identity (IMEI) identifiers for use with 3GPP cellular systems. Similarly, [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 URN type defined here, or define their own URNs.

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, 1-wire device identifiers, and cryptographically defined 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:" subtype ":" hexstring componentpart
subtype = "mac" / "ow" / "cgi"
componentpart = [ "-" component [ componentpart ] ]
component = *1(DIGIT / ALPHA)
hexstring = hexbyte /
            hexbyte hexstring
hexbyte = hexdigit hexdigit
hexdigit = DIGIT / hexletter
hexletter = "a" / "b" / "c" / "d" / "e" / "f" /
            "A" / "B" / "C" / "D" / "E" / "F"
```

The above Augmented Backus-Naur Form (ABNF) uses the DIGIT and ALPHA rules defined in [RFC5234], which are not repeated here.

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, but not case-sensitive, string match. While uppercase letters are accepted, all systems that construct or display DEV URNs MUST employ lower case letters. This is necessary to ensure that searches, processing rules, and other potentially case sensitive tools have uniformly lower-case identifiers to look at.

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. Cryptographically Generated Identifiers

DEV URNs with the "cgi" subtype represent cryptographically identified devices. These identifiers are variable length but the hexstring MUST be at least 16 characters long (128 bits). The hexstring MUST have an even number of characters.

This memo does not define the construction of the cryptographic identifiers or the algorithms used in the construction, as these are up to the specific implementations. It should be noted, however, that the use of cryptographic identifiers for anything else as tokens of identification will require the communicating parties to agree on how they are used, and what algorithms and methods are used to verify

an ownership claim, for instance. These are outside the scope of this draft, however. The authors observe that the use of plain identifiers independent of the actual cryptography that goes inside the identifier is possible. For instance, Cryptographically Generated Addresses (CGAs) [RFC3972] can be used by parties that are unaware of how they were constructed, and that ownership proofs and other advanced functionality are made separately [RFC3971].

Identifier assignment rests on individual devices and manufacturers; no coordinated identifier assignment or guaranteed uniqueness exists. However, the 128 bit or longer identifiers are very unlikely to collide, as long as their generation employs sound cryptographic principles and proper sources of randomness are used where necessary.

5. Examples

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

```
urn:dev:mac:0024beffffe804ff1  # The MAC address of
                                # Jari's laptop

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

urn:dev:cgi:fd4a5bf6ffb4ca6c    # The example hash output
                                # for a CGA from RFC 3972
                                # (before modifications to
                                # convert it to an IP address)
```

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.

7. IANA Considerations

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

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.
- [RFC3406] Daigle, L., van Gulik, D., Iannella, R., and P. Faltstrom, "Uniform Resource Names (URN) Namespace Definition Mechanisms", BCP 66, RFC 3406, October 2002.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [IEEE.EUI64] IEEE, "Guidelines For 64-bit Global Identifier (EUI-64)", IEEE ,
<<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>,
<<http://www.maxim-ic.com/app-notes/index.mvp/id/1796>>.

8.2. Informative References

- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [RFC3971] Arkko, J., Kempf, J., Zill, B., and P. Nikander, "SEcure

- Neighbor Discovery (SEND)", RFC 3971, March 2005.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, July 2005.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [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>>.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-06 (work in progress), May 2011.
- [I-D.jennings-senml]
Jennings, C., "Media Type for Sensor Markup Language (SENML)", draft-jennings-senml-05 (work in progress), March 2011.
- [I-D.arkko-core-sleepy-sensors]
Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-01 (work in progress), July 2011.
- [I-D.arkko-core-security-arch]
Arkko, J. and A. Keranen, "CoAP Security Architecture", draft-arkko-core-security-arch-00 (work in progress), July 2011.
- [I-D.montemurro-gsma-imei-urn]
Montemurro, M., "A Uniform Resource Name Namespace For The GSM Association (GSMA) and the International Mobile station Equipment Identity(IMEI)", draft-montemurro-gsma-imei-urn-01 (work in progress), February 2007.
- [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-01 (work in progress),
August 2011.

Appendix A. Acknowledgments

The authors would like to thank Christer Holmberg 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 [I-D.montemurro-gsma-imei-urn] 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 Working Group
Internet-Draft
Intended status: Informational
Expires: April 26, 2012

M. Becker, Ed.
K. Kuladinithi
T. Poetsch
ComNets, TZI, University Bremen
October 24, 2011

Transport of CoAP over SMS and GPRS
draft-becker-core-coap-sms-gprs-00

Abstract

The 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), that took the limitations of LLNs into account, is thus also applicable to telematic M2M devices. The adaption of CoAP to the SMS transport mechanism is described in this document.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 26, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
1.1. Scenarios	3
1.2. Requirements Language	5
2. Encoding of CoAP for SMS transport	5
3. Message Size Implementation Considerations	5
4. Options	5
5. Protocol Constants	6
6. Multicast	6
7. Proxying Considerations	6
8. SMS URI scheme for link-format	6
9. Acknowledgements	6
10. IANA Considerations	6
11. Security Considerations	6
12. References	7
12.1. Normative References	7
12.2. Informative References	7
Authors' Addresses	7

1. Introduction

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

1.1. Scenarios

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

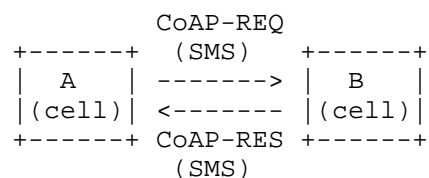


Figure 1: Cellular and Cellular Communication (only SMS-based)

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

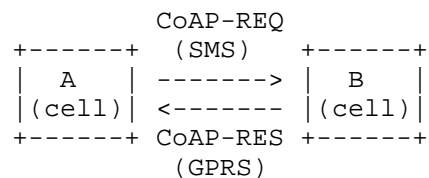


Figure 2: Cellular and Cellular Communication (SMS/GPRS-based)

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. Computer Interface to Message Distribution (CIMD [cimdl]), Universal Computer Protocol/External Machine Interface (UCP/EMI [ucpl]), Short Message Peer-to-Peer (SMPP [smpp])) to submit an SMS for delivery, which contains the CoAP Request (depicted in Figure 3).

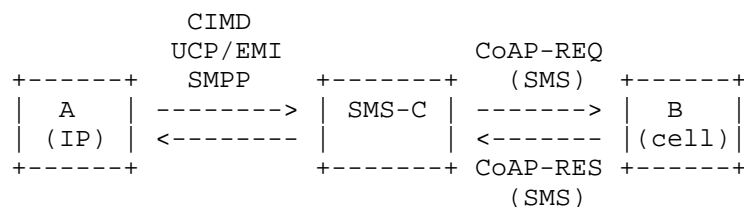


Figure 3: IP and Cellular Communication (only SMS-based)

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

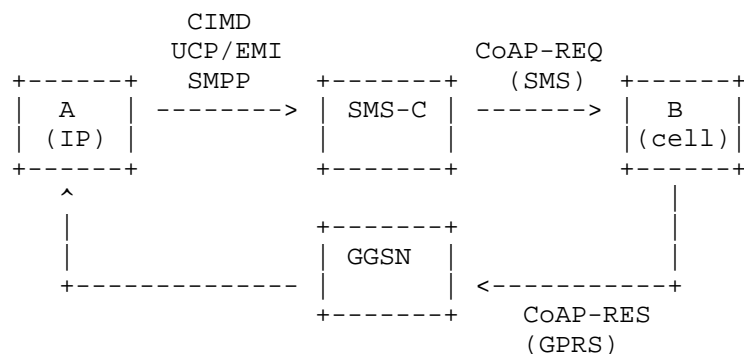


Figure 4: IP and Cellular Communication (SMS/GPRS-based)

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

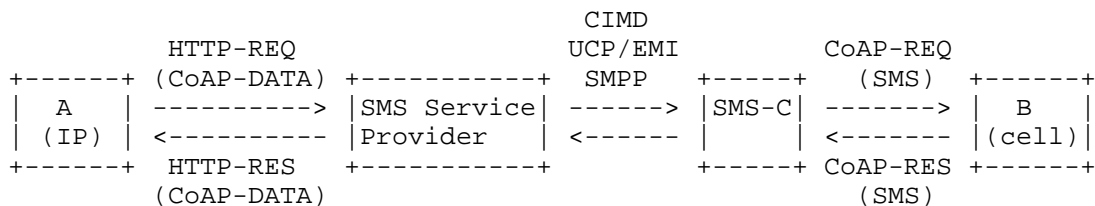


Figure 5: IP and Cellular Communication (only SMS-based, using an SMS service provider)

At the moment, this document assumes the scenarios shown in Figure 1, Figure 3 and Figure 5), i.e.\ only SMS transport.

1.2. 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. Encoding of CoAP for SMS transport

The content of SMS can be coded in 7, 8 or 16 bit characters [3gpp_ts23.038]. The advantages and disadvantages are:

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

The currently safest solution is to use 7 bit encoded SMS including Base64 encoded CoAP payload.

3. Message Size Implementation Considerations

Using 7 bit encoding 160 characters are allowed in 1 SMS, while using 8 bit encoding 140 characters are allowed. [3gpp_ts23.038]

Possible options for larger CoAP messages are:

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

4. Options

Uri-Host and Uri-Port options MUST NOT be included in the CoAP header. End-points receiving CoAP messages over SMS with such options MUST behave as specified in [I-D.ietf-core-coap].

Open question: Is the introduction of a new CoAP option Reply-To-Uri-Host necessary, if the server should use the GPRS transport for the Response? This relates to Figure 2 and Figure 4.

Number	C/E	Name	Format	Length	Default
17	Critical	Reply-To-Uri-Host	string	1-270 B	(none)
19	Critical	Reply-To-Uri-Port	uint	0-2 B	(none)

Table 1: New CoAP Option Numbers

5. Protocol Constants

The `RESPONSE_TIMEOUT` variable SHOULD be configured for a higher duration than specified in [I-D.ietf-core-coap], i.e. 10 s.

6. Multicast

Multicast MUST not be used with the SMS transport.

7. Proxying Considerations

TBD (Proxying into an IPv6/v4 network (e.g. a 6LoWPAN network) possible?)

8. SMS URI scheme for link-format

Open question: Make use of RFC5724 SMS URI scheme?

9. Acknowledgements

This document is 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.

10. IANA Considerations

This memo includes no request to IANA.

11. Security Considerations

This presents no security considerations beyond those in section 10

of the base CoAP specification [I-D.ietf-core-coap].

12. References

12.1. Normative References

[3gpp_ts23.038]

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

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-04 (work in progress), July 2011.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), July 2011.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

12.2. Informative References

[cimd] Nokia, "CIMD Interface Specification (SMSCDOC8000.00, Nokia SMS Center 8.0)", 2005.

[smpp] SMPP Developers Forum, "Short Message Peer to Peer Protocol Specification v3.4 Issue 1.2", 1999.

[ucpl] Vodafone, "Short Message Service Centre (SMSC) External Machine Interface (EMI) Description Version 4.3d", 2011.

Authors' Addresses

Markus Becker (editor)
ComNets, TZI, University Bremen
Bibliothekstrasse 1
Bremen 28359
Germany

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

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

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

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

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

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: May 3, 2012

C. Bormann
K. Hartke
Universitaet Bremen TZI
October 31, 2011

Miscellaneous additions to CoAP
draft-bormann-coap-misc-09

Abstract

This short I-D makes a number of partially interrelated proposals how to solve certain problems in the CoRE WG's main protocol, the Constrained Application Protocol (CoAP). The current version has been resubmitted to keep information about these proposals available; the proposals are not all fleshed out at this point in time.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 3, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Getting rid of artificial limitations	4
2.1. Beyond 15 options	4
2.1.1. Implementation considerations	6
2.1.2. What should we do now?	7
2.1.3. Alternatives	7
2.2. Beyond 270 bytes in a single option	7
3. URI Authorities with Binary Addresses	9
4. User-defined Option	11
5. Payload-Length Option	12
6. IANA Considerations	13
7. Security Considerations	14
8. Acknowledgements	15
9. References	16
9.1. Normative References	16
9.2. Informative References	16
Appendix A. The Cemetery (Things we won't do)	17
A.1. Stateful URI compression	17
Appendix B. Experimental Options	19
B.1. Options indicating absolute time	19
B.2. Representing Durations	20
B.3. Rationale	21
B.4. Pseudo-Floating Point	22
B.5. A Duration Type for CoAP	23
Authors' Addresses	30

1. Introduction

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

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

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

This draft attempts to address a number of problems not yet adequately solved in [I-D.ietf-core-coap]. The solutions proposed to these problems are somewhat interrelated and are therefore presented in one draft.

The appendix contains the "CoAP cemetery" (possibly later to move into its own draft), documenting roads that the WG decided not to take, in order to spare readers from reinventing them in vain.

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

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

2. Getting rid of artificial limitations

Artificial limitations are limitations of a protocol or system that are not rooted in limitations of actual capabilities, but in arbitrary design decisions. Proper system design tries to avoid artificial limitations, as these tend to cause complexity in systems that need to work with these limitations.

E.g., the original UNIX filesystem had an artificial limitation of the length of a path name component to 14 bytes. This led to all kinds of workarounds in programs that manipulate file names: E.g., systematically replacing a ".el" extension in a filename with a ".elc" for the compiled file might exceed the limit, so all ".el" files were suddenly limited to 13-byte filenames.

Note that, today, there still is a limitation in most file system implementations, typically at 255. This just happens to be high enough to rarely be of real-world concern; we will refer to this case as a "painless" artificial limitation.

CoAP-08 has two highly recognizable artificial limitations in its protocol encoding

- o The number of options in a single message is limited to 15 max.
- o The length of an option is limited to 270 max.

It is arguable that the latter limitation causes few problems, just as the 255-byte path name component limitation in filenames today causes few problems. Just in case, Section 2.2 provides a design to extend this.

2.1. Beyond 15 options

The limit of 15 options is motivated by the fixed four-bit field "OC" that is used for indicating the number of options in the fixed-length CoAP header (Figure 1).

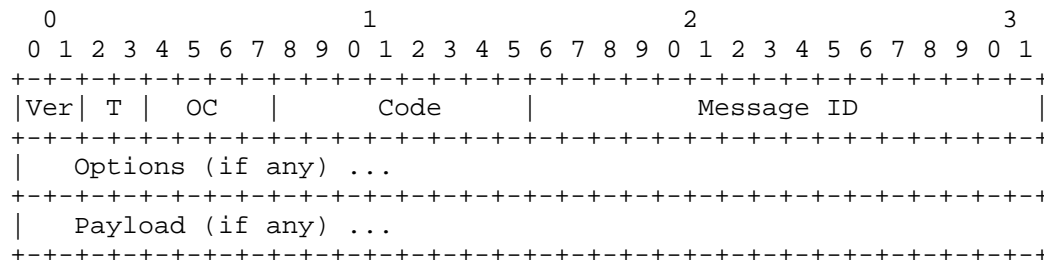


Figure 1: Four-byte fixed header in a CoAP Message

Note that there is another fixed four-bit field in CoAP: the option length (Figure 2 - note that this figure is not to the same scale as the previous figure):

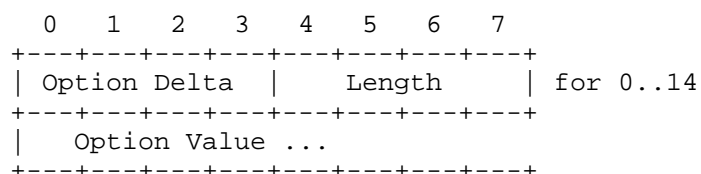


Figure 2: Short Option Header

Since 15 is unacceptable for a maximum option length, the all-ones value (15) was taken out of the set of allowable values for the short header, and a long header was introduced that allows the insertion of an extension byte (Figure 3):

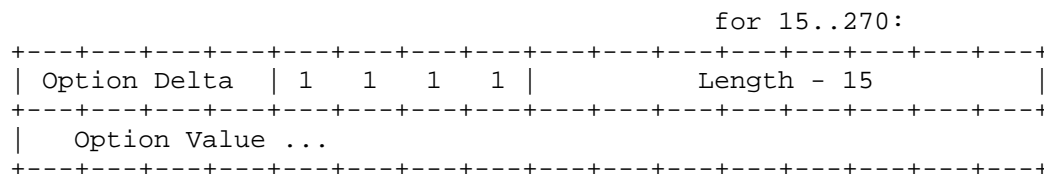


Figure 3: Long Option Header

We might want to use the same technique for the CoAP header as well. There are two obvious places where the extension byte could be placed:

1. right after the byte carrying the OC field, so the structure is the same as for the option header;
2. right after the fixed-size CoAP header.

Both solutions lose the fixed-size-ness of the CoAP header.

Solution 1 has the disadvantage that the CoAP header is also changing in structure: The extension byte is wedged between the first and the second byte of the CoAP header. This is unfortunate, as the number of options only comes into play when the option processing begins, so it is more natural to use solution 2 (Figure 4):

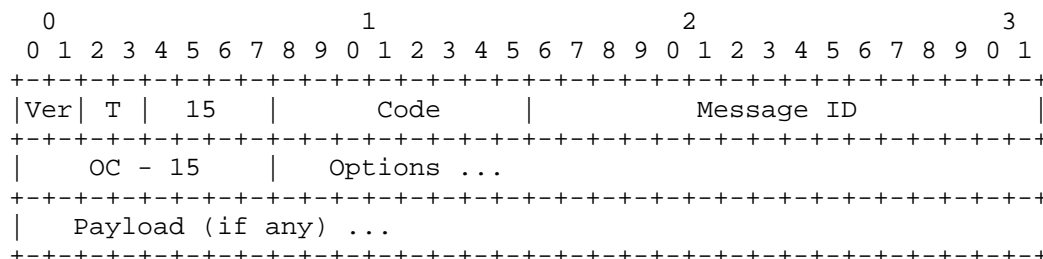


Figure 4: Extended header for CoAP Messages with 15+ options

This would allow for up to 270 options in a CoAP message, which is very likely way beyond the "painless" threshold.

2.1.1.1. Implementation considerations

For a message decoder, this extension creates relatively little pain, as the number of options only becomes interesting when the encoding turns to the options part of the message, which is then simply lead in by the extension byte if the four-bit field is 15.

For a message encoder, this extension is not so rosy. If the encoder is constructing the message serially, it may not know in advance whether the number of options will exceed 14. None of the following implementation strategies is particularly savory, but all of them do work:

1. Encode the options serially under the assumption that the number of options will be 14 or less. When the 15th option needs to be encoded, abort the option encoding, and restart it from scratch one byte further to the left.
2. Similar to 1, except that the bytes already encoded are all moved one byte to right, the extension byte is inserted, and the option encoding process is continued.
3. The encoder always leaves space for the extension byte (at least if it can't prove the number will be less than 14). If the extension byte is not needed, an Option 0 with length 0 is encoded instead (i.e., one byte is wasted - this option is elective and will be ignored by the receiver).

As a minimum, to enable strategy 3, the option 0 should be reserved at least for the case of length=0.

2.1.2. What should we do now?

As a minimum proposal for the next version of CoAP, the value 15 for OC should be marked as reserved today.

2.1.3. Alternatives

One alternative that has been discussed previously is to have an "Options" Option, which allows the carriage of multiple options in the belly of a single one. This could also be used to carry more than 15 options. However:

- o The conditional introduction of an Options option has implementation considerations that are likely to be more severe than the ones listed above;
- o since 270 bytes may not be enough for the encoding of `_all_` options, the "Options" option would need to be repeatable. This creates many different ways to encode the same message, leading to combinatorial explosion in test cases for ensuring interoperability.

2.2. Beyond 270 bytes in a single option

The authors would argue that 270 as the maximum length of an option is already beyond the "painless" threshold.

If that is not the consensus of the WG, the scheme can easily be extended as in Figure 5:

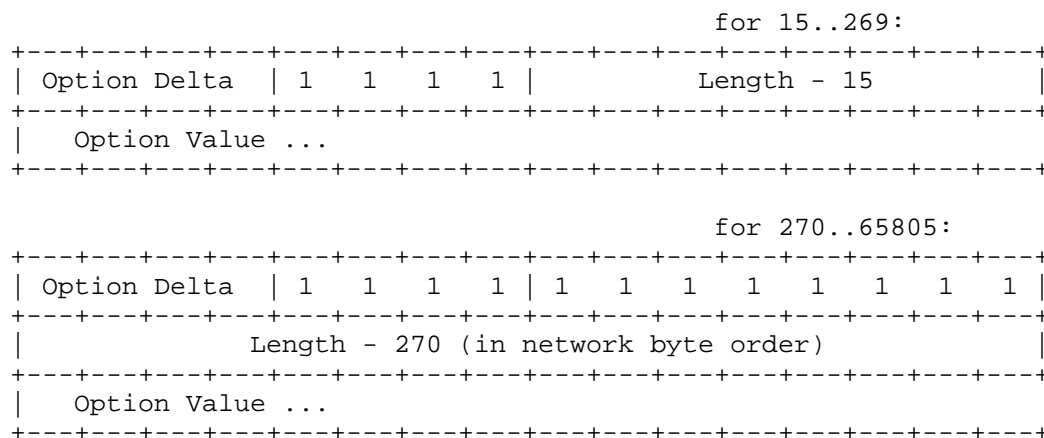


Figure 5: Ridiculously Long Option Header

The infinite number of obvious variations on this scheme are left as an exercise to the reader.

Again, as a precaution to future extensions, the current encoding for length 270 (eight ones in the extension byte) could be marked as reserved today.

3. URI Authorities with Binary Addresses

One problem with the way URI authorities are represented in the URI syntax is that the authority part can be very bulky if it encodes an IPv6 address in ASCII.

Proposal: Provide an option "Uri-Authority-Binary" that can be an even number of bytes between 2 and 18 except 12 or 14.

- o If the number of bytes is 2, the destination IP address of the packet transporting the CoAP message is implied.
- o If the number of bytes is 4 or 6, the first four bytes of the option value are an IPv4 address in binary.
- o If the number of bytes is 8 or 10, the first eight bytes are the lower 64 bits of an IPv6 address; the upper eight bytes are implied from the destination address of the packet transporting the CoAP message.
- o If the number of bytes is 16 or 18, the first 16 bytes are an IPv6 address.
- o If two more bytes remain, this is a port number (as always in network byte order).

The resulting authority is (conceptually translated into ASCII and) used in place of an Uri-Authority option, or inserted into a Proxy-Uri. Examples:

Proxy-Uri	Uri-Authority-Binary	Uri-Path	URI
(none)	(none)	(none)	"/"
(none)	(none)	'temp'	"/temp"
(none)	2 bytes: 61616	'temp'	"coap://[DA]:61616/temp"
(none)	16 bytes: 2000::1	temp	"coap://[2000::1]/temp"
'http://'	10 bytes: ::123:45 + 616	(none)	"http://[DA::123:45]:616"

'http:///te	18 bytes:	(none)	"http://[2000::1]:616/
mp'	2000::1 + 616		temp"
+-----+	+-----+	+-----+	+-----+

4. User-defined Option

To enable experimentation and community-specific options, option number 14 (the first NOP option) can also be used as a user-defined option. For this application, the option value has one or more bytes, the semantics of which are defined by prior agreement between the communicating partners.

It is RECOMMENDED to start the option value with a unique identifier, e.g., the SDNV [RFC5050] of the enterprise number of the organisation defining the option, possibly followed by additional discriminating bits or bytes.

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 0 0 0 0 0 0 1 | 0 1 1 1 0 0 1 1 | private opt-id | value... |
+-----+-----+-----+-----+-----+-----+-----+-----+
\__SDNV of enterprise number__/_

```

Figure 6: Example option value for user-defined option

User-defined Options are elective.

The User-defined Option is repeatable.

Potential options:

No.	C/E	Name	Format	Length	Default
14	Elective	User	(see below)	1-270 B	(empty)

5. Payload-Length Option

Not all transport mappings may provide an unambiguous length of the CoAP message. For UDP, it may also be desirable to pack more than one CoAP message into one UDP payload (aggregation); in that case, for all but the last message there needs to be a way to delimit the payload of that message.

This can be solved using a new option, the Payload-Length option. If this option is present, the value of this option is an unsigned integer giving the length of the payload of the message (note that this integer can be zero for a zero-length payload, which can in turn be represented by a zero-length option value). (In the UDP aggregation case, what would have been in the payload of this message after "payload-length" bytes is then actually one or more additional messages.)

6. IANA Considerations

This draft adds option numbers to Table 2 of [I-D.ietf-core-coap], resulting in:

TBD.

7. Security Considerations

TBD.

8. Acknowledgements

This work was partially funded by the Klaus Tschira Foundation.

Of course, much of the content of this draft is the result of discussions with the [I-D.ietf-core-coap] authors.

9. References

9.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-07 (work in progress), July 2011.
- [I-D.ietf-httpbis-pl-messaging]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H.,
Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., and
J. Reschke, "HTTP/1.1, part 1: URIs, Connections, and
Message Parsing", draft-ietf-httpbis-pl-messaging-17 (work
in progress), October 2011.
- [I-D.ietf-httpbis-p4-conditional]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H.,
Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., and
J. Reschke, "HTTP/1.1, part 4: Conditional Requests",
draft-ietf-httpbis-p4-conditional-17 (work in progress),
October 2011.
- [I-D.ietf-httpbis-p6-cache]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H.,
Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y.,
Nottingham, M., and J. Reschke, "HTTP/1.1, part 6:
Caching", draft-ietf-httpbis-p6-cache-17 (work in
progress), October 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

9.2. Informative References

- [REST] Fielding, R., "Architectural Styles and the Design of
Network-based Software Architectures", 2000.
- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol
Specification", RFC 5050, November 2007.

Appendix A. The Cemetery (Things we won't do)

This annex documents roads that the WG decided not to take, in order to spare readers from reinventing them in vain.

A.1. Stateful URI compression

Is the approximately 25 % average saving achievable with Huffman-based URI compression schemes worth the complexity? Probably not, because much higher average savings can be achieved by introducing state.

Henning Schulzrinne has proposed for a server to be able to supply a shortened URI once a resource has been requested using the full-length URI. Let's call such a shortened referent a `_Temporary Resource Identifier_`, `_TeRI_` for short. This could be expressed by a response option as shown in Figure 7.

```

0
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+---+---+---+---+---+---+---+---+---+
|   duration   |   TeRI...
+---+---+---+---+---+---+---+---+---+

```

Figure 7: Option for offering a TeRI in a response

The TeRI offer option indicates that the server promises to offer this resources under the TeRI given for at least the time given as the duration. Another TeRI offer can be made later to extend the duration.

Once a TeRI for a URI is known (and still within its lifetime), the client can supply a TeRI instead of a URI in its requests. The same option format as an offer could be used to allow the client to indicate how long it believes the TeRI will still be valid (so that the server can decide when to update the lifetime duration). TeRIs in requests could be distinguished from URIs e.g. by using a different option number.

Proposal: Add a TeRI option that can be used in CoAP requests and responses.

Add a way to indicate a TeRI and its duration in a link-value.

Do not add any form of stateless URI encoding.

Benefits: Much higher reduction of message size than any stateless URI encoding could achieve.

As the use of TeRIs is entirely optional, minimal complexity nodes can get by without implementing them.

Drawbacks: Adds considerable state and complexity to the protocol.

It turns out that real CoAP URIs are short enough that TeRIs are not needed.

(Discuss the security implications of TeRIs.)

Appendix B. Experimental Options

This annex documents proposals that need significant additional discussion before they can become part of (or go back to) the main CoAP specification. They are not dead, but might die if there turns out to be no good way to solve the problem.

B.1. Options indicating absolute time

HTTP has a number of headers that may indicate absolute time:

- o "Date", defined in Section 14.18 in [RFC2616] (Section 9.3 in [I-D.ietf-httpbis-pl-messaging]), giving the absolute time a response was generated;
- o "Last-Modified", defined in Section 14.29 in [RFC2616], (Section 6.6 in [I-D.ietf-httpbis-p4-conditional]), giving the absolute time of when the origin server believes the resource representation was last modified;
- o "If-Modified-Since", defined in Section 14.25 in [RFC2616], "If-Unmodified-Since", defined in Section 14.28 in [RFC2616], and "If-Range", defined in Section 14.27 in [RFC2616] can be used to supply absolute time to gate a conditional request;
- o "Expires", defined in Section 14.21 in [RFC2616] (Section 3.3 in [I-D.ietf-httpbis-p6-cache]), giving the absolute time after which a response is considered stale.
- o The more obscure headers "Retry-After", defined in Section 14.37 in [RFC2616], and "Warning", defined in section 14.46 in [RFC2616], also may employ absolute time.

[I-D.ietf-core-coap] defines a single "Date" option, which however "indicates the creation time and date of a given resource representation", i.e., is closer to a "Last-Modified" HTTP header. HTTP's caching rules [I-D.ietf-httpbis-p6-cache] make use of both "Date" and "Last-Modified", combined with "Expires". The specific semantics required for CoAP needs further consideration.

In addition to the definition of the semantics, an encoding for absolute times needs to be specified.

In UNIX-related systems, it is customary to indicate absolute time as an integer number of seconds, after midnight UTC, January 1, 1970. Unless negative numbers are employed, this time format cannot represent time values prior to January 1, 1970, which probably is not required for the uses of absolute time in CoAP.

If a 32-bit integer is used and allowance is made for a sign-bit in a local implementation, the latest UTC time value that can be represented by the resulting 31 bit integer value is 03:14:07 on January 19, 2038. If the 32-bit integer is used as an unsigned value, the last date is 2106-02-07, 06:28:15.

The reach can be extended by: - moving the epoch forward, e.g. by 40 years (= 1262304000 seconds) to 2010-01-01. This makes it impossible to represent Last-Modified times in that past (such as could be gatewayed in from HTTP). - extending the number of bits, e.g. by one more byte, either always or as one of two formats, keeping the 32-bit variant as well.

Also, the resolution can be extended by expressing time in milliseconds etc., requiring even more bits (e.g., a 48-bit unsigned integer of milliseconds would last well after year 9999.)

For experiments, an experimental "Date" option is defined with the semantics of HTTP's "Last-Modified". It can carry an unsigned integer of 32, 40, or 48 bits; 32- and 40-bit integers indicate the absolute time in seconds since 1970-01-01 00:00 UTC, while 48-bit integers indicate the absolute time in milliseconds since 1970-01-01 00:00 UTC.

However, that option is not really that useful until there is a "If-Modified-Since" option as well.

(Also: Discuss nodes without clocks.)

B.2. Representing Durations

Various message types used in CoAP need the representation of *durations*, i.e. of the length of a timespan. In SI units, these are measured in seconds. CoAP durations represent integer numbers of seconds, but instead of representing these numbers as integers, a more compact single-byte pseudo-floating-point (pseudo-FP) representation is used (Figure 8).

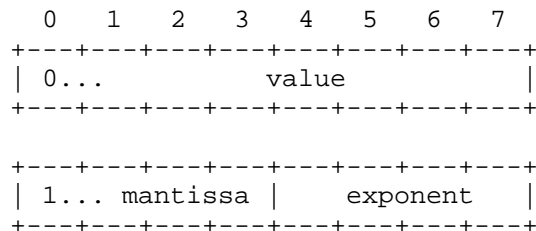


Figure 8: Duration in (8,4) pseudo-FP representation

If the high bit is clear, the entire n-bit value (including the high bit) is the decoded value. If the high bit is set, the mantissa (including the high bit, with the exponent field cleared out but still present) is shifted left by the exponent to yield the decoded value.

The (n,e)-pseudo-FP format can be decoded with a single line of code (plus a couple of constant definitions), as demonstrated in Figure 9.

```
#define N 8
#define E 4
#define HIBIT (1 << (N - 1))
#define EMASK ((1 << E) - 1)
#define MMASK ((1 << N) - 1 - EMASK)

#define DECODE_8_4(r) (r < HIBIT ? r : (r & MMASK) << (r & EMASK))
```

Figure 9: Decoding an (8,4) pseudo-FP value

Note that a pseudo-FP encoder needs to consider rounding; different applications of durations may favor rounding up or rounding down the value encoded in the message.

The highest pseudo-FP value, represented by an all-ones byte (0xFF), is reserved to indicate an indefinite duration. The next lower value (0xEF) is thus the highest representable value and is decoded as 7340032 seconds, a little more than 12 weeks.

B.3. Rationale

Where CPU power and memory is abundant, a duration can almost always be adequately represented by a non-negative floating-point number representing that number of seconds. Historically, many APIs have also used an integer representation, which limits both the resolution (e.g., if the integer represents the duration in seconds) and often the range (integer machine types have range limits that may become relevant). UNIX's "time_t" (which is used for both absolute time and durations) originally was a signed 32-bit value of seconds, but was later complemented by an additional integer to add microsecond ("struct timeval") and then later nanosecond ("struct timespec") resolution.

Three decisions need to be made for each application of the concept of duration:

- o the **resolution**. What rounding error is acceptable?
- o the **range**. What is the maximum duration that needs to be represented?
- o the **number of bits** that can be expended.

Obviously, these decisions are interrelated. Typically, a large range needs a large number of bits, unless resolution is traded. For most applications, the actual requirement for resolution are limited for longer durations, but can be more acute for shorter durations.

B.4. Pseudo-Floating Point

Constrained systems typically avoid the use of floating-point (FP) values, as

- o simple CPUs often don't have support for floating-point datatypes
- o software floating-point libraries are expensive in code size and slow.

In addition, floating-point datatypes used to be a significant element of market differentiation in CPU design; it has taken the industry a long time to agree on a standard floating point representation.

These issues have led to protocols that try to constrain themselves to integer representation even where the ability of a floating point representation to trade range for resolution would be beneficial.

The idea of introducing `_pseudo-FP_` is to obtain the increased range provided by embedding an exponent, without necessarily getting stuck with hardware datatypes or inefficient software floating-point libraries.

For the purposes of this draft, we define an (n,e)-pseudo-FP as a fixed-length value of n bits, e of which may be used for an exponent. Figure 8 illustrates an (8,4)-pseudo-FP value.

If the high bit is clear, the entire n-bit value (including the high bit) is the decoded value. If the high bit is set, the mantissa (including the high bit, but with the exponent field cleared out) is shifted left by the exponent to yield the decoded value.

The (n,e)-pseudo-FP format can be decoded with a single line of code (plus a couple of constant definition), as demonstrated in Figure 9.

Only non-negative numbers can be represented by this format. It is designed to provide full integer resolution for values from 0 to $2^{(n-1)}-1$, i.e., 0 to 127 in the (8,4) case, and a mantissa of $n-e$ bits from $2^{(n-1)}$ to $(2^n-2^e)*2^{(2^e-1)}$, i.e., 128 to 7864320 in the (8,4) case. By choosing e carefully, resolution can be traded against range.

Note that a pseudo-FP encoder needs to consider rounding; different applications of durations may favor rounding up or rounding down the value encoded in the message. This requires a little more than a single line of code (which is left as an exercise to the reader, as the most efficient expression depends on hardware details).

B.5. A Duration Type for CoAP

CoAP needs durations in a number of places. In [I-D.ietf-core-coap], durations occur in the option "Subscription-lifetime" as well as in the option "Max-age". (Note that the option "Date" is not a duration, but a point in time.) Other durations of this kind may be added later.

Most durations relevant to CoAP are best expressed with a minimum resolution of one second. More detailed resolutions are unlikely to provide much benefit.

The range of lifetimes and caching ages are probably best kept below the order of magnitude of months. An (8,4)-pseudo-FP has the maximum value of 7864320, which is about 91 days; this appears to be adequate for a subscription lifetime and probably even for a maximum cache age. Figure 10 shows the values that can be expressed. (If a larger range for the latter is indeed desired, an (8,5)-pseudo-FP could be used; this would last 15 milleniums, at the cost of having only 3 bits of accuracy for values larger than 127 seconds.)

Proposal: A single duration type is used throughout CoAP, based on an (8,4)-pseudo-FP giving a duration in seconds.

Benefits: Implementations can use a single piece of code for managing all CoAP-related durations.

In addition, length information never needs to be managed for durations that are embedded in other data structures: All durations are expressed by a single byte.

It might be worthwhile to reserve one duration value, e.g. 0xFF, for an indefinite duration.

Duration	Seconds	Encoded
----------	---------	---------

```
-----  
00:00:00 0x00000000 0x00  
00:00:01 0x00000001 0x01  
00:00:02 0x00000002 0x02  
00:00:03 0x00000003 0x03  
00:00:04 0x00000004 0x04  
00:00:05 0x00000005 0x05  
00:00:06 0x00000006 0x06  
00:00:07 0x00000007 0x07  
00:00:08 0x00000008 0x08  
00:00:09 0x00000009 0x09  
00:00:10 0x0000000a 0x0a  
00:00:11 0x0000000b 0x0b  
00:00:12 0x0000000c 0x0c  
00:00:13 0x0000000d 0x0d  
00:00:14 0x0000000e 0x0e  
00:00:15 0x0000000f 0x0f  
00:00:16 0x00000010 0x10  
00:00:17 0x00000011 0x11  
00:00:18 0x00000012 0x12  
00:00:19 0x00000013 0x13  
00:00:20 0x00000014 0x14  
00:00:21 0x00000015 0x15  
00:00:22 0x00000016 0x16  
00:00:23 0x00000017 0x17  
00:00:24 0x00000018 0x18  
00:00:25 0x00000019 0x19  
00:00:26 0x0000001a 0x1a  
00:00:27 0x0000001b 0x1b  
00:00:28 0x0000001c 0x1c  
00:00:29 0x0000001d 0x1d  
00:00:30 0x0000001e 0x1e  
00:00:31 0x0000001f 0x1f  
00:00:32 0x00000020 0x20  
00:00:33 0x00000021 0x21  
00:00:34 0x00000022 0x22  
00:00:35 0x00000023 0x23  
00:00:36 0x00000024 0x24  
00:00:37 0x00000025 0x25  
00:00:38 0x00000026 0x26  
00:00:39 0x00000027 0x27  
00:00:40 0x00000028 0x28  
00:00:41 0x00000029 0x29  
00:00:42 0x0000002a 0x2a  
00:00:43 0x0000002b 0x2b  
00:00:44 0x0000002c 0x2c  
00:00:45 0x0000002d 0x2d  
00:00:46 0x0000002e 0x2e
```

00:00:47	0x0000002f	0x2f
00:00:48	0x00000030	0x30
00:00:49	0x00000031	0x31
00:00:50	0x00000032	0x32
00:00:51	0x00000033	0x33
00:00:52	0x00000034	0x34
00:00:53	0x00000035	0x35
00:00:54	0x00000036	0x36
00:00:55	0x00000037	0x37
00:00:56	0x00000038	0x38
00:00:57	0x00000039	0x39
00:00:58	0x0000003a	0x3a
00:00:59	0x0000003b	0x3b
00:01:00	0x0000003c	0x3c
00:01:01	0x0000003d	0x3d
00:01:02	0x0000003e	0x3e
00:01:03	0x0000003f	0x3f
00:01:04	0x00000040	0x40
00:01:05	0x00000041	0x41
00:01:06	0x00000042	0x42
00:01:07	0x00000043	0x43
00:01:08	0x00000044	0x44
00:01:09	0x00000045	0x45
00:01:10	0x00000046	0x46
00:01:11	0x00000047	0x47
00:01:12	0x00000048	0x48
00:01:13	0x00000049	0x49
00:01:14	0x0000004a	0x4a
00:01:15	0x0000004b	0x4b
00:01:16	0x0000004c	0x4c
00:01:17	0x0000004d	0x4d
00:01:18	0x0000004e	0x4e
00:01:19	0x0000004f	0x4f
00:01:20	0x00000050	0x50
00:01:21	0x00000051	0x51
00:01:22	0x00000052	0x52
00:01:23	0x00000053	0x53
00:01:24	0x00000054	0x54
00:01:25	0x00000055	0x55
00:01:26	0x00000056	0x56
00:01:27	0x00000057	0x57
00:01:28	0x00000058	0x58
00:01:29	0x00000059	0x59
00:01:30	0x0000005a	0x5a
00:01:31	0x0000005b	0x5b
00:01:32	0x0000005c	0x5c
00:01:33	0x0000005d	0x5d
00:01:34	0x0000005e	0x5e

00:01:35	0x0000005f	0x5f
00:01:36	0x00000060	0x60
00:01:37	0x00000061	0x61
00:01:38	0x00000062	0x62
00:01:39	0x00000063	0x63
00:01:40	0x00000064	0x64
00:01:41	0x00000065	0x65
00:01:42	0x00000066	0x66
00:01:43	0x00000067	0x67
00:01:44	0x00000068	0x68
00:01:45	0x00000069	0x69
00:01:46	0x0000006a	0x6a
00:01:47	0x0000006b	0x6b
00:01:48	0x0000006c	0x6c
00:01:49	0x0000006d	0x6d
00:01:50	0x0000006e	0x6e
00:01:51	0x0000006f	0x6f
00:01:52	0x00000070	0x70
00:01:53	0x00000071	0x71
00:01:54	0x00000072	0x72
00:01:55	0x00000073	0x73
00:01:56	0x00000074	0x74
00:01:57	0x00000075	0x75
00:01:58	0x00000076	0x76
00:01:59	0x00000077	0x77
00:02:00	0x00000078	0x78
00:02:01	0x00000079	0x79
00:02:02	0x0000007a	0x7a
00:02:03	0x0000007b	0x7b
00:02:04	0x0000007c	0x7c
00:02:05	0x0000007d	0x7d
00:02:06	0x0000007e	0x7e
00:02:07	0x0000007f	0x7f
00:02:08	0x00000080	0x80
00:02:24	0x00000090	0x90
00:02:40	0x000000a0	0xa0
00:02:56	0x000000b0	0xb0
00:03:12	0x000000c0	0xc0
00:03:28	0x000000d0	0xd0
00:03:44	0x000000e0	0xe0
00:04:00	0x000000f0	0xf0
00:04:16	0x00000100	0x81
00:04:48	0x00000120	0x91
00:05:20	0x00000140	0xa1
00:05:52	0x00000160	0xb1
00:06:24	0x00000180	0xc1
00:06:56	0x000001a0	0xd1
00:07:28	0x000001c0	0xe1

00:08:00	0x000001e0	0xf1
00:08:32	0x00000200	0x82
00:09:36	0x00000240	0x92
00:10:40	0x00000280	0xa2
00:11:44	0x000002c0	0xb2
00:12:48	0x00000300	0xc2
00:13:52	0x00000340	0xd2
00:14:56	0x00000380	0xe2
00:16:00	0x000003c0	0xf2
00:17:04	0x00000400	0x83
00:19:12	0x00000480	0x93
00:21:20	0x00000500	0xa3
00:23:28	0x00000580	0xb3
00:25:36	0x00000600	0xc3
00:27:44	0x00000680	0xd3
00:29:52	0x00000700	0xe3
00:32:00	0x00000780	0xf3
00:34:08	0x00000800	0x84
00:38:24	0x00000900	0x94
00:42:40	0x00000a00	0xa4
00:46:56	0x00000b00	0xb4
00:51:12	0x00000c00	0xc4
00:55:28	0x00000d00	0xd4
00:59:44	0x00000e00	0xe4
01:04:00	0x00000f00	0xf4
01:08:16	0x00001000	0x85
01:16:48	0x00001200	0x95
01:25:20	0x00001400	0xa5
01:33:52	0x00001600	0xb5
01:42:24	0x00001800	0xc5
01:50:56	0x00001a00	0xd5
01:59:28	0x00001c00	0xe5
02:08:00	0x00001e00	0xf5
02:16:32	0x00002000	0x86
02:33:36	0x00002400	0x96
02:50:40	0x00002800	0xa6
03:07:44	0x00002c00	0xb6
03:24:48	0x00003000	0xc6
03:41:52	0x00003400	0xd6
03:58:56	0x00003800	0xe6
04:16:00	0x00003c00	0xf6
04:33:04	0x00004000	0x87
05:07:12	0x00004800	0x97
05:41:20	0x00005000	0xa7
06:15:28	0x00005800	0xb7
06:49:36	0x00006000	0xc7
07:23:44	0x00006800	0xd7
07:57:52	0x00007000	0xe7

	08:32:00	0x00007800	0xf7
	09:06:08	0x00008000	0x88
	10:14:24	0x00009000	0x98
	11:22:40	0x0000a000	0xa8
	12:30:56	0x0000b000	0xb8
	13:39:12	0x0000c000	0xc8
	14:47:28	0x0000d000	0xd8
	15:55:44	0x0000e000	0xe8
	17:04:00	0x0000f000	0xf8
	18:12:16	0x00010000	0x89
	20:28:48	0x00012000	0x99
	22:45:20	0x00014000	0xa9
1d	01:01:52	0x00016000	0xb9
1d	03:18:24	0x00018000	0xc9
1d	05:34:56	0x0001a000	0xd9
1d	07:51:28	0x0001c000	0xe9
1d	10:08:00	0x0001e000	0xf9
1d	12:24:32	0x00020000	0x8a
1d	16:57:36	0x00024000	0x9a
1d	21:30:40	0x00028000	0xaa
2d	02:03:44	0x0002c000	0xba
2d	06:36:48	0x00030000	0xca
2d	11:09:52	0x00034000	0xda
2d	15:42:56	0x00038000	0xea
2d	20:16:00	0x0003c000	0xfa
3d	00:49:04	0x00040000	0x8b
3d	09:55:12	0x00048000	0x9b
3d	19:01:20	0x00050000	0xab
4d	04:07:28	0x00058000	0xbb
4d	13:13:36	0x00060000	0xcb
4d	22:19:44	0x00068000	0xdb
5d	07:25:52	0x00070000	0xeb
5d	16:32:00	0x00078000	0xfb
6d	01:38:08	0x00080000	0x8c
6d	19:50:24	0x00090000	0x9c
7d	14:02:40	0x000a0000	0xac
8d	08:14:56	0x000b0000	0xbc
9d	02:27:12	0x000c0000	0xcc
9d	20:39:28	0x000d0000	0xdc
10d	14:51:44	0x000e0000	0xec
11d	09:04:00	0x000f0000	0xfc
12d	03:16:16	0x00100000	0x8d
13d	15:40:48	0x00120000	0x9d
15d	04:05:20	0x00140000	0xad
16d	16:29:52	0x00160000	0xbd
18d	04:54:24	0x00180000	0xcd
19d	17:18:56	0x001a0000	0xdd
21d	05:43:28	0x001c0000	0xed

22d	18:08:00	0x001e0000	0xfd
24d	06:32:32	0x00200000	0x8e
27d	07:21:36	0x00240000	0x9e
30d	08:10:40	0x00280000	0xae
33d	08:59:44	0x002c0000	0xbe
36d	09:48:48	0x00300000	0xce
39d	10:37:52	0x00340000	0xde
42d	11:26:56	0x00380000	0xee
45d	12:16:00	0x003c0000	0xfe
48d	13:05:04	0x00400000	0x8f
54d	14:43:12	0x00480000	0x9f
60d	16:21:20	0x00500000	0xaf
66d	17:59:28	0x00580000	0xbf
72d	19:37:36	0x00600000	0xcf
78d	21:15:44	0x00680000	0xdf
84d	22:53:52	0x00700000	0xef
91d	00:32:00	0x00780000	0xff (reserved)

Figure 10

Authors' Addresses

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

Phone: +49-421-218-63921
Fax: +49-421-218-7000
Email: cabo@tzi.org

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

Phone: +49-421-218-63905
Fax: +49-421-218-7000
Email: hartke@tzi.org

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 26, 2012

Z. Cao, Ed.
China Mobile
Y. Ma
Hitachi R&D China
H. Deng
China Mobile
October 24, 2011

HTTP-COAP Proxy Discovery using Link-format
draft-cao-core-pd-00

Abstract

This document discusses the problem of HTTP-COAP proxy discovery and proposes a method of using Link-format to do the job.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 26, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Requirements Language	3
2. Problem Formation	3
3. Link-format Proxy Discovery	4
4. Acknowledgements	5
5. IANA Considerations	5
6. Security Considerations	5
7. References	5
7.1. Normative References	5
7.2. Informative References	6
Authors' Addresses	6

1. Introduction

CoAP [I-D.ietf-core-coap] is a RESTful protocol designed for constrained devices. The ultimate goal of CoAP is to enable the "Web of Things" concept, which connects the smart sensor network with the global internet. Although CoAP has been implemented on various platforms, the rest of web is still dominated by HTTP. As a result, it is desirable to interconnect the HTTP and CoAP via some intermediary proxy. For example, the CoAP sensor client in the constrained network can access and update resources on the HTTP server, and also the HTTP client on the web can access and/or update resources on the CoAP server.

There are already some works discussing how to map HTTP to CoAP and vice versa. The basic mapping between HTTP and CoAP is described in Section 8 of [I-D.ietf-core-coap] . Further details of implementing the proxy, internal procedures and design choices are described in [I-D.castellani-core-http-mapping] .

Static configuration of HTTP-CoAP proxies is a straightforward way for the client to access the server. However, in many situations, static configuration is not enough to meet the requirements. For example, if the HTTP client would like to access a certain type of resource (temperature or humidity in a certain location, etc.), it is required that the client would find an appropriate proxy to serve the content.

1.1. Requirements Language

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

2. Problem Formation

We divide the problem into two separated parts. The first is how a CoAP client discovers a proxy to access the HTTP server. For example, the CoAP sensors want to report or get some information to a Web server. In this case, the CoAP sensor only acts as a client. In static configuration, the CoAP client is configured via DHCP or RSRA. But in dynamic environment, a mechanism for dynamic configuration is desired. This document mainly discusses this aspect.

The other case is how a HTTP client discovers a proxy to access the CoAP server. For example, the HTTP client wants to access a certain type of information in the constrained network, and would discover the proxy to the exact constrained sensor. In this case, the HTTP

Client only accesss the sensor indirectly. In this case, the HTTP Client only needs to know the address or the domain name of the proxy node, and the proxy forwards the requests to the sensor node according to the sub-domain information or the path included in the URI within the request. But in this case, we believe that the DNS-SD infrastructures are sufficient to handle this problem. For example, [I-D.vanderstok-core-bc] has described detailed considerations of a DNS-SD based proxy discovery method for Building Control use cases. So, in this document we will not talk about this direction.

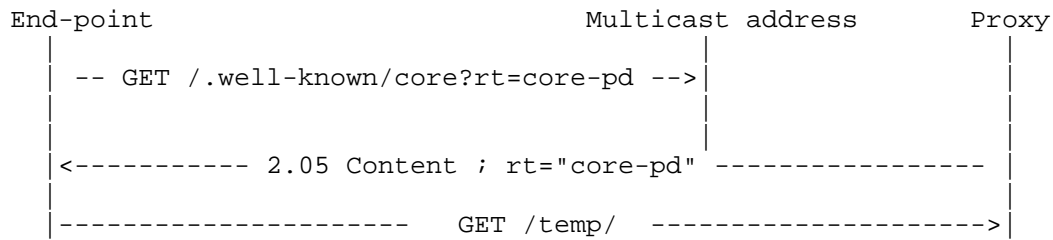
3. Link-format Proxy Discovery

Before the CoAP sensor makes use of the CoAP-HTTP proxy, it must know the location of the proxy. There can be multiple ways to discover the proxy's location, including both static and dynamic methods. DHCP is one way to do that, and documented in another document. This document describes one way to discover the proxy by the CoRE link format [I-D.ietf-core-link-format].

Note: Think of the way the user is configured with the http proxy in the enterprise network.

Discovery is performed by sending a multicast GET request to /.well-known/core and including a Resource Type (rt) parameter [I-D.ietf-core-link-format] with the value "core-pd" in the query string. Upon success, the response will contain a payload with a link format entry for each proxy discovered. The multicast IP address used will depend on the scope required and the multicast capabilities of the network. (If determined, IANA actions are required to assign a multicast address for this purpose)

The following example shows an end-point discover a locally available CoAP-HTTP proxy. The CoAP end-point sends a multicast GET request to the multicast address in the domain carrying a resource type "core-pd" indicating its discovery of a local proxy. Then the serving proxy responds the request with the rt="core-pd" and the address of the proxy is carried within the Content payload. Afterwards, the CoAP sensor initiates the data-plane communication with the proxy directly.



Req: GET coap://[ff02::1]/.well-known/core?rt=core-pd

Res: 2.05 Content
fe80::ff; rt="core-pd";

4. Acknowledgements

Some ideas in this document are according to the discussion between Zach Shelby on the problem.

5. IANA Considerations

If the ideas in this document is determined by the working group, IANA actions are required to assign a multicast address for the purpose of HTTP-CoAP proxy discovery.

6. Security Considerations

TBD.

7. References

7.1. Normative References

- [I-D.castellani-core-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best practices for HTTP-CoAP mapping implementation", draft-castellani-core-http-mapping-01 (work in progress), July 2011.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), July 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format",
draft-ietf-core-link-format-07 (work in progress),
July 2011.

[I-D.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building
Control", draft-vanderstok-core-bc-04 (work in progress),
July 2011.

7.2. Informative References

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

Authors' Addresses

Zhen Cao (editor)
China Mobile
China,
China

Phone:
Email: zehn.cao@gmail.com

Yuanchen Ma
Hitachi R&D China

Phone:
Fax:
Email: ycma@hitachi.cn
URI:

Hui Deng
China Mobile

Phone:
Fax:
Email: denghui@chinamobile.com
URI:

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: May 3, 2012

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
KoanLogic
E. Dijk
Philips Research
October 31, 2011

Best practices for HTTP-CoAP mapping implementation
draft-castellani-core-http-mapping-02

Abstract

This draft aims at being a base reference documentation for HTTP-CoAP proxy implementors. It details deployment options, discusses possible approaches for URI mapping, and provides useful considerations related to protocol translation.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 3, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	4
2. Terminology	4
3. Cross-protocol resource identification using URIs	6
3.1. URI mapping	7
3.1.1. Homogeneous mapping	7
3.1.2. Embedded mapping	8
4. HTTP-CoAP implementation	8
4.1. Placement and deployment	8
4.2. Basic mapping	11
4.2.1. Caching and congestion control	11
4.2.2. Cache Refresh via Observe	12
4.2.3. Use of CoAP blockwise transfer	13
4.2.4. Use case: HTTP/IPv4-CoAP/IPv6 proxy	13
4.3. Multiple message exchanges mapping	16
4.3.1. Relevant features of existing standards	16
4.3.2. Multicast mapping	17
4.3.3. Multicast responses caching	20
4.3.4. Subscription mapping	21
5. CoAP-HTTP implementation	21
5.1. Placement and Deployment	21
5.2. Basic mapping	22
5.2.1. Payloads and Media Types	23
5.2.2. Max-Age and ETag Options	23
5.2.3. Use of CoAP blockwise transfer	23
5.2.4. HTTP Status Codes lxx and 3xx	23
5.2.5. Examples	24
6. Security Considerations	25
6.1. Traffic overflow	26
6.2. Cross-protocol security policy mapping	27
6.3. Handling secured exchanges	27
6.4. Spoofing and Cache Poisoning	27
6.5. Subscription	28
7. Acknowledgements	28
8. References	29
8.1. Normative References	29
8.2. Informative References	30
Appendix A. Internal Mapping Functions (from an implementer's perspective)	30
A.1. URL Map Algorithm	31
A.2. Security Policy Map Algorithm	32
A.3. Content-Type Map Algorithm	33
Authors' Addresses	33

1. Introduction

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

A reference about the mapping process is provided in Section 8 of [I-D.ietf-core-coap]. However, depending on the involved application, deployment scenario, or network topology, such mapping could be realized using a wide range of intermediaries.

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

This draft is organized as follows:

- o Section 2 describes terminology to identify different mapping approaches and the related proxy deployments;
- o Section 3 discusses impact of the mapping on URI and describes notable options;
- o Section 4 and Section 5 respectively analyze the mapping from HTTP to CoAP and viceversa;
- o Section 6 discusses possible security impact related to the mapping.

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

2. Terminology

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

Regular HTTP proxies are usually same-protocol proxies, because they can map from HTTP to HTTP. CoAP same-protocol proxies are intermediaries for CoAP to CoAP exchanges. However the discussion about these entities is out-of-scope of this document.

At least two different kinds of HC proxies exist:

- o One-way cross-protocol proxy (1-way proxy): This proxy translates from a client of a protocol to a server of another protocol but

not vice-versa.

- o Two-way (or bidirectional) cross-protocol proxy (2-way proxy):
This proxy translates from a client of both protocols to a server supporting one protocol.

1-way and 2-way HC proxies are realized using the following general types of proxies:

Forward proxy (F): Is a proxy known by the client (either CoAP or HTTP) used to access a specific cross-protocol server (respectively HTTP or CoAP). Main feature: server(s) do not require to be known in advance by the proxy (ZSC: Zero Server Configuration).

Reverse proxy (R): Is a proxy known by the client to be the server, however for a subset of resources it works as a proxy, by knowing the real server(s) serving each resource. When a cross-protocol resource is accessed by a client, the request will be silently forwarded by the reverse proxy to the real server (running a different protocol). If a response is received by the reverse proxy, it will be mapped, if possible, to the original protocol and sent back to the client. Main feature: client(s) do not require to know in advance the proxy (ZCC: Zero Client Configuration).

Interception proxy (I): This proxy [RFC3040] can intercept any origin protocol request (HTTP or CoAP) and map it to the destination protocol, without any kind of knowledge about the client or server involved in the exchange. Main feature: client(s) and server(s) do not require to know or be known in advance by the proxy (ZCC and ZSC).

The proxy can be placed in the network at three different logical locations:

Server-side proxy (SS): A proxy placed on the same network domain of the server;

Client-side proxy (CS): A proxy placed on the same network domain of the client;

External proxy (E): A proxy placed in a network domain external to both endpoints, it is in the network domain neither of the client nor of the server.

3. Cross-protocol resource identification using URIs

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

URIs are formed of at least three components: scheme, authority and path. The scheme is the first part of the URI, and it often corresponds to the protocol used to access the resource. However, as noted in Section 1.2.2 of [RFC3986] the scheme does not imply that a particular protocol is used to access the resource.

Clients using URIs to identify target resources (e.g. HTTP web browsers) may support only a limited set of schemes (i.e. 'http', 'https'). If such clients need to interoperate with resources identified by an unsupported scheme (e.g. 'coap'), the existence of a URI using a scheme supported by the client is required for interoperability.

Both CoAP and HTTP implement the REST paradigm, so, in principle, the same resource can be made available in each protocol if protocol translation is applied.

In general two different procedures can be used to access cross-protocol resources:

Protocol-aware access: The client accesses the cross-protocol resource using the original URI using a cross-protocol proxy (e.g. uses 'coap' scheme URI inside the HTTP request); protocol translation is provided by a cross-protocol proxy. Both CoAP and HTTP support this access method. HTTP defines that proxy or servers MUST accept even an absolute-URI as request-target, see Section 4.1.2 of [I-D.ietf-httpbis-pl-messaging]. CoAP provides Proxy-URI option having absolute-URI as value, see Section 5.10.3 of [I-D.ietf-core-coap].

Protocol-agnostic access: The client accesses the cross-protocol resource using an URI with a scheme supported by the client (e.g. uses 'http' scheme to access a CoAP resource), URI and protocol translation is provided by a cross-protocol proxy. In order to use this method a URI identifying an equivalent resource MUST exist, and SHOULD be provided by the cross-protocol proxy.

URI mapping is NOT required when using protocol-aware access, the following section is focused on URI mapping techniques for protocol-agnostic access.

3.1. URI mapping

When accessing cross-protocol resources in a protocol-agnostic way, clients MUST use an URI with a scheme supported by the client.

Since determination of equivalence of URIs (e.g. whether or not they identify the same resource) is based on lexicographic comparison, URI domains using different schemes are fully distinct: resources identified by the same authority and path tuple change when switching the scheme.

Example: Assume that the following resource exists - "coap://node.coap.something.net/foo". The resource identified by "http://node.coap.something.net/foo" may not exist or be non-equivalent to the one identified by the 'coap' scheme.

If a cross-protocol URI exists providing an equivalent representation of the native protocol resource, it can be provided by a different URI (in terms of authority and path). The mapping of an URI between HTTP and CoAP is said HC URI mapping.

Example: The HC URI mapping to HTTP of the CoAP resource identified by "coap://node.coap.something.net/foo" is "http://node.something.net/foobar".

The process of providing the HC URI mapping could be complex, since a proper mechanism to statically or dynamically (discover) map the resource HC URI mapping is required.

Two static HC URI mappings are discussed in the following subsections.

3.1.1. Homogeneous mapping

The URI mapping between CoAP and HTTP is called homogeneous, if the same resource is identified by URIs with different schemes.

Example: The CoAP resource "//node.coap.something.net/foo" identified either by the URI "coap://node.coap.something.net/foo", and or by the URI "http://node.coap.something.net/foo" is the same. When the resource is accessed using HTTP, the mapping from HTTP to CoAP is performed by an HC proxy

When homogeneous HC URI mapping is available, HC-I proxies are easily implementable.

3.1.2. Embedded mapping

The mapping is said to be embedded, if the HC URI mapping of the resource embeds inside it the authority and path part of the native URI.

Example: The CoAP resource "coap://node.coap.something.net/foo" can be accessed at "http://hc-proxy.something.net/coap/node.coap.something.net/foo".

This mapping technique can be used to reduce the mapping complexity in an HC reverse proxy.

3.1.2.1. HTML5 scheme handler registration

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

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

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

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

4. HTTP-CoAP implementation

4.1. Placement and deployment

In typical scenarios the HC proxy is expected to be server-side (SS), in particular deployed at the edge of the constrained network.

The arguments supporting SS placement are the following:

TCP/UDP: Translation between HTTP and CoAP requires also a TCP to UDP mapping; UDP performance over the unconstrained Internet may not be adequate. In order to minimize the number of required retransmissions and overall reliability, TCP/UDP conversion SHOULD be performed at a SS placed proxy.

Caching: Efficient caching requires that all the CoAP traffic is intercepted by the same proxy, thus an SS placement, collecting all the traffic, is strategical for this need.

Multicast: To support using local-multicast functionalities available in the constrained network, the HC proxy MAY require a network interface directly attached to the constrained network.

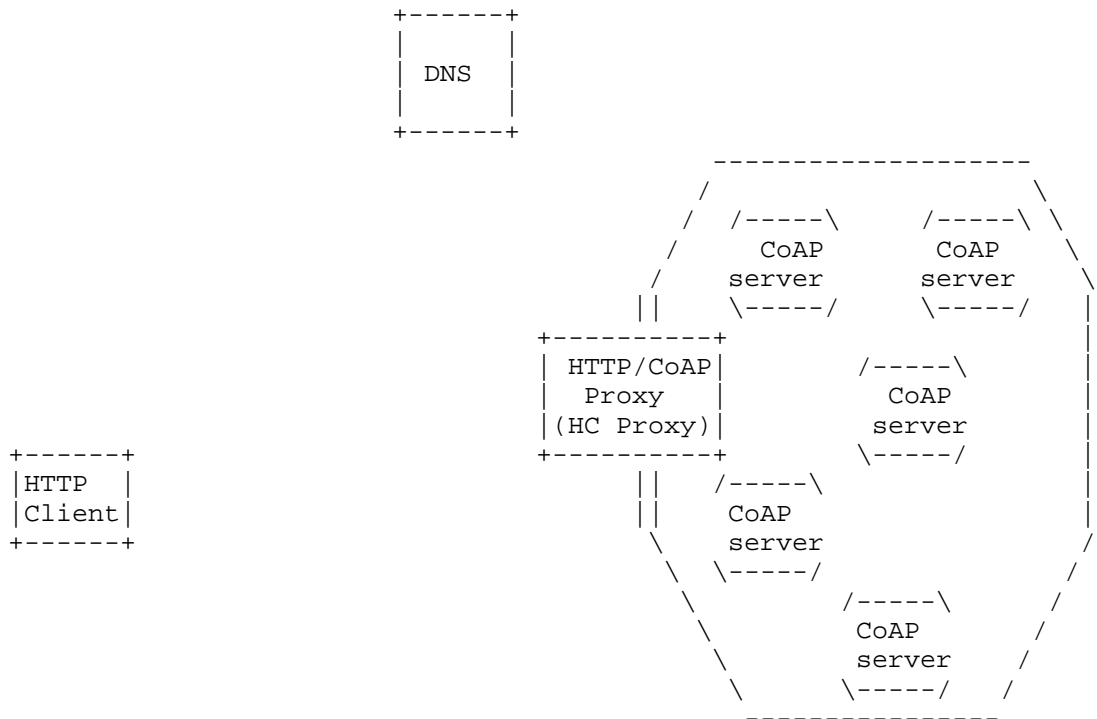


Figure 1: Server-side HC proxy deployment scenario

Other important aspects involved in the selection of which type of proxy deployment, whose choice impacts its placement too, are the following:

Client/Proxy/Network configuration overhead: Forward proxies require either static configuration or discovery support in the client. Reverse proxies require either static configuration, server discovery or embedded URI mapping in the proxy. Interception proxies typically require single router configuration for a whole network.

Scalability/Availability: Both aspects are typically addressed using redundancy. CS deployments, due to the limited catchment area and administrative-wide domain of operation, have looser requirements on this. SS deployments, in dense/popular/critical environments, have stricter requirements and MAY need to be replicated. Stateful proxies (e.g. reverse) may be complex to replicate.

Discussion about security impacts of different deployments is covered in Section 6.

Table 1 shows some interesting HC proxy deployment scenarios, and notes the advantages related to each scenario.

Feature	F CS	R SS	I SS
TCP/UDP	-	+	+
Multicast	-	+	+
Caching	-	+	+
Scalability/Availability	+	+/-	+
Configuration	-	-	+

Table 1: Interesting HC proxy deployments

Guidelines proposed in the previous paragraphs have been used to fill out the above table. In the first three rows, it can be seen that SS deployment is preferred versus CS. Scalability/Availability issues can be generally handled, but some complexity may be involved in reverse proxies scenarios. Configuration overhead could be simplified when interception proxies deployments are feasible.

When support for legacy HTTP clients is required, it may be preferable using configuration/discovery free deployments. Discovery procedures for client or proxy auto-configuration are still under active-discussion: see [I-D.vanderstok-core-bc], [I-D.bormann-core-simple-server-discovery] or [I-D.shelby-core-resource-directory]. Static configuration of multiple forward proxies is typically not feasible in existing HTTP clients.

4.2. Basic mapping

The mapping of HTTP requests to CoAP and of the response back to HTTP is defined in Section 8.2 of [I-D.ietf-core-coap].

The mapping of a CoAP response code to HTTP is not straightforward, this mapping **MUST** be operated accordingly to Table 4 of [I-D.ietf-core-coap].

No temporal upper bound is defined for a CoAP server to provide the response, thus for long delays the HTTP client or any other proxy in between **MAY** timeout. Further discussion is available in Section 7.1.4 of [I-D.ietf-httpbis-pl-messaging].

The HC proxy **MUST** define an internal timeout for each pending CoAP request, because the CoAP server may silently die before completing the request.

Even if the DNS protocol may not be used inside the constrained network, maintaining valid DNS entries describing the hosts available on such network helps offering the CoAP resources to HTTP clients.

An example of the usefulness of such entries is described in Section 4.2.4.

HTTP connection pipelining (section 7.1.2.2 of [I-D.ietf-httpbis-pl-messaging]) is transparent to the CoAP network: the HC proxy will sequentially serve the pipelined requests by issuing different CoAP requests.

4.2.1. Caching and congestion control

The HC proxy **SHOULD** limit the number of requests to CoAP servers by responding, where applicable, with a cached representation of the resource.

Duplicate idempotent pending requests to the same resource **SHOULD** in general be avoided, by duplexing the response to the relevant hosts without duplicating the request.

If the HTTP client times out and drops the HTTP session to the proxy (closing the TCP connection), the HC proxy **SHOULD** wait for the response and cache it if possible. Further idempotent requests to the same resource can use the result present in cache, or, if a response has still to come, requests will wait on the open CoAP session.

Resources experiencing a high access rate coupled with high

volatility MAY be observed [I-D.ietf-core-observe] by the HC proxy to keep their cached representation fresh while minimizing the number of needed messages. See Section 4.2.2 for a heuristics that enables the HC proxy to decide whether observing is a more convenient strategy than ordinary refreshing via Max-Age/ETag-based mechanisms.

Specific deployments may show highly congested servers/resources -- e.g. multicast resources (see Section 4.3.2), popular servers, etc. A careful analysis is required to pick the correct caching policy involving these resources, also taking into consideration the security implications that may impact these targets specifically, and the constrained network in general.

To this end when traffic reduction obtained by the caching mechanism is not adequate, the HC proxy could apply stricter policing by limiting the amount of aggregate traffic to the constrained network. In particular, the HC proxy SHOULD pose a rigid upper limit to the number of concurrent CoAP request pending on the same constrained network; further request MAY either be queued or dropped. In order to efficiently apply this congestion control, the HC proxy SHOULD be SS placed.

Further discussion on congestion control can be found in [I-D.eggert-core-congestion-control].

4.2.2. Cache Refresh via Observe

There are cases where using CoAP observe protocol [I-D.ietf-core-observe] to handle proxy cache refresh may be preferable to the validation mechanism based on ETag's defined in section 5.6.2 of [I-D.ietf-core-coap]. For example: sleeping nodes, possibly showing high variance in requests' distribution, would greatly benefit from a server driven cache update mechanism. Other expected candidates would be the crowded or very low throughput networks, where minimization of the total number of exchanged messages is a major goal.

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

Let T_R be the mean time between two client requests to resource R, F_R be the freshness lifetime of R, and M_R be the total number of messages exchanged by the cache towards resource R in order to validate its freshness. Assumed a negligible initial cost for establishing the observation relationship (one only message), an observation on R lessens M_R (i.e. it's a better cache update choice then using ETag validation) iff $T_R < 2 * F_R$, or equivalently, iff the

mean arrival time of requests for resource R is greater than half the refresh rate of R.

The above relation can easily be grasped by noticing that, in case the mean interarrival time between requests is less than the refresh rate of R, an observation does not generate any unnecessary validation message, and is therefore optimal. Further, since the number of messages used by ETag's validation is twice the observation cost (request/response vs server push), the bound on T_R can be doubled.

As a rule of thumb, volatile resources (i.e. those showing tiny freshness lifetime) with access rate in the order of half their refresh rate, are good candidates for implementing observer-based cache refresh. Imagine a sensor providing one new value every second, and clients accessing it on average once every 1.5 seconds: in one single day of usage, 28800 messages may have been saved if HC establishes an observation on the sensor resource.

4.2.3. Use of CoAP blockwise transfer

An HC proxy SHOULD support CoAP blockwise transfers [I-D.ietf-core-block] to allow transport of large CoAP payloads while avoiding link-layer fragmentation in LLNs, and to cope with small datagram buffers in CoAP end-points as described in [I-D.ietf-core-coap]. An HC proxy SHOULD attempt to retry a CoAP PUT or POST request with a payload using blockwise transfer if the destination CoAP server responded with 4.13 (Request Entity Too Large) to the original request. An HC proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than BLOCKWISE_THRESHOLD. The value of BLOCKWISE_THRESHOLD is implementation-specific, for example it may be set by an administrator, preset to a known or typical UDP datagram buffer size for CoAP end-points, to N times the size of a link-layer frame where e.g. N=5, preset to a known IP MTU value, or set to a known Path MTU value.

For improved latency an HC proxy MAY initiate a blockwise CoAP request triggered by an incoming HTTP request even when the HTTP request message has not yet been fully received, but enough data has been received to send one or more data blocks to a CoAP server already.

4.2.4. Use case: HTTP/IPv4-CoAP/IPv6 proxy

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

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

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

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

Figure 2 shows an HTTP client on IPv4 (C) accessing a CoAP server on IPv6 (S) through an HC proxy on IPv4/IPv6 (P). The DNS has an A record for "node.coap.something.net" resolving to the IPv4 address of the HC proxy, and an AAAA record with the IPv6 address of the CoAP server.

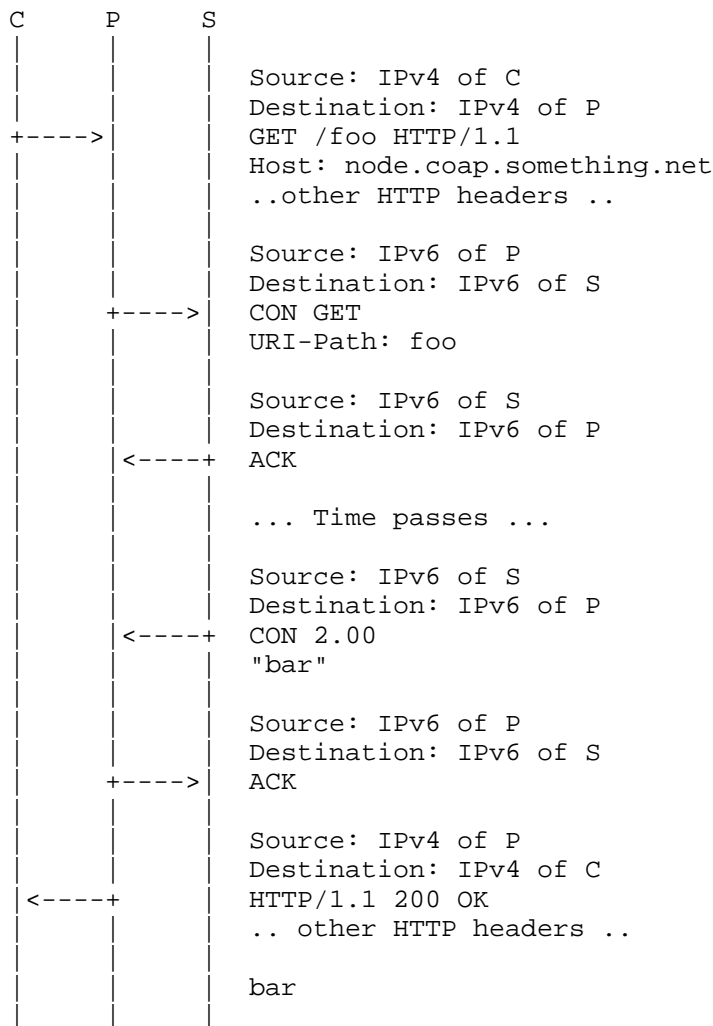


Figure 2: HTTP/IPv4 to CoAP/IPv6 mapping

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

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

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

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

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

4.3. Multiple message exchanges mapping

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

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

4.3.1. Relevant features of existing standards

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

4.3.1.1. Multipart messages

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

4.3.1.2. Immediate message delivery

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

A detailed discussion on the use of chunked Transfer-Encoding to stream data over HTTP can be found in [RFC6202]. Large delays between chunks can lead the HTTP session to timeout, more details on

this issue can be found in [I-D.thomson-hybi-http-timeout].

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

An example showing immediate delivery of CoAP responses using HTTP chunks will be provided in Section 4.3.4, while describing its application to an observe session.

4.3.1.3. Detailing source information

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

4.3.2. Multicast mapping

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

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

4.3.2.1. URI identification and mapping

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

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

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

When using IPv6 multicast paired with DNS, the mapping to IPv6 multicast is simply done using DNS resolution. If the group

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

4.3.2.2. Request handling

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

Multicast request handling consists of the following steps:

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

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

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

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

4.3.2.3. Example

Figure 3 shows an HTTP client (C) requesting the resource "/foo" to a group of CoAP servers (S1/S2/S3) through an HC proxy (P) which uses IP multicast to send the corresponding CoAP request.

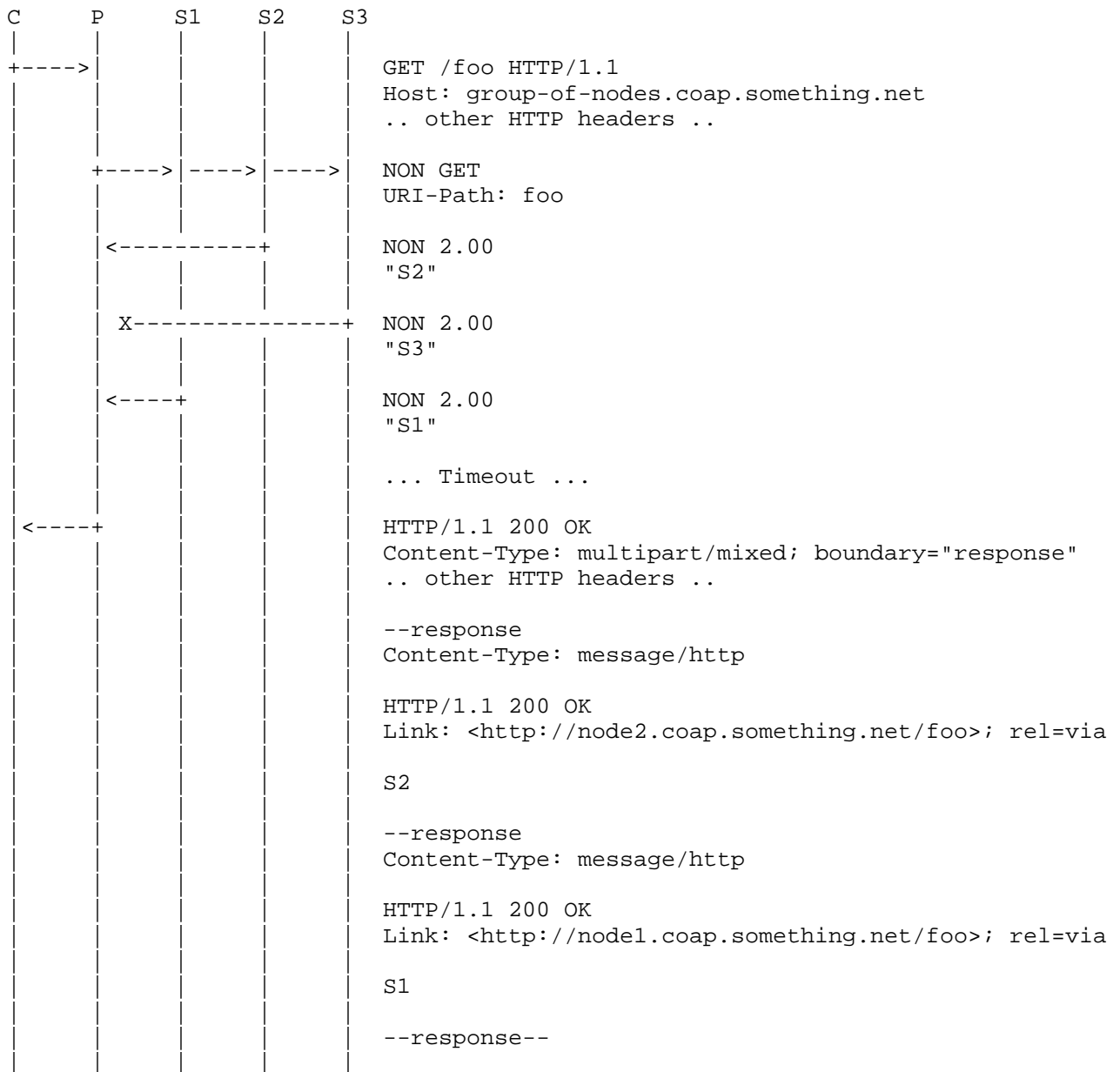


Figure 3: Unicast HTTP to multicast CoAP mapping

The example proposed in the above diagram does not make any assumption on which underlying group communication technology is

available in the constrained network. Some detailed discussion is provided about it along the following lines.

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

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

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

4.3.3. Multicast responses caching

We call perfect caching when the proxy uses only the cached representations to provide a response to the HTTP client. In the case of a multicast CoAP request, perfect caching is not adequate. This section updates the general caching guidelines of Section 4.2.1 with specific guidelines for the multicast use case.

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

Caching of multicast responses is still a valuable goal to pursue reduce network congestion, battery consumption and response latency. Some considerations to be performed when adopting a multicast caching behaviour are outlined in the following paragraph.

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

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

4.3.4. Subscription mapping

TBD

5. CoAP-HTTP implementation

The CoAP protocol [I-D.ietf-core-coap] allows CoAP clients to request CoAP proxies to perform an HTTP request on their behalf. This is accomplished by the CoAP client populating an HTTP absolute URI in the 'Proxy-URI' option of the CoAP request to the CoAP proxy. An absolute URI is an HTTP URI that does not contain a fragment component [RFC3986]. The proxy then composes an HTTP request with the given URI and sends it to the appropriate HTTP origin server. The server then returns the HTTP response to the proxy, which the proxy returns to the CoAP client via a CoAP response

5.1. Placement and Deployment

In typical scenarios, for communication from a CoAP client to an HTTP origin server, the HC proxy is expected to be located on the client-side (CS). Specifically, the HC proxy is expected to be deployed at the edge of the constrained network as shown in Figure 4.

The arguments supporting CS placement are as follows:

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

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

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

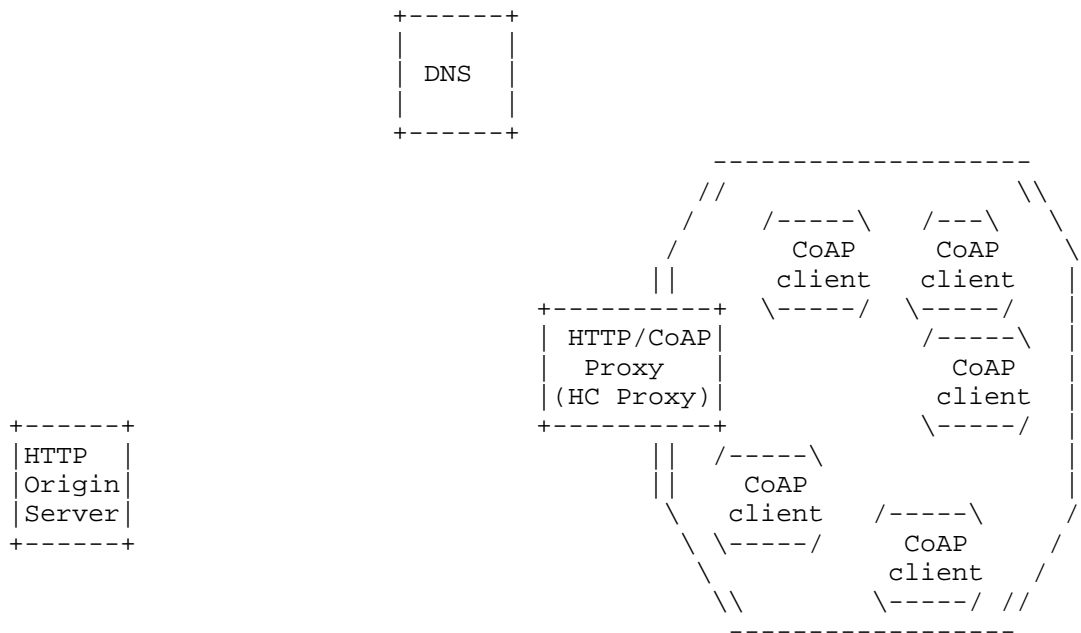


Figure 4: Client-side HC proxy deployment scenario

5.2. Basic mapping

The basic mapping of CoAP methods to HTTP is defined in [I-D.ietf-core-coap]. Specifically the {GET, PUT, POST, DELETE} set of CoAP methods are mapped to the equivalent HTTP methods.

In general, an implementation will translate and forward CoAP requests to the HTTP origin server and translate back HTTP responses to CoAP responses, typically employing a certain amount of caching to make this translation more efficient. This section gives some hints for implementing the translation. In addition, some examples are given to illustrate the mappings.

5.2.1. Payloads and Media Types

CoAP supports only a subset of media types. A proxy should convert payloads and approximate content-types as closely as possible. For example, if a HTTP server returns a resource representation in "text/plain; charset=iso-8859-1" format, the proxy should convert the payload to "text/plain; charset=utf-8" format. If conversion is not possible, the proxy can specify a media type of "application/octet-stream".

5.2.2. Max-Age and ETag Options

The proxy can determine the Max-Age Option for responses to GET requests by calculating the freshness lifetime (see Section 13.2.4 of [RFC2616]) of the HTTP resource representation retrieved. The Max-Age Option for responses to POST, PUT or DELETE requests should always be set to 0.

The proxy can assign entity tags to responses it sends to a client. These can be generated locally, if the proxy employs a cache, or be derived from the ETag header field in a response from the HTTP origin server, in which case the proxy can optimize future requests to the HTTP by using Conditional Requests. Note that CoAP does not support weak entity tags.

5.2.3. Use of CoAP blockwise transfer

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

For improved latency a CH proxy MAY initiate a HTTP request triggered by an incoming blockwise CoAP request even when blocks of the CoAP request have only been partially received by the proxy, in cases where the Content-Length field is not going to be used in the HTTP request. This is useful especially if the network between proxy and HTTP server involves low-bandwidth links.

5.2.4. HTTP Status Codes 1xx and 3xx

CoAP does not have provisional responses (HTTP Status Codes 1xx) or responses indicating that further action needs to be taken (HTTP Status Codes 3xx). When a proxy receives such a response from the HTTP server, the response should cause the proxy to complete the request, for example, by following redirects. If the proxy is unable or unwilling to do so, it can return a 5.02 (Bad Gateway) error.

5.2.5. Examples

Figure 5 shows an example implementation of a basic CoAP GET request with an HTTP URI as the value of a Proxy-URI option. The proxy retrieves a representation of the target resource from the HTTP origin server. It converts the payload to a UTF-8 charset, calculates the Max-Age Option from the Expires header field, and derives an entity-tag from the ETag header field.

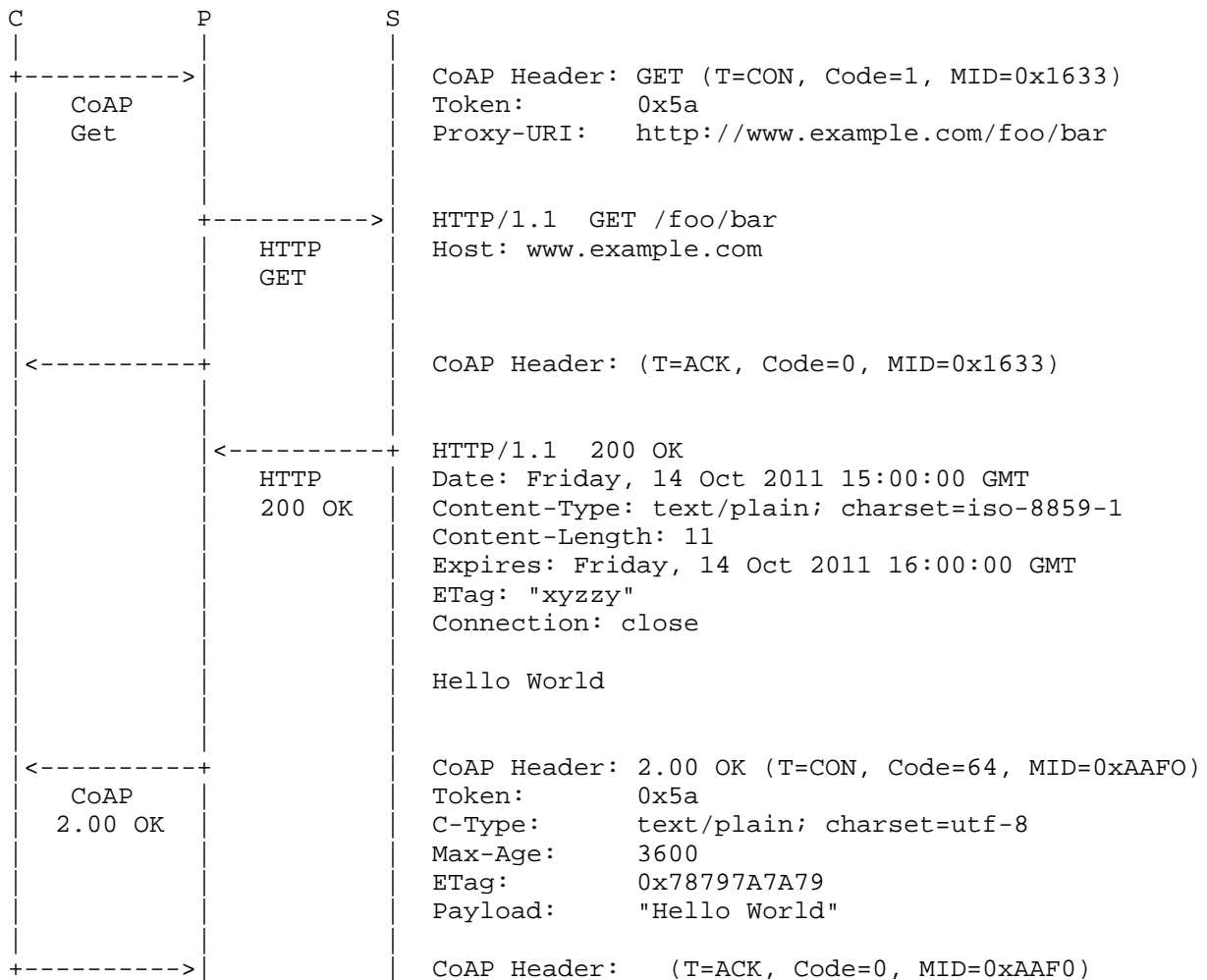


Figure 5: A basic CoAP-HTTP GET request

The example in Figure 6 builds on the previous example and shows an implementation of a GET request that includes a previously returned ETag Option. The proxy makes a Conditional Request to the HTTP origin server by including an If-None-Match header field in the HTTP GET Request. The CoAP response indicates that the response stored by the client is fresh. It includes a Max-Age Option calculated from the HTTP response's Expires header field.

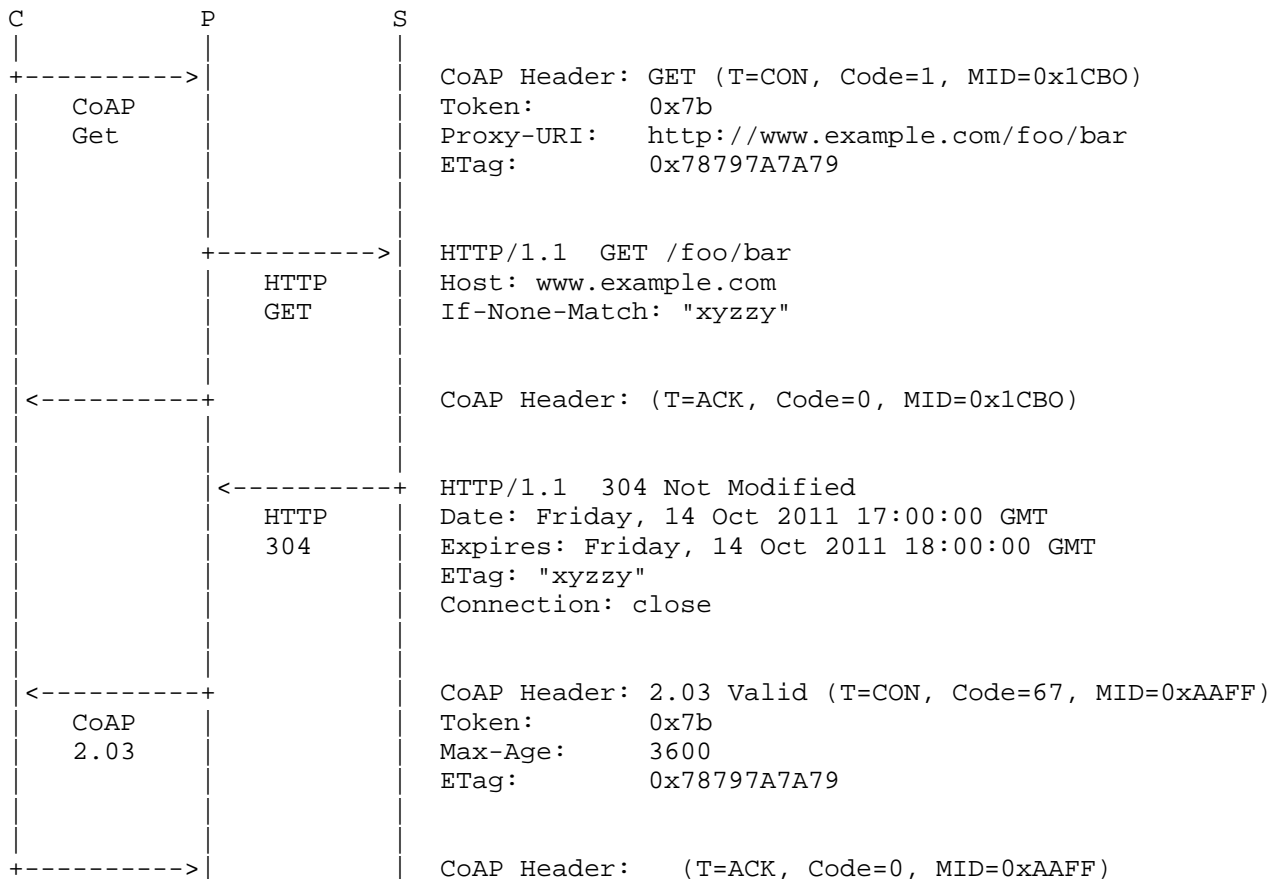


Figure 6: A CoAP-HTTP GET request with an ETag Option

6. Security Considerations

The security concerns raised in Section 15.7 of [RFC2616] also apply to the HC proxy scenario. In fact, the HC proxy is a trusted (not

rarely a transparently trusted) component in the network path.

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

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

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

6.1. Traffic overflow

Due to the typically constrained nature of CoAP nodes, particular attention SHOULD be posed in the implementation of traffic reduction mechanisms (see Section 4.2.1), because inefficient implementations can be targeted by unconstrained Internet attackers. Bandwidth or complexity involved in such attacks is very low.

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

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

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

6.2. Cross-protocol security policy mapping

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

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

6.3. Handling secured exchanges

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

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

The HC URI mapping MUST NOT map to HTTP (see Section 3.1) a CoAP resource intended to be accessed only using HTTPS.

A secured connection that is terminated at the HC proxy, i.e. the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The HC proxy SHOULD NOT cache any secured content to avoid any leak of secured information. However in some specific scenario, a security/efficiency trade-off could motivate caching secured information; in that case the caching behavior MAY be tuned to some extent on a per-resource basis (see Section 6.2).

6.4. Spoofing and Cache Poisoning

In web security jargon, the "cache poisoning" verb accounts for attacks where an evil user causes the proxy server to associate incorrect content to a cached resource, which work through especially crafted HTTP requests or request/response combos.

When working in CoAP NoSec mode, the use of UDP makes cache poisoning on the constrained network easy and effective, simple address

spoofing by a malicious host is sufficient to perform the attack. The implicit broadcast nature of typical link-layer communication technologies used in constrained networks lead this attack to be easily performed by any host, even without the requirement of being a router in the network. The ultimate outcome depends on both the order of arrival of packets (legitimate and rogue) and the processing/discarding policy at the CoAP node; attackers targeting this weakness may have less requirements on timing, thus leading the attack to succeed with high probability.

In case the threat of a rogue mote acting in the constrained network can't be winded up by appropriate procedural means, the only way to avoid such attacks is for any CoAP server to work at least in MultiKey mode with a 1:1 key with the HC proxy. SharedKey mode would just mitigate the attack, since a guessable MIDs and Tokens generation function at the HC proxy side would make it feasible for the evil mote to implement a "try until succeed" strategy. Also, (authenticated) encryption at a lower layer (MAC/PHY) could be defeated by a slightly more powerful attacker, a compromised router mote.

6.5. Subscription

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

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

7. Acknowledgements

Special credit is given to Klaus Hartke who provided the text for Section 5 and a lot of direct input to this document. Special credit about the text in Section 5 is given to Carsten Bormann who provided parts of it.

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

8. References

8.1. Normative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",
draft-ietf-core-block-04 (work in progress), July 2011.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-07 (work in progress), July 2011.
- [I-D.ietf-core-observe]
Hartke, K. and Z. Shelby, "Observing Resources in CoAP",
draft-ietf-core-observe-02 (work in progress), March 2011.
- [I-D.ietf-httpbis-pl-messaging]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H.,
Masinter, L., Leach, P., Berners-Lee, T., Reschke, J., and
Y. Lafon, "HTTP/1.1, part 1: URIs, Connections, and
Message Parsing", draft-ietf-httpbis-pl-messaging-16 (work
in progress), August 2011.
- [I-D.rahman-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP",
draft-rahman-core-groupcomm-07 (work in progress),
October 2011.
- [I-D.thomson-hybi-http-timeout]
Thomson, M., Loreto, S., and G. Wilkins, "Hypertext
Transfer Protocol (HTTP) Timeouts",
draft-thomson-hybi-http-timeout-00 (work in progress),
March 2011.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail
Extensions (MIME) Part Two: Media Types", RFC 2046,
November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,

RFC 3986, January 2005.

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

8.2. Informative References

- [I-D.bormann-core-simple-server-discovery]
Bormann, C., "CoRE Simple Server Discovery",
draft-bormann-core-simple-server-discovery-00 (work in
progress), March 2011.
- [I-D.eggert-core-congestion-control]
Eggert, L., "Congestion Control for the Constrained
Application Protocol (CoAP)",
draft-eggert-core-congestion-control-01 (work in
progress), January 2011.
- [I-D.shelby-core-resource-directory]
Shelby, Z. and S. Krco, "CoRE Resource Directory",
draft-shelby-core-resource-directory-01 (work in
progress), September 2011.
- [I-D.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building
Control", draft-vanderstok-core-bc-04 (work in progress),
July 2011.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web
Replication and Caching Taxonomy", RFC 3040, January 2001.
- [RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-
Service Considerations", RFC 4732, December 2006.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins,
"Known Issues and Best Practices for the Use of Long
Polling and Streaming in Bidirectional HTTP", RFC 6202,
April 2011.
- [W3C.HTML5]
Hickson, I., "HTML5", World Wide Web Consortium WD (work
in progress) WD-html5-20111018, October 2011,
<<http://dev.w3.org/html5/spec/>>.

Appendix A. Internal Mapping Functions (from an implementer's perspective)

At least three mapping functions have been identified, which take

place at different stages of the HC proxy processing chain, involving the URL, Content-Type and Security Policy translation.

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

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

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

A.1. URL Map Algorithm

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

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

An interception HC proxy, instead, MAY use the homogeneous mapping strategy (see Section 3.1.1 for details) to operate without any pre-configuration need.

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

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

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

INPUT
* requested URL

OUTPUT
* target URL

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

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

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

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

A.2. Security Policy Map Algorithm

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

INPUT

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

OUTPUT

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

SYNTAX

```

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

```

EXAMPLE

```
[TBD]
```

A.3. Content-Type Map Algorithm

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

INPUT

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

OUTPUT

- * mapped content-type

SYNTAX

```

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

```

EXAMPLE

```

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

```

Authors' Addresses

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

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

Akbar Rahman
InterDigital Communications, LLC

Email: Akbar.Rahman@InterDigital.com

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

Phone: +39 051 644 82 68

Email: tho@koanlogic.com

Esko Dijk
Philips Research

Email: esko.dijk@philips.com

CoRE
Internet-Draft
Intended status: Informational
Expires: May 3, 2012

O. Garcia-Morchon
S. Keoh
S. Kumar
Philips Research
R. Hummen
RWTH Aachen
R. Struik
Struik Consultancy
October 31, 2011

Security Considerations in the IP-based Internet of Things
draft-garcia-core-security-03

Abstract

A direct interpretation of the Internet of Things concept refers to the usage of standard Internet protocols to allow for human-to-thing or thing-to-thing communication. Although the security needs are well-recognized, it is still not fully clear how existing IP-based security protocols can be applied to this new setting. This Internet-Draft first provides an overview of security architecture, its deployment model and general security needs in the context of the lifecycle of a thing. Then, it presents challenges and requirements for the successful roll-out of new applications and usage of standard IP-based security protocols when applied to get a functional Internet of Things.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 3, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the

document authors. All rights reserved.

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

Table of Contents

1. Conventions and Terminology Used in this Document	4
2. Introduction	4
3. The Thing Lifecycle and Architectural Considerations	5
3.1. Threat Analysis	6
3.2. Security Aspects	10
4. State of the Art	13
4.1. IP-based Security Solutions	13
4.2. Wireless Sensor Network Security and Beyond	15
5. Challenges for a Secure Internet of Things	16
5.1. Constraints and Heterogeneous Communication	16
5.1.1. Tight Resource Constraints	16
5.1.2. Denial-of-Service Resistance	18
5.1.3. Protocol Translation and End-to-End Security	18
5.2. Bootstrapping of a Security Domain	20
5.2.1. Distributed vs. Centralized Architecture and Operation	20
5.2.2. Bootstrapping a thing's identity and keying materials	21
5.2.3. Privacy-aware Identification	22
5.3. Operation	23
5.3.1. End-to-End Security	23
5.3.2. Group Membership and Security	23
5.3.3. Mobility and IP Network Dynamics	24
6. Security Suites for the IP-based Internet of Things	25
6.1. Security Architecture	29
6.2. Security Model	30
6.3. Security Bootstrapping and Management	31
6.4. Network Security	33
6.5. Application Security	34
7. Next Steps towards a Flexible and Secure Internet of Things .	36
8. Security Considerations	38
9. IANA Considerations	38
10. Acknowledgements	39
11. References	39
11.1. Normative References	39
11.2. Informative References	42
Authors' Addresses	43

1. Conventions and Terminology Used in this Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

2. Introduction

The Internet of Things (IoT) denotes the interconnection of highly heterogeneous networked entities and networks following a number of communication patterns such as: human-to-human (H2H), human-to-thing (H2T), thing-to-thing (T2T), or thing-to-things (T2Ts). The term IoT was first coined by the Auto-ID center [AUTO-ID] in 1999. Since then, the development of the underlying concepts has ever increased its pace. Nowadays, the IoT presents a strong focus of research with various initiatives working on the (re)design, application, and usage of standard Internet technology in the IoT.

The introduction of IPv6 and web services as fundamental building blocks for IoT applications [ID-KIM] promises to bring a number of basic advantages including: (i) a homogeneous protocol ecosystem that allows simple integration with Internet hosts; (ii) simplified development of very different appliances; (iii) an unified interface for applications, removing the need for application-level proxies. Such features greatly simplify the deployment of the envisioned scenarios ranging from building automation to production environments to personal area networks, in which very different things such as a temperature sensor, a luminaire, or an RFID tag might interact with each other, with a human carrying a smart phone, or with backend services.

This Internet Draft presents an overview of the security aspects of the envisioned all-IP architecture as well as of the lifecycle of an IoT device, a thing, within this architecture. In particular, we review the most pressing aspects and functionalities that are required for a secure all-IP solution.

With this, this Internet-Draft pursues several goals. First, we aim at presenting a comprehensive view of the interactions and relationships between an IoT application and security. Second, we aim at describing challenges for a secure IoT in the specific context of the lifecycle of a resource-constrained device. The final goal of this draft is to discuss the next steps towards a secure IoT.

The rest of the Internet-Draft is organized as follows. Section 3 depicts the lifecycle of a thing and gives general definitions for

the main security aspects within the IoT domain. In Section 4, we review existing protocols and work done in the area of security for wireless sensor networks. Section 5 identifies general challenges and needs for an IoT security protocol design and discusses existing protocols and protocol proposals against the identified requirements. Section 6 proposes a number of illustrative security suites describing how different applications involve distinct security needs. Section 7 includes final remarks and conclusions.

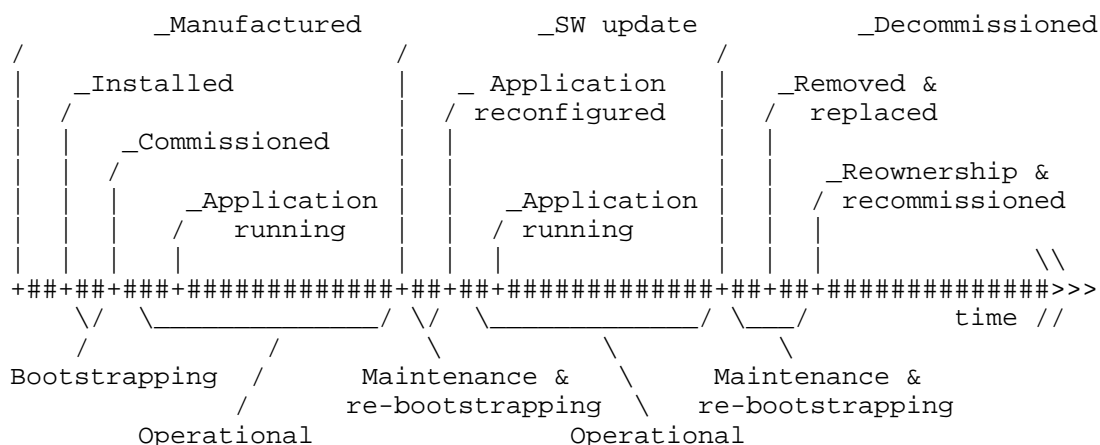
3. The Thing Lifecycle and Architectural Considerations

We consider the installation of a Building Automation and Control (BAC) system to illustrate the lifecycle of a thing in a BAC scenario. A BAC system consists of a network of interconnected nodes that perform various functions in the domains of HVAC (Heating, Ventilating, and Air Conditioning), lighting, safety etc. The nodes vary in functionality and a majority of them represent resource constrained devices such as sensors and luminaries. Some devices may also be battery operated or battery-less nodes, demanding for a focus on low energy consumption and on sleeping devices.

In our example, the life of a thing starts when it is manufactured. Due to the different application areas (i.e., HVAC, lighting, safety) nodes are tailored to a specific task. It is therefore unlikely that one single manufacturer will create all nodes in a building. Hence, interoperability as well as trust bootstrapping between nodes of different vendors is important. The thing is later installed and commissioned within a network by an installer during the bootstrapping phase. Specifically, the device identity and the secret keys used during normal operation are provided to the device during this phase. Different subcontractors may install different IoT devices for different purposes. Furthermore, the installation and bootstrapping procedures may not be a defined event but may stretch over an extended period of time. After being bootstrapped, the device and the system of things are in operational mode and run the functions of the BAC system. During this operational phase, the device is under the control of the system owner. For devices with lifetimes spanning several years, occasional maintenance cycles may be required. During each maintenance phase, the software on the device can be upgraded or applications running on the device can be reconfigured. The maintenance tasks can thereby be performed either locally or from a backend system. Depending on the operational changes of the device, it may be required to re-bootstrap at the end of a maintenance cycle. The device continues to loop through the operational phase and the eventual maintenance phase until the device is decommissioned at the end of its lifecycle. However, the end-of-life of a device does not necessarily mean that it is defective but

rather denotes a need to replace and upgrade the network to next-generation devices in order to provide additional functionality. Therefore the device can be removed and re-commissioned to be used in a different network under a different owner by starting the lifecycle over again. Figure 1 shows the generic lifecycle of a thing. This generic lifecycle is also applicable for IoT scenarios other than BAC systems.

At present, BAC systems use legacy building control standards such as BACNet [BACNET] or DALI [DALI] with independent networks for each subsystem (HVAC, lighting, etc.). However, this separation of functionality adds further complexity and costs to the configuration and maintenance of the different networks within the same building. As a result, more recent building control networks employ IP-based standards allowing seamless control over the various nodes with a single management system. While allowing for easier integration, this shift towards IP-based standards results in new requirements regarding the implementation of IP security protocols on constrained devices and the bootstrapping of security keys for devices across multiple manufacturers.



The lifecycle of a thing in the Internet of Things.

Figure 1

3.1. Threat Analysis

This section explores the security threats and vulnerabilities of a network of things in the IoTs. Security threats have been analyzed in related IP protocols including HTTPS [RFC2818], 6LoWPAN [RFC4919], ANCP [RFC5713], DNS security threats [RFC3833], SIP [RFC3261], IPv6

ND [RFC3756], and PANA [RFC4016]. Nonetheless, the challenge is about their impacts on scenarios of the IoTs. In this section, we specifically discuss the threats that could compromise an individual thing, or network as a whole, with regard to different phases in the thing's lifecycle. Note that these set of threats might go beyond the scope of Internet protocols but we gather them here for the sake of completeness.

- 1 Cloning of things: During the manufacturing process of a thing, an untrusted manufacturer can easily clone the physical characteristics, firmware/software, or security configuration of the thing. Subsequently, such a cloned thing may be sold at a cheaper price in the market, and yet be still able to function normally, as a genuine thing. For example, two cloned devices can still be associated and work with each other. In the worst case scenario, a cloned device can be used to control a genuine device. One should note here, that an untrusted manufacturer may also change functionality of the cloned thing, resulting in degraded functionality with respect to the genuine thing (thereby, inflicting potential reputational risk to the original thing manufacturer). Moreover, it can implement additional functionality with the cloned thing, such as a backdoor.
- 2 Malicious substitution of things: During the installation of a thing, a genuine thing may be substituted with a similar variant of lower quality without being detected. The main motivation may be cost savings, where the installation of lower-quality things (e.g., non-certified products) may significantly reduce the installation and operational costs. The installers can subsequently resell the genuine things in order to gain further financial benefits. Another motivation may be to inflict reputational damage on a competitor's offerings.
- 3 Eavesdropping attack: During the commissioning of a thing into a network, it may be susceptible to eavesdropping, especially if operational keying materials, security parameters, or configuration settings, are exchanged in clear using a wireless medium. After obtaining the keying material, the attacker might be able to recover the secret keys established between the communicating entities (e.g., H2T, T2Ts, or Thing to the backend management system), thereby compromising the authenticity and confidentiality of the communication channel, as well as the authenticity of commands and other traffic exchanged over this communication channel. When the network is in operation, T2T communication may be eavesdropped upon if the communication channel is not sufficiently protected or in the event of session key compromise due to a long period of usage without key renewal or updates.

- 4 Man-in-the-middle attack: The commissioning phase may also be vulnerable to man-in-the-middle attacks, e.g., when keying material between communicating entities is exchanged in the clear and the security of the key establishment protocol depends on the tacit assumption that no third party is able to eavesdrop on or sit in between the two communicating entities during the execution of this protocol. Additionally, device authentication or device authorization may be nontrivial, or may need support of a human decision process, since things usually do not have a priori knowledge about each other and can, therefore, not always be able to differentiate friends and foes via completely automated mechanisms. Thus, even if the key establishment protocol provides cryptographic device authentication, this knowledge on device identities may still need complementing with a human-assisted authorization step (thereby, presenting a weak link and offering the potential of man-in-the-middle attacks this way).
- 5 Firmware Replacement attack: When a thing is in operation or maintenance phase, its firmware or software may be updated to allow for new functionality or new features. An attacker may be able to exploit such a firmware upgrade by replacing the thing's with malicious software, thereby influencing the operational behaviour of the thing. For example, an attacker could add a piece of malicious code to the firmware that will cause it to periodically report the energy usage of the lamp to a data repository for analysis.
- 6 Extraction of security parameters: A thing deployed in the ambient environment (such as sensors, actuators, etc.) is usually physically unprotected and could easily be captured by an attacker. Such an attacker may then attempt to extract security information such as keys (e.g., device's key, private-key, group key) from this thing or try and re-program it to serve his needs. If a group key is used and compromised this way, the whole network may be compromised as well. Compromise of a thing's unique key has less security impact, since only the communication channels of this particular thing in question are compromised. Here, one should caution that compromise of the communication channel may also compromise all data communicated over this channel. In particular, one has to be weary of, e.g., compromise of group keys communicated over this channel (thus, leading to transitive exposure ripple effects).
- 7 Routing attack: As highlighted in [ID-Daniel], routing information in IoT can be spoofed, altered, or replayed, in order to create routing loops, attract/repel network traffic, extend/shorten source routes, etc. Other relevant routing attacks

include 1) Sinkhole attack (or blackhole attack), where an attacker declares himself to have a high-quality route/path to the base station, thus allowing him to do anything to all packets passing through it. 2) Selective forwarding, where an attacker may selectively forward packets or simply drop a packet. 3) Wormhole attack, where an attacker may record packets at one location in the network and tunnel them to another location, thereby influencing perceived network behaviour and potentially distorting statistics, thus greatly impacting the functionality of routing. 4) Sybil attack, whereby an attacker presents multiple identities to other things in the network.

- 8 Privacy threat: The tracking of a thing's location and usage may pose a privacy risk to its users. An attacker can infer information based on the information gathered about individual things, thus deducing behavioural patterns of the user of interest to him. Such information can subsequently be sold to interested parties for marketing purposes and targeted advertizing.
- 9 Denial-of-Service attack: Typically, things have tight memory and limited computation, they are thus vulnerable to resource exhaustion attack. Attackers can continuously send requests to be processed by specific things so as to deplete their resources. This is especially dangerous in the IoTs since an attacker might be located in the backend and target resource-constrained devices in an LLN. Additionally, DoS attack can be launched by physically jamming the communication channel, thus breaking down the T2T communication channel. Network availability can also be disrupted by flooding the network with a large number of packets.

The following table summarizes the security threats we identified above and the potential point of vulnerabilities at different layers of the communication stack. We also include related RFCs that include a threat model that might apply to the IoTs.

	Manufacturing	Installation/ Commissioning	Operation
Thing's Model	Device Cloning	Substitution	Privacy threat Extraction of security params
Application Layer		RFC2818 RFC4016	RFC2818, Firmware replacement
Transport Layer		Eavesdropping & Man-in-the-middle	Eavesdropping Man-in-the-middle
Network Layer		attack RFC4919, RFC5713 RFC3833, RFC3756	RFC4919, DoS attack Routing attack RFC3833
Physical Layer			DoS attack

The security threat analysis

Figure 2

3.2. Security Aspects

The term security subsumes a wide range of different concepts. In the first place, it refers to the basic provision of security services including confidentiality, authentication, integrity, authorization, non-repudiation, and availability, and some augmented services, such as duplicate detection and detection of stale packets (timeliness). These security services can be implemented by a combination of cryptographic mechanisms, such as block ciphers, hash functions, or signature algorithms, and non-cryptographic mechanisms, which implement authorization and other security policy enforcement aspects. For each of the cryptographic mechanisms, a solid key management infrastructure is fundamental to handling the required cryptographic keys, whereas for security policy enforcement, one needs to properly codify authorizations as a function of device roles and a security policy engine that implements these authorization checks and that can implement changes hereto throughout the system's lifecycle.

In the context of the IoT, however, the security must not only focus on the required security services, but also how these are realized in the overall system and how the security functionalities are executed.

To this end, we use the following terminology to analyze and classify security aspects in the IoT:

- 1 The security architecture refers to the system elements involved in the management of the security relationships between things and the way these security interactions are handled (e.g., centralized or distributed) during the lifecycle of a thing.
- 2 The security model of a node describes how the security parameters, processes, and applications are managed in a thing. This includes aspects such as process separation, secure storage of keying materials, etc.
- 3 Security bootstrapping denotes the process by which a thing securely joins the IoT at a given location and point in time. Bootstrapping includes the authentication and authorization of a device as well as the transfer of security parameters allowing for its trusted operation in a given network.
- 4 Network security describes the mechanisms applied within a network to ensure trusted operation of the IoT. Specifically, it prevents attackers from endangering or modifying the expected operation of networked things. Network security can include a number of mechanisms ranging from secure routing to data link layer and network layer security.
- 5 Application security guarantees that only trusted instances of an application running in the IoT can communicate with each other, while illegitimate instances cannot interfere.

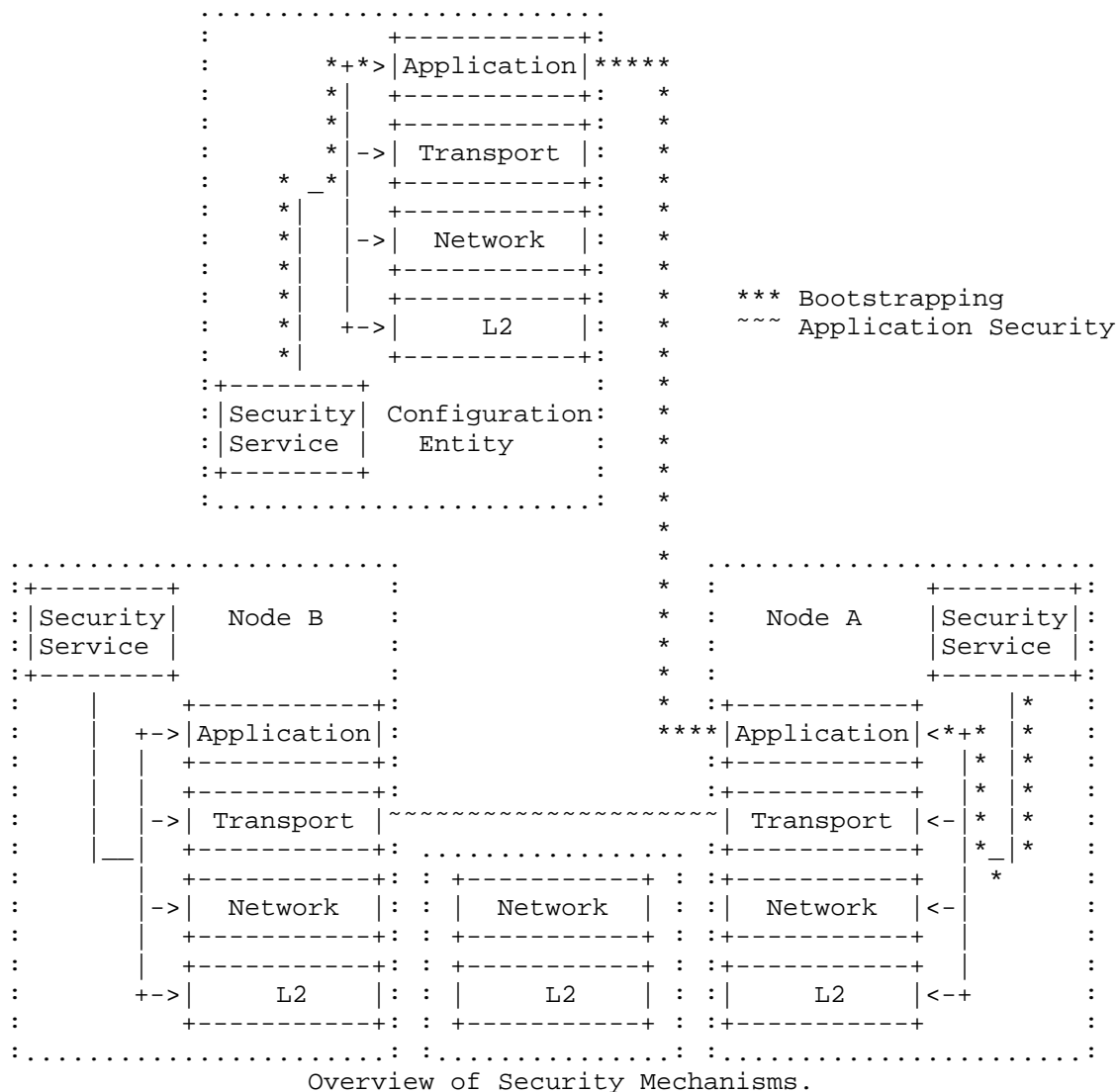


Figure 3

We now discuss an exemplary security architecture relying on a configuration entity for the management of the system with regard to the introduced security aspects (see Figure 2). Inspired by the security framework for routing over low power and lossy network [ID-Tsao], we show an example of security model and illustrates how different security concepts and the lifecycle phases map to the Internet communication stack. Assume a centralized architecture in

which a configuration entity stores and manages the identities of the things associated with the system along with their cryptographic keys. During the bootstrapping phase, each thing executes the bootstrapping protocol with the configuration entity, thus obtaining the required device identities and the keying material. The security service on a thing in turn stores the received keying material for the network layer and application security mechanisms for secure communication. Things can then securely communicate with each other during their operational phase by means of the employed network and application security mechanisms.

4. State of the Art

Nowadays, there exists a multitude of control protocols for the IoT. For BAC systems, the ZigBee standard [ZB], BACNet [BACNET], or DALI [DALI] play key roles. Recent trends, however, focus on an all-IP approach for system control.

In this setting, a number of IETF working groups are designing new protocols for resource constrained networks of smart things. The 6LoWPAN working group [WG-6LoWPAN] concentrates on the definition of methods and protocols for the efficient transmission and adaptation of IPv6 packets over IEEE 802.15.4 networks [RFC4944]. The CoRE working group [WG-CoRE] provides a framework for resource-oriented applications intended to run on constrained IP network (6LoWPAN). One of its main tasks is the definition of a lightweight version of the HTTP protocol, the Constrained Application Protocol (CoAP) [ID-CoAP], that runs over UDP and enables efficient application-level communication for things.

4.1. IP-based Security Solutions

In the context of the IP-based IoT solutions, consideration of TCP/IP security protocols is important as these protocols are designed to fit the IP network ideology and technology. While a wide range of specialized as well as general-purpose key exchange and security solutions exist for the Internet domain, we discuss a number of protocols and procedures that have been recently discussed in the context of the above working groups. The considered protocols are IKEv2/IPsec [RFC4306], TLS/SSL [RFC5246], DTLS [RFC5238], HIP [RFC5201][ID-Moskowitz], PANA [RFC5191], and EAP [RFC3748] in this Internet-Draft. Application layer solutions such as SSH [RFC4251] also exist, however, these are currently not considered. Figure 3 depicts the relationships between the discussed protocols in the context of the security terminology introduced in Section 3.1.

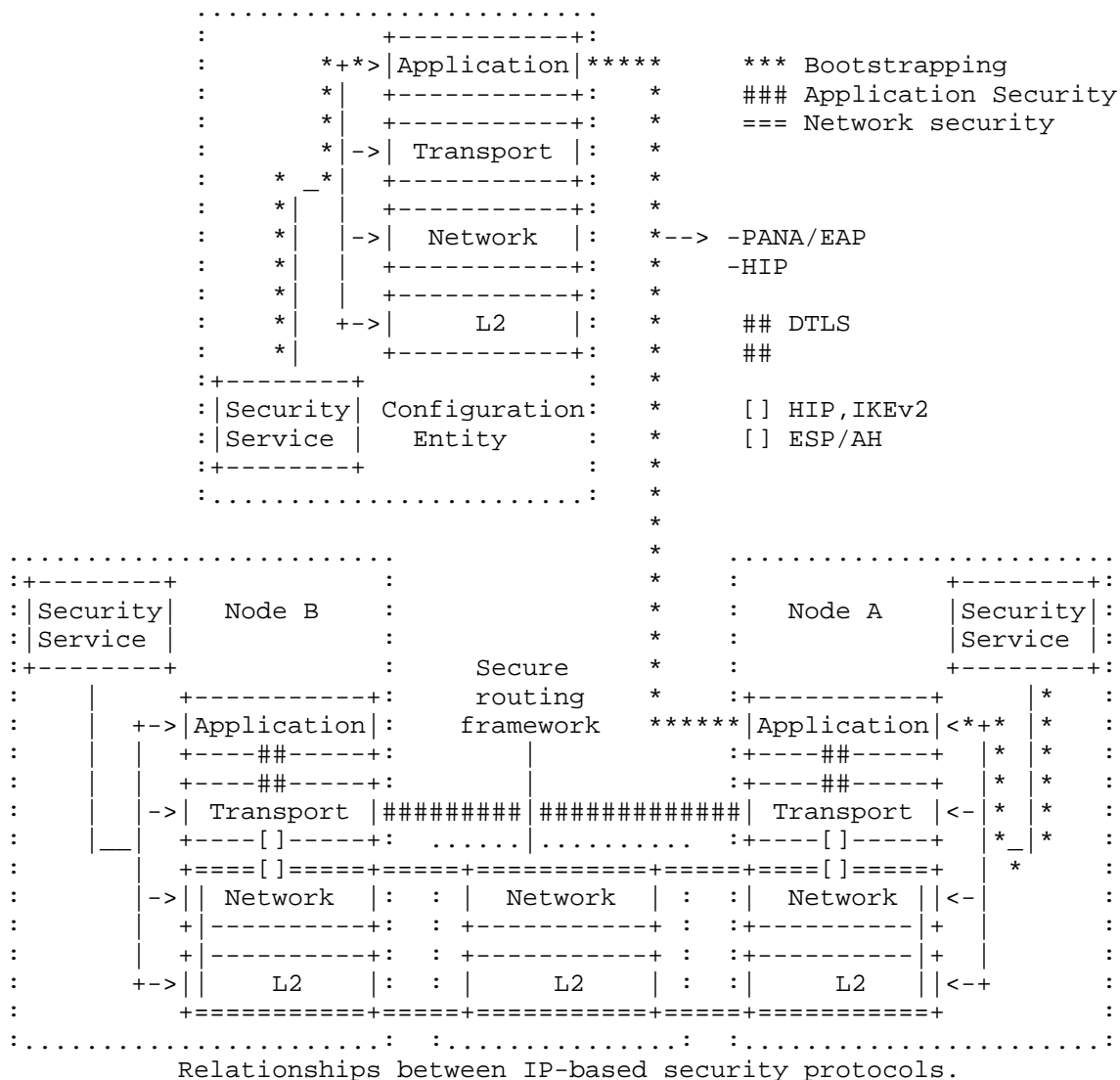


Figure 4

The Internet Key Exchange (IKEv2)/IPsec and the Host Identity protocol (HIP) reside at or above the network layer in the OSI model. Both protocols are able to perform an authenticated key exchange and set up the IPsec transforms for secure payload delivery. Currently, there are also ongoing efforts to create a HIP variant coined Diet HIP [ID-HIP] that takes lossy low-power networks into account at the authentication and key exchange level.

Transport Layer Security (TLS) and its datagram-oriented variant DTLS secure transport-layer connections. TLS provides security for TCP and requires a reliable transport, while DTLS secures and uses datagram-oriented protocols such as UDP. Both protocols are intentionally kept similar and share the same ideology and cipher suites.

The Extensible Authentication Protocol (EAP) is an authentication framework supporting multiple authentication methods. EAP runs directly over the data link layer and, thus, does not require the deployment of IP. It supports duplicate detection and retransmission, but does not allow for packet fragmentation. The Protocol for Carrying Authentication for Network Access (PANA) is a network-layer transport for EAP that enables network access authentication between clients and the network infrastructure. In EAP terms, PANA is a UDP-based EAP lower layer that runs between the EAP peer and the EAP authenticator.

4.2. Wireless Sensor Network Security and Beyond

A variety of key agreement and privacy protection protocols that are tailored to IoT scenarios have been introduced in the literature. For instance, random key pre-distribution schemes [PROC-Chan] or more centralized solutions, such as SPINS [JOURNAL-Perrig], have been proposed for key establishment in wireless sensor networks. The ZigBee standard [ZB] for sensor networks defines a security architecture based on an online trust center that is in charge of handling the security relationships within a ZigBee network. Personal privacy in ubiquitous computing has been studied extensively, e.g., in [THESIS-Langheinrich]. Due to resource constraints and the specialization to meet specific requirements, these solutions often implement a collapsed cross layer optimized communication stack (e.g., without task-specific network layers and layered packet headers). Consequently, they cannot directly be adapted to the requirements of the Internet due to the nature of their design.

Despite important steps done by, e.g., Gupta et al. [PROC-Gupta], to show the feasibility of an end-to-end standard security architecture for the embedded Internet, the Internet and the IoT domain still do not fit together easily. This is mainly due to the fact that IoT security solutions are often tailored to the specific scenario requirements without considering interoperability with Internet protocols. On the other hand, the direct use of existing Internet security protocols in the IoT might lead to inefficient or insecure operation as we show in our discussion below.

5. Challenges for a Secure Internet of Things

In this section, we take a closer look at the various security challenges in the operational and technical features of the IoT and then discuss how existing Internet security protocols cope with these technical and conceptual challenges through the lifecycle of a thing. Table 1 summarizes which requirements need to be met in the lifecycle phases as well as the considered protocols. The structure of this section follows the structure of the table. This discussion should neither be understood as a comprehensive evaluation of all protocols, nor can it cover all possible aspects of IoT security. Yet, it aims at showing concrete limitations of existing Internet security protocols in some areas rather than giving an abstract discussion about general properties of the protocols. In this regard, the discussion handles issues that are most important from the authors' perspectives.

5.1. Constraints and Heterogeneous Communication

Coupling resource constrained networks and the powerful Internet is a challenge because the resulting heterogeneity of both networks complicates protocol design and system operation. In the following we briefly discuss the resource constraints of IoT devices and the consequences for the use of Internet Protocols in the IoT domain.

5.1.1. Tight Resource Constraints

The IoT is a resource-constrained network that relies on lossy and low-bandwidth channels for communication between small nodes, regarding CPU, memory, and energy budget. These characteristics directly impact the threats to and the design of security protocols for the IoT domain. First, the use of small packets, e.g., IEEE 802.15.4 supports 127-byte sized packets at the physical layer, may result in fragmentation of larger packets of security protocols. This may open new attack vectors for state exhaustion DoS attacks, which is especially tragic, e.g., if the fragmentation is caused by large key exchange messages of security protocols. Moreover, packet fragmentation commonly downgrades the overall system performance due to fragment losses and the need for retransmissions. For instance, fate-sharing packet flight as implemented by DTLS might aggravate the resulting performance loss.

	Bootstrapping phase	Operational Phase
Requirements	Incremental deployment Identity and key management Privacy-aware identification Group creation	End-to-End security Mobility support Group membership management
Protocols	IKEv2 TLS/DTLS HIP/Diet-HIP PANA/EAP	IKEv2/MOBIKE TLS/DTLS HIP/Diet-HIP

Relationships between IP-based security protocols.

Figure 5

The size and number of messages should be minimized to reduce memory requirements and optimize bandwidth usage. In this context, layered approaches involving a number of protocols might lead to worse performance in resource-constrained devices since they combine the headers of the different protocols. In some settings, protocol negotiation can increase the number of exchanged messages. To improve performance during basic procedures such as, e.g., bootstrapping, it might be a good strategy to perform those procedures at a lower layer.

Small CPUs and scarce memory limit the usage of resource-expensive cryptoprimitives such as public-key cryptography as used in most Internet security standards. This is especially true, if the basic cryptoblocks need to be frequently used or the underlying application demands a low delay.

Independently from the development in the IoT domain, all discussed security protocols show efforts to reduce the cryptographic cost of the required public-key-based key exchanges and signatures with ECC[RFC5246][RFC5903][ID-Moskowitz][ID-HIP]. Moreover, all protocols have been revised in the last years to enable crypto agility, making cryptographic primitives interchangeable. Diet HIP takes the reduction of the cryptographic load one step further by focusing on cryptographic primitives that are to be expected to be enabled in hardware on IEEE 802.15.4 compliant devices. For example, Diet HIP does not require cryptographic hash functions but uses a CMAC [NIST] based mechanism, which can directly use the AES hardware available in standard sensor platforms. However, these improvements are only a first step in reducing the computation and communication overhead of Internet protocols. The question remains if other approaches can be

applied to leverage key agreement in these heavily resource-constrained environments.

A further fundamental need refers to the limited energy budget available to IoT nodes. Careful protocol (re)design and usage is required to reduce not only the energy consumption during normal operation, but also under DoS attacks. Since the energy consumption of IoT devices differs from other device classes, judgments on the energy consumption of a particular protocol cannot be made without tailor-made IoT implementations.

5.1.2. Denial-of-Service Resistance

The tight memory and processing constraints of things naturally alleviate resource exhaustion attacks. Especially in unattended T2T communication, such attacks are difficult to notice before the service becomes unavailable (e.g., because of battery or memory exhaustion). As a DoS countermeasure, DTLs, IKEv2, HIP, and Diet HIP implement return routability checks based on a cookie mechanism to delay the establishment of state at the responding host until the address of the initiating host is verified. The effectiveness of these defenses strongly depends on the routing topology of the network. Return routability checks are particularly effective if hosts cannot receive packets addressed to other hosts and if IP addresses present meaningful information as is the case in today's Internet. However, they are less effective in broadcast media or when attackers can influence the routing and addressing of hosts (e.g., if hosts contribute to the routing infrastructure in ad-hoc networks and meshes).

In addition, HIP implements a puzzle mechanism that can force the initiator of a connection (and potential attacker) to solve cryptographic puzzles with variable difficulties. Puzzle-based defense mechanisms are less dependent on the network topology but perform poorly if CPU resources in the network are heterogeneous (e.g., if a powerful Internet host attacks a thing). Increasing the puzzle difficulty under attack conditions can easily lead to situations, where a powerful attacker can still solve the puzzle while weak IoT clients cannot and are excluded from communicating with the victim. Still, puzzle-based approaches are a viable option for sheltering IoT devices against unintended overload caused by misconfigured or malfunctioning things.

5.1.3. Protocol Translation and End-to-End Security

Even though 6LoWPAN and CoAP progress towards reducing the gap between Internet protocols and the IoT, they do not target protocol specifications that are identical to their Internet pendants due to

performance reasons. Hence, more or less subtle differences between IoT protocols and Internet protocols will remain. While these differences can easily be bridged with protocol translators at gateways, they become major obstacles if end-to-end security measures between IoT devices and Internet hosts are used.

Cryptographic payload processing applies message authentication codes or encryption to packets. These protection methods render the protected parts of the packets immutable as rewriting is either not possible because a) the relevant information is encrypted and inaccessible to the gateway or b) rewriting integrity-protected parts of the packet would invalidate the end-to-end integrity protection.

There are essentially four solutions for this problem:

- 1 Sharing symmetric keys with gateways enables gateways to transform (e.g., de-compress, convert, etc.) packets and re-apply the security measures after transformation. This method abandons end-to-end security and is only applicable to simple scenarios with a rudimentary security model.
- 2 Reusing the Internet wire format in the IoT makes conversion between IoT and Internet protocols unnecessary. However, it leads to poor performance because IoT specific optimizations (e.g., stateful or stateless compression) are not possible.
- 3 Selectively protecting vital and immutable packet parts with a MAC or with encryption requires a careful balance between performance and security. Otherwise, this approach will either result in poor performance (protect as much as possible) or poor security (compress and transform as much as possible).
- 4 Message authentication codes that sustain transformation can be realized by considering the order of transformation and protection (e.g., by creating a signature before compression so that the gateway can decompress the packet without recalculating the signature). This enables IoT specific optimizations but is more complex and may require application-specific transformations before security is applied. Moreover, it cannot be used with encrypted data because the lack of cleartext prevents gateways from transforming packets.

To the best of our knowledge, none of the mentioned security protocols provides a fully customizable solution in this problem space. In fact, they usually offer an end-to-end secured connection. An exception is the usage layered approach as might be PANA and EAP. In such a case, this configuration (i) allows for a number of configurations regarding the location of, e.g., the EAP authenticator

and authentication server and (ii) the layered architecture might allow for authentication at different places. The drawback of this approach, however, lies in its high signaling traffic volume compared to other approaches. Hence, future work is required to ensure security, performance and interoperability between IoT and the Internet.

5.2. Bootstrapping of a Security Domain

Creating a security domain from a set of previously unassociated IoT devices is a key operation in the lifecycle of a thing and in the IoT network. In this section, we discuss general forms of network operation, how to communicate a thing's identity and the privacy implications arising from the communication of this identity.

5.2.1. Distributed vs. Centralized Architecture and Operation

Most things might be required to support both centralized and distributed operation patterns. Distributed thing-to-thing communication might happen on demand, for instance, when two things form an ad-hoc security domain to cooperatively fulfill a certain task. Likewise, nodes may communicate with a backend service located in the Internet without a central security manager. The same nodes may also be part of a centralized architecture with a dedicated node being responsible for the security management for group communication between things in the IoT domain. In today's IoT, most common architectures are fully centralized in the sense that all the security relationships within a segment are handled by a central party. In the ZigBee standard, this entity is the trust center. Current proposals for 6LoWPAN/CoRE identify the 6LoWPAN Border Router (6LBR) as such a device.

A centralized architecture allows for central management of devices and keying materials as well as for the backup of cryptographic keys. However, it also imposes some limitations. First, it represents a single point of failure. This is a major drawback, e.g., when key agreement between two devices requires online connectivity to the central node. Second, it limits the possibility to create ad-hoc security domains without dedicated security infrastructure. Third, it codifies a more static world view, where device roles are cast in stone, rather than a more dynamic world view that recognizes that networks and devices, and their roles and ownership, may change over time (e.g., due to device replacement and hand-over of control).

Decentralized architectures, on the other hand, allow creating ad-hoc security domains that might not require a single online management entity and are operative in a much more stand-alone manner. The ad-hoc security domains can be added to a centralized architecture at a

later point in time, allowing for central or remote management.

5.2.2. Bootstrapping a thing's identity and keying materials

Bootstrapping refers to the process by which a device is associated to another one, to a network, or to a system. The way it is performed depends upon the architecture: centralized or distributed. It is important to realize that bootstrapping may involve different types of information, ranging from network parameters and information on device capabilities and their presumed functionality, to management information related to, e.g., resource scheduling and trust initialization/management. Furthermore, bootstrapping may occur in stages during the lifecycle of a device and may include provisioning steps already conducted during device manufacturing (e.g., imprinting a unique identifier or a root certificate into a device during chip testing), further steps during module manufacturing (e.g., setting of application-based configurations, such as temperature read-out frequencies and push-thresholds), during personalization (e.g., fine-tuned settings depending on installation context), during hand-over (e.g., transfer of ownership from supplier to user), and, e.g., in preparation of operation in a specific network. In what follows, we focus on bootstrapping of security-related information, since bootstrapping of all other information can be conducted as ordinary secured communications, once a secure and authentic channel between devices has been put in place.

In a distributed approach, a Diffie-Hellman type of handshake can allow two peers to agree on a common secret. In general, IKEv2, HIP, TLS, DTLS, can perform key exchanges and the setup of security associations without online connections to a trust center. If we do not consider the resource limitations of things, certificates and certificate chains can be employed to securely communicate capabilities in such a decentralized scenario. HIP and Diet HIP do not directly use certificates for identifying a host, however certificate handling capabilities exist for HIP and the same protocol logic could be used for Diet HIP. It is noteworthy, that Diet HIP does not require a host to implement cryptographic hashes. Hence, some lightweight implementations of Diet HIP might not be able to verify certificates unless a hash function is implemented by the host.

In a centralized architecture, preconfigured keys or certificates held by a thing can be used for the distribution of operational keys in a given security domain. A current proposal [ID-OFlynn] refers to the use of PANA for the transport of EAP messages between the PANA client (the joining thing) and the PANA Authentication Agent (PAA), the 6LBR. EAP is thereby used to authenticate the identity of the joining thing. After the successful authentication, the PANA PAA

provides the joining thing with fresh network and security parameters.

IKEv2, HIP, TLS, and DTLS could be applied as well for the transfer of configuration parameters in a centralized scenario. While HIP's cryptographic secret identifies the thing, the other protocols do not represent primary identifiers but are used instead to bind other identifiers such as the operation keys to the public-key identities.

In addition to the protocols, operational aspects during bootstrapping are of key importance as well. Many other standard Internet protocols assume that the identity of a host is either available by using secondary services like certificate authorities or secure name resolution (e.g., DNSsec) or can be provided over a side channel (entering passwords via screen and keyboard). While these assumptions may hold in traditional networks, intermittent connectivity, localized communication, and lack of input methods complicate the situation for the IoT.

The order in which the things within a security domain are bootstrapped plays an important role as well. In [ID-Duffy], the PANA relay element is introduced, relaying PANA messages between a PaC (joining thing) and PAA of a segment [ID-OFlynn]. This approach forces commissioning based on distance to PAA, i.e., things can only be bootstrapped hop-by-hop starting from those closer to the PAA, all things that are 1-hop away are bootstrapped first, followed by those that are 2-hop away, and so on. Such an approach might impose important limitations on actual use cases in which, e.g., an installer without technical background has to roll-out the system.

5.2.3. Privacy-aware Identification

During the last years, the introduction of RFID tags has raised privacy concerns because anyone might access and track tags. As the IoT involves not only passive devices, but also includes active and sensing devices, the IoT might irrupt even deeper in people's privacy spheres. Thus, IoT protocols should be designed to avoid these privacy threats during bootstrapping and operation where deemed necessary. In H2T and T2T interactions, privacy-aware identifiers might be used to prevent unauthorized user tracking. Similarly, authentication can be used to prove membership of a group without revealing unnecessary individual information.

TLS and DTLS provide the option of only authenticating the responding host. This way, the initiating host can stay anonymous. If authentication for the initiating host is required as well, either public-key certificates or authentication via the established encrypted payload channel can be employed. Such a setup allows to

only reveal the responder's identity to possible eavesdroppers.

HIP and IKEv2 use public-key identities to authenticate the initiator of a connection. These identities could easily be traced if no additional protection were in place. IKEv2 transmits this information in an encrypted packet. Likewise, HIP provides the option to keep the identity of the initiator secret from eavesdroppers by encrypting it with the symmetric key generated during the handshake. However, Diet HIP cannot provide a similar feature because the identity of the initiator simultaneously serves as static Diffie-Hellman key. Note that all discussed solutions could use anonymous public-key identities that change for each communication. However, such identity cycling may require a considerable computational effort for generating new asymmetric key pairs. In addition to the built-in privacy features of the here discussed protocols, a large body of anonymity research for key exchange protocols exists. However, the comparison of these protocols and protocol extensions is out of scope for this work.

5.3. Operation

After the bootstrapping phase, the system enters the operational phase. During the operational phase, things can relate to the state information created during the bootstrapping phase in order to exchange information securely and in an authenticated fashion. In this section, we discuss aspects of communication patterns and network dynamics during this phase.

5.3.1. End-to-End Security

Providing end-to-end security is of great importance to address and secure individual T2T or H2T communication within one IoT domain. Moreover, end-to-end security associations are an important measure to bridge the gap between the IoT and the Internet. IKEv2 and HIP, TLS and DTLS provide end-to-end security services including peer entity authentication, end-to-end encryption and integrity protection above the network layer and the transport layer respectively. Once bootstrapped, these functions can be carried out without online connections to third parties, making the protocols applicable for decentralized use in the IoT. However, protocol translation by intermediary nodes may invalidate end-to-end protection measures (see Section 5.1).

5.3.2. Group Membership and Security

In addition to end-to-end security, group key negotiation is an important security service for the T2Ts and Ts2T communication patterns in the IoT as efficient local broadcast and multicast relies

on symmetric group keys.

All discussed protocols only cover unicast communication and therefore do not focus on group-key establishment. However, the Diffie-Hellman keys that are used in IKEv2 and HIP could be used for group Diffie-Hellman key-negotiations. Conceptually, solutions that provide secure group communication at the network layer (IPsec/IKEv2, HIP/Diet HIP) may have an advantage regarding the cryptographic overhead compared to application-focused security solutions (TLS/DTLS). This is due to the fact that application-focused solutions require cryptographic operations per group application, whereas network layer approaches may allow to share secure group associations between multiple applications (e.g., for neighbor discovery and routing or service discovery). Hence, implementing shared features lower in the communication stack can avoid redundant security measures.

A number of group key solutions have been developed in the context of the IETF working group MSEC in the context of the MIKEY architecture [WG-MSEC][RFC4738]. These are specifically tailored for multicast and group broadcast applications in the Internet and should also be considered as candidate solutions for group key agreement in the IoT. The MIKEY architecture describes a coordinator entity that disseminates symmetric keys over pair-wise end-to-end secured channels. However, such a centralized approach may not be applicable in a distributed environment, where the choice of one or several coordinators and the management of the group key is not trivial.

5.3.3. Mobility and IP Network Dynamics

It is expected that many things (e.g., wearable sensors, and user devices) will be mobile in the sense that they are attached to different networks during the lifetime of a security association. Built-in mobility signaling can greatly reduce the overhead of the cryptographic protocols because unnecessary and costly re-establishments of the session (possibly including handshake and key agreement) can be avoided. IKEv2 supports host mobility with the MOBIKE [RFC4555][RFC4621] extension. MOBIKE refrains from applying heavyweight cryptographic extensions for mobility. However, MOBIKE mandates the use of IPsec tunnel mode which requires to transmit an additional IP header in each packet. This additional overhead could be alleviated by using header compression methods or the Bound End-to-End Tunnel (BEET) mode [ID-Nikander], a hybrid of tunnel and transport mode with smaller packet headers.

HIP offers a simple yet effective mobility management by allowing hosts to signal changes to their associations [RFC5206]. However, slight adjustments might be necessary to reduce the cryptographic

costs, for example, by making the public-key signatures in the mobility messages optional. Diet HIP does not define mobility yet but it is sufficiently similar to HIP to employ the same mechanisms. TLS and DTLS do not have standards for mobility support, however, work on DTLS mobility exists in the form of an Internet draft [ID-Williams]. The specific need for IP-layer mobility mainly depends on the scenario in which nodes operate. In many cases, mobility support by means of a mobile gateway may suffice to enable mobile IoT networks, such as body sensor networks. However, if individual things change their point of network attachment while communicating, mobility support may gain importance.

6. Security Suites for the IP-based Internet of Things

Different applications have different security requirements and needs and, depending on various factors, such as device capability, availability of network infrastructure, security services needed, usage, etc., the required security protection may vary from "no security" to "full-blown security". For example, applications may have different needs regarding authentication and confidentiality. While some application might not require any authentication at all, others might require strong end-to-end authentication. In terms of secure bootstrapping of keys, some applications might assume the existence and online availability of a central key-distribution-center (KDC) within the 6LoWPAN network to distribute and manage keys; while other applications cannot rely on such a central party or their availability.

Thus, it is essential to define security profiles to better tailor security solutions for different applications with the same characteristics and requirements. This provides a means of grouping applications into profiles and then defines the minimal required security primitives to enable and support the security needs of the profile. The security elements in a security profile can be classified according to Section 3.1, namely:

- 1 Security architecture,
- 2 Security model,
- 3 Security bootstrapping,
- 4 Network security, and

5 Application security.

In order to (i) guide the design process by identifying open gaps; (ii) allow for later interoperability; and (iii) prevent possible security misconfigurations, this section defines a number of generic security profiles with different security needs. Each security profile is identified by:

- 1 a short description,
- 2 an exemplary application that might use/require such a security policy,
- 3 the security requirements for each of the above security aspects according to our classification in Section 3.1.

These security profiles can serve to guide the standardization process, since these explicitly describe the basic functionalities and protocols required to get different use cases up and running. It can allow for later interoperability since different manufacturers can describe the implemented security profile in their products. Finally, the security profiles can avoid possible security misconfigurations, since each security profile can be bound to a different application area so that it can be clearly defined which security protocols and approaches can be applied where and under which circumstances.

Note that each of these security profiles aim at summarizing the required security requirements for different applications and at providing a set of initial security features. In other words, these profiles reflect the need for different security configurations, depending on the threat and trust models of the underlying applications. In this sense, this section does not provide an overview of existing protocols as done in previous sections of the Internet Draft, but it rather explicitly describes what should be in place to ensure secure system operation. Observe also that this list of security profiles is not exhaustive and that it should be considered just as an example not related to existing legal regulations for any existing application. These security profiles are summarized in the table below:

	Application	Description
SecProf_0	No security needs	6LoWPAN/CoAP is used without security
SecProf_1	Home usage	Enables operation between home things without interaction with central device
SecProf_2	Managed Home usage	Enables operation between home things. Interaction with a central and local device is possible
SecProf_3	Industrial usage	Enables operation between things. Relies on central (local or backend) device for security
SecProf_4	Advanced Industrial usage	Enables ad-hoc operation between things and relies on central device or on a collection of control devices

Security profiles and application areas.

Figure 6

The classification in the table considers different potential applications and situations in which their security needs change due to different operational features (network size, existence of a central device, connectivity to the Internet, importance of the exchanged information, etc) or threat model (what are the assets that an attacker looks for). As already pointed out, this set of scenarios is exemplary and they should be further discussed based on a broader consensus.

SecProf_0 is meant for any application that does not require security. Examples include applications during system development, system testing, or some very basic applications in which security is not required at all.

The second security suite (SecProf_1) is catered for environments in which 6LoWPAN/CoAP can be used to enable communication between things in an ad-hoc manner and the security requirements are minimal. An example, is a home application in which two devices should exchange information and no further connection with other devices (local or with a backend) is required. In this scenario, value of the exchanged information is low and that it usually happen in a confined room, thus, it is possible to have a short period of time during

which initial secrets can be exchanged in the clear. Due to this fact, there is no requirement to enable devices from different manufacturers to interoperate in a secure way (keys are just exchanged). The expected network size of applications using this profile is expected to be small such that the provision of network security, e.g., secure routing, is of low importance.

The next security suite (SecProf_2) represents an evolution of SecProf_1 in which, e.g., home devices, can be managed locally. A first possibility for the securing domain management refers to the creation of a centrally managed security domain without any connectivity to the Internet. The central device used for management can serve as, e.g., a key distribution center including policies for key update, storage, etc. The presence of a central device can help in the management of larger networks. Network security becomes more relevant in this scenario since the 6LoWPAN/CoAP network can be prone to Denial of Service attacks (e.g., flooding if L2 is not protected) or routing attacks.

SecProf_3 considers that a central device is always required for network management. Example applications of this profile include building control and automation, sensor networks for industrial use, environmental monitoring, etc. As before, the network manager can be located in the 6LoWPAN/CoAP network and handle key management. In this case, the first association of devices to the network is required to be done in a secure way. In other words, the threat model requires measurements to protect against any vulnerable period of time. This step can involve the secure transmission of keying materials used for network security at different layers. The information exchanged in the network is considered to be valuable and it should be protected in the sense of pairwise links. Commands should be secured and broadcast should be secured with entity authentication [ID-CoAPMulticast]. Network should be protected from attacks. A further extension to this use case is to allow for remote management. A "backend manager" is in charge of managing SW or information exchanged or collected within the 6LoWPAN/CoAP network. This requires connection of devices to the Internet over a 6LBR involving a number of new threats that were not present before. A list of potential attacks include: resource-exhaustion attacks from the Internet; amplification attacks; trust issues related a HTTP-CoAP proxy [ID-proHTTPCoAP], etc. This use case requires protecting the communication from a device in the backend to a device in the 6LoWPAN/CoAP network, end-to-end. This use case also requires measures to provide the 6LBR with the capability of dropping fake requests coming from the Internet. This becomes especially challenging when the 6LBR is not trusted and access to the exchanged information is limited; and even more in the case of a HTTP-CoAP proxy since protocol translation is required. This use case should

take care of protecting information accessed from the backend due to privacy issues (e.g., information such as type of devices, location, usage, type and amount of exchanged information, or mobility patterns can be gathered at the backend threatening the privacy sphere of users) so that only required information is disclosed.

The last security suite (SecProf_4) essentially represents interoperability of all the security profiles defined previously. It considers applications with some additional requirements regarding operation such as: (i) ad-hoc establishment of security relationships between things (potentially from different manufacturers) in non-secure environments or (ii) dynamic roaming of things between different 6LoWPAN/CoAP security domains. Such operational requirements pose additional security requirements, e.g., in addition to secure bootstrapping of a device within a 6LoWPAN/CoAP security domain and the secure transfer of network operational key, there is a need to enable inter-domains secure communication to facilitate data sharing.

The above description illustrates how different applications of 6LoWPAN/CoAP networks involve different security needs. In the following sections, we summarize the expected security features or capabilities for each the security profile with regards to "Security Architecture", "Security Model", "Security Bootstrapping", "Network Security", and "Application Security".

6.1. Security Architecture

The choice of security architecture has many implications regarding key management, access control, or security scope. A distributed (or ad-hoc) architecture means that security relationships between things are setup on the fly between a number of objects and kept in a decentralized fashion. A locally centralized security architecture means that a central device, e.g., the 6LBR, handles the keys for all the devices in the security domain. Alternatively, a central security architecture could also refer to the fact that smart objects are managed from the backend. The security architecture for the different security profiles is classified as follows.

	Description
SecProf_0	-
SecProf_1	Distributed
SecProf_2	Distributed able to move centralized (local)
SecProf_3	Centralized (local &/or backend)
SecProf_4	Distributed & centralized (local &/or backend)

Security architectures in different security profiles.

Figure 7

In "SecProf_1", management mechanisms for the distributed assignment and management of keying materials is required. Since this is a very simple use case, access control to the security domain can be enabled by means of a common secret known to all devices. In the next security suite (SecProf_2), a central device can assume key management responsibilities and handle the access to the network. The last two security suites (SecProf_3 and SecProf_4) further allow for the management of devices or some keying materials from the backend.

6.2. Security Model

While some applications might involve very resource-constrained things such as, e.g., a humidity, pollution sensor, other applications might target more powerful devices aimed at more exposed applications. Security parameters such as keying materials, certificates, etc must be protected in the thing, for example by means of tamper-resistant hardware. Keys may be shared across a thing's networking stack to provide authenticity and confidentiality in each networking layer. This would minimize the number of key establishment/agreement handshake and incurs less overhead for constrained thing. While more advance applications may require key separation at different networking layers, and possibly process separation and sandboxing to isolate one application from another. In this sense, this section reflects the fact that different applications require different sets of security mechanisms.

	Description
SecProf_0	-
SecProf_1	No tamper resistant Sharing keys between layers
SecProf_2	No tamper resistant Sharing keys between layers
SecProf_3	Tamper resistant Key and process separation
SecProf_4	(no) Tamper resistant Sharing keys between layers/Key and process separation Sandbox

Thing security models in different security profiles.

Figure 8

6.3. Security Bootstrapping and Management

Bootstrapping refers to the process by which a thing initiates its life within a security domain and includes the initialization of secure and/or authentic parameters bound to the thing and at least one other device in the network. Here, different mechanisms may be used to achieve confidentiality and/or authenticity of these parameters, depending on deployment scenario assumptions and the communication channel(s) used for passing these parameters. The simplest mechanism for initial set-up of secure and authentic parameters is via communication in the clear using a physical interface (USB, wire, chip contact, etc.). Here, one commonly assumes this communication channel is secure, since eavesdropping and/or manipulation of this interface would generally require access to the physical medium and, thereby, to one or both of the devices themselves. This mechanism was used with the so-called original "resurrecting duckling" model, as introduced in [PROC-Stajano]. This technique may also be used securely in wireless, rather than wired, set-ups, if the prospect of eavesdropping and/or manipulating this channel are dim (a so-called "location-limited" channel [PROC-Smetters-04, PROC-Smetters-02]). Examples hereof include the communication of secret keys in the clear using near field communication (NFC) - where the physical channel is purported to have very limited range (roughly 10cm), thereby thwarting eavesdropping by

far-away adversarial devices, and in-the-clear communication during a small time window (triggered by, e.g., a button-push) - where eavesdropping is presumed absent during this small time window. With the use of public-key based techniques, assumptions on the communication channel can be relaxed even further, since then the cryptographic technique itself provides for confidentiality of the channel set-up and the location-limited channel - or use of certificates - rules out man-in-the-middle attacks, thereby providing authenticity [PROC-Smetters-02]. The same result can be obtained using password-based public-key protocols [SPEKE], where authenticity depends on the (weak) password not being guessed during execution of the protocol. It should be noted that while most of these techniques realize a secure and authentic channel for passing parameters, these generally do not provide for explicit authorization. As an example, with use of certificate-based public-key based techniques, one may obtain hard evidence on whom one shares secret and/or authentic parameters with, but this does not answer the question as to whether one wishes to share this information at all with this specifically identified device (the latter usually involves a human-decision element). Thus, the bootstrapping mechanisms above should generally be complemented by mechanisms that regulate (security policies for) authorization. Furthermore, the type of bootstrapping is very related to the required type of security architecture. Distributed bootstrapping means that a pair of devices can setup a security relationship on the fly, without interaction with a central device elsewhere within the system. In many cases, it is handy to have a distributed bootstrapping protocol based on existing security protocols (e.g., DTLS in CoAP) required for other purposes: this reduces the amount of required software. A centralized bootstrapping protocol is one in which a central device manages the security relationships within a network. This can happen locally, e.g., handled by the 6LBR, or remotely, e.g., from a server connected via the Internet. The security bootstrapping for the different security profiles is as follows.

	Description
SecProf_0	-
SecProf_1	* Distributed, (e.g., Resurrecting duckling) * First key distribution happens in the clear
SecProf_2	* Distributed, (e.g., Resurrecting duckling) * Centralized (local), 6LBR acts as KDC * First key distribution occurs in the clear, if the KDC is available, the KDC can manage network access
SecProf_3	* 6LBR acts as KDC. It handles node joining, provides them with keying material from L2 to application layers * Bootstrapping occurs in a secure way - either in secure environment or the security mechanisms ensure that eavesdropping is not possible. * KDC and backend can implement secure methods for network access
SecProf_4	* As in SecProf_3.

Security bootstrapping methods in different security profiles

Figure 9

6.4. Network Security

Network security refers to the mechanisms used to ensure the secure transport of 6LoWPAN frames. This involves a multitude of issues ranging from secure discovery, frame authentication, routing security, detection of replay, secure group communication, etc. Network security is important to thwart potential attacks such as denial-of-service (e.g., through message flooding) or routing attacks.

The Internet Draft [ID-Tsao] presents a very good overview of attacks and security needs classified according to the confidentiality, integrity, and availability needs. A potential limitation is that there exist no differentiation in security between different use cases and the framework is limited to L3. The security suites gathered in the present ID aim at solving this by allowing for a more flexible selection of security needs at L2 and L3.

	Description
SecProf_0	-
SecProf_1	<ul style="list-style-type: none"> * Network key creating a home security domain at L2 ensuring authentication and freshness of exchanged data * Secure multicast does not ensure origin authentication * No need for secure routing at L3
SecProf_2	<ul style="list-style-type: none"> * Network key creating a home security domain at L2 ensuring authentication and freshness of exchanged data * Secure multicast does not ensure origin authentication * No need for secure routing at L3
SecProf_3	<ul style="list-style-type: none"> * Network key creating an industry security domain at L2 ensuring authentication and freshness of exchanged data * Secure routing needed (integrity & availability) at L3 within 6LoWPAN/CoAP * Secure multicast requires origin authentication
SecProf_4	<ul style="list-style-type: none"> * Network key creating an industry security domain at L2 ensuring authentication and freshness of exchanged data * Inter-domain authentication/secure handoff * Secure routing needed at L3 * Secure multicast requires origin authentication * 6LBR (HTTP-CoAP proxy) requires verification of forwarded messages and messages leaving or entering the 6LoWPAN/CoAP network.

Network security needs in different security profiles

Figure 10

6.5. Application Security

In the context of 6LoWPAN/CoAP networks, application security refers firstly to the configuration of DTLS used to protect the exchanged information. It further refers to the measures required in potential translation points (e.g., a (HTTP-CoAP) proxy) where information can be collected and the privacy sphere of users in a given security domain is endangered. Application security for the different security profiles is as follows.

	Description
SecProf_0	-
SecProf_1	-
SecProf_2	<ul style="list-style-type: none"> * DTLS is used for end-to-end application security between management device and things and between things * DTLS ciphersuites configurable to provide confidentiality and/or authentication and/or freshness * Key transport and policies for generation of session keys are required
SecProf_3	<ul style="list-style-type: none"> * Requirements as in SecProf_2 and * DTLS is used for end-to-end application security between management device and things and between things * Communication between KDC and each thing secured by pairwise keys * Group keys for communication in a group distributed by KDC * Privacy protection should be provided in translation points
SecProf_4	<ul style="list-style-type: none"> * Requirements as in SecProf_3 and * TLS or DTLS can be used to send commands from the backend to the 6LBR or things in a 6LoWPAN/CoAP network * End-to-end secure connectivity from backend required * Secure broadcast in a network from backend required

Application security methods in different security profiles

Figure 11

The first two security profiles do not include any security at the application layer. The reason is that, in the first case, security is not provided and, in the second case, it seems reasonable to provide basic security at L2. In the third security profile (SecProf_2), DTLS becomes the way of protecting messages at application layer between things and with the KDC running on a 6LBR. A key option refers to the capability of easily configuring DTLS to provide a subset of security services (e.g., some applications do not require confidentiality) to reduce the impact of security in the system operation of resource-constrained things. In addition to basic key management mechanisms running within the KDC, communication protocols for key transport or key update are required. These

protocols could be based on DTLS. The next security suite (SecProf_3) requires pairwise keys for communication between things within the security domain. Furthermore, it can involve the usage of group keys for group communication. If secure multicast is implemented, it should provide origin authentication. Finally, privacy protection should be taken into account to limit access to valuable information -- such as identifiers, type of collected data, traffic patterns -- in potential translation points (proxies) or in the backend. The last security suite (SecProf_4) further extends the previous set of requirements considering security mechanisms to deal with translations between TLS and DTLS or for the provision of secure multicast within a 6LoWPAN/CoAP network from the backend.

7. Next Steps towards a Flexible and Secure Internet of Things

As evident from the discussions of the lifecycle of a thing, the IP security challenges in the Internet of Things, and the security suits, it is important to define specific steps towards a feasible security concept for the IP-based IoT with special emphasis on the employed security protocols. Due to the resource constraints of IoT devices and the specific limitations of IoT network scenarios, this security concept will differ from today's Internet security architectures. Therefore, focusing on the protection of a single protocol such as CoAP will not suffice. The aim is to put together the key security aspects of IoT, making clear how the architectural, operational, and technical aspects impact the protocol design and choices. Next, we list the most important topics towards achieving this goal.

- 1 Performance assessment of candidate IP security protocols on resource constrained devices in the context of the IoT and its requirements. To the best of our knowledge, the implementation feasibility of existing protocols on constrained devices (e.g., 8-bit CPU and limited RAM budget) has not been fully assessed so far. Hence, a performance evaluation of candidate security solutions is required in terms of CPU and communication overhead, energy consumption, and memory requirements for a better understanding of the impact of existing IP security solutions on the IoT ecosystem. Such analysis should be carried out on a well-defined set of standard node platforms as a reference for the subsequent performance evaluation and comparison. This benchmarking should not just involve cryptographic constructs and protocols, but also include implementation benchmarks for security policies, since these may impact overall system performance and network traffic (an example of this would be policies including frequent key updates, which would necessitate securely propagating these to all devices in the network). These

results then serve as a basis for the design of a suitable security architecture for the IoT.

- 2 In-depth evaluation of the security mechanisms employed in IP security protocols in order to design possible adaptations for these protocols fitting the IoT requirements. Here, we focus on the discussion of the challenges for IP security solutions in the IoT and hint at IP security protocol properties that violate IoT requirements. Possible adaptations should allow existing protocols to not only fulfill the performance requirements of the IoT, but also to provide the security properties they have been designed for in the context of the Internet architecture. An example might be the incorporation of new security mechanisms for IoT-specific DoS protection. This is due to the fact that Internet protocols have been designed with comparably high assumptions regarding MTU size. However, IEEE 802.15.4 networks have physical packets of 127 B. Thus, IoT candidate security solutions should avoid prohibitively long messages in order to (i) prevent high resource usage in the network and individual nodes due to fragmentation, and (ii) mitigate attackers from launching fragmentation-based DoS attacks.
- 3 Definition of a flexible security architecture matching the different operational scenarios during the lifecycle of a thing. IoT scenarios might comprise both centralized and ad-hoc security domains. Hence, the IoT security architecture should incorporate the properties of a fully centralized architecture as well as allow devices to be paired together initially without the need for a trusted third party to create ad-hoc security domains comprising a number of nodes. These ad-hoc security domains could then be added later to the Internet via a single, central node or via a collection of nodes (thus, facilitating implementation of a centralized or distributed architecture, respectively). The architecture should also facilitate scenarios, where an operational network may be partitioned or merged, and where hand-over of control functionality of a single device or even of a complete subnetworks may occur over time may (if only to facilitate smooth device repair/replacement without the need for a hard "system reboot" or to realize ownership transfer). Thus, the IoT can transparently and effortlessly move from an ad-hoc security domain to a centrally-managed single security domain or a heterogeneous collection of security domains and vice-versa. We propose to further define the security suits included in Section 6, and focus first in simple deployment with basic security needs, extending them later to more complex deployments.

- 4 Selection and coordination of an IP security suite. With a good understanding of IP security in the IoT and adapted candidate solutions, a security protocol suite can be chosen that fulfills the IoT requirements during the different phases in the lifecycle of a thing. Such a protocol suite must be closely interconnected across layers to ensure security efficiency as resource limitations make it challenging to secure all layers individually. In this regard, securing only the application layer leaves the network open to attacks, while security focused only at the network and link layer might introduce possible inter-application security threats. Hence, the limited resources of things may require sharing of keying material and common security mechanisms between layers. It is required that the data format of the keying material is standardized to facilitate cross layer interaction. Additionally, cross layer concepts should be considered for an IoT-driven re-design of Internet security protocols. To our knowledge, such a "holistic approach to security architectural design is still a nascent area.
- 5 Definition of a standard lightweight bootstrapping protocol for the commissioning of devices with keying materials, addresses, and network parameters in order to allow for secure network communication. The bootstrapping protocol should be reusable and lightweight to fit into small devices. Such a standard bootstrapping protocol must allow for commissioning of devices from different manufacturers in both centralized and ad-hoc scenarios and facilitate transitions of control devices during the device's and system's lifecycle. Examples of the latter include scenarios that involve hand-over of control, e.g., from a configuration device to an operational management console and involving replacement of such a control device. A key challenge for secure bootstrapping of a device in a centralized architecture is that it is currently not feasible to commission a device when the adjacent devices have not been commissioned yet.

8. Security Considerations

This document reflects upon the requirements and challenges of the security architectural framework for Internet of Things.

9. IANA Considerations

This document contains no request to IANA.

10. Acknowledgements

We gratefully acknowledge feedback and fruitful discussion with Tobias Heer and Robert Moskowitz.

11. References

11.1. Normative References

- [ID-CoAP] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), Jan 2011.
- [ID-CoAPMulticast]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-rahman-core-groupcomm-05 (work in progress), May 2011.
- [ID-Daniel]
Park, S., Kim, K., Haddad, W., Chakrabarti, S., and J. Laganier, "IPv6 over Low Power WPAN Security Analysis", Internet Draft draft-daniel-6lowpan-security-analysis-05, Mar 2011.
- [ID-Duffy]
Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA) Relay Element", draft-ohba-pana-relay-03 (work in progress), Nov 2010.
- [ID-HIP] Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-02 (work in progress), Jul 2010.
- [ID-KIM] Kim, E., Kaspar, D., and J. Vasseur, "Design and Application Spaces for 6LoWPANs", draft-ietf-6lowpan-usecases-09 (work in progress), January 2011.
- [ID-Moskowitz]
Moskowitz, R., Jokela, P., Henderson, T., and T. Heer, "Host Identity Protocol Version 2", draft-ietf-hip-rfc5201-bis-03 (work in progress), Oct 2010.
- [ID-Nikander]
Nikander, P. and J. Melen, "A Bound End-to-End Tunnel

(BEET) mode for ESP", Internet
Draft draft-nikander-esp-beet-mode-09, Feb 2009.

[ID-OFlynn]

O'Flynn, C., Sarikaya, B., Ohba, Y., Cao, Z., and R.
Cragie, "Security Bootstrapping of Resource-Constrained
Devices", draft-oflynn-core-bootstrapping-03 (work in
progress), Nov 2010.

[ID-Tsao]

Tsao, T., Alexander, R., Dohler, M., Daza, V., and A.
Lozano, "A Security Framework for Routing over Low Power
and Lossy Networks", Internet
Draft draft-ietf-roll-security-framework-04, Jan 2011.

[ID-Williams]

Williams, M. and J. Barrett, "Mobile DTLS", Internet
Draft draft-barrett-mobile-dtls-00, Sept 2009.

[ID-proHTTPCoAP]

Castellani, A., Loreto, S., Rahman, A., Fossati, T., and
E. Dijk, "Best practices for HTTP-CoAP mapping
implementation", draft-castellani-core-http-mapping-00
(work in progress), July 2011.

[RFC2818]

Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3261]

Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston,
A., Peterson, J., Sparks, R., Handley, M., and E.
Schooler, "SIP: Session Initiation Protocol", RFC 3261,
June 2002.

[RFC3748]

Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H.
Levkowetz, "Extensible Authentication Protocol (EAP)",
RFC 3748, June 2004.

[RFC3756]

Nikander, P., Kempf, J., and E. Nordmark, "IPv6 Neighbor
Discovery (ND) Trust Models and Threats", RFC 3756,
May 2004.

[RFC3833]

Atkins, D. and R. Austein, "Threat Analysis of the Domain
Name System (DNS)", RFC 3833, Aug 2004.

[RFC4016]

Parthasarathy, M., "Protocol for Carrying Authentication
and Network Access (PANA)", RFC 4016, Mar 2005.

[RFC4246]

Dierks, T. and E. Rescorla, "The Transport Layer Security
(TLS) Protocol version 1.2", RFC 5246, Aug 2008.

- [RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, Jan 2006.
- [RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol (updated by RFC5282)", RFC 4306, Aug 2008.
- [RFC4555] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, Jun 2006.
- [RFC4621] Kivinen, T. and H. Tschofenig, "Design of the IKEv2 Mobility and Multihoming (MOBIKE) Protocol", RFC 4621, Aug 2006.
- [RFC4738] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 4738, Aug 2004.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, Aug 2007.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, Sept 2007.
- [RFC5191] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.
- [RFC5201] Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, "Host Identity Protocol", RFC 5201, Apr 2008.
- [RFC5206] Nikander, P., Henderson, T., Vogt, C., and J. Arkko, "End-Host Mobility and Multi-homing with the Host Identity Protocol", RFC 5206, Aug 2006.
- [RFC5238] Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, May 2008.
- [RFC5246] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 5246, May 2006.
- [RFC5713] Moustafa, H., Tschofenig, H., and S. De Cnodder, "Security Threats and Security Requirements for the Access Node Control Protocol (ANCP)", RFC 5317, Jan 2010.

- [RFC5903] Fu, D. and J. Solinas, "Elliptic Curve Groups Modulo a Prime (ECP Groups) for IKE and IKEv2", RFC 5903, June 2010.

11.2. Informative References

- [AUTO-ID] "AUTO-ID LABS", Web <http://www.autoidlabs.org/>, Sept 2010.
- [BACNET] "BACnet", Web <http://www.bacnet.org/>, Feb 2011.
- [DALI] "DALI", Web <http://www.dalibydesign.us/dali.html>, Feb 2011.
- [JOURNAL-Perrig]
Perrig, A., Szewczyk, R., Wen, V., Culler, D., and J. Tygar, "SPINS: Security protocols for Sensor Networks", Journal Wireless Networks, Sept 2002.
- [NIST] Dworkin, M., "NIST Specification Publication 800-38B", 2005.
- [PROC-Chan]
Chan, H., Perrig, A., and D. Song, "Random Key Predistribution Schemes for Sensor Networks", Proceedings IEEE Symposium on Security and Privacy, 2003.
- [PROC-Gupta]
Gupta, V., Wurm, M., Zhu, Y., Millard, M., Fung, S., Gura, N., Eberle, H., and S. Shantz, "Sizzle: A Standards-based End-to-End Security Architecture for the Embedded Internet", Proceedings Pervasive Computing and Communications (PerCom), 2005.
- [PROC-Smetters-02]
Balfanz, D., Smetters, D., Steward, P., and H. Chi Wong, "Talking To Strangers: Authentication in Ad-Hoc Wireless Networks", Paper NDSS, 2002.
- [PROC-Smetters-04]
Balfanz, D., Durfee, G., Grinter, R., Smetters, D., and P. Steward, "Network-in-a-Box: How to Set Up a Secure Wireless Network in Under a Minute", Paper USENIX 2004, 2004.
- [PROC-Stajano-99]
Stajano, F. and R. Anderson, "Resurrecting Duckling - Security Issues for Adhoc Wireless Networks", Paper Security Protocols, 7th International Workshop

Proceedings, Nov 1999.

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

[THESIS-Langheinrich]
Langheinrich, M., "Personal Privacy in Ubiquitous Computing", PhD Thesis ETH Zurich, 2005.

[WG-6LoWPAN]
"IETF 6LoWPAN Working Group",
Web <http://tools.ietf.org/wg/6lowpan/>, Feb 2011.

[WG-CoRE] "IETF Constrained RESTful Environment (CoRE) Working Group", Web <https://datatracker.ietf.org/wg/core/charter/>, Feb 2011.

[WG-MSEC] "MSEC Working Group",
Web <http://datatracker.ietf.org/wg/msec/>.

[ZB] "ZigBee Alliance", Web <http://www.zigbee.org/>, Feb 2011.

Authors' Addresses

Oscar Garcia-Morchon
Philips Research
High Tech Campus
Eindhoven, 5656 AA
The Netherlands

Email: oscar.garcia@philips.com

Sye Loong Keoh
Philips Research
High Tech Campus
Eindhoven, 5656 AA
The Netherlands

Email: sye.loong.keoh@philips.com

Sandeep S. Kumar
Philips Research
High Tech Campus
Eindhoven, 5656 AA
The Netherlands

Email: sandeep.kumar@philips.com

RenA(C) Hummen
RWTH Aachen University
Templergraben 55
Aachen, 52056
Germany

Email: rene.hummen@cs.rwth-aachen.de

Rene Struik
Struik Security Consultancy
Toronto,
Canada

Email: rstruik.ext@gmail.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 24, 2012

X. He
YC.Ma
Hitachi (China) Research and
Development Corporation
Z.Cao
China Mobile
October 22, 2011

Energy Aware Proxy Discovery for CoAP
draft-he-core-energy-aware-pd-00.txt

Abstract

CoRE defines a mechanism for resource discovery based on Web linking with discovery, registration, modification, and other procedures. But energy efficiency is very important for resource constrained devices. This specification shows an efficient method for CoAP proxy finding the resource from end-points by reducing multicast messages.

The current version -00 of this document is just an initial draft that is intended to spark discussion.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on April 11, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
2. Resource discovery analysis.	4
3. Energy aware proxy discovery	5
3.1. Reduced Protocol Operations	5
3.2. Example	6
4. IANA Consideration	6
5. Security Considerations	6
6. Normative References	7

1. Introduction

The Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained nodes and networks. CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation [I-D.shelby-core-coap-req]. As being the main work of CoRE, CoAP defined a proxy mechanism for CoAP end-point, that proxy can be tasked by CoAP clients to perform requests on their behalf [I-D.ietf-core-coap].

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

The proxy mechanism should support this RD function in resource constrained environments. There are several methods to discovery proxy by a CoAP end-point, including assuming a default location (e.g. on an Edge Router in a LoWPAN), by assigning an anycast address to the proxy, using DHCP, or using the CoRE Link Format. While reducing energy consumption is essential for battery operated nodes in some devices, which is one of the most important work in M2M communication. Node's energy usage depends on network messages it has to receive and or respond. Thus the discovery procedure should be optimized with energy aware consideration.

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. Resource discovery analysis

As mentioned in other drafts, resource discovery is performed by sending either a multicast or unicast GET request to `/.well-known/core` and including a Resource Type (`rt`) parameter [I-D.ietf-core-link-format] with the value "core-rd" in the query string. Upon success, the response will contain a payload with a link format entry for each RD discovered, with the URL indicating the root resource of the RD.

After discovering the location of an RD, an end-point MAY register its resources to the RD's registration interface. This interface accepts a POST from an end-point containing the list of resources to be added to the directory as the message payload in the CoRE Link Format along with query string parameters indicating the name of the end-point, an optional node identifier and the lifetime of the registration. Upon success, the response will be 2.01 "Created" [I-D.shelby-core-resource-directory].

That means once resource discovery needs twice communication process between end-point and proxy. For energy saving point of view, this procedure should be optimized. Another draft indicates more efficient method. A CoAP server that wants to make itself discoverable sends a POST request to the default discovery URI of any Candidate CoAP Server Discovery Server [I-D.bormann-core-simple-server-discovery].

This draft shows more details about energy aware proxy discovery mechanism for CoAP.

3. Energy aware proxy discovery

3.1. Reduced Protocol Operations

The Energy aware proxy reduces discovery and registration processes into one.

The end-point acting as a server will use server IP address, the CoAP default port[I-D.ietf-core-coap], and the absolute path `"/.well-known/core"`[I-D.ietf-core-link-format] to build its POST request. And this request will be send to its Neighbor by unicast (as using 6LOWPAN or IPv6) or to a multicast address.

The POST request is a link-format message, which indicates the service list that requesting server wants to make known to the proxy.

End-point resources in the proxy are kept active for the period also indicated by the lifetime parameter. But unless the end-point needs to refreshing the data with update message, the data will be kept in this lifetime. Then the data will be delete automatically as expired.

The discovery interface is specified as follows:

Interaction: EP -> Proxy

Path: `/.well-known/core`

Method: POST

Content-Type: `application/link-format`

Parameters:

Lifetime (lt): Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, the response will be Failure: 4.00 "Bad Request" .

Host (h): The host identifier or name of the registering node. The maximum length of this parameter is 63 octets. This parameter is combined with the Instance parameter (if any) to form the end-point name. If not included, the proxy MUST generate a unique Host name on behalf of the node.

Instance (ins): The instance of the end-point on this host, if there are multiple. The maximum length of this parameter is 63 octets.

Type (rt): The semantic type of end-point. The maximum length of this parameter is 63 octets.

Success: 2.01 "Created". The Location header of the new resource entry for the end-point could be e.g. in the form `/rd-base/{end-point name}`

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

3.2 Example

The following example shows an end-point with the name "node1" POST temperature resource to a proxy using this interface.

End-point	Neighbor (as a proxy)
--- POST /.well-known/core?rt=core-rd----->	
<-- 2.01 Created Location: /rd/node1 -----	

Req: POST coap://[ff02::1]/.well-known/core?rt=core-rd

Payload:

</sensors/temp>;ct=41;rt="TemperatureC";if="sensor",

Res: 2.01 Created

Location: /rd/node1

4. Security Considerations

TBD.

5. IANA Considerations

This document does not require any IANA actions.

6. Normative References

- [I-D.shelby-core-coap-req]
Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R. Kelsey, "CoAP Requirements and Features", draft-shelby-core-coap-req-01 (work in progress), April 2010.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-06, May 2011.
- [I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-07, July 2011.
- [I-D.bormann-core-simple-server-discovery]
Bormann, C., "CoRE Simple Server Discovery" draft-bormann-core-simple-server-discovery-00, June, 2011
- [I-D.shelby-core-resource-directory]
Shelby, Z., Krco, S., "CoRE Resource Directory" draft-shelby-core-resource-directory-01, September, 2011

Authors' Addresses

Xuan He
Yuanchen Ma
Hitachi R&D China
301 of North Tower C, Raycom
@ Kexuyuan Nanlu, Haidian District
Beijing 100190
China

Email:
xhe@hitachi.cn
ycma@hitachi.cn

Zhen Cao
China Mobile
Unit2, 28 Xuanwumenxi Ave, Xuanwu District
Beijing 100053
China

Email: zehn.cao@gmail.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 12, 2012

C. Bormann
Universitaet Bremen TZI
Z. Shelby, Ed.
Sensinode
July 11, 2011

Blockwise transfers in CoAP
draft-ietf-core-block-04

Abstract

CoAP is a RESTful transfer protocol for constrained nodes and networks. CoAP is based on datagram transport, which limits the maximum size of resource representations that can be transferred without too much fragmentation. The Block options provide a minimal way to transfer larger representations in a block-wise fashion.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 12, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Block-wise transfers	5
2.1. The Block Options	5
2.2. Using the Block Options	7
3. Examples	10
3.1. HTTP Mapping Considerations	15
4. IANA Considerations	17
5. Security Considerations	18
5.1. Mitigating Resource Exhaustion Attacks	18
5.2. Mitigating Amplification Attacks	19
6. Acknowledgements	20
7. References	21
7.1. Normative References	21
7.2. Informative References	21
Appendix A. Historical Note	22
Authors' Addresses	23

1. Introduction

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

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

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

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

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

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

options include:

- o Transfers larger than can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- o No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.
- o The transfer of each block is acknowledged, enabling retransmission if required.
- o Both sides have a say in the block size that actually will be used.
- o The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.
- o If needed, the Block options can also be used as is to provide random access to power-of-two sized blocks within a resource representation.

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

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

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

2. Block-wise transfers

2.1. The Block Options

Type	C/E	Name	Format	Length	Default
19	Critical	Block1	uint	1-3 B	0 (see below)
17	Critical	Block2	uint	1-3 B	0 (see below)

Table 1: Block Option Numbers

The Block1 Option pertains to request payloads (the `_first_` part of the request-response exchange, hence Block1), and the Block2 Option pertains to response payloads (the `_second_` part of the request-response exchange). (Note that either option can be used in both requests and responses, see more below.)

Implementation of either Block option is intended to be optional. However, when it is present in a CoAP message, it **MUST** be processed (or the message rejected); therefore it is identified as a critical option.

The size of the blocks is not fixed by the protocol. To keep the implementation as simple as possible, the Block options support a small range of power-of-two block sizes, from 2^4 (16) to 2^{10} (1024) bytes. One of these seven values can be encoded in three bits (0 for 2^4 to 6 for 2^{10} bytes), which we call the "SZX" (size exponent); the actual block size is then $1 \ll (\text{SZX} + 4)$. (The actual size of the final block in a block-wise transfer may be less than or equal to the power-of-two block size.)

When a resource representation is larger than can be comfortably transferred in a single UDP datagram, a Block option can be used to indicate a block-wise transfer. The value of the option is a 1-, 2- or 3-byte integer, the four least significant bits of which indicate the size and whether the current block-wise transfer is the last block being transferred (indicated by the M or "more" bit). The option value divided by sixteen (the NUM field) is the number of the block currently being transferred, starting from zero. The current transfer is therefore about the "size" bytes starting at byte "block number $\ll (\text{SZX} + 4)$ " (although the power-of-two size need not be reached in the final block). The default value of each of the Block options is zero, indicating that the current block is the first (block number 0) and only (M bit not set) block of the transfer; however, there is no explicit size implied by this default value.

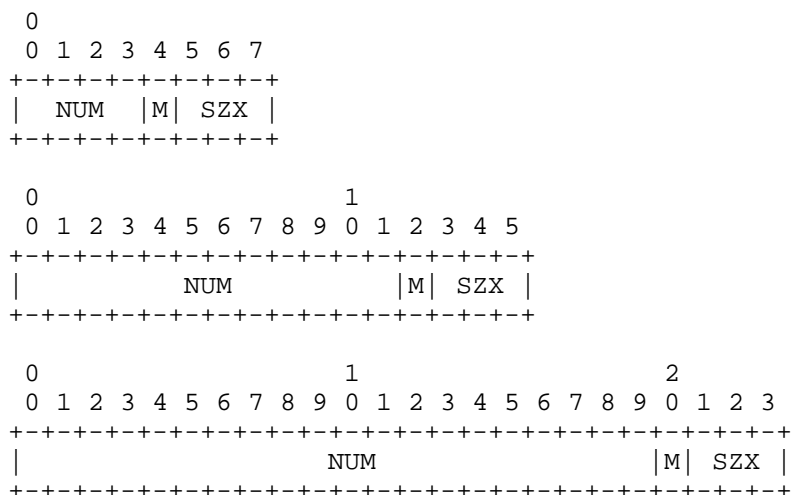


Figure 1: Block option value

(Note that, as an implementation convenience, the option value with the last 4 bits masked out, shifted to the left by the value of SZX, gives the byte position of the block.)

NUM: Block Number. The block number is a variable-size (4, 12, or 20 bit) unsigned integer (uint, see Appendix A of [I-D.ietf-core-coap]) indicating the block number being requested or provided. Block number 0 indicates the first block of a representation.

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

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

The Block options are used in one of three roles:

- o In a request (e.g., GET), the Block2 Option gives the block number requested to be returned in the response and suggests a block size (in the case of block number 0) or echoes the block size of previous blocks received (in the case of block numbers other than 0). In this case, the M bit has no function and MUST be set to zero.
- o A Block2 Option in a response (e.g., for GET), or a Block1 Option in a request (e.g., PUT or POST), describes what block number is contained in the payload of this message, and whether further blocks are required to complete the transfer of that body (M bit). If the M bit is set, the size of the payload body in bytes MUST indeed be the power of two given by the block size. With certain exceptions given below, all blocks for a REST transfer MUST use the same block size, except for the last block (M bit not set).
- o In a response, the Block1 Option indicates what block number is being acknowledged (e.g., for a PUT or POST request). In this case, if the M bit is set it indicates that this response does not carry the final response to the request; this can occur when the M bit was set in the request and the server implements the request atomically (i.e., acts only upon reception of the last block of payload). Conversely, if the M bit is unset, it indicates the block-wise request was enacted now, and the response carries the final response to this request (and to any previous ones with the M bit set in the response's Block1 Option in this sequence of block-wise transfers). Finally, the block size given in such a Block1 Option indicates the largest block size preferred by the server for transfers toward the resource that is the same or smaller than the one used in the initial exchange; the client SHOULD use this block size or a smaller one in all further requests in the transfer sequence.

2.2. Using the Block Options

Using one or both Block options, a single REST operation can be split into multiple CoAP message exchanges. Each of these message exchanges uses their own CoAP Message ID.

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

To influence the block size used in a response, the requester also uses the Block2 Option, giving the desired size, a block number of

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

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

Block-wise transfers can be used to GET resources the representations of which are entirely static (not changing over time at all, such as in a schema describing a device), or for dynamically changing resources. In the latter case, the Block2 Option SHOULD be used in conjunction with the ETag Option, to ensure that the blocks being reassembled are from the same version of the representation: The server SHOULD include an ETag option in each response. If an ETag option is available, the client's reassembler, when reassembling the representation from the blocks being exchanged, MUST compare ETag Options. If the ETag Options do not match in a GET transfer, the requester has the option of attempting to retrieve fresh values for the blocks it retrieved first. To minimize the resulting inefficiency, the server MAY cache the current value of a representation for an ongoing sequence of requests, but there is no requirement for the server to establish any state. The client MAY facilitate identifying the sequence by using the Token Option with a non-default value.

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

In response to a request with a payload (e.g., a PUT or POST transfer), the block size given in the Block1 Option indicates the block size preference of the server for this resource. Obviously, at this point the first block has already been transferred by the client without benefit of this knowledge. Still, the client SHOULD heed the preference and use the block size preferred by the server or a

smaller one. Note that any reduction in the block size may mean that the second request starts with a block number larger than one, as the first request already transferred multiple blocks as counted in the smaller size.

In a blockwise transfer of a request payload (e.g., a PUT or POST) that is intended to be implemented in an atomic fashion at the server, the actual creation/replacement takes place at the time the final block, i.e. a block with the M bit unset in the Block1 Option, is received. If not all previous blocks are available at the server at this time, the transfer fails and error code 4.08 (Request Entity Incomplete) MUST be returned. The error code 4.13 (Request Entity Too Large) can be returned at any time by a server that does not currently have the resources to store blocks for a block-wise request payload transfer that it would intend to implement in an atomic fashion.

If multiple concurrently proceeding block-wise request payload transfer (e.g., PUT or POST) operations are possible, the requester SHOULD use the Token Option to clearly separate the different sequences. In this case, when reassembling the representation from the blocks being exchanged to enable atomic processing, the reassembler MUST compare any Token Options present (and, as usual, taking an absent Token Option to default to the empty Token). If atomic processing is not desired, there is no need to process the Token Option (but it is still returned in the response as usual).

3. Examples

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

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

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

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

Figure 2: Simple blockwise GET

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

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

Figure 3: Blockwise GET with early negotiation

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

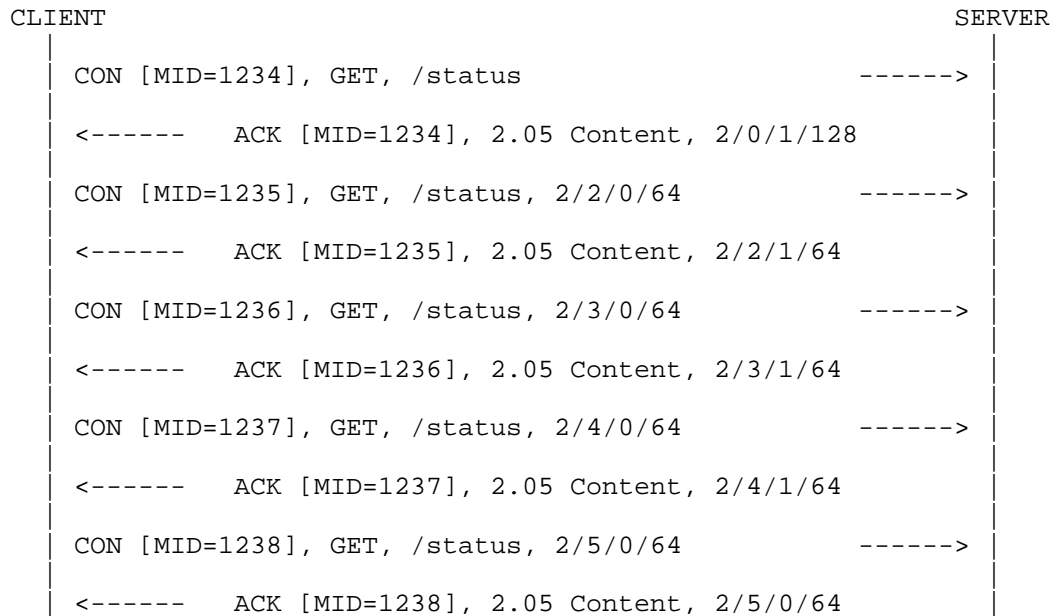


Figure 4: Blockwise GET with late negotiation

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

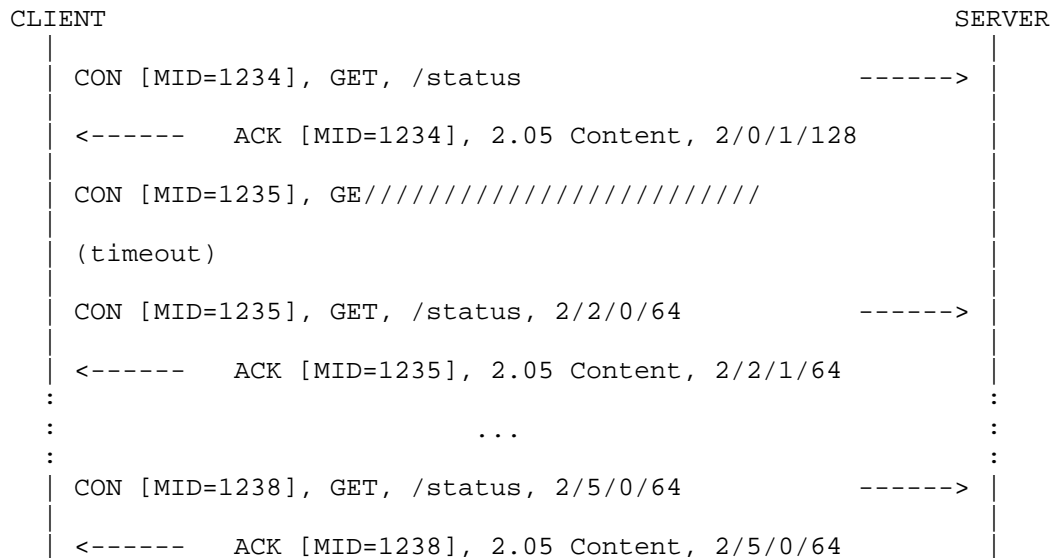


Figure 5: Blockwise GET with late negotiation and lost CON

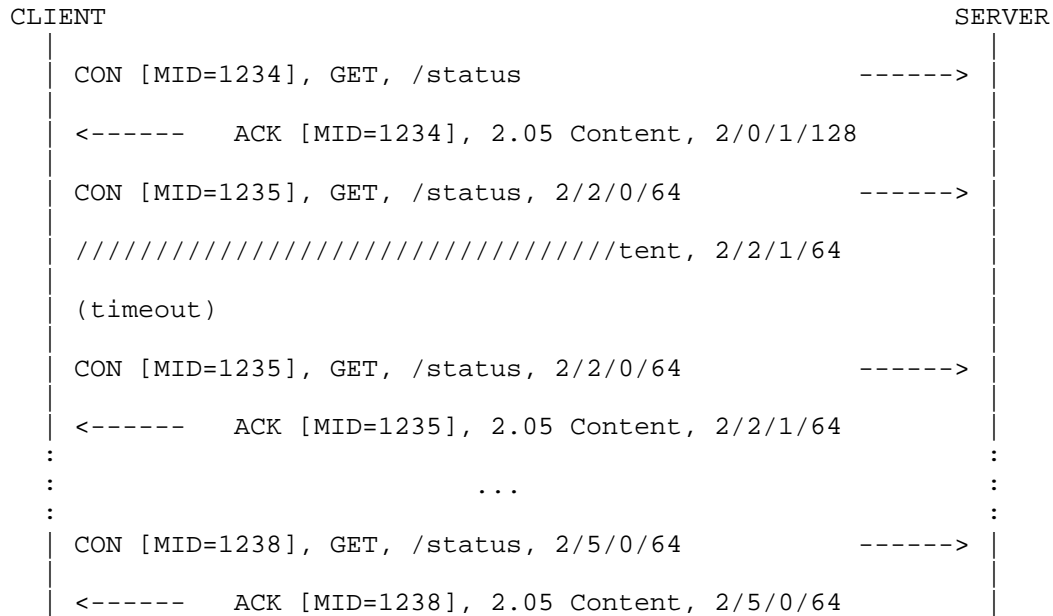


Figure 6: Blockwise GET with late negotiation and lost ACK

The following examples demonstrate a PUT exchange; a POST exchange looks the same, with different requirements on atomicity/idempotence. To ensure that the blocks relate to the same version of the resource representation carried in the request, the client in Figure 7 sets the Token to "v17" in all requests. Note that, as with the GET, the responses to the requests that have a more bit in the request Block2 Option are provisional; only the final response tells the client that the PUT succeeded.

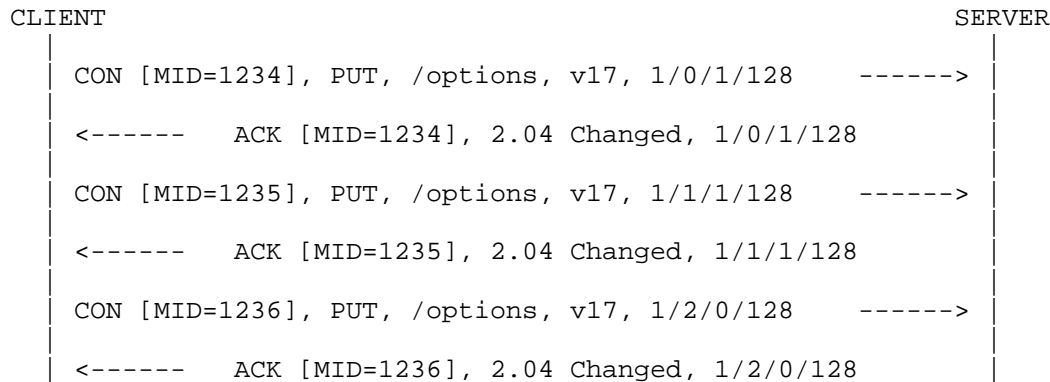


Figure 7: Simple atomic blockwise PUT

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

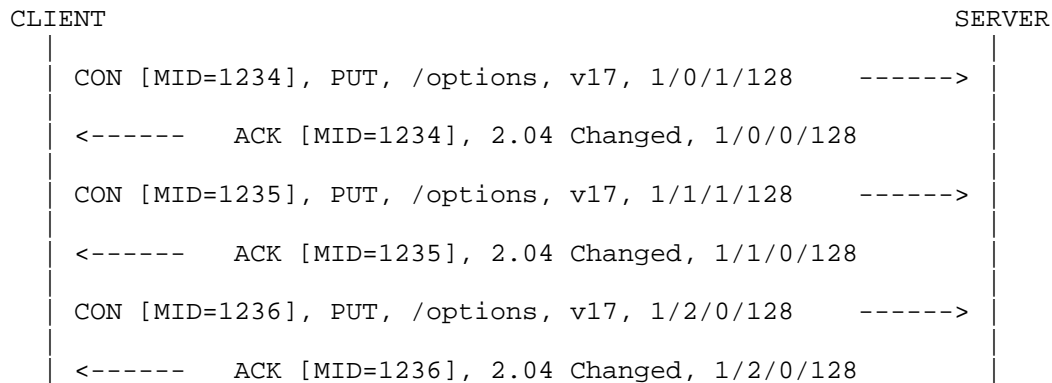


Figure 8: Simple stateless blockwise PUT

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

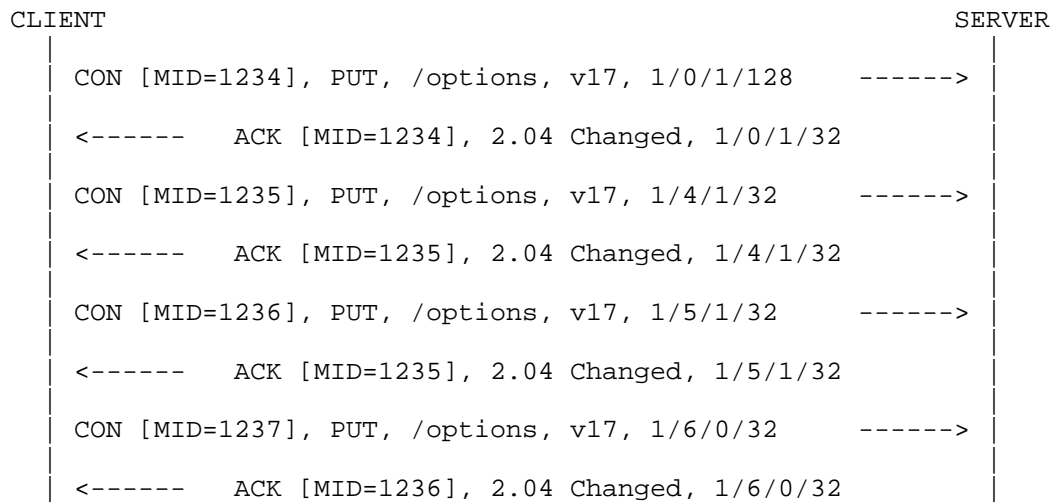


Figure 9: Simple atomic blockwise PUT with negotiation

3.1. HTTP Mapping Considerations

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

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

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

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

(Note that, while the semantics of CoAP 4.08 and HTTP 408 differ, this difference is largely due to the different way the two protocols are mapped to transport. HTTP has an underlying TCP connection, which supplies connection state, so a HTTP 408 status code can immediately be used to indicate that a timeout occurred during transmitting a request through that active TCP connection. The CoAP 4.08 response code indicates one or more missing blocks, which may be due to timeouts or resource constraints; as there is no connection state, there is no way to deliver such a response immediately; instead, it is delivered on the next block transfer. Still, HTTP 408 is probably the best mapping back to HTTP, as the timeout is the most likely cause for a CoAP 4.08. Note that there is no way to distinguish a timeout from a missing block for a server without creating additional state, the need for which we want to avoid.)

4. IANA Considerations

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

Number	Name	Reference
17	Block2	[RFCXXXX]
19	Block1	[RFCXXXX]

Table 2: CoAP Option Numbers

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

Code	Description	Reference
136	4.08 Request Entity Incomplete	[RFCXXXX]

Table 3: CoAP Response Codes

5. Security Considerations

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

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

5.1. Mitigating Resource Exhaustion Attacks

Certain blockwise requests may induce the server to create state, e.g. to create a snapshot for the blockwise GET of a fast-changing resource to enable consistent access to the same version of a resource for all blocks, or to create temporary resource representations that are collected until pressed into service by a final PUT or POST with the more bit unset. All mechanisms that induce a server to create state that cannot simply be cleaned up create opportunities for denial-of-service attacks. Servers SHOULD avoid being subject to resource exhaustion based on state created by untrusted sources. But even if this is done, the mitigation may cause a denial-of-service to a legitimate request when it is drowned out by other state-creating requests. Wherever possible, servers should therefore minimize the opportunities to create state for untrusted sources, e.g. by using stateless approaches.

Performing segmentation at the application layer is almost always better in this respect than at the transport layer or lower (IP fragmentation, adaptation layer fragmentation), e.g. because there is

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

5.2. Mitigating Amplification Attacks

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

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

6. Acknowledgements

Much of the content of this draft is the result of discussions with the [I-D.ietf-core-coap] authors, and via many CoRE WG discussions. Tokens were suggested by Gilman Tolle and refined by Klaus Hartke.

Charles Palmer provided extensive editorial comments to a previous version of this draft, some of which the authors hope to have covered in this version.

7. References

7.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-07 (work in progress), July 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

7.2. Informative References

- [REST] Fielding, R., "Architectural Styles and the Design of
Network-based Software Architectures", 2000.

Appendix A. Historical Note

An earlier version of this draft used a single option:

Type	C/E	Name	Format	Length	Default
13	Critical	Block	uint	1-3 B	0 (see below)

Note that this option number has since been reallocated in [I-D.ietf-core-coap]; no backwards compatibility is provided after July 1st, 2011.

Authors' Addresses

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

Phone: +49-421-218-63921
Fax: +49-421-218-7000
Email: cabo@tzi.org

Zach Shelby (editor)
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2012

Z. Shelby
Sensinode
K. Hartke
C. Bormann
Universitaet Bremen TZI
B. Frank
SkyFoundry
November 1, 2011

Constrained Application Protocol (CoAP)
draft-ietf-core-coap-08

Abstract

This document specifies the Constrained Application Protocol (CoAP), a specialized web transfer protocol for use with constrained networks and nodes for machine-to-machine applications such as smart energy and building automation. These constrained nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while networks such as 6LoWPAN often have high packet error rates and a typical throughput of 10s of kbit/s. CoAP provides a method/response interaction model between application end-points, supports built-in resource discovery, and includes key web concepts such as URIs and content-types. CoAP easily translates to HTTP for integration with the web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 4, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	5
1.1. Features	5
1.2. Terminology	6
2. Constrained Application Protocol	8
2.1. Messaging Model	8
2.2. Request/Response Model	10
2.3. Intermediaries and Caching	12
2.4. Resource Discovery	12
3. Message Syntax	12
3.1. Message Format	13
3.1.1. Message Size Implementation Considerations	14
3.2. Option Format	15
4. Message Semantics	16
4.1. Reliable Messages	17
4.2. Unreliable Messages	18
4.3. Message Matching Rules	19
4.4. Message Types	19
4.4.1. Confirmable (CON)	19
4.4.2. Non-Confirmable (NON)	20
4.4.3. Acknowledgement (ACK)	20
4.4.4. Reset (RST)	20
4.5. Multicast	20
4.6. Congestion Control	20
5. Request/Response Semantics	21
5.1. Requests	21
5.2. Responses	22
5.2.1. Piggy-backed	23
5.2.2. Separate	23
5.2.3. Non-Confirmable	24
5.3. Request/Response Matching	24
5.4. Options	25
5.4.1. Critical/Elective	26

5.4.2.	Length	26
5.4.3.	Default Values	27
5.4.4.	Repeating Options	27
5.4.5.	Option Numbers	27
5.5.	Payload	27
5.6.	Caching	28
5.6.1.	Freshness Model	29
5.6.2.	Validation Model	29
5.7.	Proxying	29
5.8.	Method Definitions	31
5.8.1.	GET	31
5.8.2.	POST	31
5.8.3.	PUT	31
5.8.4.	DELETE	32
5.9.	Response Code Definitions	32
5.9.1.	Success 2.xx	32
5.9.2.	Client Error 4.xx	33
5.9.3.	Server Error 5.xx	35
5.10.	Option Definitions	36
5.10.1.	Token	36
5.10.2.	Uri-Host, Uri-Port, Uri-Path and Uri-Query	37
5.10.3.	Proxy-Uri	38
5.10.4.	Content-Type	38
5.10.5.	Accept	38
5.10.6.	Max-Age	39
5.10.7.	ETag	39
5.10.8.	Location-Path and Location-Query	40
5.10.9.	If-Match	40
5.10.10.	If-None-Match	41
6.	CoAP URIs	41
6.1.	coap URI Scheme	41
6.2.	coaps URI Scheme	42
6.3.	Normalization and Comparison Rules	42
6.4.	Decomposing URIs into Options	43
6.5.	Composing URIs from Options	44
7.	Finding and Addressing CoAP End-Points	45
7.1.	Resource Discovery	45
7.1.1.	Content-type code 'ct' attribute	46
7.2.	Default Ports	46
8.	HTTP Mapping	46
8.1.	CoAP-HTTP Mapping	47
8.1.1.	GET	48
8.1.2.	PUT	48
8.1.3.	DELETE	48
8.1.4.	POST	49
8.2.	HTTP-CoAP Mapping	49
8.2.1.	OPTIONS and TRACE	49
8.2.2.	GET	49

8.2.3.	HEAD	50
8.2.4.	POST	50
8.2.5.	PUT	51
8.2.6.	DELETE	51
8.2.7.	CONNECT	51
9.	Protocol Constants	51
10.	Security Considerations	52
10.1.	Securing CoAP with DTLS	53
10.1.1.	PreSharedKey Mode	54
10.1.2.	RawPublicKey Mode	54
10.1.3.	Certificate Mode	54
10.2.	Using CoAP with IPsec	55
10.3.	Threat analysis and protocol limitations	56
10.3.1.	Protocol Parsing, Processing URIs	56
10.3.2.	Proxying and Caching	57
10.3.3.	Risk of amplification	57
10.3.4.	IP Address Spoofing Attacks	58
10.3.5.	Cross-Protocol Attacks	59
11.	IANA Considerations	60
11.1.	CoAP Code Registry	61
11.1.1.	Method Codes	61
11.1.2.	Response Codes	62
11.2.	Option Number Registry	63
11.3.	Media Type Registry	65
11.4.	URI Scheme Registration	66
11.5.	Secure URI Scheme Registration	67
11.6.	Service Name and Port Number Registration	68
11.7.	Secure Service Name and Port Number Registration	68
12.	Acknowledgements	69
13.	References	69
13.1.	Normative References	69
13.2.	Informative References	72
Appendix A.	Integer Option Value Format	73
Appendix B.	Examples	73
Appendix C.	URI Examples	79
Appendix D.	Security Provisioning and Access Control	80
D.1.	RawPublicKey Identity	81
D.2.	Provisioning	81
D.3.	Access Control	81
D.3.1.	PreSharedKey Mode	81
D.3.2.	RawPublicKey Mode	81
D.3.3.	Certificate Mode	82
Appendix E.	Changelog	82
Authors' Addresses	88

1. Introduction

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

The Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). Constrained networks like 6LoWPAN support the expensive fragmentation of IPv6 packets into small link-layer frames. One design goal of CoAP has been to keep message overhead small, thus limiting the use of fragmentation.

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

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

1.1. Features

CoAP has the following main features:

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

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

1.2. Terminology

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

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

Piggy-backed Response

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

Separate Response

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

Critical Option

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

Elective Option

An option that is intended be ignored by an end-point that does not understand it, which nonetheless still can correctly process the message (Section 5.4.1).

Resource Discovery

The process where a CoAP client queries a server for its list of hosted resources (i.e., links, Section 7.1).

End-Point

An entity participating in the CoAP protocol. Colloquially, a synonym is "Node", although "Host" would be more consistent with Internet standards usage.

Sender

The originating end-point of a message.

Recipient

The destination end-point of a message.

Client

The originating end-point of a request; the destination end-point of a response.

Server

The destination end-point of a request; the originating end-point of a response.

Origin Server

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

Intermediary

A CoAP end-point that acts both as a server and as a client towards (possibly via further intermediaries) an origin server. There are two common forms of intermediary: proxy and reverse proxy. In some cases, a single end-point might act as an origin server, proxy, or reverse proxy, switching behavior based on the nature of each request.

Proxy

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

Reverse Proxy

A "reverse proxy" is an end-point that acts as a layer above some other server(s) and satisfies requests on behalf of these, doing any necessary translations. Unlike a proxy, a reverse proxy receives requests as if it was the origin server for the target resource; the requesting client will not be aware that it is communicating with a reverse proxy.

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

In this specification, the operator "^" stands for exponentiation.

2. Constrained Application Protocol

The interaction model of CoAP is similar to the client/server model of HTTP. However, machine-to-machine interactions typically result in a CoAP implementation acting in both client and server roles (called an end-point). A CoAP request is equivalent to that of HTTP, and is sent by a client to request an action (using a method code) on a resource (identified by a URI) on a server. The server then sends a response with a response code; this response may include a resource representation.

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

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

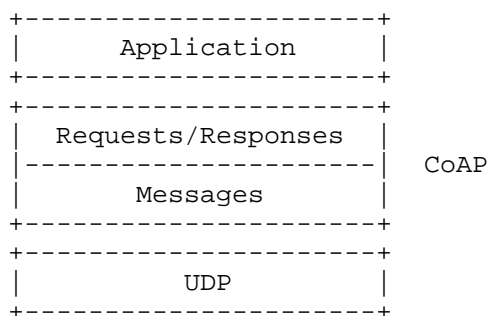


Figure 1: Abstract layering of CoAP

2.1. Messaging Model

The CoAP messaging model is based on the exchange of messages over UDP between end-points.

CoAP uses a short fixed-length binary header (4 bytes) that may be followed by compact binary options and a payload. This message

format is shared by requests and responses. The CoAP message format is specified in Section 3. Each message contains a Message ID used to detect duplicates and for optional reliability.

Reliability is provided by marking a message as Confirmable (CON). A Confirmable message is retransmitted using a default timeout and exponential back-off between retransmissions, until the recipient sends an Acknowledgement message (ACK) with the same Message ID (for example, 0x7d34) from the corresponding end-point; see Figure 2. When a recipient is not able to process a Confirmable message, it replies with a Reset message (RST) instead of an Acknowledgement (ACK).

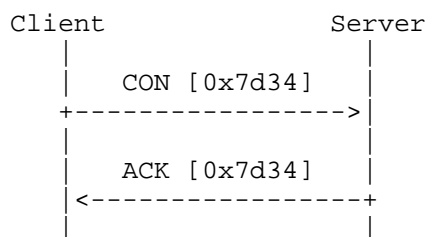


Figure 2: Reliable message delivery

A message that does not require reliable delivery, for example each single measurement out of a stream of sensor data, can be sent as a Non-confirmable message (NON). These are not acknowledged, but still have a Message ID for duplicate detection; see Figure 3.

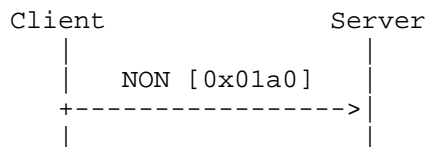


Figure 3: Unreliable message delivery

See Section 4 for details of CoAP messages.

As CoAP is based on UDP, it also supports the use of multicast IP destination addresses, enabling multicast CoAP requests. Section 4.5 discusses the proper use of CoAP messages with multicast addresses and precautions for avoiding response congestion.

Several security modes are defined for CoAP in Section 10 ranging from no security to certificate-based security. The use of IPsec along with a binding to DTLS are specified for securing the protocol.

2.2. Request/Response Model

CoAP request and response semantics are carried in CoAP messages, which include either a method code or response code, respectively. Optional (or default) request and response information, such as the URI and payload content-type are carried as CoAP options. A Token Option is used to match responses to requests independently from the underlying messages (Section 5.3).

A request is carried in a Confirmable (CON) or Non-confirmable (NON) message, and if immediately available, the response to a request carried in a Confirmable message is carried in the resulting Acknowledgement (ACK) message. This is called a piggy-backed response, detailed in Section 5.2.1. Two examples for a basic GET request with piggy-backed response are shown in Figure 4.

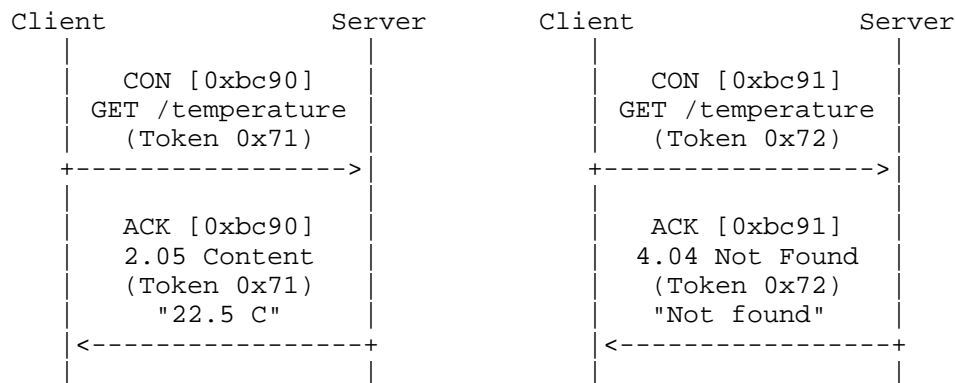


Figure 4: Two GET requests with piggy-backed responses, one successful, one not found

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

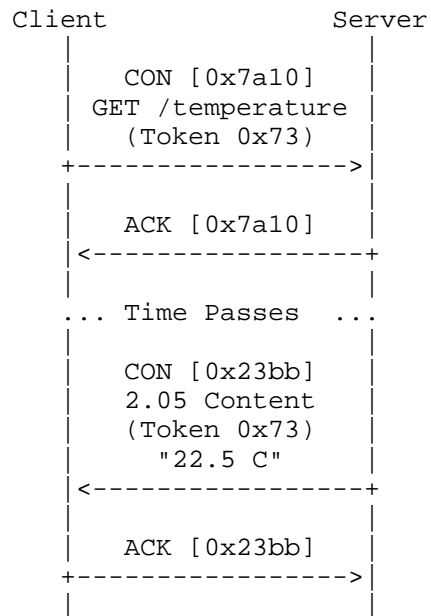


Figure 5: A GET request with a separate response

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

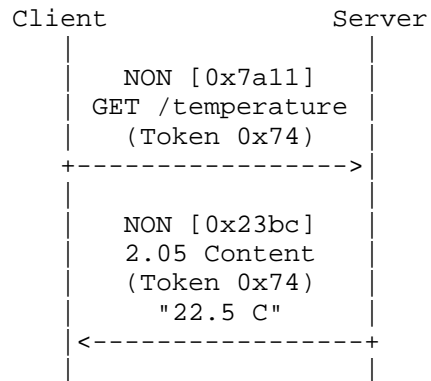


Figure 6: A NON request and response

CoAP makes use of GET, PUT, POST and DELETE methods in a similar manner to HTTP, with the semantics specified in Section 5.8. (Note that the detailed semantics of CoAP methods are "almost, but not

entirely unlike" those of HTTP methods: Intuition taken from HTTP experience generally does apply well, but there are enough differences that make it worthwhile to actually read the present specification.)

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

2.3. Intermediaries and Caching

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

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

As CoAP was designed according to the REST architecture and thus exhibits functionality similar to that of the HTTP protocol, it is quite straightforward to map between HTTP-CoAP or CoAP-HTTP. Such a mapping may be used to realize an HTTP REST interface using CoAP, or for converting between HTTP and CoAP. This conversion can be carried out by a proxy, which converts the method or response code, content-type and options to the corresponding HTTP feature. Section 8 provides more detail about HTTP mapping.

2.4. Resource Discovery

Resource discovery is important for machine-to-machine interactions, and is supported using the CoRE Link Format [I-D.ietf-core-link-format] as discussed in Section 7.1.

3. Message Syntax

CoAP is based on the exchange of short messages which, by default, are transported over UDP (i.e. each CoAP message occupies the data section of one UDP datagram). CoAP may be used with Datagram

Transport Layer Security (DTLS) (see Section 10.1). It could also be used over other transports such as TCP or SCTP, the specification of which is out of this document's scope.

3.1. Message Format

CoAP messages are encoded in a simple binary format. A message consists of a fixed-sized CoAP Header followed by options in Type-Length-Value (TLV) format and a payload. The number of options is determined by the header. The payload is made up of the bytes after the options, if any; its length is calculated from the datagram length.

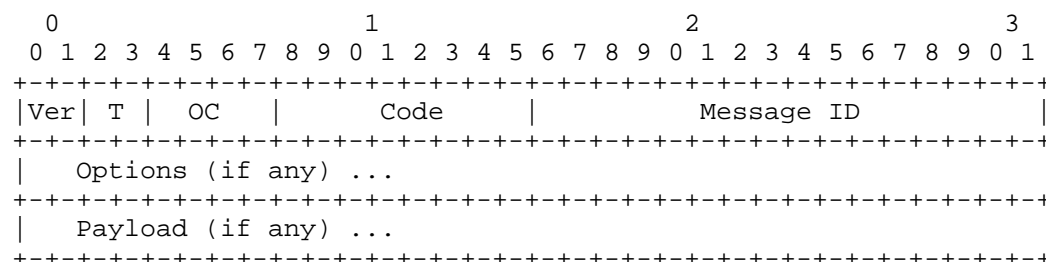


Figure 7: Message Format

The fields in the header are defined as follows:

Version (Ver): 2-bit unsigned integer. Indicates the CoAP version number. Implementations of this specification **MUST** set this field to 1. Other values are reserved for future versions.

Type (T): 2-bit unsigned integer. Indicates if this message is of type Confirmable (0), Non-Confirmable (1), Acknowledgement (2) or Reset (3). See Section 4 for the semantics of these message types.

Option Count (OC): 4-bit unsigned integer. Indicates the number of options after the header. If set to 0, there are no options and the payload (if any) immediately follows the header. The format of options is defined below.

Code: 8-bit unsigned integer. Indicates if the message carries a request (1-31) or a response (64-191), or is empty (0). (All other code values are reserved.) In case of a request, the Code field indicates the Request Method; in case of a response a Response Code. Possible values are maintained in the CoAP Code Registry (Section 11.1). See Section 5 for the semantics of requests and responses.

Message ID: 16-bit unsigned integer. Used for the detection of message duplication, and to match messages of type Acknowledgement/Reset and messages of type Confirmable. See Section 4 for Message ID generation rules and how messages are matched.

While specific link layers make it beneficial to keep CoAP messages small enough to fit into their link layer packets (see Section 1), this is a matter of implementation quality. The CoAP specification itself provides only an upper bound to the message size. Messages larger than an IP fragment result in undesired packet fragmentation. A CoAP message, appropriately encapsulated, SHOULD fit within a single IP packet (i.e., avoid IP fragmentation) and MUST fit within a single IP datagram. If the Path MTU is not known for a destination, an IP MTU of 1280 bytes SHOULD be assumed; if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size.

3.1.1. Message Size Implementation Considerations

Note that CoAP's choice of message size parameters works well with IPv6 and with most of today's IPv4 paths. (However, with IPv4, it is harder to absolutely ensure that there is no IP fragmentation. If IPv4 support on unusual networks is a consideration, implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes; worse, the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload. Implementations extremely focused on this problem set might also set the IPv4 DF bit and perform some form of path MTU discovery; this should generally be unnecessary in most realistic use cases for CoAP, however.) A more important kind of fragmentation in many constrained networks is that on the adaptation layer (e.g., 6LoWPAN L2 packets are limited to 127 bytes including various overheads); this may motivate implementations to be frugal in their packet sizes and to move to block-wise transfers [I-D.ietf-core-block] when approaching three-digit message sizes.

Note that message sizes are also of considerable importance to implementations on constrained nodes. Many implementations will need to allocate a buffer for incoming messages. If an implementation is too constrained to allow for allocating the above-mentioned upper bound, it could apply the following implementation strategy: Implementations receiving a datagram into a buffer that is too small are usually able to determine if the trailing portion of a datagram was discarded and to retrieve the initial portion. So, if not all of the payload, at least the CoAP header and options are likely to fit within the buffer. A server can thus fully interpret a request and

return a 4.13 (Request Entity Too Large) response code if the payload was truncated. A client sending an idempotent request and receiving a response larger than would fit in the buffer can repeat the request with a suitable value for the Block Option [I-D.ietf-core-block].

3.2. Option Format

Options MUST appear in order of their Option Number (see Section 5.4.5). A delta encoding is used between options, with the Option Number for each Option calculated as the sum of its Option Delta field and the Option Number of the preceding Option in the message, if any, or zero otherwise. Multiple options with the same Option Number can be included by using an Option Delta of zero. Following the Option Delta, each option has a Length field which specifies the length of the Option Value, in bytes. The Length field can be extended by one byte for options with values longer than 14 bytes. The Option Value immediately follows the Length field.

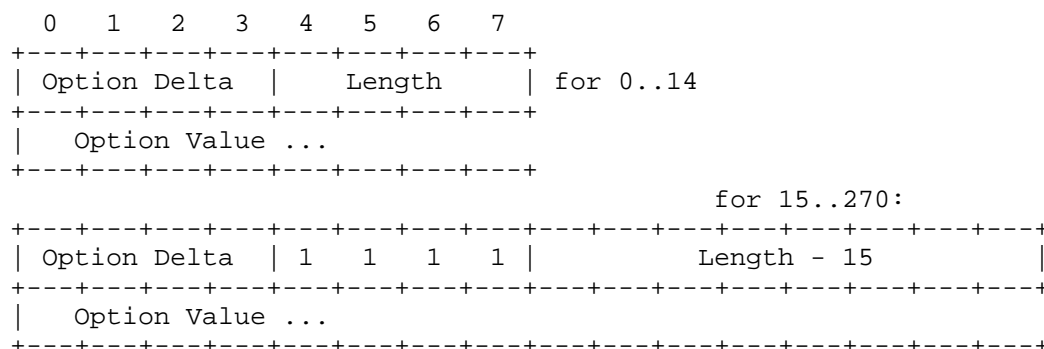


Figure 8: Option Format

The fields in an option are defined as follows:

Option Delta: 4-bit unsigned integer. Indicates the difference between the Option Number of this option and the previous option (or zero for the first option). In other words, the Option Number is calculated by simply summing the Option Delta fields of this and previous options before it. The Option Numbers 14, 28, 42, ... are reserved for no-op options when they are sent with an empty value (they are ignored) and can be used as "fenceposts" if deltas larger than 15 would otherwise be required.

Length: Indicates the length of the Option Value, in bytes. Normally Length is a 4-bit unsigned integer allowing value lengths of 0-14 bytes. When the Length field is set to 15, another byte is added as an 8-bit unsigned integer whose value is added to the

15, allowing option value lengths of 15-270 bytes.

The length and format of the Option Value depends on the respective option, which MAY define variable length values. Options defined in this document make use of the following formats for option values:

uint: A non-negative integer which is represented in network byte order using a variable number of bytes (see Appendix A).

string: A Unicode string which is encoded using UTF-8 [RFC3629] in Net-Unicode form [RFC5198]. Note that here and in all other places where UTF-8 encoding is used in the CoAP protocol, the intention is that the encoded strings can be directly used and compared as opaque byte strings by CoAP protocol implementations. There is no expectation and no need to perform normalization within a CoAP implementation unless Unicode strings that are not known to be normalized are imported from sources outside the CoAP protocol. Note also that ASCII strings (that do not make use of special control characters) are always valid UTF-8 Net-Unicode strings.

opaque: An opaque sequence of bytes.

Option Numbers are maintained in the CoAP Option Number Registry (Section 11.2). See Section 5.10 for the semantics of the options defined in this document.

4. Message Semantics

CoAP messages are exchanged asynchronously between CoAP end-points. They are used to transport CoAP requests and responses, the semantics of which are defined in Section 5.

As CoAP is bound to non-reliable transports such as UDP, CoAP messages may arrive out of order, appear duplicated, or go missing without notice. For this reason, CoAP implements a lightweight reliability mechanism, without trying to re-create the full feature set of a transport like TCP. It has the following features:

- o Simple stop-and-wait retransmission reliability with exponential back-off for "confirmable" messages.
- o Duplicate detection for both "confirmable" and "non-confirmable" messages.
- o Multicast support.

4.1. Reliable Messages

The reliable transmission of a message is initiated by marking the message as "confirmable" in the CoAP header. A recipient **MUST** acknowledge such a message with an acknowledgement message (or, if it lacks context to process the message properly, **MUST** reject it with a reset message). The sender retransmits the confirmable message at exponentially increasing intervals, until it receives an acknowledgement (or reset message), or runs out of attempts.

Retransmission is controlled by two things that a CoAP end-point **MUST** keep track of for each confirmable message it sends while waiting for an acknowledgement (or reset): a timeout and a retransmission counter. For a new confirmable message, the initial timeout is set to a random number between `RESPONSE_TIMEOUT` and $(\text{RESPONSE_TIMEOUT} * \text{RESPONSE_RANDOM_FACTOR})$, and the retransmission counter is set to 0. When the timeout is triggered and the retransmission counter is less than `MAX_RETRANSMIT`, the message is retransmitted, the retransmission counter is incremented, and the timeout is doubled. If the retransmission counter reaches `MAX_RETRANSMIT` on a timeout, or if the end-point receives a reset message, then the attempt to transmit the message is canceled and the application process informed of failure. On the other hand, if the end-point receives an acknowledgement message in time, transmission is considered successful.

An acknowledgement or reset message is related to a confirmable message by means of a Message ID along with additional address information of the corresponding end-point as described in Section 4.3. The Message ID is a 16-bit unsigned integer that is generated by the sender of a confirmable message and included in the CoAP header. The Message ID **MUST** be echoed in the acknowledgement or reset message by the recipient.

Several implementation strategies can be employed for generating Message IDs. In the simplest case a CoAP end-point generates Message IDs by keeping a single Message ID variable, which is changed each time a new confirmable message is sent regardless of the destination address or port. End-points dealing with large numbers of transactions could keep multiple Message ID variables, for example per prefix or destination address. The initial variable value **SHOULD** be randomized. The same Message ID **MUST NOT** be re-used (per Message ID variable) within the potential retransmission window, calculated as $\text{RESPONSE_TIMEOUT} * \text{RESPONSE_RANDOM_FACTOR} * (2 ^ \text{MAX_RETRANSMIT} - 1)$ plus the expected maximum round trip time.

A recipient **MUST** be prepared to receive the same confirmable message (as indicated by the Message ID and additional address information of the corresponding end-point as described in Section 4.3) multiple

times, for example, when its acknowledgement went missing or didn't reach the original sender before the first timeout. The recipient SHOULD acknowledge each duplicate copy of a confirmable message using the same acknowledgement or reset message, but SHOULD process any request or response in the message only once. This rule MAY be relaxed in case the confirmable message transports a request that is idempotent (see Section 5.1). Examples for relaxed message deduplication:

- o A server MAY relax the requirement to answer all retransmissions of an idempotent request with the same response (Section 4.1), so that it does not have to maintain state for Message IDs. For example, an implementation might want to process duplicate transmissions of a GET, PUT or DELETE request as separate requests if the effort incurred by duplicate processing is less expensive than keeping track of previous responses would be.
- o (As an implementation consideration, a constrained server MAY even want to relax this requirement for certain non-idempotent requests if the application semantics make this trade-off favorable. For example, if the result of a POST request is just the creation of some short-lived state at the server, it may be less expensive to incur this effort multiple times for a request than keeping track of whether a previous transmission of the same request already was processed.)

Implementation notes: Note that a CoAP end-point that sent a confirmable message MAY give up in attempting to obtain an ACK even before the MAX_RETRANSMIT counter value is reached: E.g., the application has canceled the request as it no longer needs a response, or there is some other indication that the CON message did arrive. In particular, a CoAP request message may have elicited a separate response, in which case it is clear to the requester that only the ACK was lost and a retransmission of the request would serve no purpose. However, a responder MUST NOT in turn rely on this cross-layer behavior from a requester, i.e. it SHOULD retain the state to create the ACK for the request, if needed, even if a confirmable response was already acknowledged by the requester.

4.2. Unreliable Messages

As a more lightweight alternative, a message can be transmitted less reliably by marking the message as "non-confirmable". A non-confirmable message MUST NOT be acknowledged by the recipient. If a recipient lacks context to process the message properly, it MAY reject the message with a reset message or otherwise MUST silently ignore it.

There is no way to detect if a non-confirmable message was received or not at the CoAP-level. A sender MAY choose to transmit a non-confirmable message multiple times which, for this purpose, specifies a Message ID as well. The same rules for generating the Message ID apply.

A recipient MUST be prepared to receive the same non-confirmable message (as indicated by the Message ID and source address information) multiple times. As a general rule that may be relaxed based on the specific semantics of a message, the recipient SHOULD silently ignore any duplicated non-confirmable message, and SHOULD process any request or response in the message only once.

4.3. Message Matching Rules

The exact rules for matching an ACK or RST to a CON message or a RST to a NON message are as follows. The Message ID of the response MUST match that of the original message. For unicast messages, the source of the response MUST match the destination of the original message. How this is determined depends on the security mode used (see Section 10): With NoSec, the IP address and port number of the message destination and response source must match. With other security modes, in addition to the IP address and UDP port matching, the request and response MUST have the same security context.

4.4. Message Types

The different types of messages are summarized below. The type of a message is specified by the T field of the CoAP header.

Separate from the message type, a message may carry a request, a response, or be empty. This is signaled by the Code field in the CoAP header and is relevant to the request/response model. Possible values for the Code field are maintained by the CoAP Code Registry (Section 11.1).

An empty message has the Code field set to 0. The OC field SHOULD be set to 0 and no bytes SHOULD be present after the Message ID field. The OC field and any bytes trailing the header MUST be ignored by any recipient.

4.4.1. Confirmable (CON)

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

A confirmable message always carries either a request or response and MUST NOT be empty.

4.4.2. Non-Confirmable (NON)

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

A non-confirmable message always carries either a request or response, as well, and MUST NOT be empty.

4.4.3. Acknowledgement (ACK)

An Acknowledgement message acknowledges that a specific confirmable message (identified by its Message ID) arrived. It does not indicate success or failure of any encapsulated request.

The acknowledgement message MUST echo the Message ID of the confirmable message, and MUST carry a response or be empty (see Section 5.2.1 and Section 5.2.2).

4.4.4. Reset (RST)

A Reset message indicates that a specific confirmable message was received, but some context is missing to properly process it. This condition is usually caused when the receiving node has rebooted and has forgotten some state that would be required to interpret the message.

A reset message MUST echo the Message ID of the confirmable message, and MUST be empty.

4.5. Multicast

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

4.6. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.1.

In order not to cause congestion, Clients (including proxies) SHOULD strictly limit the number of simultaneous outstanding interactions

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

Further congestion control optimizations and considerations are expected in the future, which may for example provide automatic initialization of the CoAP constants defined in Section 9.

5. Request/Response Semantics

CoAP operates under a similar request/response model as HTTP: a CoAP end-point in the role of a "client" sends one or more CoAP requests to a "server", which services the requests by sending CoAP responses. Unlike HTTP, requests and responses are not sent over a previously established connection, but exchanged asynchronously over CoAP messages.

5.1. Requests

A CoAP request consists of the method to be applied to the resource, the identifier of the resource, a payload and Internet media type (if any), and optional meta-data about the request.

CoAP supports the basic methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP. They have the same properties of safe (only retrieval) and idempotent (you can invoke it multiple times with the same effects) as HTTP (see Section 9.1 of [RFC2616]). The GET method is safe, therefore it MUST NOT take any other action on a resource other than retrieval. The GET, PUT and DELETE methods MUST be performed in such a way that they are idempotent. POST is not idempotent, because its effect is determined by the origin server and dependent on the target resource; it usually results in a new resource being created or the target resource being updated.

A request is initiated by setting the Code field in the CoAP header of a confirmable or a non-confirmable message to a Method Code and including request information.

The methods used in requests are described in detail in Section 5.8.

5.2. Responses

After receiving and interpreting a request, a server responds with a CoAP response, which is matched to the request by means of a client-generated token.

A response is identified by the Code field in the CoAP header being set to a Response Code. Similar to the HTTP Status Code, the CoAP Response Code indicates the result of the attempt to understand and satisfy the request. These codes are fully defined in Section 5.9. The Response Code numbers to be set in the Code field of the CoAP header are maintained in the CoAP Response Code Registry (Section 11.1.2).

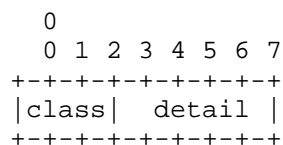


Figure 9: Structure of a Response Code

The upper three bits of the 8-bit Response Code number define the class of response. The lower five bits do not have any categorization role; they give additional detail to the overall class (Figure 9). There are 3 classes:

- 2 - Success: The request was successfully received, understood, and accepted.
- 4 - Client Error: The request contains bad syntax or cannot be fulfilled.
- 5 - Server Error: The server failed to fulfill an apparently valid request.

The response codes are designed to be extensible: Response Codes in the Client Error and Server Error class that are unrecognized by an end-point MUST be treated as being equivalent to the generic Response Code of that class. However, there is no generic Response Code indicating success, so a Response Code in the Success class that is unrecognized by an end-point can only be used to determine that the request was successful without any further details.

As a human readable notation for specifications and protocol diagnostics, the numeric value of a response code is indicated by

giving the upper three bits in decimal, followed by a dot and then the lower five bits in a two-digit decimal. E.g., "Not Found" is written as 4.04 -- indicating a value of hexadecimal 0x84 or decimal 132. In other words, the dot "." functions as a short-cut for "*32+".

The possible response codes are described in detail in Section 5.9.

Responses can be sent in multiple ways, which are defined below.

5.2.1. Piggy-backed

In the most basic case, the response is carried directly in the acknowledgement message that acknowledges the request (which requires that the request was carried in a confirmable message). This is called a "Piggy-backed" Response.

The response is returned in the acknowledgement message independent of whether the response indicates success or failure. In effect, the response is piggy-backed on the acknowledgement message, so no separate message is required to both acknowledge that the request was received and return the response.

5.2.2. Separate

It may not be possible to return a piggy-backed response in all cases. For example, a server might need longer to obtain the representation of the resource requested than it can wait sending back the acknowledgement message, without risking the client to repeatedly retransmit the request message. Responses to requests carried in a Non-Confirmable message are always sent separately (as there is no acknowledgement message).

The server maybe initiates the attempt to obtain the resource representation and times out an acknowledgement timer, or it immediately sends an acknowledgement knowing in advance that there will be no piggy-backed response. The acknowledgement effectively is a promise that the request will be acted upon.

When the server finally has obtained the resource representation, it sends the response. To ensure that this message is not lost, it is again sent as a confirmable message and answered by the client with an acknowledgement, echoing the new Message ID chosen by the server.

(Implementation notes: Note that, as the underlying datagram transport may not be sequence-preserving, the confirmable message carrying the response may actually arrive before or after the acknowledgement message for the request. Note also that, while the

CoAP protocol itself does not make any specific demands here, there is an expectation that the response will come within a time frame that is reasonable from an application point of view; as there is no underlying transport protocol that could be instructed to run a keep-alive mechanism, the requester MAY want to set up a timeout that is unrelated to CoAP's retransmission timers in case the server is destroyed or otherwise unable to send the response.)

For a separate exchange, both the acknowledgement to the confirmable request and the acknowledgement to the confirmable response MUST be an empty message, i.e. one that carries neither a request nor a response.

5.2.3. Non-Confirmable

If the request message is non-confirmable, then the response SHOULD be returned in a non-confirmable message as well. However, an end-point MUST be prepared to receive a non-confirmable response (preceded or followed an empty acknowledgement message) in reply to a confirmable request, or a confirmable response in reply to a non-confirmable request.

5.3. Request/Response Matching

Regardless of how a response is sent, it is matched to the request by means of a token that is included by the client in the request as one of the options along with additional address information of the corresponding end-point. The token MUST be echoed by the server in any resulting response without modification.

The exact rules for matching a response to a request are as follows:

1. For requests sent in a unicast message, the source of the response MUST match the destination of the original request. How this is determined depends on the security mode used (see Section 10): With NoSec, the IP address and port number of the request destination and response source must match. With other security modes, in addition to the IP address and UDP port matching, the request and response MUST have the same security context.
2. In a piggy-backed response, both the Message ID of the confirmable request and the acknowledgement, and the token of the response and original request MUST match. In a separate response, just the token of the response and original request MUST match.

The client SHOULD generate tokens in a way that tokens currently in

use for a given source/destination pair are unique. (Note that a client can use the same token for any request if it uses a different source port number each time.)

An end-point receiving a token MUST treat it as opaque and make no assumptions about its format. (Note that there is a default value for the Token Option, so every message carries a token, even if it is not explicitly expressed in a CoAP option.)

In case a confirmable message carrying a response is unexpected (i.e. the client is not waiting for a response with the specified address and/or token), the confirmable response SHOULD be rejected with a reset message and MUST NOT be acknowledged.

5.4. Options

Both requests and responses may include a list of one or more options. For example, the URI in a request is transported in several options, and meta-data that would be carried in an HTTP header in HTTP is supplied as options as well.

CoAP defines a single set of options that are used in both requests and responses:

- o Content-Type
- o ETag
- o Location-Path
- o Location-Query
- o Max-Age
- o Proxy-Uri
- o Token
- o Uri-Host
- o Uri-Path
- o Uri-Port
- o Uri-Query
- o Accept

- o If-Match
- o If-None-Match

The semantics of these options along with their properties are defined in detail in Section 5.10.

Not all options have meaning with all methods and response codes. The possible options for methods and response codes are defined in Section 5.8 and Section 5.9 respectively. In case an option has no meaning, it SHOULD NOT be included by the sender and MUST be ignored by the recipient.

5.4.1. Critical/Elective

Options fall into one of two classes: "critical" or "elective". The difference between these is how an option unrecognized by an endpoint is handled:

- o Upon reception, unrecognized options of class "elective" MUST be silently ignored.
- o Unrecognized options of class "critical" that occur in a confirmable request MUST cause the return of a 4.02 (Bad Option) response. This response SHOULD include a human-readable error message describing the unrecognized option(s) (see Section 5.5).
- o Unrecognized options of class "critical" that occur in a confirmable response SHOULD cause the response to be rejected with a reset message.
- o Unrecognized options of class "critical" that occur in a non-confirmable message MUST cause the message to be silently ignored.

Note that, whether critical or elective, an option is never "mandatory" (it is always optional): These rules are defined in order to enable implementations to reject options they do not understand or implement.

5.4.2. Length

Option values are defined to have a specific length, often in the form of an upper and lower bound. If the length of an option value in a request is outside the defined range, that option MUST be treated like an unrecognized option (see Section 5.4.1).

5.4.3. Default Values

Options may be defined to have a default value. If the value of option is intended to be this default value, the option SHOULD NOT be included in the message. If the option is not present, the default value MUST be assumed.

5.4.4. Repeating Options

Each definition of an option specifies whether it is defined to occur only at most once or whether it can occur multiple times. If a message includes an option with more instances than the option is defined for, the additional option instances MUST be treated like an unrecognized option (see Section 5.4.1).

5.4.5. Option Numbers

Options are identified by an option number. Odd numbers indicate a critical option, while even numbers indicate an elective option. (Note that this is not just a convention, it is a feature of the protocol: Whether an option is elective or critical is entirely determined by whether its option number is even or odd.)

The numbers 14, 28, 42, ... are reserved for "fenceposting", as described in Section 3.2. As these option numbers are even, they stand for elective options, and unless assigned a meaning, these MUST be silently ignored.

The option numbers for the options defined in this document are listed in the CoAP Option Number Registry (Section 11.2).

5.5. Payload

Both requests and responses may include payload, depending on the method or response code respectively. Methods with payload are PUT and POST, and the response codes with payload are 2.05 (Content) and the error codes.

The payload of PUT, POST and 2.05 (Content) is typically a resource representation. Its format is specified by the Internet media type given by the Content-Type Option. No default value is assumed in the absence of this option.

2.01 (Created), 2.02 (Deleted), 2.04 (Changed) MAY include payload that is describing the result of the action. Again, the format of this payload is specified by the Internet media type given by the Content-Type Option; no default value is assumed in the absence of this option.

A response with a code indicating a Client or Server Error SHOULD include a brief human-readable diagnostic message as payload, explaining the error situation. This diagnostic message MUST be encoded using UTF-8 [RFC3629], more specifically using Net-Unicode form [RFC5198]. The Content-Type Option has no meaning and SHOULD NOT be included. (Similar to what one would find as a Reason-Phrase on an HTTP status line, the message is not intended for end-users but for software engineers that during debugging need to interpret it in the context of the present, English-language specification; therefore no language tagging is foreseen.)

If a method or response code is not defined to have a payload, then the sender SHOULD NOT include one, and the recipient MUST ignore it.

5.6. Caching

CoAP end-points MAY cache responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests.

The goal of caching in CoAP is to reuse a prior response message to satisfy a current request. In some cases, a stored response can be reused without the need for a network request, reducing latency and network round-trips; a "freshness" mechanism is used for this purpose (see Section 5.6.1). Even when a new request is required, it is often possible to reuse the payload of a prior response to satisfy the request, thereby reducing network bandwidth usage; a "validation" mechanism is used for this purpose (see Section 5.6.2).

Unlike HTTP, the cacheability of CoAP responses does not depend on the request method, but the Response Code. The cacheability of each Response Code is defined along the Response Code definitions in Section 5.9. Response Codes that indicate success and are unrecognized by an end-point MUST NOT be cached.

For a presented request, a CoAP end-point MUST NOT use a stored response, unless:

- o the presented request method and that used to obtain the stored response match,
- o all options match between those in the presented request and those of the request used to obtain the stored response (which includes the request URI), except that there is no need for a match of the Token, Max-Age, or ETag request option(s), and
- o the stored response is either fresh or successfully validated as defined below.

5.6.1. Freshness Model

When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.

The mechanism for determining freshness is for an origin server to provide an explicit expiration time in the future, using the Max-Age Option (see Section 5.10.6). The Max-Age Option indicates that the response is to be considered not fresh after its age is greater than the specified number of seconds.

As the Max-Age Option defaults to a value of 60, if it is not present in a cacheable response, then the response is considered not fresh after its age is greater than 60 seconds. If an origin server wishes to prevent caching, it **MUST** explicitly include a Max-Age Option with a value of zero seconds.

5.6.2. Validation Model

When an end-point has one or more stored responses for a GET request, but cannot use any of them (e.g., because they are not fresh), it can use the ETag Option in the GET request to give the origin server an opportunity to both select a stored response to be used, and to update its freshness. This process is known as "validating" or "revalidating" the stored response.

When sending such a request, the end-point **SHOULD** add an ETag Option specifying the entity-tag of each stored response that is applicable.

A 2.03 (Valid) response indicates the stored response identified by the entity-tag given in the response's ETag Option can be reused, after updating its freshness with the value of the Max-Age Option that is included with the response (see Section 5.9.1.3).

Any other response code indicates that none of the stored responses nominated in the request is suitable. Instead, the response **SHOULD** be used to satisfy the request and **MAY** replace the stored response.

5.7. Proxying

CoAP distinguishes between requests to an origin server and a request made through a proxy. A proxy is a CoAP end-point that can be tasked by CoAP clients to perform requests on their behalf. This may be useful, for example, when the request could otherwise not be made, or to service the response from a cache in order to reduce response time and network bandwidth or energy consumption.

CoAP requests to a proxy are made as normal confirmable or non-confirmable requests to the proxy end-point, but specify the request URI in a different way: The request URI in a proxy request is specified as a string in the Proxy-Uri Option (see Section 5.10.3), while the request URI in a request to an origin server is split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options (see Section 5.10.2).

When a proxy request is made to an end-point and the end-point is unwilling or unable to act as proxy for the request URI, it MUST return a 5.05 (Proxying Not Supported) response. If the authority (host and port) is recognized as identifying the proxy end-point, then the request MUST be treated as a local request.

Unless a proxy is configured to forward the proxy request to another proxy, it MUST translate the request as follows: The origin server's IP address and port are determined by the authority component of the request URI, and the request URI is decoded and split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options.

All options present in a proxy request MUST be processed at the proxy. Critical options in a request that are not recognized by the proxy MUST lead to a 4.02 (Bad Option) response being returned by the proxy. Elective options not recognized by the proxy MUST NOT be forwarded to the origin server. Similarly, critical options in a response that are not recognized by the proxy server MUST lead to a 5.02 (Bad Gateway) response. Again, elective options that are not recognized MUST NOT be forwarded.

If the proxy does not employ a cache, then it simply forwards the translated request to the determined destination. Otherwise, if it does employ a cache but does not have a stored response that matches the translated request and is considered fresh, then it needs to refresh its cache according to Section 5.6.

If the request to the destination times out, then a 5.04 (Gateway Timeout) response MUST be returned. If the request to the destination returns an response that cannot be processed by the proxy, then a 5.02 (Bad Gateway) response MUST be returned. Otherwise, the proxy returns the response to the client.

If a response is generated out of a cache, it MUST be generated with a Max-Age Option that does not extend the max-age originally set by the server, considering the time the resource representation spent in the cache. E.g., the Max-Age Option could be adjusted by the proxy for each response using the formula: $\text{proxy-max-age} = \text{original-max-age} - \text{cache-age}$. For example if a request is made to a proxied resource that was refreshed 20 seconds ago and had an original Max-Age of 60

seconds, then that resource's proxied max-age is now 40 seconds.

5.8. Method Definitions

In this section each method is defined along with its behavior. A request with an unrecognized or unsupported Method Code MUST generate a 4.05 (Method Not Allowed) response.

5.8.1. GET

The GET method retrieves a representation for the information that currently corresponds to the resource identified by the request URI. If the request includes one or more Accept Options, they indicate the preferred content-type of a response. If the request includes an ETag Option, the GET method requests that ETag be validated and that the representation be transferred only if validation failed. Upon success a 2.05 (Content) or 2.03 (Valid) response SHOULD be sent.

The GET method is safe and idempotent.

5.8.2. POST

The POST method requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and dependent on the target resource. It usually results in a new resource being created or the target resource being updated.

If a resource has been created on the server, a 2.01 (Created) response that includes the URI of the new resource in a sequence of one or more Location-Path Options and/or a Location-Query Option SHOULD be returned. If the POST succeeds but does not result in a new resource being created on the server, a 2.04 (Changed) response SHOULD be returned. If the POST succeeds and results in the target resource being deleted, a 2.02 (Deleted) response SHOULD be returned.

POST is neither safe nor idempotent.

5.8.3. PUT

The PUT method requests that the resource identified by the request URI be updated or created with the enclosed representation. The representation format is specified by the media type given in the Content-Type Option.

If a resource exists at the request URI the enclosed representation SHOULD be considered a modified version of that resource, and a 2.04 (Changed) response SHOULD be returned. If no resource exists then

the server MAY create a new resource with that URI, resulting in a 2.01 (Created) response. If the resource could not be created or modified, then an appropriate error response code SHOULD be sent.

Further restrictions to a PUT can be made by including the If-Match (see Section 5.10.9) or If-None-Match (see Section 5.10.10) options in the request.

PUT is not safe, but idempotent.

5.8.4. DELETE

The DELETE method requests that the resource identified by the request URI be deleted. A 2.02 (Deleted) response SHOULD be sent on success or in case the resource did not exist before the request.

DELETE is not safe, but idempotent.

5.9. Response Code Definitions

Each response code is described below, including any options required in the response. Where appropriate, some of the codes will be specified in regards to related response codes in HTTP [RFC2616]; this does not mean that any such relationship modifies the HTTP mapping specified in Section 8.

5.9.1. Success 2.xx

This class of status code indicates that the clients request was successfully received, understood, and accepted.

5.9.1.1. 2.01 Created

Like HTTP 201 "Created", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result. The representation format is specified by the media type given in the Content-Type Option.

If the response includes one or more Location-Path Options and/or a Location-Query Option, the values of these options specify the location at which the resource was created. Otherwise, the resource was created at the request URI. A cache SHOULD mark any stored response for the created resource as not fresh.

This response is not cacheable.

5.9.1.2. 2.02 Deleted

Like HTTP 204 "No Content", but only used in response to DELETE requests. The payload returned with the response, if any, is a representation of the action result. The representation format is specified by the media type given in the Content-Type Option.

This response is not cacheable. However, a cache SHOULD mark any stored response for the deleted resource as not fresh.

5.9.1.3. 2.03 Valid

Related to HTTP 304 "Not Modified", but only used to indicate that the response identified by the entity-tag identified by the included ETag Option is valid. Accordingly, the response MUST include an ETag Option.

When a cache receives a 2.03 (Valid) response, it needs to update the stored response with the value of the Max-Age Option included in the response (see Section 5.6.2).

5.9.1.4. 2.04 Changed

Like HTTP 204 "No Content", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result. The representation format is specified by the media type given in the Content-Type Option.

This response is not cacheable. However, a cache SHOULD mark any stored response for the changed resource as not fresh.

5.9.1.5. 2.05 Content

Like HTTP 200 "OK", but only used in response to GET requests.

The payload returned with the response is a representation of the target resource. The representation format is specified by the media type given in the Content-Type Option.

This response is cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1) and (if present) the ETag Option for validation (see Section 5.6.2).

5.9.2. Client Error 4.xx

This class of response code is intended for cases in which the client seems to have erred. These response codes are applicable to any request method.

The server SHOULD include a brief human-readable message as payload, as detailed in Section 5.5.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

5.9.2.1. 4.00 Bad Request

Like HTTP 400 "Bad Request".

5.9.2.2. 4.01 Unauthorized

The client is not authorized to perform the requested action. The client SHOULD NOT repeat the request without previously improving its authentication status to the server. Which specific mechanism can be used for this is outside this document's scope; see also Section 10.

5.9.2.3. 4.02 Bad Option

The request could not be understood by the server due to one or more unrecognized or malformed critical options. The client SHOULD NOT repeat the request without modification.

5.9.2.4. 4.03 Forbidden

Like HTTP 403 "Forbidden".

5.9.2.5. 4.04 Not Found

Like HTTP 404 "Not Found".

5.9.2.6. 4.05 Method Not Allowed

Like HTTP 405 "Method Not Allowed", but with no parallel to the "Allow" header field.

5.9.2.7. 4.06 Not Acceptable

Like HTTP 406 "Not Acceptable", but with no response entity.

5.9.2.8. 4.12 Precondition Failed

Like HTTP 412 "Precondition Failed".

5.9.2.9. 4.13 Request Entity Too Large

Like HTTP 413 "Request Entity Too Large".

5.9.2.10. 4.15 Unsupported Media Type

Like HTTP 415 "Unsupported Media Type".

5.9.3. Server Error 5.xx

This class of response code indicates cases in which the server is aware that it has erred or is incapable of performing the request. These response codes are applicable to any request method.

The server SHOULD include a human-readable message as payload, as detailed in Section 5.5.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

5.9.3.1. 5.00 Internal Server Error

Like HTTP 500 "Internal Server Error".

5.9.3.2. 5.01 Not Implemented

Like HTTP 501 "Not Implemented".

5.9.3.3. 5.02 Bad Gateway

Like HTTP 502 "Bad Gateway".

5.9.3.4. 5.03 Service Unavailable

Like HTTP 503 "Service Unavailable", but using the Max-Age Option in place of the "Retry-After" header field.

5.9.3.5. 5.04 Gateway Timeout

Like HTTP 504 "Gateway Timeout".

5.9.3.6. 5.05 Proxying Not Supported

The server is unable or unwilling to act as a proxy for the URI specified in the Proxy-Uri Option (see Section 5.10.3).

5.10. Option Definitions

The individual CoAP options are summarized in Table 1 and explained below.

No.	C/E	Name	Format	Length	Default
1	Critical	Content-Type	uint	0-2 B	(none)
2	Elective	Max-Age	uint	0-4 B	60
3	Critical	Proxy-Uri	string	1-270 B	(none)
4	Elective	ETag	opaque	1-8 B	(none)
5	Critical	Uri-Host	string	1-270 B	(see below)
6	Elective	Location-Path	string	1-270 B	(none)
7	Critical	Uri-Port	uint	0-2 B	(see below)
8	Elective	Location-Query	string	1-270 B	(none)
9	Critical	Uri-Path	string	1-270 B	(none)
11	Critical	Token	opaque	1-8 B	(empty)
12	Elective	Accept	uint	0-2 B	(none)
13	Critical	If-Match	opaque	0-8 B	(none)
15	Critical	Uri-Query	string	1-270 B	(none)
21	Critical	If-None-Match	(none)	0 B	(none)

Table 1: Options

5.10.1. Token

The Token Option is used to match a response with a request. Every request has a client-generated token which the server **MUST** echo in any response. A default value of a zero-length token is assumed in the absence of the option. Thus when the token value is empty, the Token Option **SHOULD** be elided for efficiency.

A token is intended for use as a client-local identifier for differentiating between concurrent requests (see Section 5.3). A client **SHOULD** generate tokens in a way that tokens currently in use for a given source/destination pair are unique. An empty token value is appropriate e.g. when no other tokens are in use to a destination, or when requests are made serially per destination. There are however multiple possible implementation strategies to fulfill this. An end-point receiving a token **MUST** treat it as opaque and make no assumptions about its format.

This option is "critical". It **MUST NOT** occur more than once.

5.10.2. Uri-Host, Uri-Port, Uri-Path and Uri-Query

The Uri-Host, Uri-Port, Uri-Path and Uri-Query Options are used to specify the target resource of a request to a CoAP origin server. The options encode the different components of the request URI in a way that no percent-encoding is visible in the option values and that the full URI can be reconstructed at any involved end-point. The syntax of CoAP URIs is defined in Section 6.

The steps for parsing URIs into options is defined in Section 6.4. These steps result in zero or more Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in a request, where each option holds the following values:

- o the Uri-Host Option specifies the Internet host of the resource being requested,
- o the Uri-Port Option specifies the port number of the resource,
- o each Uri-Path Option specifies one segment of the absolute path to the resource, and
- o each Uri-Query Option specifies one argument parameterizing the resource.

Note: Fragments ([RFC3986], Section 3.5) are not part of the request URI and thus will not be transmitted in a CoAP request.

The default value of the Uri-Host Option is the IP literal representing the destination IP address of the request message. Likewise, the default value of the Uri-Port Option is the destination UDP port. The default Uri-Host and Uri-Port options are sufficient for requests to most servers, and are typically used when an end-point hosts multiple virtual servers.

The Uri-Path and Uri-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Uri-Path Option MUST NOT be "." or ".." (as the request URI must be resolved before parsing it into options).

The steps for constructing the request URI from the options are defined in Section 6.5. Note that an implementation does not necessarily have to construct the URI; it can simply look up the target resource by looking at the individual options.

Examples can be found in Appendix C.

All of the options are "critical". Uri-Host and Uri-Port MUST NOT

occur more than once; Uri-Path and Uri-Query MAY occur one or more times.

5.10.3. Proxy-Uri

The Proxy-Uri Option is used to make a request to a proxy (see Section 5.7). The proxy is requested to forward the request or service it from a valid cache, and return the response.

The option value is an absolute-URI ([RFC3986], Section 4.3). In case the absolute-URI doesn't fit within a single option, the Proxy-Uri Option MAY be included multiple times in a request such that the concatenation of the values results in the single absolute-URI.

All but the last instance of the Proxy-Uri Option MUST have a value with a length of 270 bytes, and the last instance MUST NOT be empty.

Note that the proxy MAY forward the request on to another proxy or directly to the server specified by the absolute-URI. In order to avoid request loops, a proxy MUST be able to recognize all of its server names, including any aliases, local variations, and the numeric IP addresses.

An end-point receiving a request with a Proxy-Uri Option that is unable or unwilling to act as a proxy for the request MUST cause the return of a 5.05 (Proxying Not Supported) response.

This option is "critical". It MAY occur one or more times and MUST take precedence over any of the Uri-Host, Uri-Port, Uri-Path or Uri-Query options (which MUST NOT be included at the same time).

5.10.4. Content-Type

The Content-Type Option indicates the representation format of the message payload. The representation format is given as a numeric media type identifier that is defined in the CoAP Media Type registry (Section 11.3). No default value is assumed in the absence of the option.

This option is "critical". It MUST NOT occur more than once.

5.10.5. Accept

The CoAP Accept option indicates when included one or more times in a request, one or more media types, each of which is an acceptable media type for the client, in the order of preference. The representation format is given as a numeric media type identifier that is defined in the CoAP Media Type registry (Section 11.3). If

no Accept options are given, the client does not express a preference (thus no default value is assumed). The client prefers the representation returned by the server to be in one of the media types indicated. The server SHOULD return one of the preferred media types if available. If none of the preferred media types can be returned, then a 4.06 "Not Acceptable" SHOULD be sent as a response.

Note that as a server might not support the Accept option (and thus would ignore it as it is elective), the client needs to be prepared to receive a representation in a different media type. The client can simply discard a representation it can not make use of.

This option is "elective". It MAY occur more than once.

5.10.6. Max-Age

The Max-Age Option indicates the maximum time a response may be cached before it MUST be considered not fresh (see Section 5.6.1).

The option value is an integer number of seconds between 0 and $2^{32}-1$ inclusive (about 136.1 years). A default value of 60 seconds is assumed in the absence of the option in a response.

This option is "elective". It MUST NOT occur more than once.

5.10.7. ETag

The ETag Option in a response provides the current value of the entity-tag for the enclosed representation of the target resource.

An entity-tag is intended for use as a resource-local identifier for differentiating between representations of the same resource that vary over time. It may be generated in any number of ways including a version, checksum, hash or time. An end-point receiving an entity-tag MUST treat it as opaque and make no assumptions about its format. (End-points generating an entity-tag are encouraged to use the most compact representation possible, in particular in regards to clients and intermediaries that may want to store multiple ETag values.)

An end-point that has one or more representations previously obtained from the resource can specify the ETag Option in a request for each stored response to determine if any of those representations is current (see Section 5.6.2).

This option is "elective". It MUST NOT occur more than once in a response, and MAY occur one or more times in a request.

5.10.8. Location-Path and Location-Query

The Location-Path and Location-Query Options indicates the location of a resource as an absolute path URI. The Location-Path Option is similar to the Uri-Path Option, and the Location-Query Option similar to the Uri-Query Option.

The two options MAY be included in a response to indicate the location of a new resource created with POST.

If a response with a Location-Path and/or Location-Query Option passes through a cache and the implied URI identifies one or more currently stored responses, those entries SHOULD be marked as not fresh.

Both options are "elective" and MAY occur one or more times.

5.10.9. If-Match

The If-Match Option MAY be used to make a request conditional on the current existence or value of an ETag for one or more representations of the target resource. If-Match is generally useful for resource update requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource (i.e., the "lost update" problem).

The value of an If-Match option is either an ETag or the empty string. An empty string places the precondition on the existence of any current representation for the target resource.

The If-Match Option can occur multiple times. If any of the ETags given as an option value match the ETag of the selected representation for the target resource, or if an If-Match Option with an empty string as option value is given and any current representation exists for the target resource, then the server MAY perform the request method as if the If-Match Option was not present.

If none of the ETags match and, if an empty string is given, no current representation exists at all, the server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

If the request would, without the If-Match Options, result in anything other than a 2.xx or 4.12 response code, then any If-Match Options MUST be ignored.

This option is "critical". It MAY occur more than once.

5.10.10. If-None-Match

The If-None-Match Option MAY be used to make a request conditional on the non-existence of the target resource. If-None-Match is useful for resource creation requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource. The If-None-Match Option carries no value.

If the target resource does exist, then the server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

This option is "critical". It MAY NOT occur more than once.

6. CoAP URIs

CoAP uses the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. Resources are organized hierarchically and governed by a potential CoAP origin server listening for CoAP requests ("coap") or DTLS-secured CoAP requests ("coaps") on a given UDP port. The CoAP server is identified via the generic syntax's authority component, which includes a host identifier and optional UDP port number. The remainder of the URI is considered to be identifying a resource which can be operated on by the methods defined by the CoAP protocol. The "coap" and "coaps" URI schemes can thus be compared to the "http" and "https" URI schemes respectively.

The syntax of the "coap" and "coaps" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", "query", "segment", "IP-literal", "IPv4address" and "reg-name" are adopted from [RFC3986].

6.1. coap URI Scheme

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

If host is provided as an IP-literal or IPv4address, then the CoAP server is located at that IP address. If host is a registered name, then that name is considered an indirect identifier and the end-point might use a name resolution service, such as DNS, to find the address of that host. The host MUST NOT be empty. The port subcomponent indicates the UDP port at which the CoAP server is located. If it is empty or not given, then the default port 5683 is assumed.

The path identifies a resource within the scope of the host and port.

It consists of a sequence of path segments separated by a slash character (U+002F SOLIDUS "/").

The query serves to further parameterize the resource. It consists of a sequence of arguments separated by an ampersand character (U+0026 AMPERSAND "&"). An argument is often in the form of a "key=value" pair.

The "coap" URI scheme supports the path prefix "/.well-known/" defined by [RFC5785] for "well-known locations" in the name-space of a host. This enables discovery of policy or other information about a host ("site-wide metadata"), such as hosted resources (see Section 7.1).

Application designers are encouraged to make use of short, but descriptive URIs. As the environments that CoAP is used in are usually constrained for bandwidth and energy, the trade-off between these two qualities should lean towards the shortness, without ignoring descriptiveness.

6.2. coaps URI Scheme

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

All of the requirements listed above for the "coap" scheme are also requirements for the "coaps" scheme, except that a default UDP port of [IANA_TBD_PORT] is assumed if the port subcomponent is empty or not given, and the UDP datagrams MUST be secured for privacy through the use of DTLS as described in Section 10.1.

Unlike the "coap" scheme, responses to "coaps" identified requests are never "public" and thus MUST NOT be reused for shared caching. They can, however, be reused in a private cache if the message is cacheable by default in CoAP.

Resources made available via the "coaps" scheme have no shared identity with the "coap" scheme even if their resource identifiers indicate the same authority (the same host listening to the same UDP port). They are distinct name spaces and are considered to be distinct origin servers.

6.3. Normalization and Comparison Rules

Since the "coap" and "coaps" schemes conform to the URI generic syntax, such URIs are normalized and compared according to the algorithm defined in [RFC3986], Section 6, using the defaults described above for each scheme.

If the port is equal to the default port for a scheme, the normal form is to elide the port subcomponent. Likewise, an empty path component is equivalent to an absolute path of "/", so the normal form is to provide a path of "/" instead. The scheme and host are case-insensitive and normally provided in lowercase; IP-literals are in recommended form [RFC5952]; all other components are compared in a case-sensitive manner. Characters other than those in the "reserved" set are equivalent to their percent-encoded octets (see [RFC3986], Section 2.1): the normal form is to not encode them.

For example, the following three URIs are equivalent, and cause the same options and option values to appear in the CoAP messages:

```
coap://example.com:5683/~sensors/temp.xml
coap://EXAMPLE.com/%7Esensors/temp.xml
coap://EXAMPLE.com:/%7esensors/temp.xml
```

6.4. Decomposing URIs into Options

The steps to parse a request's options from a string `/url/` are as follows. These steps either result in zero or more of the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in the request, or they fail.

1. If the `/url/` string is not an absolute URI ([RFC3986]), then fail this algorithm.
2. Resolve the `/url/` string using the process of reference resolution defined by [RFC3986], with the URL character encoding set to UTF-8 [RFC3629].

NOTE: It doesn't matter what it is resolved relative to, since we already know it is an absolute URL at this point.

3. If `/url/` does not have a `<scheme>` component whose value, when converted to ASCII lowercase, is "coap" or "coaps", then fail this algorithm.
4. If `/url/` has a `<fragment>` component, then fail this algorithm.
5. If the `<host>` component of `/url/` does not represent the request's destination IP address as an IP-literal or IPv4address, include a Uri-Host Option and let that option's value be the value of the `<host>` component of `/url/`, converted to ASCII lowercase, and then converting all percent-encodings ("% followed by two hexadecimal digits) to the corresponding characters.

NOTE: In the usual case where the request's destination IP

address is derived from the host part, this ensures that Uri-Host Options are only used for host parts of the form reg-name.

6. If /url/ has a <port> component, then let /port/ be that component's value interpreted as a decimal integer; otherwise, let /port/ be the default port for the scheme.
7. If /port/ does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be /port/.
8. If the value of the <path> component of /url/ is empty or consists of a single slash character (U+002F SOLIDUS "/"), then move to the next step.

Otherwise, for each segment in the <path> component, include a Uri-Path Option and let that option's value be the segment (not including the delimiting slash characters) after converting all percent-encodings ("% followed by two hexadecimal digits) to the corresponding characters.

9. If /url/ has a <query> component, then, for each argument in the <query> component, include a Uri-Query Option and let that option's value be the argument (not including the question mark and the delimiting ampersand characters) after converting all percent-encodings to the corresponding characters.

Note that these rules completely resolve any percent-encoding.

6.5. Composing URIs from Options

The steps to construct a URI from a request's options are as follows. These steps either result in a URI, or they fail. In these steps, percent-encoding a character means replacing each of its (UTF-8 encoded) bytes by a "%" character followed by two hexadecimal digits representing the byte, where the digits A-F are in upper case (as defined in [RFC3986] Section 2.1; to reduce variability, the hexadecimal notation in CoAP URIs MUST use uppercase letters).

1. If the request is secured using DTLS, let /url/ be the string "coaps://". Otherwise, let /url/ be the string "coap://".
2. If the request includes a Uri-Host Option, let /host/ be that option's value, where any non-ASCII characters are replaced by their corresponding percent-encoding. If /host/ is not a valid reg-name or IP-literal or IPv4address, fail the algorithm. Otherwise, let /host/ be the IP-literal (making use of the conventions of [RFC5952]) or IPv4address representing the request's destination IP address.

3. Append /host/ to /url/.
4. If the request includes a Uri-Port Option, let /port/ be that option's value. Otherwise, let /port/ be the request's destination UDP port.
5. If /port/ is not the default port for the scheme, then append a single U+003A COLON character (:) followed by the decimal representation of /port/ to /url/.
6. Let /resource name/ be the empty string. For each Uri-Path Option in the request, append a single character U+002F SOLIDUS (/) followed by the option's value to /resource name/, after converting any character that is not either in the "unreserved" set, "sub-delims" set, a U+003A COLON (:) or U+0040 COMMERCIAL AT (@) character, to its percent-encoded form.
7. If /resource name/ is the empty string, set it to a single character U+002F SOLIDUS (/).
8. For each Uri-Query Option in the request, append a single character U+003F QUESTION MARK (?) (first option) or U+0026 AMPERSAND (&) (subsequent options) followed by the option's value to /resource name/, after converting any character that is not either in the "unreserved" set, "sub-delims" set (except U+0026 AMPERSAND (&)), a U+003A COLON (:), U+0040 COMMERCIAL AT (@), U+002F SOLIDUS (/) or U+003F QUESTION MARK (?) character, to its percent-encoded form.
9. Append /resource name/ to /url/.
10. Return /url/.

Note that these steps have been designed to lead to a URI in normal form (see Section 6.3).

7. Finding and Addressing CoAP End-Points

7.1. Resource Discovery

The discovery of resources offered by a CoAP end-point is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. A CoAP end-point SHOULD support the CoRE Link Format of discoverable resources as described in [I-D.ietf-core-link-format]. It is up to the server which resources are made discoverable (if any).

7.1.1. Content-type code 'ct' attribute

This section defines a new Web Linking [RFC5988] attribute for use with [I-D.ietf-core-link-format]. The Content-type code "ct" attribute provides a hint about the Internet media type(s) this resource returns. Note that this is only a hint, and does not override the Content-type Option of a CoAP response obtained by actually following the link. The value is in the CoAP identifier code format as a decimal ASCII integer and MUST be in the range of 0-65535 (16-bit unsigned integer). For example application/xml would be indicated as "ct=41". If no Content-type code attribute is present then nothing about the type can be assumed. The Content-type code attribute MAY appear more than once in a link, indicating that multiple content-types are available.

```
link-extension    = <Defined in RFC5988>
link-extension    = ( "ct" "=" cardinal ) ; Range of 0-65535
cardinal          = "0" / %x31-39 *DIGIT
```

7.2. Default Ports

The CoAP default port number 5683 MUST be supported by a server for resource discovery and SHOULD be supported for providing access to other resources. The DTLS-secured CoAP default port number [IANA_TBD_PORT] MAY be supported by a server for resource discovery and for providing access to other resources. In addition other endpoints may be hosted in the dynamic port space.

When a CoAP server is hosted by a 6LoWPAN node, it SHOULD also support a port in the 61616-61631 compressed UDP port space defined in [RFC4944].

8. HTTP Mapping

CoAP supports a limited subset of HTTP functionality, and thus a mapping to HTTP is straightforward. There might be several reasons for mapping between CoAP and HTTP, for example when designing a web interface for use over either protocol or when realizing a CoAP-HTTP proxy. Likewise, CoAP could equally be mapped to other protocols such as XMPP [RFC6120] or SIP [RFC3264]; the definition of these mappings is out of scope of this specification.

There are two possible mappings via a forward proxy:

CoAP-HTTP Mapping: Enables CoAP clients to access resources on HTTP servers through an intermediary. This is initiated by including the Proxy-Uri Option with an "http" or "https" URI in a CoAP request to a CoAP-HTTP proxy.

HTTP-CoAP Mapping: Enables HTTP clients to access resources on CoAP servers through an intermediary. This is initiated by specifying a "coap" or "coaps" URI in the Request-Line of an HTTP request to an HTTP-CoAP proxy.

Either way, only the Request/Response model of CoAP is mapped to HTTP. The underlying model of confirmable or non-confirmable messages, etc., is invisible and MUST have no effect on a proxy function. The following sections describe the handling of requests to a forward proxy. Reverse proxies are not specified as the proxy function is transparent to the client with the proxy acting as if it was the origin server.

8.1. CoAP-HTTP Mapping

If a request contains a Proxy-URI Option with an 'http' or 'https' URI [RFC2616], then the receiving CoAP end-point (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated HTTP resource and return the result to the client.

This section specifies for any CoAP request the CoAP response that the proxy should return to the client. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to an HTTP origin server.

Since HTTP and CoAP share the basic set of request methods, performing a CoAP request on an HTTP resource is not so different from performing it on a CoAP resource. The meanings of the individual CoAP methods when performed on HTTP resources are explained below.

If the proxy is unable or unwilling to service a request with an HTTP URI, a 5.05 (Proxying Not Supported) response SHOULD be returned to the client. If the proxy services the request by interacting with a third party (such as the HTTP origin server) and is unable to obtain a result within a reasonable time frame, a 5.04 (Gateway Timeout) response SHOULD be returned; if a result can be obtained but is not understood, a 5.02 (Bad Gateway) response SHOULD be returned.

8.1.1. GET

The GET method requests the proxy to return a representation of the HTTP resource identified by the request URI.

Upon success, a 2.05 (Content) response SHOULD be returned. The payload of the response MUST be a representation of the target HTTP resource, and the Content-Type Option be set accordingly. The response MUST indicate a Max-Age value that is no greater than the remaining time the representation can be considered fresh. If the HTTP entity has an entity tag, the proxy SHOULD include an ETag Option in the response and process ETag Options in requests as described below.

A client can influence the processing of a GET request by including the following option:

Accept: The request MAY include one or more Accept Options, identifying the preferred response content-type.

ETag: The request MAY include one or more ETag Options, identifying responses that the client has stored. This requests the proxy to send a 2.03 (Valid) response whenever it would send a 2.05 (Content) response with an entity tag in the requested set otherwise.

8.1.2. PUT

The PUT method requests the proxy to update or create the HTTP resource identified by the request URI with the enclosed representation.

If a new resource is created at the request URI, a 2.01 (Created) response MUST be returned to the client. If an existing resource is modified, a 2.04 (Changed) response MUST be returned to indicate successful completion of the request.

8.1.3. DELETE

The DELETE method requests the proxy to delete the HTTP resource identified by the request URI at the HTTP origin server.

A 2.02 (Deleted) response MUST be returned to client upon success or if the resource does not exist at the time of the request.

8.1.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the HTTP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 2.04 (Changed) response MUST be returned to the client. If a resource has been created on the origin server, a 2.01 (Created) response MUST be returned.

8.2. HTTP-CoAP Mapping

If an HTTP request contains a Request-URI with a 'coap' or 'coaps' URI, then the receiving HTTP end-point (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated CoAP resource and return the result to the client.

This section specifies for any HTTP request the HTTP response that the proxy should return to the client. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to a CoAP origin server. The meanings of the individual HTTP methods when performed on CoAP resources are explained below.

If the proxy is unable or unwilling to service a request with a CoAP URI, a 501 (Not Implemented) response SHOULD be returned to the client. If the proxy services the request by interacting with a third party (such as the CoAP origin server) and is unable to obtain a result within a reasonable time frame, a 504 (Gateway Timeout) response SHOULD be returned; if a result can be obtained but is not understood, a 502 (Bad Gateway) response SHOULD be returned.

8.2.1. OPTIONS and TRACE

As the OPTIONS and TRACE methods are not supported in CoAP a 501 (Not Implemented) error MUST be returned to the client.

8.2.2. GET

The GET method requests the proxy to return a representation of the CoAP resource identified by the Request-URI.

Upon success, a 200 (OK) response SHOULD be returned. The payload of the response MUST be a representation of the target CoAP resource,

and the Content-Type Option be set accordingly. The response MUST indicate a Max-Age value that is no greater than the remaining time the representation can be considered fresh. If the CoAP entity has an entity tag, the proxy SHOULD include an ETag Option in the response.

A client can influence the processing of a GET request by including the following option:

Accept: Each individual Media-type of the HTTP Accept header in a request is mapped to a CoAP Accept option. HTTP Accept Media-type ranges, parameters and extensions are not supported by the CoAP Accept option. If the proxy cannot send a response which is acceptable according to the combined Accept field value, then the proxy SHOULD send a 406 (not acceptable) response.

Conditional GETs: Conditional HTTP GET requests that include an "If-Match" or "If-None-Match" request-header field can be mapped to a corresponding CoAP request. The "If-Modified-Since" and "If-Unmodified-Since" request-header fields are not directly supported by CoAP, but SHOULD be implemented locally by a caching proxy.

8.2.3. HEAD

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response.

Although there is no direct equivalent of HTTP's HEAD method in CoAP, an HTTP-CoAP proxy responds to HEAD requests for CoAP resources, and the HTTP headers are returned without a message-body.

8.2.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the CoAP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 200 (OK) or 204 (No Content) response MUST be returned to the client. If a resource has been created on the origin server, a 201 (Created) response MUST be returned.

8.2.5. PUT

The PUT method requests the proxy to update or create the CoAP resource identified by the Request-URI with the enclosed representation.

If a new resource is created at the Request-URI, a 201 (Created) response MUST be returned to the client. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.

8.2.6. DELETE

The DELETE method requests the proxy to delete the CoAP resource identified by the Request-URI at the CoAP origin server.

A successful response SHOULD be 200 (OK) if the response includes an entity describing the status or 204 (No Content) if the action has been enacted but the response does not include an entity.

8.2.7. CONNECT

This method can not currently be satisfied by an HTTP-CoAP proxy function as TLS to DTLS tunneling has not been specified. It is however expected that such a tunneling mapping will be defined in the future. A 501 (Not Implemented) error SHOULD be returned to the client.

9. Protocol Constants

This section defines the relevant protocol constants defined in this document:

RESPONSE_TIMEOUT 2 seconds

RESPONSE_RANDOM_FACTOR 1.5

MAX_RETRANSMIT 4

Future specifications are expected that will allow implementations to use other sources for initializing RESPONSE_TIMEOUT. The RESPONSE_TIMEOUT variable MAY be configured with a different value for special environments that exhibit very short or very long RTTs.

10. Security Considerations

This section defines the DTLS binding for CoAP, the alternative use of IPsec, and analyzes the possible threats to the protocol and its limitations.

During the provisioning phase, a CoAP device is provided with the security information that it needs, including keying materials and access control lists. This specification defines provisioning for the RawPublicKey mode in Appendix D.2. At the end of the provisioning phase, the device will be in one of four security modes with the following information for the given mode. The NoSec and RawPublicKey modes are mandatory to implement for this specification.

NoSec: There is no protocol level security (DTLS is disabled).
Alternative techniques to provide lower layer security SHOULD be used when appropriate. The use of IPsec is discussed in Section 10.2.

PreSharedKey: DTLS is enabled and there is a list of pre-shared keys and each key includes a list of which nodes it can be used to communicate with as described in Section 10.1.1. At the extreme there may be one key for each node this CoAP node needs to communicate with (1:1 node/key ratio).

RawPublicKey: DTLS is enabled and the device has an asymmetric key pair, but without an X.509 certificate as described in Section 10.1.2. The device also has an identity calculated from the public key and a list of identities of the nodes it can communicate with.

Certificate: DTLS is enabled and the device has an asymmetric key pair with an X.509 [RFC5280] certificate that binds it to its Authority Name and is signed by some common trust root as described in Section 10.1.3. The device also has a list of root trust anchors that can be used for validating a certificate.

In the "NoSec" mode, the system simply sends the packets over normal UDP over IP and is indicated by the "coap" scheme and the CoAP default port. The system is secured only by keeping attackers from being able to send or receive packets from the network with the CoAP nodes; see Section 10.3.5 for an additional complication with this approach.

The other three security modes are achieved using DTLS and are indicated by the "coaps" scheme and DTLS-secured CoAP default port. The result is a security association that can be used to authenticate (within the limits of the security model) and, based on this

authentication, authorize the communication partner. CoAP itself does not provide protocol primitives for authentication or authorization; where this is required, it can either be provided by communication security (i.e., IPsec or DTLS) or by object security (within the payload). Devices that require authorization for certain operations are expected to require one of these two forms of security. Necessarily, where an intermediary is involved, communication security only works when that intermediary is part of the trust relationships; CoAP does not provide a way to forward different levels of authorization that clients may have with an intermediary to further intermediaries or origin servers -- it therefore may be required to perform all authorization at the first intermediary.

10.1. Securing CoAP with DTLS

Just as HTTP is secured using Transport Layer Security (TLS) over TCP, CoAP is secured using Datagram TLS (DTLS) [RFC4347] over UDP. This section defines the CoAP binding to DTLS, along with the minimal MUST implement configurations appropriate for constrained environments. DTLS is in practice TLS with added features to deal with the unreliable nature of the UDP transport.

In some constrained nodes (limited flash and/or RAM) and networks (limited bandwidth or high scalability requirements), and depending on the specific cipher suites in use, DTLS may not be applicable. Some of DTLS' cipher suites can add significant implementation complexity as well as some initial handshake overhead needed when setting up the security association. Once the initial handshake is completed, DTLS adds a limited per-datagram overhead of approximately 13 bytes, not including any initialization vectors (which are generally implicitly derived with DTLS), integrity check values (e.g., 8 bytes with TLS_PSK_WITH_AES_128_CCM_8 [I-D.mcgregw-tls-aes-ccm]) and padding required by the cipher suite. Whether and which mode of using DTLS is applicable for a CoAP-based application should be carefully weighed considering the specific cipher suites that may be applicable, and whether the session maintenance makes it compatible with application flows and sufficient resources are available on the constrained nodes and for the added network overhead. DTLS is not applicable to group keying (multicast communication); however, it may be a component in a future group key management protocol.

Devices SHOULD support the Server Name Indication (SNI) to indicate their Authority Name in the SNI HostName field as defined in Section 3 of [RFC6066]. This is needed so that when a host that acts as a virtual server for multiple Authorities receives a new DTLS connection, it knows which keys to use for the DTLS session.

DTLS connections in RawPublicKey and Certificate mode are set up using mutual authentication so they can remain up and be reused for future message exchanges in either direction. Devices can close a DTLS connection when they need to recover resources but in general they should keep the connection up for as long as possible. Closing the DTLS connection after every CoAP message exchange is very inefficient.

10.1.1. PreSharedKey Mode

When forming a connection to a new node, the system selects an appropriate key based on which nodes it is trying to reach then forms a DTLS session using a PSK (Pre-Shared Key) mode of DTLS. Implementations in these modes MUST support the mandatory to implement cipher suite TLS_PSK_WITH_AES_128_CCM_8 as specified in [I-D.mcgreww-tls-aes-ccm].

The security considerations of [RFC4279] (Section 7) apply. In particular, applications should carefully weigh whether they need Perfect Forward Secrecy (PFS) or not and select an appropriate cipher suite (7.1). The entropy of the PSK must be sufficient to mitigate against brute-force and (where the PSK is not chosen randomly but by a human) dictionary attacks (7.2). The cleartext communication of client identities may leak data or compromise privacy (7.3).

10.1.2. RawPublicKey Mode

In this mode the device has an asymmetric key pair but without an X.509 certificate (called a raw public key). A device MAY be configured with multiple raw public keys. The type and length of the raw public key depends on the cipher suite used. Implementations in RawPublicKey mode MUST support the mandatory to implement cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as specified in [RFC5246], [RFC4492].

TLS does not currently define a way to carry a raw public key during the handshake phase. The raw public key is therefore wrapped in an X.509 certificate [RFC5280]. The only requirement on this certificate is that it MUST have a subjectPublicKeyInfo field with the algorithm set to that of the cipher suite used and the public key placed in subjectPublicKey field. The certificate MAY additionally have validity information. If the validity field is present and not currently valid, the certificate MUST be rejected.

10.1.3. Certificate Mode

Implementations in Certificate Mode MUST support the mandatory to implement cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as

specified in [RFC5246].

The Authority Name in the certificate is the name that would be used in the Authority part of a CoAP URI. It is worth noting that this would typically not be either an IP address or DNS name but would instead be a long term unique identifier for the device such as the EUI-64 [EUI64]. The discovery process used in the system would build up the mapping between IP addresses of the given devices and the Authority Name for each device. Some devices could have more than one Authority and would need more than a single certificate.

When a new connection is formed, the certificate from the remote device needs to be verified. If the CoAP node has a source of absolute time, then the node SHOULD check the validity dates of the certificate are within range. The certificate MUST also be signed by an appropriate chain of trust. If the certificate contains a SubjectAltName, then the Authority Name MUST match at least one of the authority names of any CoAP URI found in a URI type fields in the SubjectAltName set. If there is no SubjectAltName in the certificate, then the Authoritative Name must match the CN found in the certificate using the matching rules defined in [RFC2818] with the exception that certificates with wildcards are not allowed. Further access control is performed as described in Appendix D.3.3.

If the system has a shared key in addition to the certificate, then a cipher suite that includes the shared key such as TLS_RSA_PSK_WITH_AES_128_CBC_SHA SHOULD be used.

10.2. Using CoAP with IPsec

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

Necessarily for AES-CCM, but much preferably also for AES-CBC, static keying should be avoided and the initial keying material be derived into transient session keys, e.g. using a low-overhead mode of IKEv2 [RFC5996] as described in [I-D.kivinen-ipsecme-ikev2-minimal]; such a

protocol for managing keys and sequence numbers is also the only way to achieve anti-replay capabilities. However, no recommendation can be made at this point on how to manage group keys (i.e., for multicast) in a constrained environment. Once any initial setup is completed, IPsec ESP adds a limited overhead of approximately 10 bytes per packet, not including initialization vectors, integrity check values and padding required by the cipher suite.

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

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

10.3. Threat analysis and protocol limitations

This section is meant to inform protocol and application developers about the security limitations of CoAP as described in this document. As CoAP realizes a subset of the features in HTTP/1.1, the security considerations in Section 15 of [RFC2616] are also pertinent to CoAP. This section concentrates on describing limitations specific to CoAP.

10.3.1. Protocol Parsing, Processing URIs

A network-facing application can exhibit vulnerabilities in its processing logic for incoming packets. Complex parsers are well-known as a likely source of such vulnerabilities, such as the ability to remotely crash a node, or even remotely execute arbitrary code on it. CoAP attempts to narrow the opportunities for introducing such vulnerabilities by reducing parser complexity, by giving the entire range of encodable values a meaning where possible, and by aggressively reducing complexity that is often caused by unnecessary choice between multiple representations that mean the same thing. Much of the URI processing has been moved to the clients, further reducing the opportunities for introducing vulnerabilities into the servers. Even so, the URI processing code in CoAP implementations is likely to be a large source of remaining vulnerabilities and should be implemented with special care. The most complex parser remaining could be the one for the link-format, although this also has been designed with a goal of reduced implementation complexity [I-D.ietf-core-link-format]. (See also section 15.2 of [RFC2616].)

10.3.2. Proxying and Caching

As mentioned in 15.7 of [RFC2616], which see, proxies are by their very nature men-in-the-middle, breaking any IPsec or DTLS protection that a direct CoAP message exchange might have. They are therefore interesting targets for breaking confidentiality or integrity of CoAP message exchanges. As noted in [RFC2616], they are also interesting targets for breaking availability.

The threat to confidentiality and integrity of request/response data is amplified where proxies also cache. Note that CoAP does not define any of the cache-suppressing Cache-Control options that HTTP/1.1 provides to better protect sensitive data.

Finally, a proxy that fans out Separate Responses (as opposed to Piggy-backed Responses) to multiple original requesters may provide additional amplification (see below).

10.3.3. Risk of amplification

CoAP servers generally reply to a request packet with a response packet. This response packet may be significantly larger than the request packet. An attacker might use CoAP nodes to turn a small attack packet into a larger attack packet, an approach known as amplification. There is therefore a danger that CoAP nodes could become implicated in denial of service (DoS) attacks by using the amplifying properties of the protocol: An attacker that is attempting to overload a victim but is limited in the amount of traffic it can generate, can use amplification to generate a larger amount of traffic.

This is particularly a problem in nodes that enable NoSec access, that are accessible from an attacker and can access potential victims (e.g. on the general Internet), as the UDP protocol provides no way to verify the source address given in the request packet. An attacker need only place the IP address of the victim in the source address of a suitable request packet to generate a larger packet directed at the victim.

As a mitigating factor, many constrained networks will only be able to generate a small amount of traffic, which may make CoAP nodes less attractive for this attack. However, the limited capacity of the constrained network makes the network itself a likely victim of an amplification attack.

A CoAP server can reduce the amount of amplification it provides to an attacker by using slicing/blocking modes of CoAP [I-D.ietf-core-block] and offering large resource representations

only in relatively small slices. E.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

CoAP also supports the use of multicast IP addresses in requests, an important requirement for M2M. Multicast CoAP requests may be the source of accidental or deliberate denial of service attacks, especially over constrained networks. This specification attempts to reduce the amplification effects of multicast requests by limiting when a response is returned. To limit the possibility of malicious use, CoAP servers SHOULD NOT accept multicast requests that can not be authenticated. If possible a CoAP server SHOULD limit the support for multicast requests to specific resources where the feature is required.

On some general purpose operating systems providing a Posix-style API, it is not straightforward to find out whether a packet received was addressed to a multicast address. While many implementations will know whether they have joined a multicast group, this creates a problem for packets addressed to multicast addresses of the form FF0x::1, which are received by every IPv6 node. Implementations SHOULD make use of modern APIs such as IPV6_RECVPKTINFO [RFC3542], if available, to make this determination.

10.3.4. IP Address Spoofing Attacks

Due to the lack of a handshake in UDP, a rogue endpoint which is free to read and write messages carried by the constrained network (i.e. NoSec or PreSharedKey deployments with nodes/key ratio > 1:1), may easily attack a single endpoint, a group of endpoints, as well as the whole network e.g. by:

1. spoofing RST in response to a CON message, thus making an endpoint "deaf"; or
2. spoofing the entire response with forged payload/options (this has different levels of impact: from single response disruption, to much bolder attacks on the supporting infrastructure, e.g. poisoning proxy caches, or tricking validation / lookup interfaces in resource directories and, more generally, any component that stores global network state and uses CoAP as the messaging facility to handle state set/update's is a potential target.); or
3. spoofing a multicast request for a target node which may result in both network congestion/collapse and victim DoS'ing / forced wakeup from sleeping; or

4. spoofing observe messages, etc.

In principle, spoofing can be detected by CoAP only in case CON semantics is used, because of unexpected ACK/RSTs coming from the deceived endpoint. But this imposes keeping track of the used MIDs which is not always possible, and moreover detection becomes available usually after the damage is already done. This kind of attack can be prevented using security modes other than NoSec.

10.3.5. Cross-Protocol Attacks

The ability to incite a CoAP end-point to send packets to a fake source address can be used not only for amplification, but also for cross-protocol attacks:

- o the attacker sends a message to a CoAP end-point with a fake source address,
- o the CoAP end-point replies with a message to the given source address,
- o the victim at the given source address receives a UDP packet that it interprets according to the rules of a different protocol.

This may be used to circumvent firewall rules that prevent direct communication from the attacker to the victim, but happen to allow communication from the CoAP end-point (which may also host a valid role in the other protocol) to the victim.

Also, CoAP end-points may be the victim of a cross-protocol attack generated through an end-point of another UDP-based protocol such as DNS. In both cases, attacks are possible if the security properties of the end-points rely on checking IP addresses (and firewalling off direct attacks sent from outside using fake IP addresses). In general, because of their lack of context, UDP-based protocols are relatively easy targets for cross-protocol attacks.

Finally, CoAP URIs transported by other means could be used to incite clients to send messages to end-points of other protocols.

One mitigation against cross-protocol attacks is strict checking of the syntax of packets received, combined with sufficient difference in syntax. As an example, it might help if it were difficult to incite a DNS server to send a DNS response that would pass the checks of a CoAP end-point. Unfortunately, the first two bytes of a DNS reply are an ID that can be chosen by the attacker, which map into the interesting part of the CoAP header, and the next two bytes are then interpreted as CoAP's Message ID (i.e., any value is

acceptable). The DNS count words may be interpreted as multiple instances of a (non-existent, but elective) CoAP option 0. The echoed query finally may be manufactured by the attacker to achieve a desired effect on the CoAP end-point; the response added by the server (if any) might then just be interpreted as added payload.

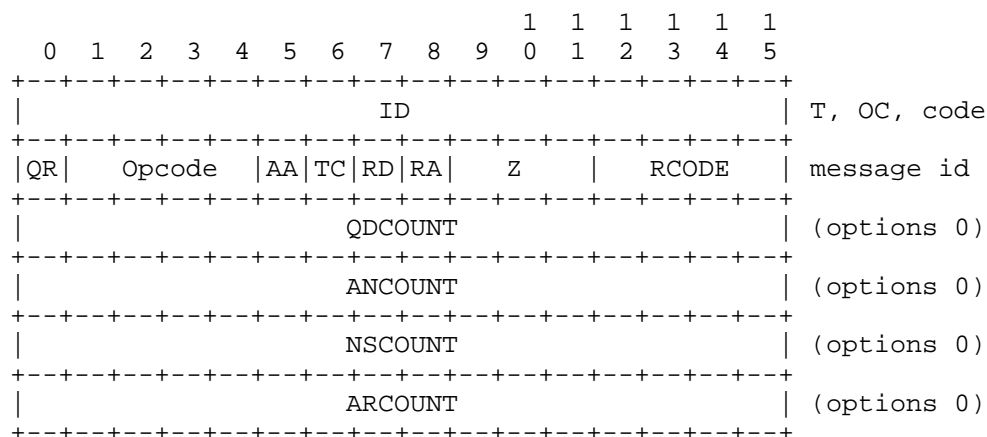


Figure 10: DNS Header vs. CoAP Message

In general, for any pair of protocols, one of the protocols can very well have been designed in a way that enables an attacker to cause the generation of replies that look like messages of the other protocol. It is often much harder to ensure or prove the absence of viable attacks than to generate examples that may not yet completely enable an attack but might be further developed by more creative minds. Cross-protocol attacks can therefore only be completely mitigated if end-points don't authorize actions desired by an attacker just based on trusting the source IP address of a packet. Conversely, a NoSec environment that completely relies on a firewall for CoAP security not only needs to firewall off the CoAP end-points but also all other end-points that might be incited to send UDP messages to CoAP end-points using some other UDP-based protocol.

In addition to the considerations above, the security considerations for DTLS with respect to cross-protocol attacks apply. E.g., if the same DTLS security association ("connection") is used to carry data of multiple protocols, DTLS no longer provides protection against cross-protocol attacks between these protocols.

11. IANA Considerations

11.1. CoAP Code Registry

This document defines a registry for the values of the Code field in the CoAP header. The name of the registry is "CoAP Codes".

All values are assigned by sub-registries according to the following ranges:

- 0 Indicates an empty message (see Section 4.4).
- 1-31 Indicates a request. Values in this range are assigned by the "CoAP Method Codes" sub-registry (see Section 11.1.1).
- 32-63 Reserved
- 64-191 Indicates a response. Values in this range are assigned by the "CoAP Response Codes" sub-registry (see Section 11.1.2).
- 192-255 Reserved

11.1.1. Method Codes

The name of the sub-registry is "CoAP Method Codes".

Each entry in the sub-registry must include the Method Code in the range 1-31, the name of the method, and a reference to the method's documentation.

Initial entries in this sub-registry are as follows:

Code	Name	Reference
1	GET	[RFCXXXX]
2	POST	[RFCXXXX]
3	PUT	[RFCXXXX]
4	DELETE	[RFCXXXX]

Table 2: CoAP Method Codes

All other Method Codes are Unassigned.

The IANA policy for future additions to this registry is "IETF Review" as described in [RFC5226].

The documentation of a method code should specify the semantics of a

request with that code, including the following properties:

- o The response codes the method returns in the success case.
- o Whether the method is idempotent, safe, or both.
- o Whether the request causes a cache to mark responses stored for the request URI as not fresh.

11.1.2. Response Codes

The name of the sub-registry is "CoAP Response Codes".

Each entry in the sub-registry must include the Response Code in the range 64-191, a description of the Response Code, and a reference to the Response Code's documentation.

Initial entries in this sub-registry are as follows:

Code	Description	Reference
65	2.01 Created	[RFCXXXX]
66	2.02 Deleted	[RFCXXXX]
67	2.03 Valid	[RFCXXXX]
68	2.04 Changed	[RFCXXXX]
69	2.05 Content	[RFCXXXX]
128	4.00 Bad Request	[RFCXXXX]
129	4.01 Unauthorized	[RFCXXXX]
130	4.02 Bad Option	[RFCXXXX]
131	4.03 Forbidden	[RFCXXXX]
132	4.04 Not Found	[RFCXXXX]
133	4.05 Method Not Allowed	[RFCXXXX]
134	4.06 Not Acceptable	[RFCXXXX]
140	4.12 Precondition Failed	[RFCXXXX]
141	4.13 Request Entity Too Large	[RFCXXXX]
143	4.15 Unsupported Media Type	[RFCXXXX]
160	5.00 Internal Server Error	[RFCXXXX]
161	5.01 Not Implemented	[RFCXXXX]
162	5.02 Bad Gateway	[RFCXXXX]
163	5.03 Service Unavailable	[RFCXXXX]
164	5.04 Gateway Timeout	[RFCXXXX]
165	5.05 Proxying Not Supported	[RFCXXXX]

Table 3: CoAP Response Codes

The Response Codes 96-127 are Reserved for future use. All other

Response Codes are Unassigned.

The IANA policy for future additions to this registry is "IETF Review" as described in [RFC5226].

The documentation of a response code should specify the semantics of a response with that code, including the following properties:

- o The methods the response code applies to.
- o Whether payload is required, optional or not allowed.
- o The semantics of the payload. For example, the payload of a 2.05 (Content) response is a representation of the target resource; the payload in an error response is a human-readable diagnostic message.
- o The format of the payload. For example, the format in a 2.05 (Content) response is indicated by the Content-Type Option; the format of the payload in an error response is always Net-Unicode text.
- o Whether the response is cacheable according to the freshness model.
- o Whether the response is validatable according to the validation model.
- o Whether the response causes a cache to mark responses stored for the request URI as not fresh.

11.2. Option Number Registry

This document defines a registry for the Option Numbers used in CoAP options. The name of the registry is "CoAP Option Numbers".

Each entry in the registry must include the Option Number, the name of the option and a reference to the option's documentation.

Initial entries in this registry are as follows:

Number	Name	Reference
1	Content-Type	[RFCXXXX]
2	Max-Age	[RFCXXXX]
3	Proxy-Uri	[RFCXXXX]
4	ETag	[RFCXXXX]
5	Uri-Host	[RFCXXXX]
6	Location-Path	[RFCXXXX]
7	Uri-Port	[RFCXXXX]
8	Location-Query	[RFCXXXX]
9	Uri-Path	[RFCXXXX]
11	Token	[RFCXXXX]
12	Accept	[RFCXXXX]
13	If-Match	[RFCXXXX]
15	Uri-Query	[RFCXXXX]
21	If-None-Match	[RFCXXXX]

Table 4: CoAP Option Numbers

The Option Number 0 is Reserved for future use. The Option Numbers 14, 28, 42, ... are Reserved for "fenceposting" (see Section 3.2). All other Option Numbers are Unassigned.

The IANA policy for future additions to this registry is "IETF Review" as described in [RFC5226].

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

- o The meaning of the option in a request.
- o The meaning of the option in a response.
- o Whether the option is critical or elective, as determined by the Option Number.
- o The format and length of the option's value.
- o Whether the option must occur at most once or whether it can occur multiple times.
- o The default value, if any.

11.3. Media Type Registry

Media types are identified by a string, such as "application/xml" [RFC2046]. In order to minimize the overhead of using these media types to indicate the format of payloads, this document defines a registry for a subset of Internet media types to be used in CoAP and assigns each a numeric identifier. The name of the registry is "CoAP Media Types".

Each entry in the registry must include the media type registered with IANA, the numeric identifier in the range 0-65535 to be used for that media type in CoAP, and a reference to a document describing what payload with that media type means semantically.

Initial entries in this registry are as follows:

Media type	Id.	Reference
text/plain; charset=utf-8	0	[RFC2046][RFC3676][RFC5147]
application/link-format	40	[I-D.ietf-core-link-format]
application/xml	41	[RFC3023]
application/octet-stream	42	[RFC2045][RFC2046]
application/exi	47	[EXIMIME]
application/json	50	[RFC4627]

Table 5: CoAP Media Types

The identifiers between 201 and 255 inclusive are reserved for Private Use. All other identifiers are Unassigned.

Because the name space of single-byte identifiers is so small, the IANA policy for future additions in the range 0-200 inclusive to the registry is "Expert Review" as described in [RFC5226]. The IANA policy for additions in the range 256-65535 inclusive is "First Come First Served" as described in [RFC5226].

In machine to machine applications, it is not expected that generic Internet media types such as text/plain, application/xml or application/octet-stream are useful for real applications in the long term. It is recommended that M2M applications making use of CoAP will request new Internet media types from IANA indicating semantic information about how to create or parse a payload. For example, a Smart Energy application payload carried as XML might request a more specific type like application/se+xml or application/se+exi.

11.4. URI Scheme Registration

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

URI scheme name.
coap

Status.
Permanent.

URI scheme syntax.
Defined in Section 6.1 of [RFCXXXX].

URI scheme semantics.
The "coap" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP). The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "http" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP end-points to access CoAP resources.

Interoperability considerations.
None.

Security considerations.
See Section 10.3.1 of [RFCXXXX].

Contact.
IETF Chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
[RFCXXXX]

11.5. Secure URI Scheme Registration

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

URI scheme name.
 coaps

Status.
 Permanent.

URI scheme syntax.
 Defined in Section 6.2 of [RFCXXXX].

URI scheme semantics.
 The "coaps" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using DTLS for session security. The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "https" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.
 The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
 The scheme is used by CoAP end-points to access CoAP resources using DTLS.

Interoperability considerations.
 None.

Security considerations.
 See Section 10.3.1 of [RFCXXXX].

Contact.
 IETF Chair <chair@ietf.org>

Author/Change controller.
 IESG <iesg@ietf.org>

References.
[RFCXXXX]

11.6. Service Name and Port Number Registration

One of the functions of CoAP is resource discovery: a CoAP client can ask a CoAP server about the resources offered by it (see Section 7.1). To enable resource discovery just based on the knowledge of an IP address, the CoAP port for resource discovery needs to be standardized.

IANA has assigned the port number 5683 and the service name "coap", in accordance with [I-D.ietf-tsvwg-iana-ports].

Besides unicast, CoAP can be used with both multicast and anycast.

Service Name.
coap

Transport Protocol.
UDP

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFCXXXX]

Port Number.
5683

11.7. Secure Service Name and Port Number Registration

CoAP resource discovery may also be provided using the DTLS-secured CoAP "coaps" scheme. Thus the CoAP port for secure resource discovery needs to be standardized.

This document requests the assignment of the port number [IANA_TBD_PORT] and the service name "coaps", in accordance with [I-D.ietf-tsvwg-iana-ports].

Besides unicast, Secure CoAP can be used with anycast.

Service Name.
coaps

Transport Protocol.
UDP

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
DTLS-secured CoAP

Reference.
[RFCXXXX]

Port Number.
[IANA_TBD_PORT]

12. Acknowledgements

Special thanks to Peter Bigot and Cullen Jennings for substantial contributions to the ideas and text in the document, along with countless detailed reviews and discussions.

Thanks to Michael Stuber, Richard Kelsey, Guido Moritz, Peter Van Der Stok, Adriano Pezzuto, Lisa Dussealt, Alexey Melnikov, Gilbert Clark, Salvatore Loreto, Petri Mutka, Szymon Sasin, Robert Quattlebaum, Robert Cragie, Angelo Castellani, Tom Herbst, Ed Beroaset, Gilman Tolle, Robby Simpson, Colin O'Flynn, Eric Rescorla, Matthieu Vial, Linyi Tian, Kerry Lynn, Dale Seed, Akbar Rahman, Charles Palmer, Thomas Fossati and David Ryan for helpful comments and discussions that have shaped the document.

Some of the text has been lifted from the working documents of the IETF httpbis working group.

13. References

13.1. Normative References

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format",
draft-ietf-core-link-format-07 (work in progress),

July 2011.

- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.
- [RFC3602] Frankel, S., Glenn, R., and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec", RFC 3602, September 2003.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [RFC3676] Gellens, R., "The Text/Plain Format and DelSp Parameters", RFC 3676, February 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.
- [RFC4309] Housley, R., "Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)", RFC 4309, December 2005.
- [RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer

Security", RFC 4347, April 2006.

- [RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.
- [RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC4835] Manral, V., "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 4835, April 2007.
- [RFC5147] Wilde, E. and M. Duerst, "URI Fragment Identifiers for the text/plain Media Type", RFC 5147, April 2008.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen,

"Internet Key Exchange Protocol Version 2 (IKEv2)",
RFC 5996, September 2010.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions:
Extension Definitions", RFC 6066, January 2011.

13.2. Informative References

[EUI64] "GUIDELINES FOR 64-BIT GLOBAL IDENTIFIER (EUI-64)
REGISTRATION AUTHORITY", April 2010, <<http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>>.

[EXIMIME] "Efficient XML Interchange (EXI) Format 1.0",
December 2009, <<http://www.w3.org/TR/2009/CR-exi-20091208/#mediaTypeRegistration>>.

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

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",
draft-ietf-core-block-04 (work in progress), July 2011.

[I-D.ietf-httpbis-pl-messaging]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H.,
Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., and
J. Reschke, "HTTP/1.1, part 1: URIs, Connections, and
Message Parsing", draft-ietf-httpbis-pl-messaging-17 (work
in progress), October 2011.

[I-D.ietf-tsvwg-iana-ports]
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",
draft-ietf-tsvwg-iana-ports-10 (work in progress),
February 2011.

[I-D.kivinen-ipsecme-ikev2-minimal]
Kivinen, T., "Minimal IKEv2",
draft-kivinen-ipsecme-ikev2-minimal-00 (work in progress),
February 2011.

[I-D.mcgrew-tls-aes-ccm]
McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for TLS",

draft-mcgrew-tls-aes-ccm-01 (work in progress),
March 2011.

- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

Appendix A. Integer Option Value Format

Options of type uint contain a non-negative integer that is represented in network byte order using a variable number of bytes, as shown below.

```

Length = 0      (implies value of 0)

                0
                0 1 2 3 4 5 6 7
                +---+---+---+---+---+
Length = 1      |           0-255           |
                +---+---+---+---+---+

                0                               1
                0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
                +---+---+---+---+---+---+---+---+---+---+
Length = 2      |           0-65535           |
                +---+---+---+---+---+---+---+---+---+

Length = 3 is 24 bits, Length = 4 is 32 bits etc.
```

Appendix B. Examples

This section gives a number of short examples with message flows for GET requests. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and multicast.

Figure 11 shows a basic GET request causing a piggy-backed response: The client sends a Confirmable GET request for the resource `coap://server/temperature` to the server with a Message ID of `0x7d34`. The request includes one Uri-Path Option (Delta $0 + 9 = 9$, Length 11, Value "temperature"); the Token is left at its default value (empty). This request is a total of 16 bytes long. A 2.05 (Content) response is returned in the Acknowledgement message that acknowledges the Confirmable request, echoing both the Message ID `0x7d34` and the (implicitly empty) Token value. The response includes a Payload of "22.3 C" and is 10 bytes long.

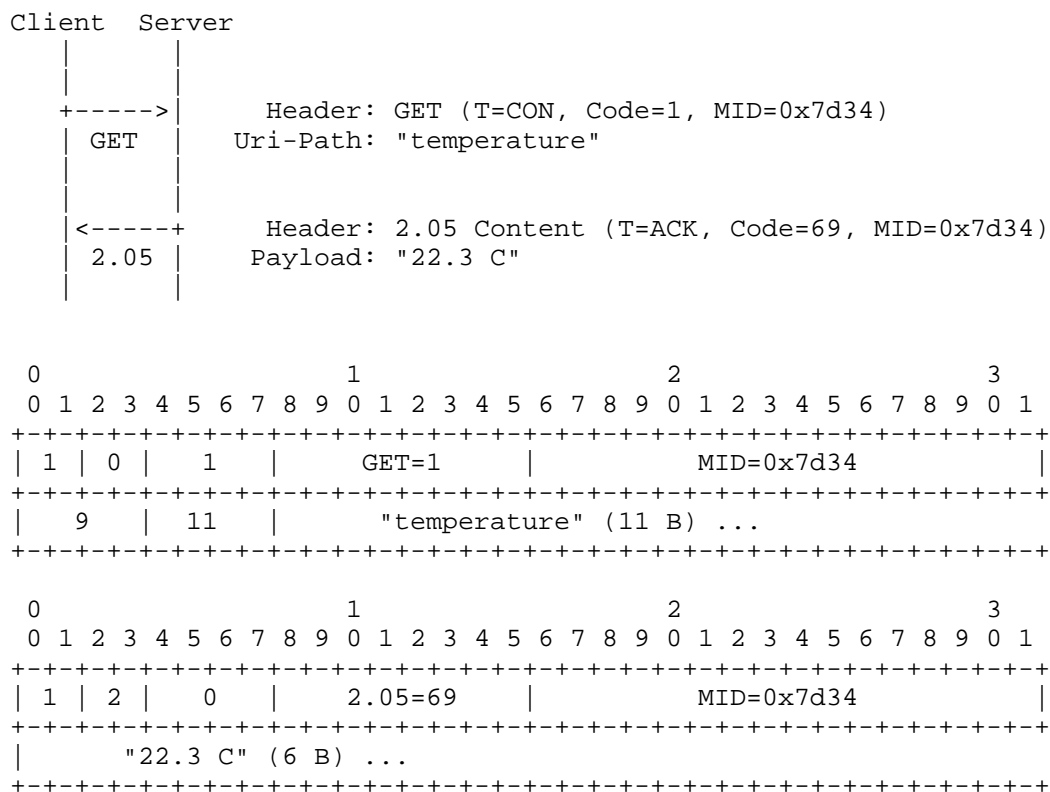


Figure 11: Confirmable request; piggy-backed response

Figure 12 shows a similar example, but with the inclusion of an explicit Token Option (Delta $9 + 2 = 11$, Length 1, Value 0x20) in the request and (Delta $11 + 0 = 11$) in the response, increasing the sizes to 18 and 12 bytes, respectively.

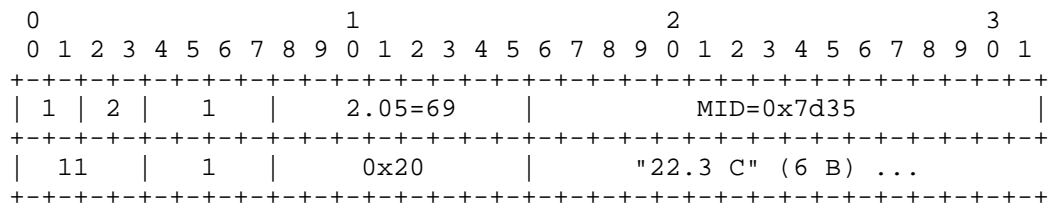
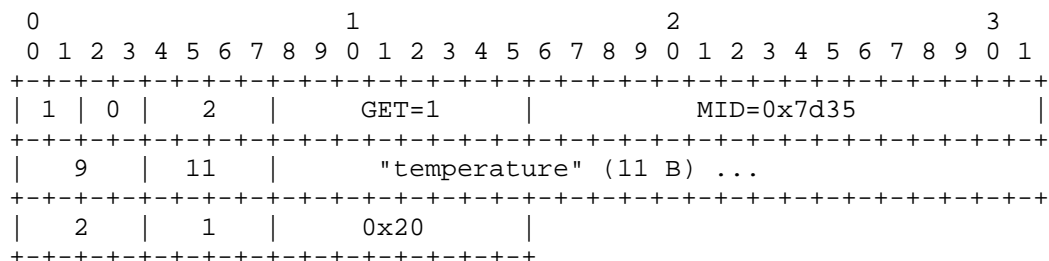
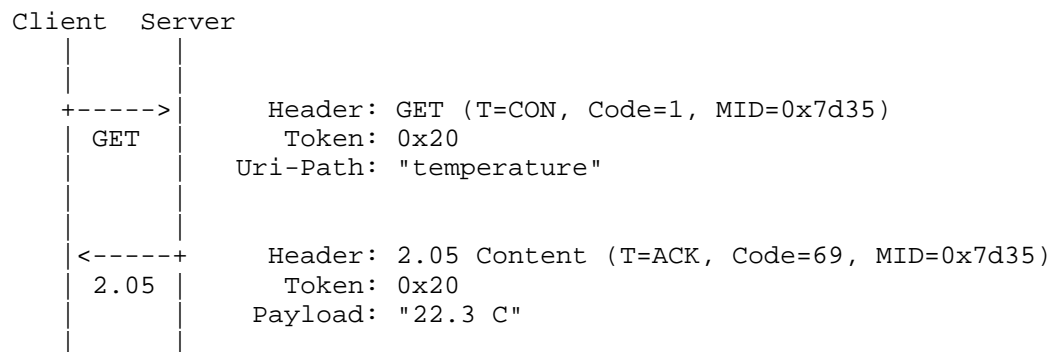


Figure 12: Confirmable request; piggy-backed response

In Figure 13, the Confirmable GET request is lost. After RESPONSE_TIMEOUT seconds, the client retransmits the request, resulting in a piggy-backed response as in the previous example.

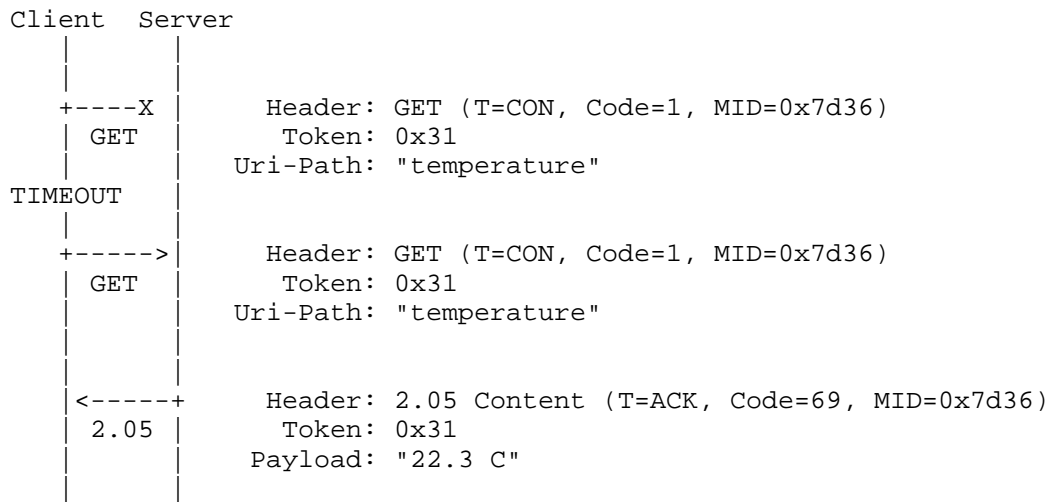


Figure 13: Confirmable request (retransmitted); piggy-backed response

In Figure 14, the first Acknowledgement message from the server to the client is lost. After RESPONSE_TIMEOUT seconds, the client retransmits the request.

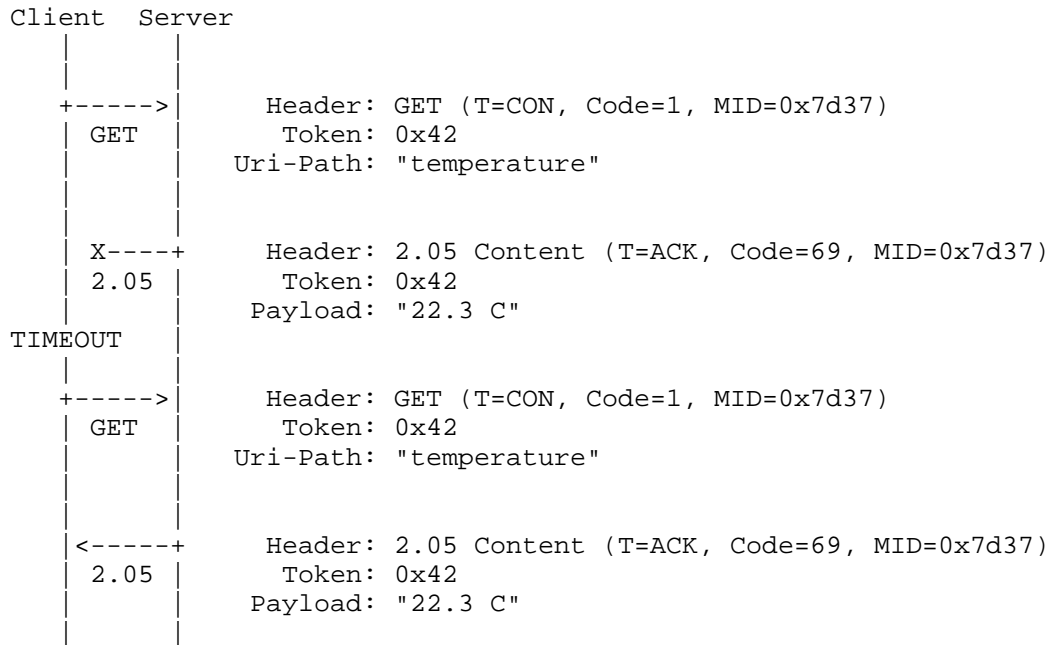


Figure 14: Confirmable request; piggy-backed response (retransmitted)

In Figure 15, the server acknowledges the Confirmable request and sends a 2.05 (Content) response separately in a Confirmable message. Note that the Acknowledgement message and the Confirmable response do not necessarily arrive in the same order as they were sent. The client acknowledges the Confirmable response.

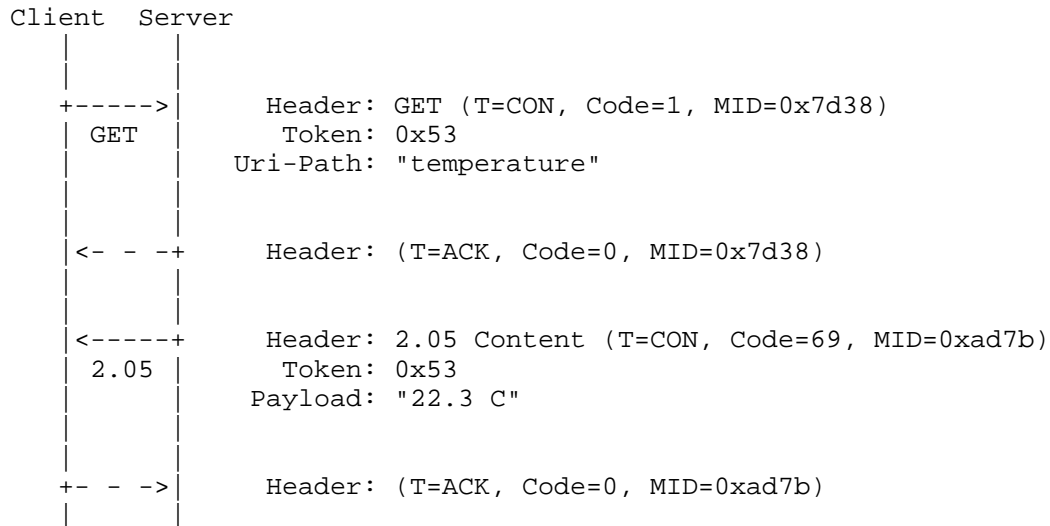


Figure 15: Confirmable request; separate response

Figure 16 shows an example where the client loses its state (e.g., crashes and is rebooted) right after sending a Confirmable request, so the separate response arriving some time later comes unexpected. In this case, the client rejects the Confirmable response with a Reset message. Note that the unexpected ACK is silently ignored.

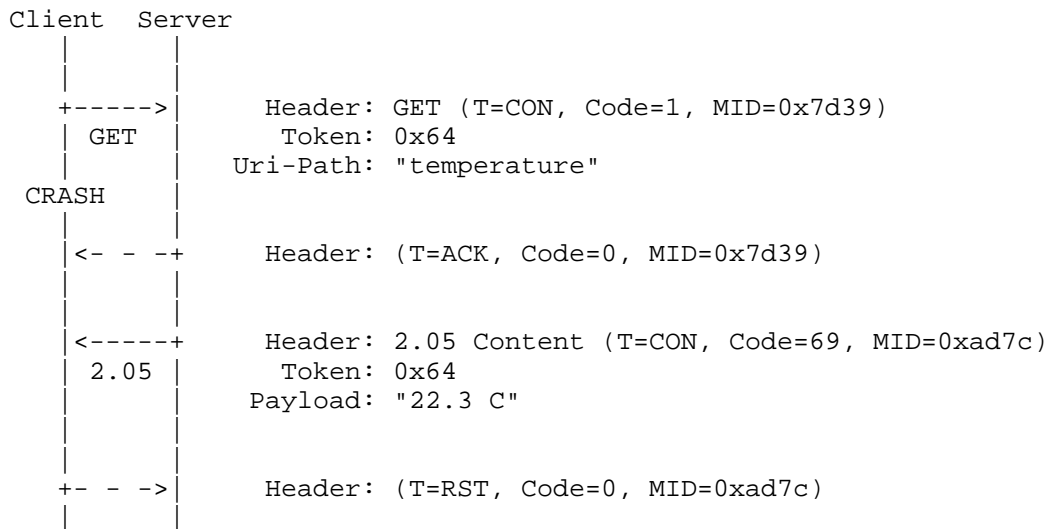


Figure 16: Confirmable request; separate response (unexpected)

Figure 17 shows a basic GET request where the request and the response are non-confirmable, so both may be lost without notice.

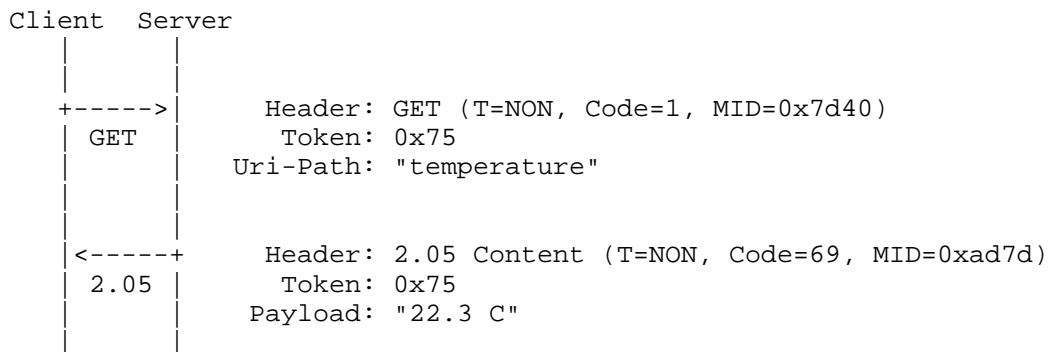


Figure 17: Non-confirmable request; Non-confirmable response

In Figure 18, the client sends a Non-confirmable GET request to a multicast address: all nodes in link-local scope. There are 3 servers on the link: A, B and C. Servers A and B have a matching resource, therefore they send back a Non-confirmable 2.05 (Content) response. The response sent by B is lost. C does not have matching response, therefore it sends a Non-confirmable 4.04 (Not Found) response.

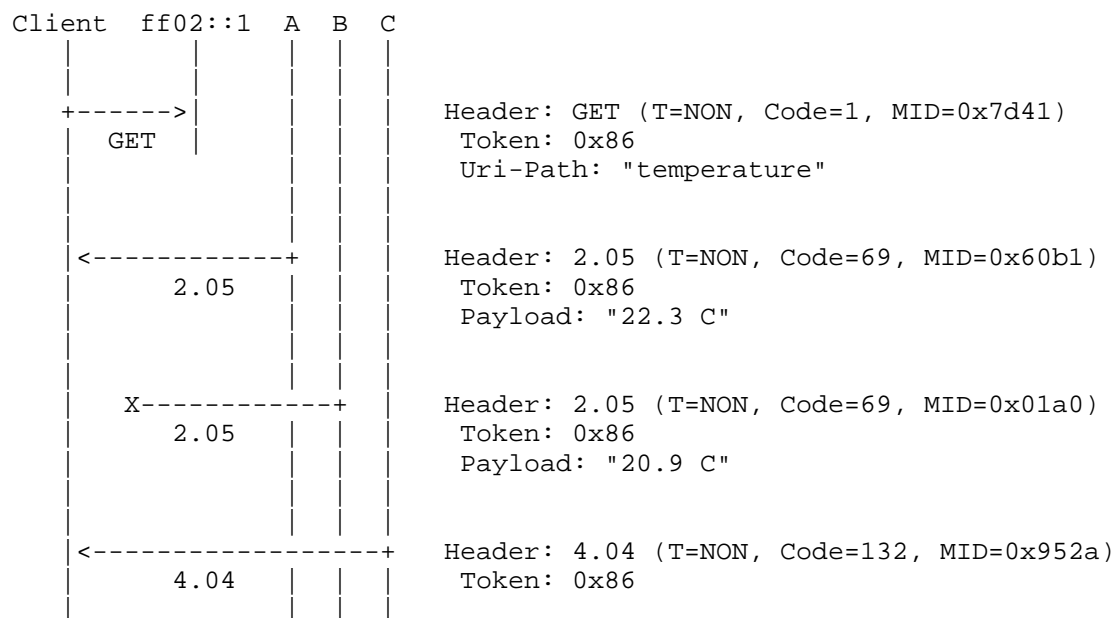


Figure 18: Non-confirmable request (multicast); Non-confirmable response

Appendix C. URI Examples

The following examples demonstrate different sets of Uri options, and the result after constructing an URI from them.

- o coap://[2001:db8::2:1]/
 - Destination IP Address = [2001:db8::2:1]
 - Destination UDP Port = 5683
- o coap://example.net/
 - Destination IP Address = [2001:db8::2:1]
 - Destination UDP Port = 5683
 - Uri-Host = "example.net"
- o coap://example.net/.well-known/core

Destination IP Address = [2001:db8::2:1]

Destination UDP Port = 5683

Uri-Host = "example.net"

Uri-Path = ".well-known"

Uri-Path = "core"

- o coap://xn--18j4d.example/%E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF

Destination IP Address = [2001:db8::2:1]

Destination UDP Port = 5683

Uri-Host = "xn--18j4d.example"

Uri-Path = the string composed of the Unicode characters U+3053 U+3093 U+306b U+3061 U+306f, usually represented in UTF-8 as E38193E38293E381ABE381A1E381AF hexadecimal

- o coap://198.51.100.1:61616//%2F//?%2F%2F?%26

Destination IP Address = 198.51.100.1

Destination UDP Port = 61616

Uri-Path = ""

Uri-Path = "/"

Uri-Path = ""

Uri-Path = ""

Uri-Query = "/"

Uri-Query = "?&"

Appendix D. Security Provisioning and Access Control

This Annex contains further information about ways to perform provisioning and access control for CoAP Security.

D.1. RawPublicKey Identity

An identity for the device configured with this asymmetric key pair is calculated from the public key and is used for provisioning devices and performing access control. The identity is an (TBD)-bit one-way hash of the public key. This is calculated by performing a (TBD) hash over the raw public key.

D.2. Provisioning

The RawPublicKey mode was designed to be easily provisioned in M2M deployments. It is assumed that each device has an appropriate asymmetric public key pair installed, and the identity of that public key has been calculated as described in Appendix D.1. During provisioning, the identity of each node is collected, for example by reading a barcode on the outside of the device or by obtaining a pre-compiled list of the identities. These identities are then installed in the corresponding end-point, for example an M2M data collection server. The identity is used for two purposes, to associate the end-point with further device information and to perform access control. During provisioning, an access control list of identities the device may start DTLS sessions with SHOULD also be installed.

D.3. Access Control

D.3.1. PreSharedKey Mode

In this mode in order to perform access control, identity needs to be assigned when installing or negotiating keys for the device. This identity may also be needed to choose the correct key to use in a DTLS session. The exact mechanism for provisioning keys, maintaining identities and using those for access control in PreSharedKey mode is out of scope for this specification.

D.3.2. RawPublicKey Mode

In this mode the identity of the public key for a device is used for access control. An end-point SHOULD keep a list of identities that it allows to access its resource, and MAY also support more detailed access control on the method or resource level. When a DTLS session is negotiated, a CoAP server that has an access control list MUST check the identity of the client. This is done by calculating the identity of the client's public key as described in Appendix D.1. A client SHOULD also verify the identity of the server if it has been configured with the appropriate access control list.

D.3.3. Certificate Mode

When in Certificate mode, access control is performed using the Authority Name from the certificate (e.g. the EUI-64 of the device). An end-point is provisioned with the list of Authority Names it can communicate with, and MAY also support more detailed access control on the method or resource level. When a DTLS session is negotiated, a CoAP server that has an access control list MUST check the Authority Name of the client's certificate. A client SHOULD also verify the identity of the server if it has been configured with the appropriate access control list.

Appendix E. Changelog

Changed from ietf-07 to ietf-08:

- o Clarified matching rules for messages (#175)
- o Fixed a bug in Section 8.2.2 on Etags (#168)
- o Added an IP address spoofing threat analysis contribution (#167)
- o Re-focused the security section on raw public keys (#166)
- o Added an 4.06 error to Accept (#165)

Changed from ietf-06 to ietf-07:

- o application/link-format added to Media types registration (#160)
- o Moved content-type attribute to the document from link-format.
- o Added coaps scheme and DTLS-secured CoAP default port (#154)
- o Allowed 0-length Content-type options (#150)
- o Added congestion control recommendations (#153)
- o Improved text on PUT/POST response payloads (#149)
- o Added an Accept option for content-negotiation (#163)
- o Added If-Match and If-None-Match options (#155)
- o Improved Token Option explanation (#147)

- o Clarified mandatory to implement security (#156)
- o Added first come first server policy for 2-byte Media type codes (#161)
- o Clarify matching rules for messages and tokens (#151)
- o Changed OPTIONS and TRACE to always return 501 in HTTP-CoAP mapping (#164)

Changed from ietf-05 to ietf-06:

- o HTTP mapping section improved with the minimal protocol standard text for CoAP-HTTP and HTTP-CoAP forward proxying (#137).
- o Eradicated percent-encoding by including one Uri-Query Option per &-delimited argument in a query.
- o Allowed RST message in reply to a NON message with unexpected token (#134).
- o Cache Invalidation only happens upon successful responses (#135).
- o 50% jitter added to the initial retransmit timer (#142).
- o DTLS cipher suites aligned with ZigBee IP, DTLS clarified as default CoAP security mechanism (#138, #139)
- o Added a minimal reference to draft-kivinen-ipsecme-ikev2-minimal (#140).
- o Clarified the comparison of UTF-8s (#136).
- o Minimized the initial media type registry (#101).

Changed from ietf-04 to ietf-05:

- o Renamed Immediate into Piggy-backed and Deferred into Separate -- should finally end the confusion on what this is about.
- o GET requests now return a 2.05 (Content) response instead of 2.00 (OK) response (#104).
- o Added text to allow 2.02 (Deleted) responses in reply to POST requests (#105).
- o Improved message deduplication rules (#106).

- o Section added on message size implementation considerations (#103).
- o Clarification made on human readable error payloads (#109).
- o Definition of CoAP methods improved (#108).
- o Max-Age removed from requests (#107).
- o Clarified uniqueness of tokens (#112).
- o Location-Query Option added (#113).
- o ETag length set to 1-8 bytes (#123).
- o Clarified relation between elective/critical and option numbers (#110).
- o Defined when to update Version header field (#111).
- o URI scheme registration improved (#102).
- o Added review guidelines for new CoAP codes and numbers.

Changes from ietf-03 to ietf-04:

- o Major document reorganization (#51, #63, #71, #81).
- o Max-age length set to 0-4 bytes (#30).
- o Added variable unsigned integer definition (#31).
- o Clarification made on human readable error payloads (#50).
- o Definition of POST improved (#52).
- o Token length changed to 0-8 bytes (#53).
- o Section added on multiplexing CoAP, DTLS and STUN (#56).
- o Added cross-protocol attack considerations (#61).
- o Used new Immediate/Deferred response definitions (#73).
- o Improved request/response matching rules (#74).
- o Removed unnecessary media types and added recommendations for their use in M2M (#76).

- o Response codes changed to base 32 coding, new Y.XX naming (#77).
- o References updated as per AD review (#79).
- o IANA section completed (#80).
- o Proxy-Uri Option added to disambiguate between proxy and non-proxy requests (#82).
- o Added text on critical options in cached states (#83).
- o HTTP mapping sections improved (#88).
- o Added text on reverse proxies (#72).
- o Some security text on multicast added (#54).
- o Trust model text added to introduction (#58, #60).
- o AES-CCM vs. AES-CCB text added (#55).
- o Text added about device capabilities (#59).
- o DTLS section improvements (#87).
- o Caching semantics aligned with RFC2616 (#78).
- o Uri-Path Option split into multiple path segments.
- o MAX_RETRANSMIT changed to 4 to adjust for RESPONSE_TIME = 2.

Changes from ietf-02 to ietf-03:

- o Token Option and related use in asynchronous requests added (#25).
- o CoAP specific error codes added (#26).
- o Erroring out on unknown critical options changed to a MUST (#27).
- o Uri-Query Option added.
- o Terminology and definitions of URIs improved.
- o Security section completed (#22).

Changes from ietf-01 to ietf-02:

- o Sending an error on a critical option clarified (#18).
- o Clarification on behavior of PUT and idempotent operations (#19).
- o Use of Uri-Authority clarified along with server processing rules; Uri-Scheme Option removed (#20, #23).
- o Resource discovery section removed to a separate CoRE Link Format draft (#21).
- o Initial security section outline added.

Changes from ietf-00 to ietf-01:

- o New cleaner transaction message model and header (#5).
- o Removed subscription while being designed (#1).
- o Section 2 re-written (#3).
- o Text added about use of short URIs (#4).
- o Improved header option scheme (#5, #14).
- o Date option removed while being designed (#6).
- o New text for CoAP default port (#7).
- o Completed proxying section (#8).
- o Completed resource discovery section (#9).
- o Completed HTTP mapping section (#10).
- o Several new examples added (#11).
- o URI split into 3 options (#12).
- o MIME type defined for link-format (#13, #16).
- o New text on maximum message size (#15).
- o Location Option added.

Changes from shelby-01 to ietf-00:

- o Removed the TCP binding section, left open for the future.

- o Fixed a bug in the example.
- o Marked current Sub/Notify as (Experimental) while under WG discussion.
- o Fixed maximum datagram size to 1280 for both IPv4 and IPv6 (for CoAP-CoAP proxying to work).
- o Temporarily removed the Magic Byte header as TCP is no longer included as a binding.
- o Removed the Uri-code Option as different URI encoding schemes are being discussed.
- o Changed the rel= field to desc= for resource discovery.
- o Changed the maximum message size to 1024 bytes to allow for IP/UDP headers.
- o Made the URI slash optimization and method impotence MUSTs
- o Minor editing and bug fixing.

Changes from shelby-00 to shelby-01:

- o Unified the message header and added a notify message type.
- o Renamed methods with HTTP names and removed the NOTIFY method.
- o Added a number of options field to the header.
- o Combines the Option Type and Length into an 8-bit field.
- o Added the magic byte header.
- o Added new ETag Option.
- o Added new Date Option.
- o Added new Subscription Option.
- o Completed the HTTP Code - CoAP Code mapping table appendix.
- o Completed the Content-type Identifier appendix and tables.
- o Added more simplifications for URI support.

- o Initial subscription and discovery sections.
- o A Flag requirements simplified.

Authors' Addresses

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

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

Phone: +49-421-218-63905
Fax: +49-421-218-7000
Email: hartke@tzi.org

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

Phone: +49-421-218-63921
Fax: +49-421-218-7000
Email: cabo@tzi.org

Brian Frank
SkyFoundry
Richmond, VA
USA

Phone:
Email: brian@skyfoundry.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2012

K. Hartke
Universitaet Bremen TZI
Z. Shelby, Ed.
Sensinode
October 31, 2011

Observing Resources in CoAP
draft-ietf-core-observe-03

Abstract

CoAP is a RESTful application protocol for constrained nodes and networks. The state of a resource on a CoAP server can change over time. This document specifies a simple protocol extension for CoAP that gives clients the ability to observe such changes.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 3, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Background	3
1.2. Protocol Overview	3
1.3. Design Philosophy	5
1.4. Conformance Requirements	6
2. Options	6
2.1. Observe	6
2.2. Max-OFE	7
3. Client-side Requirements	7
3.1. Request	7
3.2. Notifications	8
3.3. Caching	8
3.4. Reordering	9
4. Server-side Requirements	10
4.1. Request	10
4.2. Notifications	11
4.3. Caching	11
4.4. Reordering	12
4.5. Retransmission	13
5. Intermediaries	13
6. Block-wise Transfers	14
7. Discovery	15
8. Security Considerations	15
9. IANA Considerations	15
10. Acknowledgements	16
11. References	16
11.1. Normative References	16
11.2. Informative References	16
Appendix A. Examples	18
A.1. Proxying	22
A.2. Block-wise Transfer	24
Appendix B. Modeling Resources to Tailor Notifications	25
Appendix C. Changelog	25
Authors' Addresses	27

1. Introduction

1.1. Background

CoAP [I-D.ietf-core-coap] is an Application Protocol for Constrained Nodes/Networks. It is intended to provide RESTful services [REST] not unlike HTTP [RFC2616] while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

The communication model of REST is that of a client exchanging resource representations with an origin server. The origin server is the definitive source for representations of the resources in its namespace. A client interested in a resource sends a request to the origin server that returns a response with a representation that is current at the time of the request.

This model does not work well when a client is interested in having a current representation of a resource over a period of time. Existing approaches from the HTTP world, such as repeated polling or long-polls [RFC6202], generate significant complexity and/or overhead and thus are less applicable in the constrained CoAP world.

The protocol specified in this document extends the CoAP core protocol with a mechanism to push resource representations from servers to interested clients, while still keeping the properties of REST.

Note that there is no intention for this mechanism to solve the full set of problems that the existing HTTP solutions solve, to replace publish/subscribe networks that solve a much more general problem [RFC5989], or to enable general two-way communication between clients and servers [I-D.ietf-hybi-thewebsocketprotocol].

1.2. Protocol Overview

The protocol is based on the well-known observer design pattern [GOF].

In this design pattern, components, called `_observers_`, register at a specific, known provider, called the `_subject_`, that they are interested in being notified whenever the subject undergoes a change in state. The subject is responsible for administering its list of registered observers. If multiple subjects are of interest, an observer must register separately for all of them. The pattern is typically used when a clean separation between related components is required, such as data storage and user interface.

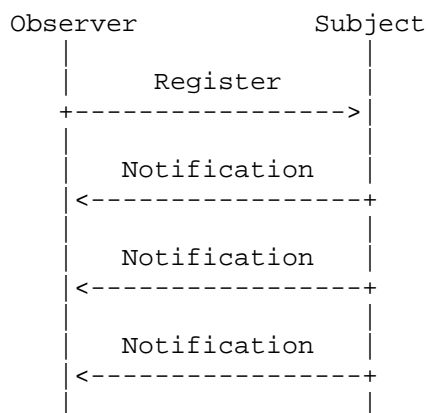


Figure 1: Observer Design Pattern

The observer design pattern is realized in CoAP as follows:

Subject: In the context of CoAP, the subject is a resource in the namespace of a CoAP server. The state of the resource can change over time, ranging from infrequent updates to continuous state transformations.

Observer: An observer is a CoAP client that is interested in the current state of the resource at any given time.

Registration: A client registers its interest by sending an extended GET request to the server. In addition to returning a representation of the target resource, this request causes the server to add the client to the list of observers of the resource.

Notification: Whenever the state of a resource changes, the server notifies each client registered as observer for the resource. Each notification is an additional CoAP response sent by the server in reply to the GET request and includes a complete representation of the new resource state.

Figure 2 shows an example of a CoAP client registering and receiving three notifications: the first upon registration and then two when the state of the resource changes. Registration request and notifications are identified by the presence of the Observe Option defined in this document. Notifications also echo the token specified by the client in the request, so the client can easily correlate them to the request.

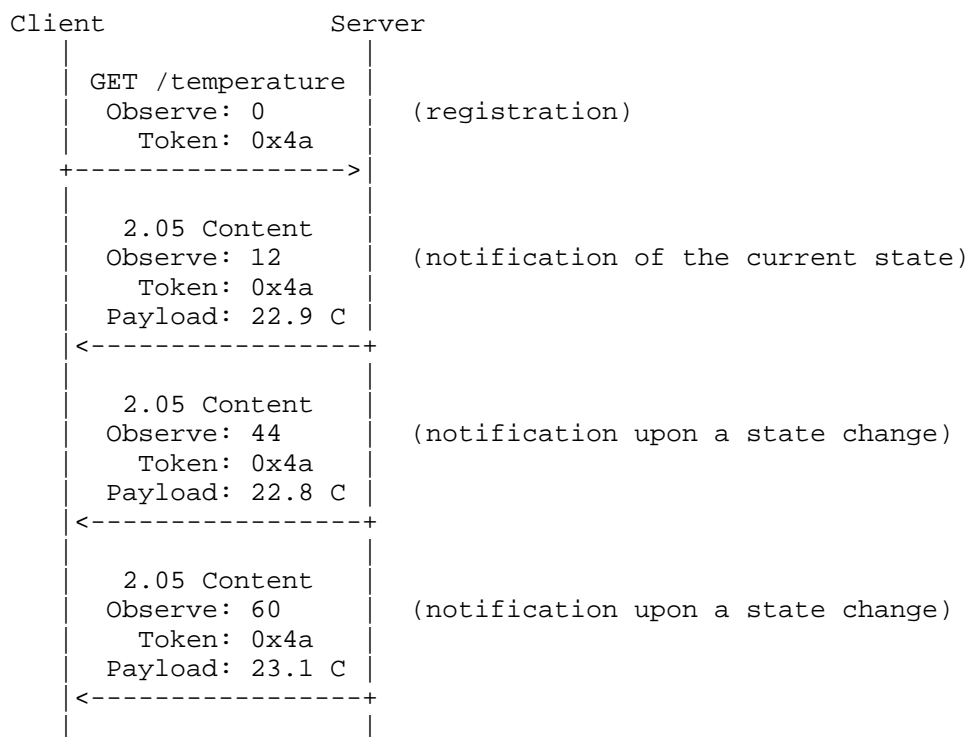


Figure 2: Observing a Resource in CoAP

The client is removed from the list of observers when it is no longer interested in the observed resource. The server can determine the client's continued interest from the client's acknowledgement of confirmable notifications. If a client wants to receive notifications after it has been removed from the list of observers, it needs to register again. The client can determine that it's still on the list of observers from the fact that it receives notifications. The protocol includes clear rules for what to do when a client does not receive a notification for some time, or a server does not receive acknowledgements.

1.3. Design Philosophy

The protocol builds on the architectural elements of REST: a server that is responsible for the state and representation of the resources in its namespace, a client that is responsible for keeping the application state, and the stateless exchange of resource representations. (A server needs to keep track of the observers though, similar to how HTTP servers need to keep track of the TCP connections from their clients.) The protocol enables high

scalability and efficiency through the support of caches and intermediaries that multiplex the interest of multiple clients in the same resource into a single association.

The server is the authority for determining under what conditions resources change their state and how often observers are notified. The protocol does not offer explicit means for setting up triggers, thresholds or other conditions; it is up to the server to expose observable resources that change their state in a way that is meaningful for the application. Resources can be parameterized to achieve similar effects though; see Appendix B for examples.

Since bandwidth is in short supply in constrained environments, servers must adapt the rate of notifications to each client. This implies that a client cannot rely on observing every single state a resource goes through. Instead, the protocol is designed on the principle of eventual consistency: it guarantees that if the resource does not undergo a new change in state, eventually all observers will have a current representation of the last resource state.

1.4. Conformance Requirements

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. Options

No.	C/E	Name	Format	Length	Default
10	Elective	Observe	uint	0-2 B	(none)
14	Elective	Max-OFE	uint	0-4 B	0

2.1. Observe

The Observe Option, when present, modifies the GET method so it does not only retrieve a representation of the current state of the resource identified by the request URI, but also requests the server to add the client to the list of observers of the resource. The exact semantics are defined in the sections below. The value of the option in a request MUST be zero on transmission and MUST be ignored on reception.

In a response, the Observe Option identifies the message as a

notification, which implies that the client has been added to the list of observers and that the server will notify the client of further changes to the resource state. The option's value is a sequence number that can be used for reordering detection (see Section 3.4 and Section 4.4). It is encoded as a variable-length unsigned integer as defined in Appendix A of RFC XXXX [I-D.ietf-core-coap].

Since the Observe Option is elective, a GET request that includes the Observe Option will automatically fall back to a normal GET request if the server does not support the protocol specified in this document.

The Observe Option MUST NOT occur more than once in a request or response.

2.2. Max-OFE

The freshness of a notification for caching purposes is determined by the Max-Age Option. However, a server may want to enable a cache to continue to optimistically use a cached representation even when the freshness indicated by the Max-Age Option has expired (see Section 3.3 and Section 4.3).

The time span for which this optimism is justified is under control of the server: it can use the Max-OFE Option to indicate a desired "optimistic freshness extension". This is also a promise by the server that it intends to send another notification within this time period. The exact semantics are defined in the sections below. The value of this option is a time span in seconds, measured from the time of expiry of Max-Age. The option is elective and defaults to zero (which means that no optimistic freshness extension is granted).

The Max-OFE Option MUST NOT occur more than once in a response.

3. Client-side Requirements

3.1. Request

A client can register its interest in a resource by issuing a GET request that includes an empty Observe Option. If the server returns a 2.xx response that includes an Observe Option as well, the server has added the client successfully to the list of observers of the target resource and the client will be notified of changes to the resource state for as long as the server can assume the client's interest.

3.2. Notifications

Notifications are additional responses sent by the server in reply to the GET request. Each notification includes an Observe Option with a sequence number (see Section 3.4), a Token Option that matches the token specified by the client in the GET request, and a payload in the same representation format as the initial response.

A notification can be confirmable or non-confirmable (i.e. sent in a confirmable or non-confirmable message). If a client does not recognize the token in a confirmable notification, it **MUST NOT** acknowledge the message and **SHOULD** reject it with a RST message. Otherwise, the client **MUST** acknowledge the message with an ACK message as usual.

An acknowledgement signals to the server that the client is alive and interested in receiving further notifications; if the server does not receive an acknowledgement in reply to a confirmable notification, it will assume that the client is no longer interested and will eventually remove it from the list of observers.

Notifications will have a 2.05 (Content) response code in most cases. They may also have a 2.03 (Valid) response code, if the client includes an ETag Option in its request (see Section 3.3). In the event that the state of an observed resource is changed in a way that would cause a normal GET request to return an error (for example, when the resource is deleted), the server will send a notification with an error response code (4.xx/5.xx) and empty the list of observers of the resource.

3.3. Caching

As notifications are just additional responses, notifications partake in caching as defined by Section 5.6 of RFC XXXX [I-D.ietf-core-coap]. Both the freshness model and the validation model are supported. The freshness model also serves as the model for the client to determine if it's still on the list of observers or if it needs to re-register its interest in the resource.

A client **MAY** store a notification like a response in its cache and use a stored response/notification that is fresh without contacting the origin server. A notification is considered fresh while its age is not greater than its Max-Age and if it has not been invalidated by a newer notification or as the result of a request.

Ideally, the server will provide a new notification exactly when the freshness of the latest notification expires. This may not always be possible though, due to network latency and/or resources that change

their state in unpredictable intervals. In this case, the client MAY optimistically use a stale (non-fresh) notification while the notification's age is not greater than Max-Age plus Max-OFE and the notification has not been invalidated.

If the client does not receive a notification before Max-Age plus Max-OFE expires, the client can assume it has been removed from the list of observers (e.g., due to a loss of server state). In this case, it needs to re-register by issuing a new GET request with an Observe Option.

To make sure it has a fresh representation and/or it is on the list of observers, a client MAY issue another GET request with an Observe Option at any time. The new GET request SHOULD specify a new token to avoid ambiguity. It is RECOMMENDED that the client does not issue the request before the Max-Age of the latest notification expires (i.e. while it still has a fresh notification).

When a client has one or more notifications stored, it can use the ETag Option in its request to give the server an opportunity to select a stored response to be used. The client MAY include an ETag Option for each stored response that is applicable. It needs to keep those responses in the cache until it is no longer interested in receiving notifications for the target resource or it issues a new GET request with a new set of entity-tags. When the observed resource changes its state to a representation identified by one of the ETag Options, the server can send a 2.03 (Valid) notification instead of a 2.05 (Content) notification.

3.4. Reordering

Messages that carry notifications can arrive in a different order than they were sent. Since the goal is eventual consistency (see Section 1.3), a client can safely skip a notification that arrives later than a newer notification. For this purpose, the server sets the value of the Observe Option in each notification to a sequence number.

A client MAY treat a notification as outdated (not fresh) under the following condition:

$$(V1 - V2) \% (2^{**16}) < (2^{**15}) \quad \text{and} \quad T2 < (T1 + (2^{**14}))$$

where V1 is the value of the Observe Option of the latest valid notification received, V2 the value of the Observe Option of the present notification, T1 a client-local timestamp of the latest valid notification received (in seconds), and T2 a client-local timestamp of the present notification.

Design Note: The first condition essentially verifies that $V2 > V1$ holds in 16-bit sequence number arithmetic [RFC1982]. The second condition checks that the time expired between the two incoming messages is not so large that the sequence number might have wrapped around and the first check is therefore invalid. (In other words, after about 2^{14} seconds elapse without any notification, the client does not need to check the sequence numbers in order to assume an incoming notification is new.) The constants of 2^{14} and 2^{15} are non-critical, as is the even speed or precision of the clock involved.

4. Server-side Requirements

4.1. Request

A GET request that includes an Observe Option requests the server not only to return a representation of the resource identified by the request URI, but also to add the client to the list of observers of the target resource. If no error occurs, the server **MUST** return a response with the representation of the current resource state and **MUST** notify the client of subsequent changes to the state as long as the client is on the list of observers.

A server that is unable or unwilling to add the client to the list of observers of the target resource **MAY** silently ignore the Observe Option and process the GET request as usual. The resulting response **MUST NOT** include an Observe Option, the absence of which signals to the client that it will not be notified of changes to the resource state and, e.g., needs to poll the resource instead.

If the client is already on the list of observers, the server **MUST NOT** add it a second time but **MUST** replace or update the existing entry. If the server receives a GET request that does not include an Observe Option, it **MUST** remove the client from the list of observers.

Two requests relate to the same list entry if both the request URI and the source of the requests match. The source of a request is determined by the security mode used (see Section 10 of RFC XXXX [I-D.ietf-core-coap]): With NoSec, it is determined by the source IP address and UDP port number. With other security modes, the source is also determined by the security context. Message IDs and Token Options **MUST NOT** be taken into account.

Any request with a method other than GET **MUST NOT** have a direct effect on a list of observers of a resource. However, such a request can have the indirect consequence of causing the server to send an error notification which affects the list of observers (e.g., when a

DELETE request is successful and an observed resource no longer exists).

4.2. Notifications

A client is notified of a resource state change by an additional response sent by the server in reply to the GET request. Each such notification response MUST include an Observe Option and MUST echo the token specified by the client in the GET request. If there are multiple clients, the order in which they are notified is not defined; the server is free to use any method to determine the order.

A notification SHOULD have a 2.05 (Content) or 2.03 (Valid) response code. However, in the event that the state of a resource changes in a way that would cause a normal GET request to return an error (for example, if the resource is deleted), the server SHOULD notify the client by sending a notification with an appropriate error response code (4.xx/5.xx) and MUST empty the list of observers of the resource.

The representation format/media type used in a notification MUST be the same format used in the initial response to the GET request. If the server is unable to continue sending notifications in this format, it SHOULD send a 5.00 (Internal Server Error) notification and MUST empty the list of observers of the resource.

A notification can be sent as a confirmable or a non-confirmable message. The message type used is typically application-dependent and MAY be determined by the server for each notification individually. For example, for resources that change in a somewhat predictable or regular fashion, notifications can be sent in non-confirmable messages; for resources that change infrequently, notifications can be sent in confirmable messages. The server can combine these two approaches depending on the frequency of state changes and the importance of individual notifications.

The acknowledgement of a confirmable notification implies the client's continued interest in being notified. If the client rejects a confirmable notification with a RST message, the server MUST remove the client from the list of observers.

4.3. Caching

The Max-Age Option of a notification SHOULD be set to a value that indicates when the server will send the next notification. For example, if the server sends a notification every 30 seconds, a Max-Age Option with value 30 should be included. The server MAY send a new notification before Max-Age ends. The server SHOULD also include

a Max-OFE Option so the client can continue to use a notification in case the next notification arrives a bit later due to network latency. If the client does not receive a new notification before Max-Age plus Max-OFE ends, it will assume that it was removed from the list of observers (e.g., due to a loss of server state) and may issue a new GET request to re-register its interest.

It may not always be possible to predict when the server will send the next notification, for example, when a resource does not change its state in regular intervals. In this case, the server SHOULD set Max-Age to a good approximation and Max-OFE to a time span for which the server is willing to keep the client in the list of observers.

Setting the values for Max-Age and Max-OFE is a trade-off between increased usage of bandwidth and the risk of stale information. Smaller values lead to more notifications and more GET requests, while greater values result in network or device failures being detected later and data becoming stale.

When the observed resource changes its state and the origin server is about to send a 2.05 (Content) notification, then, whenever that notification has an entity-tag in the set of entity-tags specified by the client, it MAY send a 2.03 (Valid) response with an appropriate ETag Option instead. The server MUST NOT assume that the recipient has any response stored other than those identified by the entity-tags in the most recent GET request.

4.4. Reordering

Because messages can get reordered, the client needs a way to determine if a notification arrived later than a newer notification. For this purpose, the server MUST set the value of the Observe Option in each notification to the 16 least-significant bits of a strictly increasing sequence number. The sequence number MAY start at any value. The server MUST NOT reuse the same option value with the same client, token and resource within approximately 2^{16} seconds (roughly 18.2 hours).

Implementation Note: A simple implementation that satisfies the requirements is to use a timestamp (in seconds) provided by the device's clock, or a 16-bit unsigned integer variable that is incremented every second and wraps around every 2^{16} seconds. It is not necessary that the clock reflects the correct local time or that it ticks exactly every second. Note that, on average, a server cannot send more than one notification per second per client, token and resource.

4.5. Retransmission

In CoAP, confirmable messages are retransmitted in exponentially increasing intervals for a certain number of attempts until they are acknowledged by the client. In the context of observing a resource, it is undesirable to continue transmitting the representation of a resource state when the state has changed in the meantime.

When a server is in the process of delivering a confirmable notification and is waiting for an acknowledgement, and it wants to notify the client of a state change using a new confirmable message, it **MUST** stop retransmitting the old notification and **SHOULD** attempt to deliver the new notification with the number of attempts remaining from the old notification. When the last attempt to retransmit a confirmable message with a notification for a resource times out, the server **SHOULD** remove the client from the list of observers and **MAY** additionally remove the client from the lists of observers of all resources in its namespace.

The server **SHOULD** use a number of retransmit attempts (`MAX_RETRANSMIT`) such that removing a client from the list of observers before `Max-Age` plus `Max-OFU` ends is avoided.

A server **MAY** choose to skip a notification if it knows that it will send another notification soon (e.g., when the state is changing frequently). Similarly, it **MAY** choose to send a notification more than once. For example, when state changes occur in bursts, the server can skip some notifications, send notifications in non-confirmable messages, and make sure that the client observes the latest state change after the burst by repeating the last notification in a confirmable message.

5. Intermediaries

A client may be interested in a resource in the namespace of an origin server that is reached through one or more CoAP-to-CoAP intermediaries. In this case, the client registers its interest with the first intermediary towards the origin server, acting as if it was communicating with the origin server itself as specified in Section 3. It is the task of this intermediary to provide the client with a current representation of the target resource and send notifications upon changes to the target resource state, much like an origin server as specified in Section 4.

To perform this task, the intermediary **SHOULD** make use of the protocol specified in this document, taking the role of the client and registering its own interest in the target resource with the next

hop. If the next hop does not return a response with an Observe Option, the intermediary MAY resort to polling the next hop, or MAY itself return a response without an Observe Option. Note that the communication between each pair of hops is independent, i.e. each hop in the server role MUST determine individually how many notifications to send, of which type, and so on, MUST generate its own values for the Observe Option, and MUST set the values of the Max-Age Option and Max-OFE Option according to the age of the local current representation.

Because a client (or an intermediary in the client role) can only be once in the list of observers of a resource at a server (or an intermediary in the server role) -- as it makes no sense to observe the same resource multiple times -- an intermediary MUST observe a resource only once, even if there are multiple clients for which it observes the resource.

Note that an intermediary is not required to have a client to observe a resource; an intermediary MAY observe a resource, for instance, just to keep its own cache up to date.

See Appendix A.1 for examples.

6. Block-wise Transfers

Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. CoAP provides a block-wise transfer mechanism to address this problem [I-D.ietf-core-block]. The following rules apply to the combination of block-wise transfers with notifications.

As with basic GET transfers, the client can indicate its desired block size in a Block2 Option in the GET request. If the server supports block-wise transfers, it SHOULD take note of the block size for all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the server receives a new GET request from the client).

When sending a 2.05 (Content) notification, the server always sends all blocks of the representation, suitably sequenced by its congestion control mechanism, even if only some of the blocks have changed with respect to a previous value. The server performs the block-wise transfer by making use of the Block2 Option in each block. When reassembling representations that are transmitted in multiple blocks, the client MUST NOT combine blocks carrying different Observe Option values, or blocks that have been received more than approximately 2**14 seconds apart.

See Appendix A.2 for an example.

7. Discovery

A web link [RFC5988] to a resource accessible by the CoAP protocol MAY indicate that the server encourages the observation of this resource by including the target attribute "obs". This is particularly useful in link-format documents [I-D.ietf-core-link-format].

This target attribute is used as a flag, and thus it has no value component -- a value given for the attribute MUST NOT be given for this version of the specification and MUST be ignored if present. The target attribute "obs" MUST NOT be given more than once for this version of the specification.

8. Security Considerations

The security considerations of RFC XXXX [I-D.ietf-core-coap] apply.

Note that the considerations about amplification attacks are somewhat amplified when observing resources. In NoSec mode, a server MUST therefore strictly limit the number of notifications that it sends between receiving acknowledgements that confirm the actual interest of the client in the data; i.e., any notifications sent in non-confirmable messages MUST be interspersed with confirmable messages. (An attacker may still spoof the acknowledgements if the confirmable messages are sufficiently predictable.)

As with any protocol that creates state, attackers may attempt to exhaust the resources that the server has available for maintaining the list of observers for each resource. Servers MAY want to access-control this creation of state. As degraded behavior, the server can always fall back to processing the request as a normal GET request (without an Observe Option) if it is unwilling or unable to add a client to the list of observers of a resource, including if system resources are exhausted or nearing exhaustion.

Intermediaries MUST be careful to ensure that notifications cannot be employed to create a loop. A simple way to break any loops is to employ caches for forwarding notifications in intermediaries.

9. IANA Considerations

The following entries are added to the CoAP Option Numbers registry:

Number	Name	Reference
10	Observe	[RFCXXXX]
14	Max-OFE	[RFCXXXX]

10. Acknowledgements

Carsten Bormann was an original author of this draft and is acknowledged for significant contribution to this document.

Thanks to Daniele Alessandrelli, Jari Arkko, Peter Bigot, Angelo Castellani, Gilbert Clark, Esko Dijk, Brian Frank, Salvatore Loreto and Charles Palmer for helpful comments and discussions that have shaped the document.

Klaus Hartke was funded by the Klaus Tschira Foundation.

11. References

11.1. Normative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-04 (work in progress), July 2011.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), July 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

11.2. Informative References

- [GOF] Gamma, E., Helm, R., Johnson, R., and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, USA, November 1994.
- [I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format",

draft-ietf-core-link-format-07 (work in progress),
July 2011.

- [I-D.ietf-hybi-thewebsocketprotocol]
Fette, I. and A. Melnikov, "The WebSocket protocol",
draft-ietf-hybi-thewebsocketprotocol-17 (work in
progress), September 2011.
- [REST] Fielding, R., "Architectural Styles and the Design of
Network-based Software Architectures", 2000, <<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>>.
- [RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982,
August 1996.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC5989] Roach, A., "A SIP Event Package for Subscribing to Changes
to an HTTP Resource", RFC 5989, October 2010.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins,
"Known Issues and Best Practices for the Use of Long
Polling and Streaming in Bidirectional HTTP", RFC 6202,
April 2011.

Appendix A. Examples

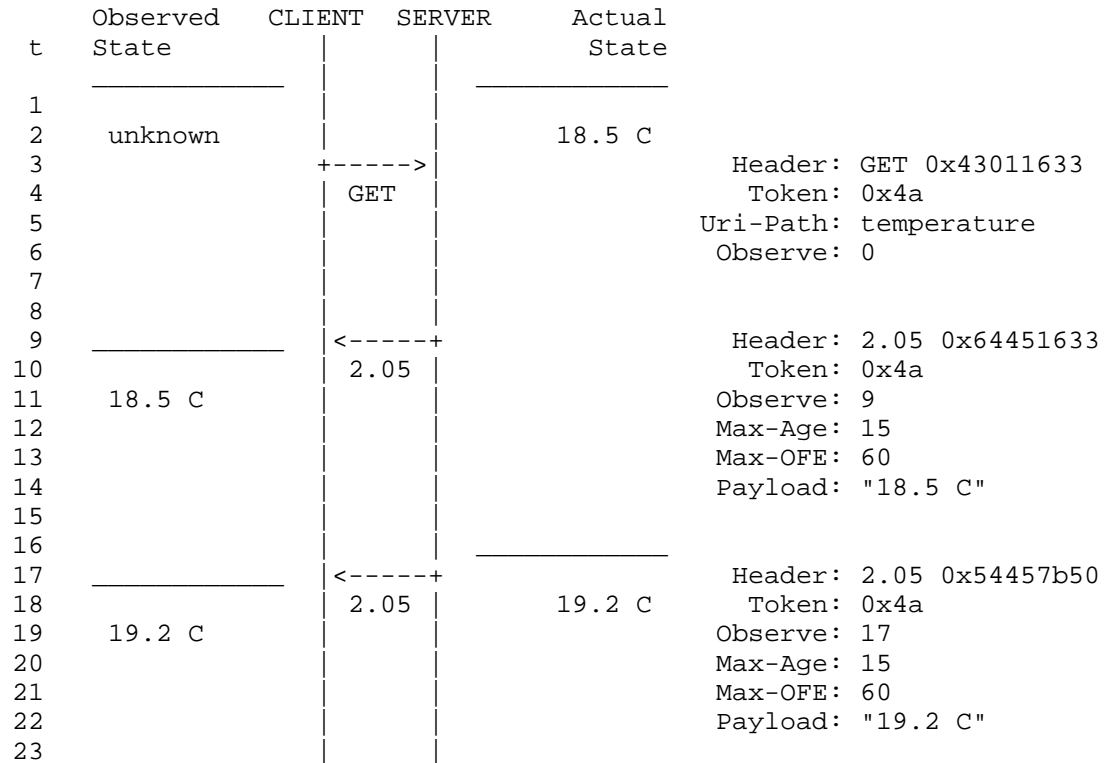


Figure 3: A client registers and receives a notification of the current state and upon a state change

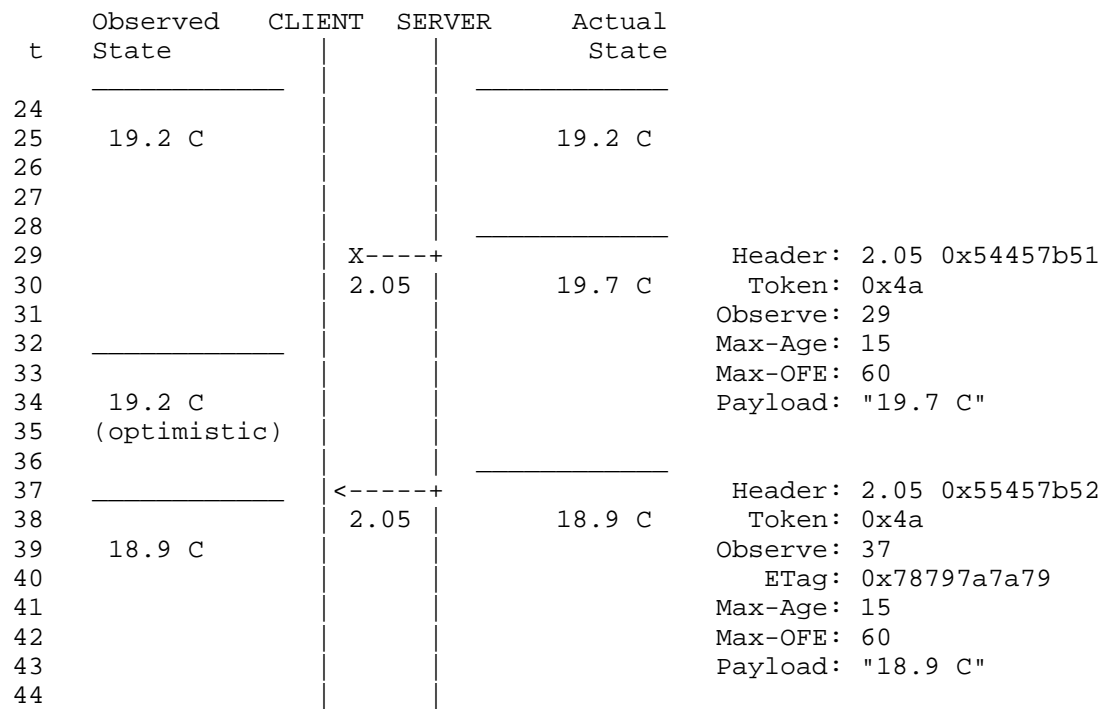


Figure 4: The client optimistically assumes that the state did not change after Max-Age ended

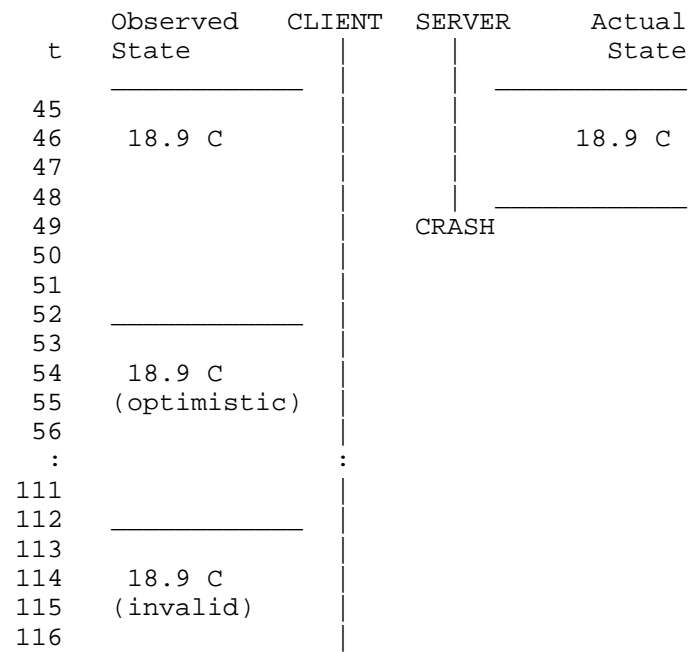


Figure 5: The server crashes and leaves the client with stale information

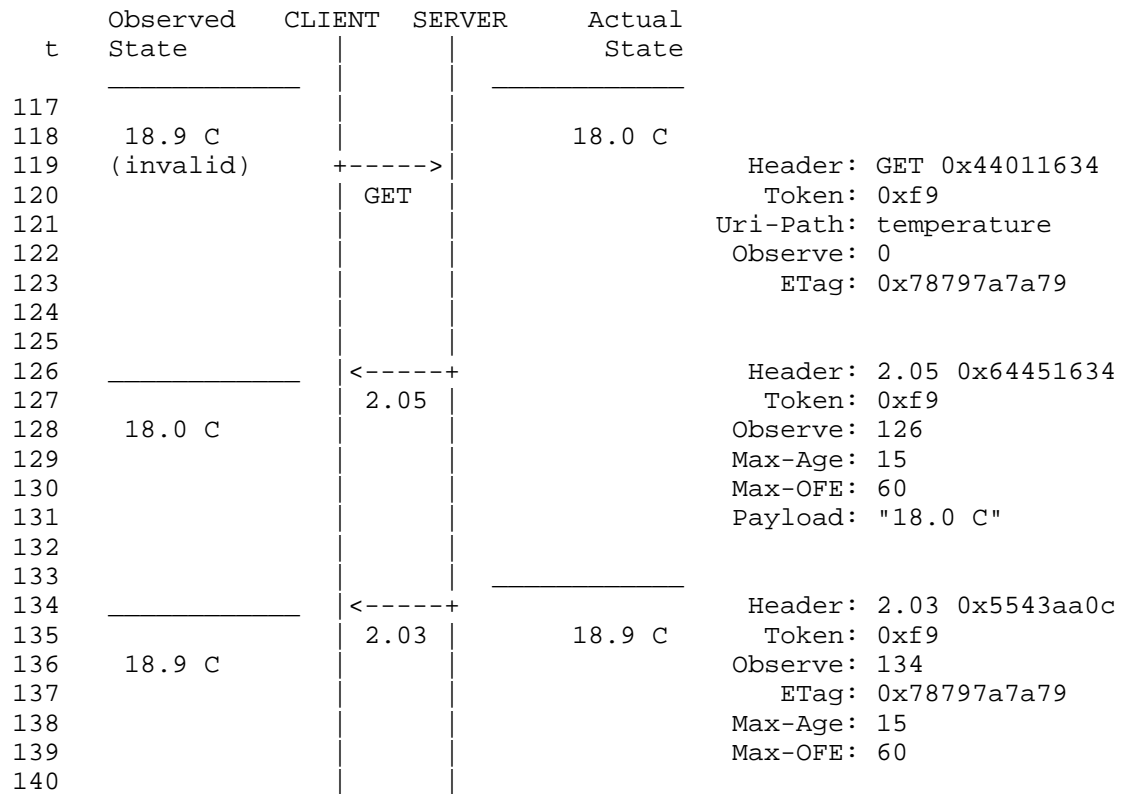


Figure 6: The client re-registers and gives the server the opportunity to select a stored response

A.1. Proxying

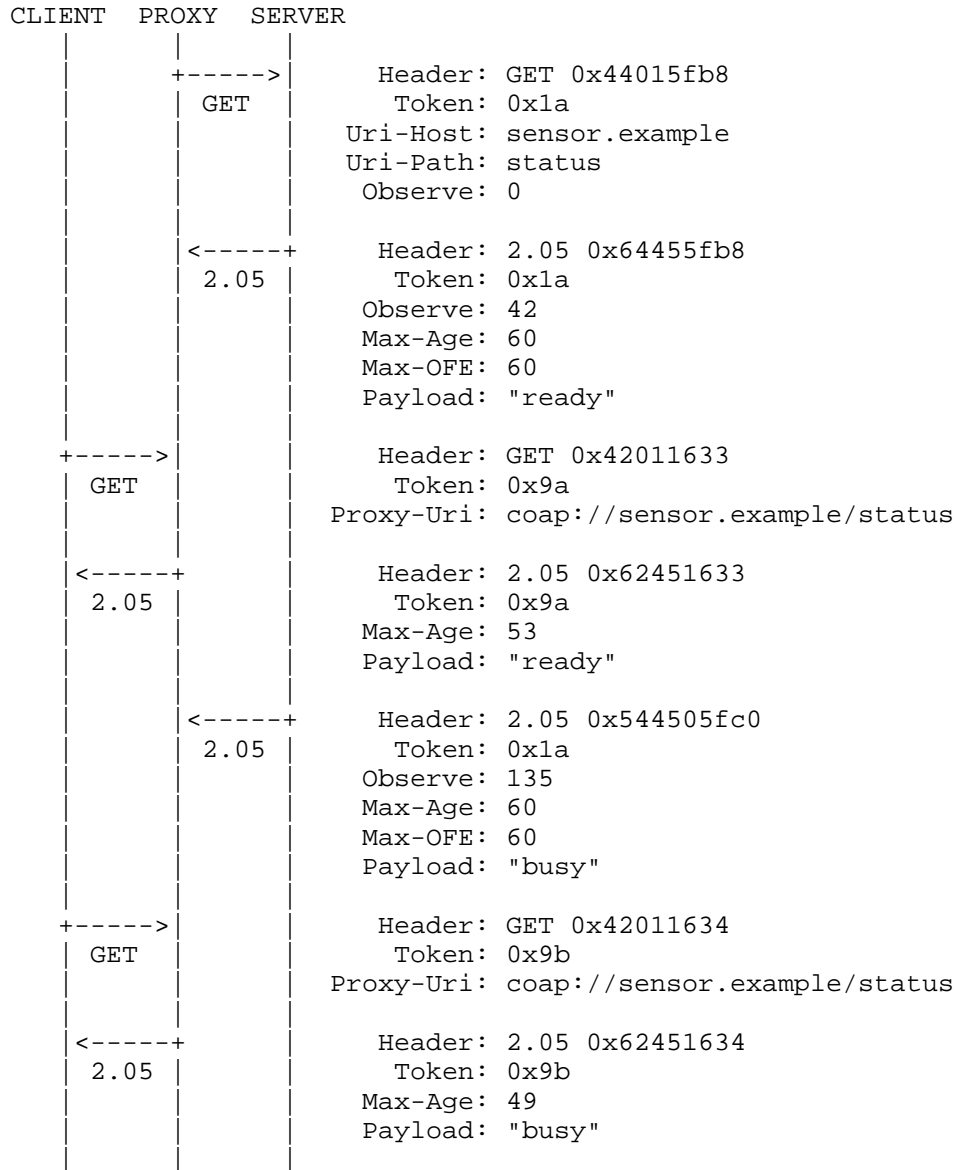


Figure 7: A proxy observes a resource to keep its cache up to date

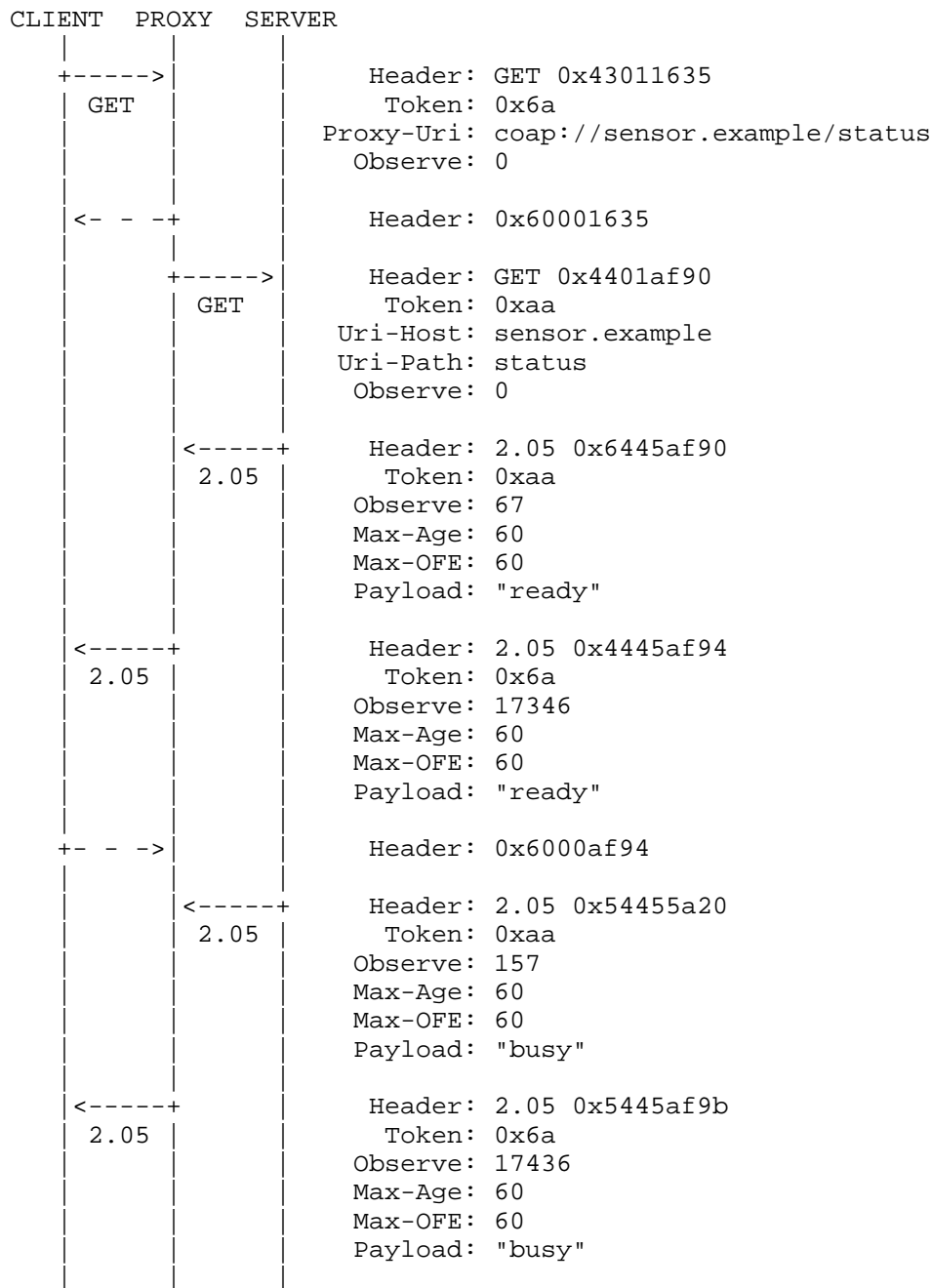


Figure 8: A client observes a resource through a proxy

A.2. Block-wise Transfer

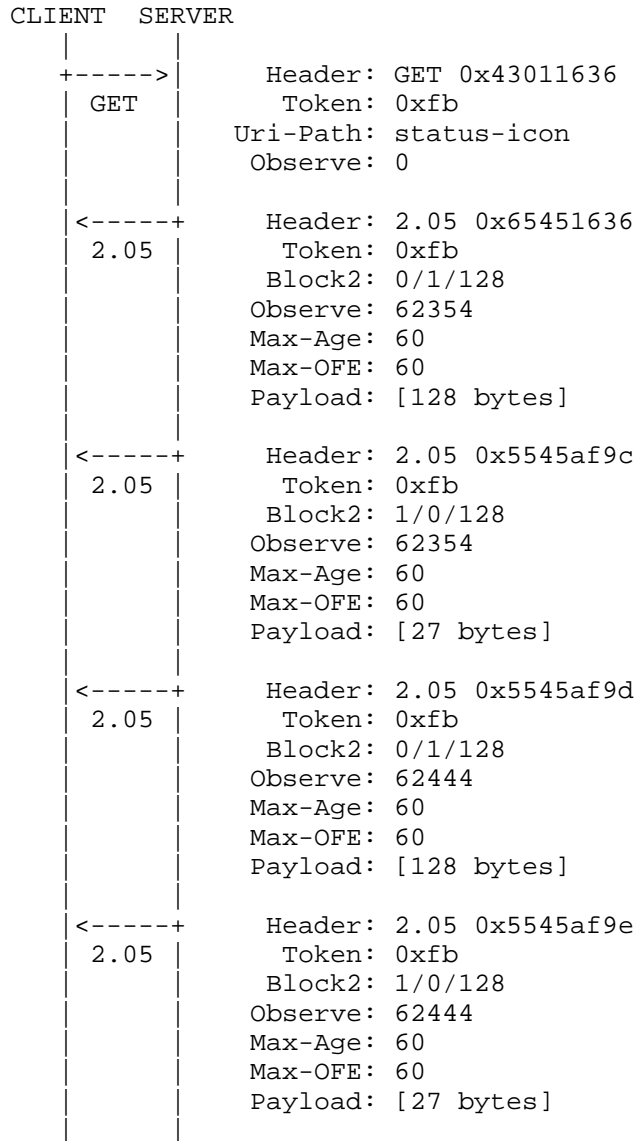


Figure 9: A server sends two notifications of two blocks each

Appendix B. Modeling Resources to Tailor Notifications

A server may want to provide notifications that respond to very specific conditions on some state. This is best done by modeling the resources that the server exposes according to these needs.

For example, for a CoAP server with an attached temperature sensor,

- o the server could, in the simplest form, expose a resource `<http://server/temperature>` that changes its state every second to the current temperature measured by the sensor;
- o the server could, however, also expose a resource `<http://server/temperature/felt>` that changes its state to "cold" when the temperature drops below a preconfigured threshold, and to "warm" when the temperature exceeds a second, higher threshold;
- o the server could expose a parameterized resource `<http://server/temperature/critical?above=45>` that changes its state to the current temperature if the temperature exceeds the specified value, and changes its state to "OK" when the temperature drops below; or
- o the server could expose a parameterized resource `<http://server/temperature?query=select+avg(temperature)+from+Sensor.window:time(30sec)>` that accepts expressions of arbitrary complexity and changes its state accordingly.

In any case, the client is notified about the current state of the resource whenever the state of the appropriately modeled resource changes. By designing resources that change their state on certain conditions, it is possible to notify the client only when these conditions occur instead of continuously supplying it with information it doesn't need. With parametrized resources, this is not limited to conditions defined by the server, but can be extended to arbitrarily complex conditions defined by the client. Thus, the server designer can choose exactly the right level of complexity for the application envisioned and devices used, and is not constrained to a "one size fits all" mechanism built into the protocol.

Appendix C. Changelog

Changes from ietf-02 to ietf-03:

- o Separated client-side and server-side requirements.

- o Fixed uncertainty if client is still in the list of observers by introducing a liveliness model based on Max-Age and a new option called Max-OFE (#174).
- o Simplified the text on message reordering (#129).
- o Clarified requirements for intermediaries.
- o Clarified the combination of block-wise transfers with notifications (#172).
- o Updated examples to show how the state observed by the client becomes eventually consistent with the actual state on the server.
- o Added examples for parameterization of observable resource.

Changes from ietf-01 to ietf-02:

- o Removed the requirement of periodic refreshing (#126).
- o The new "Observe" Option replaces the "Lifetime" Option.
- o Introduced a new mechanism to detect message reordering.
- o Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:

- o Changed terminology from "subscriptions" to "observation relationships" (#33).
- o Changed the name of the option to "Lifetime".
- o Clarified establishment of observation relationships.
- o Clarified that an observation is only identified by the URI of the observed resource and the identity of the client (#66).
- o Clarified rules for establishing observation relationships (#68).
- o Clarified conditions under which an observation relationship is terminated.
- o Added explanation on how clients can terminate an observation relationship before the lifetime ends (#34).
- o Clarified that the overriding objective for notifications is eventual consistency of the actual and the observed state (#67).

- o Specified how a server needs to deal with clients not acknowledging confirmable messages carrying notifications (#69).
- o Added a mechanism to detect message reordering (#35).
- o Added an explanation of how notifications can be cached, supporting both the freshness and the validation model (#39, #64).
- o Clarified that non-GET requests do not affect observation relationships, and that GET requests without "Lifetime" Option affecting relationships is by design (#65).
- o Described interaction with block-wise transfers (#36).
- o Added Resource Discovery section (#99).
- o Added IANA Considerations.
- o Added Security Considerations (#40).
- o Added examples (#38).

Authors' Addresses

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

Phone: +49-421-218-63905
Fax: +49-421-218-7000
Email: hartke@tzi.org

Zach Shelby (editor)
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2012

C. Jennings
Cisco
Z. Shelby
Sensinode
J. Arkko
Ericsson
Nov 1, 2011

Media Types for Sensor Markup Language (SENML)
draft-jennings-senml-07

Abstract

This specification defines media types for representing simple sensor measurements and device parameters in the Sensor Markup Language (SenML). Representations are defined in JavaScript Object Notation (JSON), eXtensible Markup Language (XML) and Efficient XML Interchange (EXI), which share the common SenML data model. A simple sensor, such as a temperature sensor, could use this media type in protocols such as HTTP or CoAP to transport the measurements of the sensor or to be configured.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 4, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Overview	3
2. Requirements and Design Goals	3
3. Terminology	4
4. Semantics	4
5. Associating Meta-data	7
6. JSON Representation (application/senml+json)	7
6.1. Examples	9
6.1.1. Single Datapoint	9
6.1.2. Multiple Datapoints	9
6.1.3. Multiple Measurements	10
6.1.4. Collection of Resources	10
7. XML Representation (application/senml+xml)	11
8. EXI Representation (application/senml+exi)	12
9. Usage Considerations	14
10. IANA Considerations	15
10.1. Units Registry	15
10.2. Media Type Registration	17
10.2.1. senml+json Media Type Registration	17
10.2.2. senml+xml Media Type Registration	18
10.2.3. senml+exi Media Type Registration	19
11. Security Considerations	20
12. Privacy Considerations	20
13. Acknowledgement	20
14. References	20
14.1. Normative References	20
14.2. Informative References	21
Authors' Addresses	22

1. Overview

Connecting sensors to the internet is not new, and there have been many protocols designed to facilitate it. This specification defines new media types for carrying simple sensor information in a protocol such as HTTP or CoAP[I-D.ietf-core-coap] called the Sensor Markup Language (SenML). This format was designed so that processors with very limited capabilities could easily encode a sensor measurement into the media type, while at the same time a server parsing the data could relatively efficiently collect a large number of sensor measurements. There are many types of more complex measurements and measurements that this media type would not be suitable for. A decision was made not to carry most of the meta data about the sensor in this media type to help reduce the size of the data and improve efficiency in decoding. Instead meta-data about a sensor resource can be described out-of-band using the CoRE Link Format [I-D.ietf-core-link-format]. The markup language can be used for a variety of data flow models, most notably data feeds pushed from a sensor to a collector, and the web resource model where the sensor is requested as a resource representation (GET /sensor/temperature).

SenML is defined by a data model for measurements and simple meta-data about measurements and devices. The data is structured as a single object (with attributes) that contains an array of entries. Each entry is an object that has attributes such as a unique identifier for the sensor, the time the measurement was made, and the current value. Serializations for this data model are defined for JSON [RFC4627], XML and Efficient XML Interchange (EXI).

For example, the following shows a measurement from a temperature gauge encoded in the JSON syntax.

```
{"e":[{"n": "urn:dev:ow:10e2073a01080063", "v":23.5, "u":"degC" }]}
```

In the example above, the array in the object has a single measurement for a sensor named "urn:dev:ow:10e2073a01080063" with a temperature of 23.5 degrees Celsius.

2. Requirements and Design Goals

The design goal is to be able to send simple sensor measurements in small packets on mesh networks from large numbers of constrained devices. Keeping the total size under 80 bytes makes this easy to use on a wireless mesh network. It is always difficult to define what small code is, but there is a desire to be able to implement this in roughly 1 KB of flash on a 8 bit microprocessor. Experience with Google power meter and large scale deployments has indicated that the solution needs to support allowing multiple measurements to

be batched into a single HTTP or CoAP request. This "batch" upload capability allows the server side to efficiently support a large number of devices. It also conveniently supports batch transfers from proxies and storage devices, even in situations where the sensor itself sends just a single data item at a time. The multiple measurements could be from multiple related sensors or from the same sensor but at different times.

3. Terminology

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

4. Semantics

Each representation carries a single SenML object that represents a set of measurements and/or parameters. This object contains several optional attributes described below and a mandatory array of one or more entries.

Base Name

This is a string that is prepended to the names found in the entries. This attribute is optional.

Base Time

A base time that is added to the time found in an entry. This attribute is optional.

Base Units

A base unit that is assumed for all entries, unless otherwise indicated. This attribute is optional. Acceptable values are specified in Section 10.1.

Version

Version number of media type format. This attribute is optional positive integer and defaults to 1 if not present.

Measurement or Parameter Entries

Array of values for sensor measurements or other generic parameters (such as configuration parameters). If present there must be at least one entry in the array.

Each array entry contains several attributes, some of which are optional and some of which are mandatory.

Name

Name of the sensor or parameter. When appended to the Base Name attribute, this must result in a globally unique identifier for the resource. The name is optional, if the Base Name is present. If the name is missing Base Name must uniquely identify the resource. This can be used to represent a large array of measurements from the same sensor without having to repeat its identifier on every measurement.

Units

Units for a measurement value. Optional, if Base Unit is present or if not required for a parameter. Acceptable values are specified in Section 10.1.

Value

Value of the entry. Optional if a Sum value is present, otherwise required. Values are represented using three basic data types, Floating point numbers ("v" field for "Value"), Booleans ("bv" for "Boolean Value") and Strings ("sv" for "String Value"). Exactly one of these three fields MUST appear.

Sum

Integrated sum of the values over time. Optional. This attribute is in the units specified in the Unit value multiplied by seconds.

Time

Time when value was recorded. Optional.

Update Time

Update time. A time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement. This can be used to detect the failure of sensors or

communications path from the sensor. Optional.

The SenML format can be extended with further custom attributes placed in the base object, or in an entry. Extensions in the base object pertain to all entries, whereas extensions in an entry object only pertain to that.

Systems reading one of the objects MUST check for the Version attribute. If this value is a version number larger than the version which the system understands, the system SHOULD NOT use this object. This allows the version number to indicate that the object contains mandatory to understand attributes. New version numbers can only be defined in RFC which updates this specification or its successors.

The Name value is concatenated to the Base Name value to get the name of the sensor. The resulting name needs to uniquely identify and differentiate the sensor from all others. If the object is a representation resulting from the request of a URI [RFC3986], then in the absence of the Base Name attribute, this URI is used as the default value of Base Name. Thus in this case the Name field needs to be unique for that URI, for example an index or subresource name of sensors handled by the URI.

Alternatively, for objects not related to a URI, a unique name is required. In any case, it is RECOMMENDED that the full names are represented as URIs or URNs [RFC2141]. One way to create a unique name is to include a EUI-48 or EUI-64 identifier (A MAC address) or some other bit string that is guaranteed uniqueness (such as a 1-wire address) that is assigned to the device. Some of the examples in this draft use the device URN type as specified in [I-D.arkko-core-dev-urn]. UUIDs [RFC4122] are another way to generate a unique name.

The resulting concatenated name MUST consist only of characters out of the set "A" to "Z", "a" to "z", "0" to "9", "-", ":", ".", or "_" and it MUST start with a character out of the set "A" to "Z", "a" to "z", or "0" to "9". This restricted character set was chosen so that these names can be directly used as in other types of URI including segments of an HTTP path with no special encoding. [RFC5952] contains advice on encoding an IPv6 address in a name.

If either the Base Time or Time value is missing, the missing attribute is considered to have a value of zero. The Base Time and Time values are added together to get the time of measurement. A time of zero indicates that the sensor does not know the absolute time and the measurement was made roughly "now". A negative value is used to indicate seconds in the past from roughly "now". A positive

value is used to indicate the number of seconds, excluding leap seconds, since the start of the year 1970 in UTC .

Representing the statistical characteristics of measurements can be very complex. Future specification may add new attributes to provide better information about the statistical properties of the measurement.

5. Associating Meta-data

SenML is designed to carry the minimum dynamic information about measurements, and for efficiency reasons does not carry more static meta-data about the device, object or sensors. Instead, it is assumed that this meta-data is carried out of band. For web resources using SenML representations, this meta-data can be made available using the CoRE Link Format [I-D.ietf-core-link-format].

The CoRE Link Format provides a simple way to describe Web Links, and in particular allows a web server to describe resources it is hosting. The list of links that a web server has available, can be discovered by retrieving the /.well-known/core resource, which returns the list of links in the CoRE Link Format. Each link may contain attributes, for example title, resource type, interface description and content-type.

The most obvious use of this link format is to describe that a resource is available in a SenML format in the first place. The relevant media type indicator is included in the Content-Type (ct=) attribute.

Further semantics about a resource can be included in the Resource Type and Interface Description attributes. The Resource Type (rt=) attribute is meant to give a semantic meaning to that resource. For example rt="OutdoorTemperature" would indicate static semantic meaning in addition to the unit information included in SenML. The Interface Description (if=) attribute is used to describe the REST interface of a resource, and may include e.g. a reference to a WADL description [WADL].

6. JSON Representation (application/senml+json)

Root variables:

	SenML	JSON	Type
Base Name	bn		String
Base Time	bt		Number
Base Units	bu		Number
Version	ver		Number
Measurement or Parameters	e		Array

Measurement or Parameter Entries:

	SenML	JSON	Notes
Name	n		String
Units	u		String
Value	v		Floating point
String Value	sv		String
Boolean Value	bv		Boolean
Value Sum	s		Floating point
Time	t		Number
Update Time	ut		Number

All of the data is UTF-8, but since this is for machine to machine communications on constrained systems, only characters with code points between U+0001 and U+007F are allowed which corresponds to the ASCII[RFC0020] subset of UTF-8.

The root contents MUST consist of exactly one JSON object as specified by [RFC4627]. This object MAY contain a "bn" attribute with a value of type string. This object MAY contain a "bt" attribute with a value of type number. The object MAY contain a "bu" attribute with a value of type string. The object MAY contain a "ver" attribute with a value of type number. The object MAY contain other attribute value pairs, and the object MUST contain exactly one "e" attribute with a value of type array. The array MUST have one or more measurement or parameter objects.

Inside each measurement or parameter object the "n", "u", and "sv" attributes are of type string, the "t" and "ut" attributes are of type number, the "bv" attribute is of type boolean, and the "v" and "s" attributes are of type floating point. All the attributes are optional, but as specified in Section 4, one of the "v", "sv", or "bv" attributes MUST appear unless the "s" attribute is also present. The "v", and "sv", and "bv" attributes MUST NOT appear together.

Systems receiving measurements MUST be able to process the range of floating point numbers that are representable as an IEEE double-precision floating-point numbers [IEEE.754.1985]. The number of significant digits in any measurement is not relevant, so a reading of 1.1 has exactly the same semantic meaning as 1.10. If the value has an exponent, the "e" MUST be in lower case. The mantissa SHOULD be less than 19 characters long and the exponent SHOULD be less than 5 characters long. This allows time values to have better than micro second precision over the next 100 years.

6.1. Examples

6.1.1. Single Datapoint

The following shows a temperature reading taken approximately "now" by a 1-wire sensor device that was assigned the unique 1-wire address of 10e2073a01080063:

```
{"m":[{"n": "urn:dev:ow:10e2073a01080063", "v":23.5 }]}
```

6.1.2. Multiple Datapoints

The following example shows voltage and current now, i.e., at an unspecified time. The device has an EUI-64 MAC address of 0024beffffe804ff1.

```
{"e":[
  { "n": "voltage", "t": 0, "u": "V", "v": 120.1 },
  { "n": "current", "t": 0, "u": "A", "v": 1.2 }],
  "bn": "urn:dev:mac:0024beffffe804ff1/"
}
```

The next example is similar to the above one, but shows current at Tue Jun 8 18:01:16 UTC 2010 and at each second for the previous 5 seconds.

```
{"e":[
  { "n": "voltage", "u": "V", "v": 120.1 },
  { "n": "current", "t": -5, "v": 1.2 },
  { "n": "current", "t": -4, "v": 1.30 },
  { "n": "current", "t": -3, "v": 0.14e1 },
  { "n": "current", "t": -2, "v": 1.5 },
  { "n": "current", "t": -1, "v": 1.6 },
  { "n": "current", "t": 0, "v": 1.7 }],
  "bn": "urn:dev:mac:0024beffffe804ff1/",
  "bt": 1276020076,
  "ver": 1,
  "bu": "A"
}
```

```
}

```

6.1.3. Multiple Measurements

The following example shows humidity measurements from a mobile device with an IPv6 address 2001:db8::1, starting at Mon Oct 31 13:24:24 UTC 2011. The device also provide position data, which is provided in the same measurement or parameter array as separate entries. Note time is used to for correlating data that belongs together, e.g., a measurement and a parameter associated with it. Finally, the device also reports extra data about its battery status at a separate time.

```
{ "e": [
  { "v": 20.0, "t": 0 },
  { "sv": "E 24' 30.621", "u": "lon", "t": 0 },
  { "sv": "N 60' 7.965", "u": "lat", "t": 0 },
  { "v": 20.3, "t": 60 },
  { "sv": "E 24' 30.622", "u": "lon", "t": 60 },
  { "sv": "N 60' 7.965", "u": "lat", "t": 60 },
  { "v": 20.7, "t": 120 },
  { "sv": "E 24' 30.623", "u": "lon", "t": 120 },
  { "sv": "N 60' 7.966", "u": "lat", "t": 120 },
  { "v": 98.0, "u": "%EL", "t": 150 },
  { "v": 21.2, "t": 180 },
  { "sv": "E 24' 30.628", "u": "lon", "t": 180 },
  { "sv": "N 60' 7.967", "u": "lat", "t": 180 }],
  "bn": "http://[2001:db8::1]",
  "bt": 1320067464,
  "bu": "%RH"
}
```

6.1.4. Collection of Resources

The following example shows how to query one device that can provide multiple measurements. The example assumes that a client has fetched information from a device at 2001:db8::2 by performing a GET operation on `http://[2001:db8::2]` at Mon Oct 31 16:27:09 UTC 2011, and has gotten two separate values as a result, a temperature and humidity measurement.

```
{ "e": [
  { "n": "temperature", "v": 27.2, "u": "degC" },
  { "n": "humidity", "v": 80, "u": "%RH" }],
  "bn": "http://[2001:db8::2]/",
  "bt": 1320078429,
  "ver": 1
}
```


7. XML Representation (application/senml+xml)

A SenML object can also be represented in XML format as defined in this section. The following example shows an XML example for the same sensor measurement as in Section 6.1.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<senml xmlns="urn:ietf:params:xml:ns:senml"
  bn="urn:dev:mac:0024beffffe804ff1/"
  bt="1276020076"
  ver="1" bu="A">

  <e n="voltage" u="V" v="120.1" />

  <e n="current" t="-5" v="1.2" />

  <e n="current" t="-4" v="1.30" />

  <e n="current" t="-3" v="0.14e1" />

  <e n="current" t="-2" v="1.5" />

  <e n="current" t="-1" v="1.6" />

  <e n="current" t="0" v="1.7" />

</senml>
```

The RelaxNG schema for the XML is:

```
default namespace = "urn:ietf:params:xml:ns:senml"
namespace rng = "http://relaxng.org/ns/structure/1.0"
```

```
e = element e {
  attribute n { xsd:string }?,
  attribute u { xsd:string }?,
  attribute v { xsd:float }?,
  attribute sv { xsd:string }?,
  attribute bv { xsd:boolean }?,
  attribute s { xsd:decimal }?,
  attribute t { xsd:integer }?,
  attribute ut { xsd:integer }?,
  p*
}

senml =
  element senml {
    attribute bn { xsd:string }?,
    attribute bt { xsd:integer }?,
    attribute bu { xsd:string }?,
    attribute ver { xsd:integer }?,
    e*
  }

start = senml
```

8. EXI Representation (application/senml+exi)

For efficient transmission of SenML over e.g. a constrained network, Efficient XML Interchange (EXI) can be used. This encodes the XML Schema structure of SenML into binary tags and values rather than ASCII text. An EXI representation of SenML SHOULD be made using the strict schema-mode of EXI. This mode however does not allow tag extensions to the schema, and therefore any extensions will be lost in the encoding. For uses where extensions need to be preserved in EXI, the non-strict schema mode of EXI MAY be used.

The following XSD Schema is generated from the RelaxNG and used for strict schema guided EXI processing. (TODO: define a better schema that include the common strings. I'd like to have something where the stream ended up defined in a way that it was trivial to encode sensor measurement into EXI in a small amount of C code.)

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="urn:ietf:params:xml:ns:senml"
  xmlns:ns1="urn:ietf:params:xml:ns:senml">

  <xs:element name="e">
    <xs:complexType>
      <xs:attribute name="n" type="xs:string"/>
      <xs:attribute name="u" type="xs:string"/>
      <xs:attribute name="v" type="xs:float"/>
      <xs:attribute name="sv" type="xs:string"/>
      <xs:attribute name="bv" type="xs:boolean"/>
      <xs:attribute name="s" type="xs:decimal"/>
      <xs:attribute name="t" type="xs:integer"/>
      <xs:attribute name="ut" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="senml">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="ns1:e"/>
      </xs:sequence>
      <xs:attribute name="bn" type="xs:string"/>
      <xs:attribute name="bt" type="xs:integer"/>
      <xs:attribute name="bu" type="xs:string"/>
      <xs:attribute name="ver" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

The following shows a hexdump of the EXI produced from encoding the following XML example. Note that while this example is similar to the the first example in Section 6.1.2 in JSON format, it has been simplified by omitting the URNs and units.

```

<?xml version="1.0" encoding="UTF-8"?>
<senml xmlns="urn:ietf:params:xml:ns:senml">
  <e n="voltage" t="0" v="120.1" />
  <e n="current" t="0" v="1.2" />
</senml>

```

Result:

```

00000000  90 60 84 bb 37 b6 3a 30 b3 b2 a0 02 58 84 c0 00
00000020  84 b1 ba b9 39 32 b7 3a 20 02 06 40 08

```

9. Usage Considerations

The measurements support sending both the current value of a sensor as well as the an integrated sum. For many types of measurements, the sum is more useful than the current value. For example, an electrical meter that measures the energy a given computer uses will typically want to measure the cumulative amount of energy used. This is less prone to error than reporting the power each second and trying to have something on the server side sum together all the power measurements. If the network between the sensor and the meter goes down over some period of time, when it comes back up, the cumulative sum helps reflect what happened while the network was down. A meter like this would typically report a measurement with the units set to watts, but it would put the sum of energy used in the "s" attribute of the measurement. It might optionally include the current power in the "v" attribute.

While the benefit of using the integrated sum is fairly clear for measurements like power and energy, it is less obvious for something like temperature. Reporting the sum of the temperature makes it easy to compute averages even when the individual temperature values are not reported frequently enough to compute accurate averages. Implementors are encouraged to report the cumulative sum as well as the raw value of a given sensor.

Applications that use the cumulative sum values need to understand they are very loosely defined by this specification, and depending on the particular sensor implementation may behave in unexpected ways. Applications should be able to deal with the following issues:

1. Many sensors will allow the cumulative sums to "wrap" back to zero after the value gets sufficiently large.
2. Some sensors will reset the cumulative sum back to zero when the device is reset, loses power, or is replaced with a different sensor.
3. Applications cannot make assumptions about when the device started accumulating values into the sum.

Typically applications can make some assumptions about specific sensors that will allow them to deal with these problems. A common assumption is that for sensors whose measurement values are always positive, the sum should never get smaller; so if the sum does get smaller, the application will know that one of the situations listed above has happened.

10. IANA Considerations

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

10.1. Units Registry

IANA will create a registry of unit symbols. The primary purpose of this registry is to make sure that symbols uniquely map to give type of measurement. Definitions for many of these units can be found in [NIST822] and [BIPM].

Symbol	Description	Reference
m	meter	RFC-AAAA
kg	kilogram	RFC-AAAA
s	second	RFC-AAAA
A	ampere	RFC-AAAA
K	kelvin	RFC-AAAA
cd	candela	RFC-AAAA
mol	mole	RFC-AAAA
Hz	hertz	RFC-AAAA
rad	radian	RFC-AAAA
sr	steradian	RFC-AAAA
N	newton	RFC-AAAA
Pa	pascal	RFC-AAAA
J	joule	RFC-AAAA
W	watt	RFC-AAAA
C	coulomb	RFC-AAAA
V	volt	RFC-AAAA
F	farad	RFC-AAAA
Ohm	ohm	RFC-AAAA
S	siemens	RFC-AAAA
Wb	weber	RFC-AAAA
T	tesla	RFC-AAAA
H	henry	RFC-AAAA
degC	degrees Celsius	RFC-AAAA
lm	lumen	RFC-AAAA
lx	lux	RFC-AAAA
Bq	becquerel	RFC-AAAA
Gy	gray	RFC-AAAA
Sv	sievert	RFC-AAAA
kat	katal	RFC-AAAA
pH	pH acidity	RFC-AAAA
%	Value of a switch. A value of 0.0 indicates the switch is off while 100.0 indicates on.	RFC-AAAA
count	counter value	RFC-AAAA

%RH	Relative Humidity	RFC-AAAA
m2	area	RFC-AAAA
l	volume in liters	RFC-AAAA
m/s	velocity	RFC-AAAA
m/s2	acceleration	RFC-AAAA
l/s	flow rate in liters per second	RFC-AAAA
W/m2	irradiance	RFC-AAAA
cd/m2	luminance	RFC-AAAA
Bspl	bel sound pressure level	RFC-AAAA
bit/s	bits per second	RFC-AAAA
lat	degrees latitude. Assumed to be in WGS84 unless another reference frame is known for the sensor.	RFC-AAAA
lon	degrees longitude. Assumed to be in WGS84 unless another reference frame is known for the sensor.	RFC-AAAA
%EL	remaining battery energy level in percents	RFC-AAAA

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. Experts should exercise their own good judgment but need to consider the following guidelines:

1. There needs to be a real and compelling use for any new unit to be added.
2. Units should define the semantic information and be chosen carefully. Implementors need to remember that the same word may be used in different real-life contexts. For example, degrees when measuring latitude have no semantic relation to degrees when measuring temperature; thus two different units are needed.
3. These measurements are produced by computers for consumption by computers. The principle is that conversion has to be easily be done when both reading and writing the media type. The value of a single canonical representation outweighs the convenience of easy human representations or loss of precision in a conversion.
4. Use of SI prefixes such as "k" before the unit is not allowed. Instead one can represent the value using scientific notation such a 1.2e3.
5. For a given type of measurement, there will only be one unit type defined. So for length, meters are defined and other lengths such as mile, foot, light year are not allowed. For most cases, the SI unit is preferred.
6. Symbol names that could be easily confused with existing common units or units combined with prefixes should be avoided. For example, selecting a unit name of "mph" to indicate something that had nothing to do with velocity would be a bad choice, as "mph" is commonly used to mean miles per hour.

7. The following should not be used because they are common SI prefixes: Y, Z, E, P, T, G, M, k, h, da, d, c, n, u, p, f, a, z, y, Ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi.
8. The following units should not be used as they are commonly used to represent other measurements: Ky, Gal, dyn, etg, P, St, Mx, G, Oe, Gb, sb, Lmb, ph, Ci, R, RAD, REM, gal, bbl, qt, degF, Cal, BTU, HP, pH, B/s, psi, Torr, atm, at, bar, kWh.
9. The unit names are case sensitive and the correct case needs to be used, but symbols that differ only in case should not be allocated.
10. A number after a unit typically indicates the previous unit raised to that power, and the / indicates that the units that follow are the reciprocal. A unit should have only one / in the name.

10.2. Media Type Registration

The following registrations are done following the procedure specified in [RFC4288] and [RFC3023].

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

10.2.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC4627]. Specifically, only the ASCII[RFC0020] subset of the UTF-8 characters are allowed. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such as the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is

adequate.

Interoperability considerations: Applications should ignore any JSON key value pairs that they do not understand. This allows backwards compatibility extensions to this specification. The "ver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <c.jennings@ieee.org>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <c.jennings@ieee.org>

Change controller: IESG

10.2.2. senml+xml Media Type Registration

Type name: application

Subtype name: senml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <c.jennings@ieee.org>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <c.jennings@ieee.org>

Change controller: IESG

10.2.3. senml+exi Media Type Registration

Type name: application

Subtype name: senml+exi

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <c.jennings@ieee.org>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <c.jennings@ieee.org>

Change controller: IESG

11. Security Considerations

See Section 12. Further discussion of security proprieties can be found in Section 10.2.

12. Privacy Considerations

Sensor data can range from information with almost no security considerations, such as the current temperature in a given city, to highly sensitive medical or location data. This specification provides no security protection for the data but is meant to be used inside another container or transport protocol such as S/MIME or HTTP with TLS that can provide integrity, confidentiality, and authentication information about the source of the data.

13. Acknowledgement

We would like to thank Lisa Dusseault, Joe Hildebrand, Lyndsay Campbell, Martin Thomson, John Klensin, Bjoern Hoehrmann, and Carsten Bormann for their review comments.

14. References

14.1. Normative References

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media

Types", RFC 3023, January 2001.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

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

[IEEE.754.1985]
Institute of Electrical and Electronics Engineers,
"Standard for Binary Floating-Point Arithmetic",
IEEE Standard 754, August 1985.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, May 2008.

14.2. Informative References

[RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.

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

[I-D.ietf-core-coap]
Shelby, Z. and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), January 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "Constrained Application Protocol (CoAP)", draft-ietf-core-link-format-07 (work in progress), January 2011.

[BIPM] Bureau International des Poids et Mesures, "The International System of Units (SI)", 8th edition, 2006 .

[NIST822] Thompson, A. and B. Taylor, "Guide for the Use of the International System of Units (SI)", NIST Special Publication 811, 2008 Edition .

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

- [RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.
- [I-D.arkko-core-dev-urn]
Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-00 (work in progress), October 2011.
- [WADL] Hadley, M., "Web Application Description Language (WADL)", 2009, <<http://java.net/projects/wadl/sources/svn/content/trunk/www/wadl20090202.pdf>>.

Authors' Addresses

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

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

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach@sensinode.com

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

core
Internet-Draft
Intended status: BCP
Expires: April 23, 2012

K. Li
Huawei Technologies
October 21, 2011

CoAP Over SMS
draft-li-core-coap-over-sms-00

Abstract

This document explains how to use CoAP in cellular networks, by using SMS (Short Message Service) as the transport protocol.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 23, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Motivation	3
1.2. Terminology	3
2. CoAP Over SMS	3
2.1. Addressing	3
2.2. Mapping to SMS Messages	4
2.3. Parameters Mapping	5
2.4. Interaction with the Block option	5
3. Security Considerations	5
4. IANA Considerations	5
5. Acknowledgements	5
6. Normative References	5
Author's Address	6

1. Introduction

In cellular networks, it is possible that constrained end-points don't support an IP stack, i.e. TCP or UDP, but support the SMS protocol instead. Compared to the UDP protocol stack, SMS provides a much smaller message size: SMS can transfer up to 140 bytes in each message. So, some optimizations need to be done to reduce the CoAP message size to make it fit for SMS. Also, some adaptations in CoAP need to be specified, to cater for SMS specific parameters.

1.1. Motivation

In some environments, internet connectivity is not supported by the constrained end-points, but a cellular network connection is supported instead. In this situation, SMS will be supported, instead of UDP.

In OMA, there is a new approved work item named "the Lightweight M2M Protocol", which aims at identifying requirements and defining protocols for M2M applications in cellular networks.

In 3GPP, SMS is identified as the transport protocol for small data transmissions (See 3GPP-23.888).

Currently, there are already some SMS based deployments for binary M2M protocols, which are quite similar to CoAP.

1.2. Terminology

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

2. CoAP Over SMS

2.1. Addressing

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 SMS message contains a CoAP message, the TP-DATA-Coding-Scheme SHOULD be included.

2.2. Mapping to SMS Messages

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

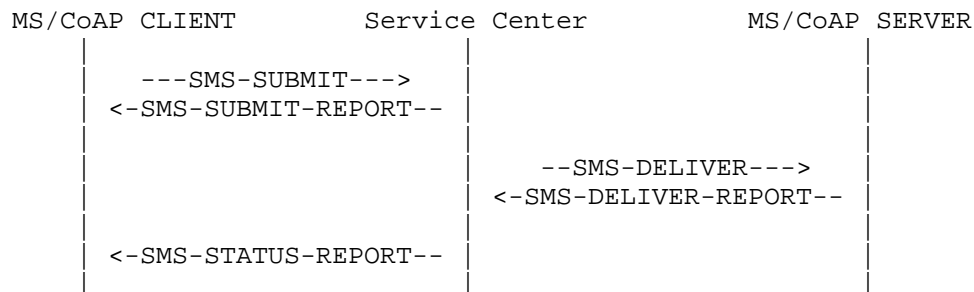


Figure 1: CoAP Messages over SMS

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

Step 1: The CoAP Client sends a CoAP request in a SMS-SUBMIT message to the Service Center. The CoAP Server address is specified as TP-Destination-Address (see 3GPP-23.040).

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

Step 3: The Service Center stores the received SMS message and forwards it to the CoAP Server, using an SMS-DELIVER message. The CoAP Client address is specified as a TP Originating Address (see 3GPP-23.040).

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

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

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

2.3. Parameters Mapping

In case of SMS transport, MSISDN MUST be used as the value of the Uri-Host option. The Uri-Port option SHOULD not be sent, as it is not used for SMS transport.

2.4. Interaction with the Block option

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

3. Security Considerations

Security mechanisms defined in 3GPP-23.888 are used to guarantee transport security.

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 a 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.

4. IANA Considerations

N/A.

5. Acknowledgements

The authors of this draft would like to thank Bert Greevenbosch for the discussion.

6. Normative References

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-04 (work in progress), July 2011.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), July 2011.

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

Author's Address

Kepeng Li
Huawei Technologies
Huawei Base, Bantian, Longgang District
Shenzhen, Guangdong 518129
P. R. China

Phone: +86-755-28974289
Email: likepeng@huawei.com

core
Internet-Draft
Intended status: Standards Track
Expires: April 23, 2012

K. Li
L. Tian
B. Leiba
Huawei Technologies
October 21, 2011

CoAP Option Extension : Size
draft-li-core-coap-size-option-02

Abstract

This document defines an extension to the Constrained Application Protocol (CoAP) to add a new option Size, which is used to indicate the resource size in a PUT/POST request or in a GET response.

Note

Discussion and suggestions for improvement are requested, and should be sent to core@ietf.org.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 23, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
1.1. Justification	3
1.2. Terminology	4
2. Size Option Extension	4
2.1. Size Option Definition	4
2.2. Using the Size Option	4
3. Interaction with Block option	5
3.1. Usage in POST/PUT Request	5
3.2. Usage in GET Response	5
4. How to merge into Block draft	5
4.1. The Size option	5
4.2. Using the Size option	6
4.3. Example	6
5. Examples	6
6. Security Considerations	7
7. IANA Considerations	8
8. Acknowledgements	8
9. Normative References	8
Authors' Addresses	8

1. Introduction

This specification adds a new option Size to the Constrained Application Protocol (CoAP). The main purpose is to indicate the resource size in a PUT/POST request, or in a GET response.

1.1. Justification

If the requester wants to retrieve large resource data using a GET request, it is better to know in advance the size of the resource data. Currently in the Link Format [I-D.ietf-core-link-format] specification, the maximum size estimate attribute "sz" is defined to give an indication of the estimated maximum size of the resource data. By using this, the requester is able to know whether it is capable to accept the resource data. However it is not possible for the requester to know exactly how many blocks will be transmitted, therefore, concurrent GET can't be supported.

Also in a POST/PUT request (for example, a firmware update), it is not possible for the recipient to know in advance what is the size of the data to be transmitted. According to the current CoAP [I-D.ietf-core-coap] specification, when transmitting large data, the recipient will return an error code 4.13 (Request Entity Too Large) to the requester when the data size is too big to be accepted by the recipient. In this case the whole transmission has failed, and the previous received data will be useless. This is a waste of transmission resources.

This document adds the new Size Option to provide the capability to indicate the accurate size in a GET response or in a POST/PUT request.

By using the Size Option in a GET response, the CoAP Server can let the requester know the actual size of the resource in advance. This is especially useful for large resources, and can facilitate the requester to allocate enough buffer space before transmission. Also, using the block size, the requester can calculate the total number of blocks and can use concurrent GET requests to retrieve resource data using the Block Option. Finally, the recipient can check the resource size after the data transmission has been completed.

By using the Size Option in a PUT/POST request, the requester can indicate the resource size in the first Block Option message, to let the recipient know the resource data size in advance. If the recipient is not able to receive the data with the indicated size, the recipient can tell the requester in a response code, avoiding the cost of the actual data transmission.

1.2. Terminology

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

2. Size Option Extension

2.1. Size Option Definition

Type	C/E	Name	Data type	Length	Default
18	E	Size	uint	0-4 B	

2.2. Using the Size Option

The Size Option is used to indicate the size of the resource data measured in bytes.

The GET request including Size=0 is treated as a request to get the size of the resource representation (but not the resource payload).

The GET request including an empty Size option is treated as a request to get the size of the resource representation with the resource payload.

The Size option MUST be included in the GET response, if the Size option is present in the request.

Also it SHOULD be used in a POST/PUT request in the first Block Option message.

The Size option SHOULD be included for resources larger than a single PDU, if the Size information is available. And it MAY be included for resources smaller than a single MTU.

In the absence of the option, the size of the resource data is calculated after the data has been transmitted to the recipient, either from the CoAP payload length or based on number of blocks and block size.

If the Size option is specified it SHOULD be accurate at that time, and SHOULD NOT be an estimate.

But due to the dynamic change of the resource data, the Size may not be accurate. If the value of Size option is not the same as the actual transmitted data, the recipient MUST take the size of the actual transmitted data as accurate, and ignore the Size option. In case that the recipient gets all the data but it is still smaller than the announced Size, the recipient SHOULD stop the transmission. If the recipient finds out the transmitted data reaches the Size limit, and there's more data left, the recipient SHOULD continue to transmit the remaining data.

This option is "Elective". It MUST NOT occur more than once.

3. Interaction with Block option

3.1. Usage in POST/PUT Request

In a PUT/POST request for large resource data, the requester SHOULD use the Size option to indicate the size of the resource. If the recipient is not capable to receive the data with the indicated size, the recipient MUST return a 4.13 (Request Entity Too Large) response code to the requester, and the data transmission is avoided, so that the cost of the actual data transmission is saved.

3.2. Usage in GET Response

In a GET response for large resource data, the CoAP Server SHOULD use the Size option to indicate the resource size and return the first block data. The requester can calculate the number of blocks to be transferred based on the block size and the resource size, and use concurrent GET requests to retrieve resource data. Also, when the client determines it cannot process data of this Size, it MAY choose to abort and not to send subsequent GETs.

4. How to merge into Block draft

This section introduces how to merge the Size option draft into Block draft with the minimum functionalities.

4.1. The Size option

This section will work as section 2.3 in Block draft.

Type	C/E	Name	Data type	Length	Default
18	E	Size	uint	1-4 B	

4.2. Using the Size option

This section will work as section 2.4 in Block draft.

The Size Option is used to indicate the size of the resource data measured in bytes.

The Size option SHOULD be used in a POST/PUT request in the first Block Option message. If the recipient is not capable to receive the data with the indicated size, the recipient MUST return a 4.13 (Request Entity Too Large) response code to the requester, and the data transmission is avoided, so that the cost of the actual data transmission is saved.

For a GET request, if it includes an empty Size option, the Size option MUST be included in the response. If the GET request includes a Block option, the Size option SHOULD be included in the first Block response. In other cases the GET response MAY contain a Size option.

If the Size option is specified, it SHOULD be accurate at that time, and SHOULD NOT be an estimate.

The option is "Elective". It MUST NOT occur more than once.

4.3. Example

Example as indicated as Figure 2 in this draft can be added in section 3 in the Block draft.

5. Examples

This section gives a number of short examples with message flows to illustrate the use of Size option in a GET response, or in a PUT/POST request.

The first example (Figure 1) shows that the requester does not know the resource data size, and sends the GET request, the recipient can send back the resource size using the Size option and the first block. In the subsequent GET request, the requester can calculate the number of blocks and use concurrent GET requests to retrieve the resource data.

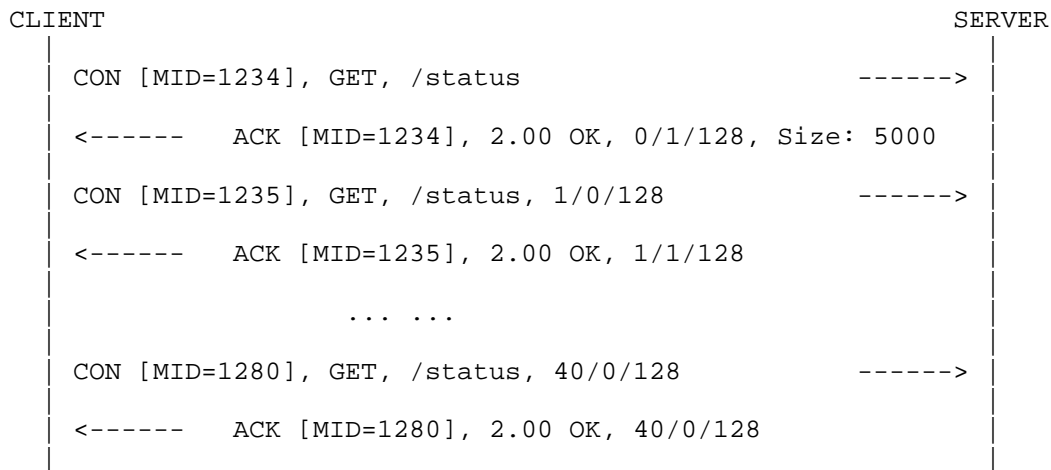


Figure 1: Size Option in a GET response

The second example (Figure 2) shows the requester sending a PUT request with the Size option to indicate the resource data size, and since the recipient determines that the resource data is too large to be accepted, it sends back a 4.13 (Request Entity Too Large) response code.

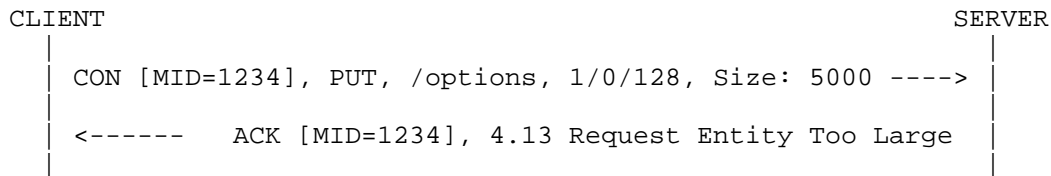


Figure 2: Size Option in a PUT request

6. Security Considerations

As the size option is used to determine whether or not the recipient will accept the data, lying about it can cause the recipient to make a wrong decision. For example, an attacker might reduce the reported size such that the recipient will accept, even when it cannot process the complete data.

Related is another attack, where the attacker changes the reported size to a higher value, leading to the recipient rejecting even when it has the capability to receive.

The latter attack is similar to an attack where the attacker blocks

the packets altogether; although it is more efficient since the attacker only needs to modify one message. The former attack needs serious consideration at implementation level, especially concerning possible buffer overflows that might lead to data leaking into the code.

7. IANA Considerations

The IANA is requested to add the following Option Number entry.

Number	Name	Reference
18	Size	Section 2

8. Acknowledgements

The authors of this draft would like to thank the participants of the email discussion on this issue. Thanks to Bert Greevenbosch, Charles Palmer and Carsten Bormann for the detailed reviews and suggestions.

9. Normative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",
draft-ietf-core-block-04 (work in progress), July 2011.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-07 (work in progress), July 2011.
- [I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format",
draft-ietf-core-link-format-07 (work in progress),
July 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.

Authors' Addresses

Kepeng Li
Huawei Technologies
Huawei Base, Bantian, Longgang District
Shenzhen, Guangdong 518129
P. R. China

Phone: +86-755-28974289
Email: likepeng@huawei.com

Linyi Tian
Huawei Technologies
Huawei Base, Bantian, Longgang District
Shenzhen, Guangdong 518129
P. R. China

Phone: +86-755-28978078
Email: tianlinyi@huawei.com

Barry Leiba
Huawei Technologies

Phone: +1 646 827 0648
Email: barryleiba@computer.org
URI: <http://internetmessagingtechnology.org/>

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: April 3, 2012

Y. Ma
X. He
Hitachi (China) Research and
Development Corporation
Z. Cao
China Mobile
Oct 2011

dhcp option for CoAP Proxy Discovery
draft-ma-core-dhcp-pd-00

Abstract

CoAP utilizes DNS to discovery the IP address of the CoAP server. However DNS is heavy for the most resource constrained end-points. In this case the assistance from CoAP proxy or research directory (RD) is needed for CoAP transaction. This specification proposes to define one new dhcp option for proxy/RD discovery for the most resource constrained end-points.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 3, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
1.1. Conventions used in this document	3
2. dhcp option for proxy discovery	3
3. Security Considerations	4
4. IANA Considerations	4
5. Normative References	4
Authors' Addresses	4

1. Introduction

CoAP [I-D.ietf-core-coap] is a RESTful protocol designed for constrained devices. The ultimate goal of CoAP is to enable the "Web of Things" concept, which connects the smart sensor network with the global internet.

CoAP utilizes DNS for CoAP server IP address discovery. However in some circumstances, DNS is heavy to be implemented in the resource constrained nodes. In this case the assistance from CoAP proxy is needed for CoAP transaction.

Also 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)[I-D.shelby-core-resource-directory], which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources .

Before the CoAP sensor makes use of the CoAP proxy or RD, it must know the location of the proxy or RD. There can be multiple ways to discover the proxy's location, including both static and dynamic methods. Static configuration is a straightforward way for the client to access the server. However, in many situations, static configuration is not enough to meet the requirements.

1.1. Conventions used in this document

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

2. dhcp option for proxy discovery

dhcp [RFC2131] provides dynamic methods to deliver configuration information to the end node. dhcp options [RFC2132] are defined to specify server information. This document specifies one new dhcp option for CoAP proxy/rd discovery.

The CoAP proxy/rd option specifies a list of CoAP proxy or Research Directory servers available to the client. Servers SHOULD be listed in order of preference.

The code for the name server option is x. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

Code	Len	Address 1				Address 2			
x	n	a1	a2	a3	a4	a1	a2	...	

CoAP proxy/rd option

3. Security Considerations

TBD.

4. IANA Considerations

This document needs to register one new dhcp option number at IANA.

5. Normative References

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-07 (work in progress), July 2011.

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

Shelby, Z. and S. Krco, "CoRE Resource Directory",
draft-shelby-core-resource-directory-01 (work in
progress), September 2011.

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

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol",
RFC 2131, March 1997.

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor
Extensions", RFC 2132, March 1997.

Authors' Addresses

Yuanchen Ma
Hitachi (China) Research and Development Corporation
301, Tower C North, Raycom, 2 Kexuyuan Nanlu, Haidian District
Beijing 100190
China

Email: ycma@hitachi.cn

Xuan He
Hitachi (China) Research and Development Corporation
301, Tower C North, Raycom, 2 Kexuyuan Nanlu, Haidian District
Beijing 100190
China

Email: xhe@hitachi.cn

Zhen Cao
China Mobile
Unit2, 28 Xuanwumenxi Ave, Xuanwu District
Beijing 100053
China

Email: zehn.cao@gmail.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2012

J. Nieminen, Ed.
B. Patil
T. Savolainen
M. Isomaki
Nokia
Z. Shelby
Sensinode
C. Gomez
Universitat Politecnica de
Catalunya/i2CAT
October 31, 2011

Configuration and service discovery of IPv6 capable sensors
draft-nieminen-core-service-discovery-01

Abstract

The number and type of Internet connected devices continues to grow rapidly. Sensors are a new category of devices that are becoming Internet connected. While sensors have existed for a long time, it is only now that we see sensors being capable of communicating via an IP stack. Sensors are used in a multitude of industries and applications. By enabling these devices to be Internet connected, they open up new vistas in terms of applications and uses. Most sensors are generally small with minimal power and processing capabilities. The ability to configure these devices is a challenge. However, in order to make the usage and deployment of Internet connected sensors easier, it is essential that there exist a well defined configuration, control, and service discovery mechanism. This document illustrates various use cases of sensors in different types of deployments and a solution for configuring and controlling them in addition to the ability for sensors and devices to discover services.

Status of this Memo

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

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

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

material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 3, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
1.1. Requirements Language	4
1.2. Terminology	4
2. Problem statement	5
3. Sensor communication models	5
3.1. Local reporting, local consumption	5
3.2. Local reporting, remote consumption	6
3.3. Local reporting, Internet service	6
3.4. Internet service	7
3.5. Internet service with ability to control the sensor	7
3.6. Internet service, direct sensor control	7
4. Sensor configuration	7
5. Resource and service discovery	8
6. IANA Considerations	9
7. Security Considerations	9
8. Acknowledgements	9
9. Normative References	9
Authors' Addresses	10

1. Introduction

During the recent years, mechanisms for connecting sensors with the Internet of Things (IoT) have been developed and standardized. 6LoWPAN standard [RFC4944] defines how to run IPv6 over 802.15.4 family radios (ZigBee, Wireless HART) and [I-D.ietf-6lowpan-btle] proposes a solution for adapting 6LoWPAN stack for ultra-low power Bluetooth Low Energy (BT-LE). However, being able to transmit IPv6 packets is not enough for providing a complete Internet connectivity solution for the sensors. Mechanisms such as sensor configuration, control, resource and service discovery are a crucial part of the solution. This document describes the typical communication models between sensors and the Internet and discusses the related functional and technical requirements.

1.1. Requirements Language

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

1.2. Terminology

Bluetooth Low Energy

Bluetooth Low Energy is a low power air interface technology specified by the Bluetooth Special Interest Group (SIG). BT-LE is specified in Revision 4.0 of the Bluetooth specifications (BT 4.0).

Gateway

Network element connecting sensors to the Internet. Can be e.g a home gateway or a mobile device.

6LR and 6LBR

These terms correspond to those defined in [I-D.ietf-6lowpan-nd]

Consumer

Network element monitoring the state of the sensors. Can be e.g a server or a mobile device.

2. Problem statement

Sensors are generally purpose built devices which have very limited processing power, memory or battery life. There are various types of sensors and they can communicate either via a wired or wireless interface. Sensors for detecting temperature, luminescence, humidity, hazardous chemicals, heartrate, pulse etc. are just a few examples. They can be used in a variety of industries and applications including healthcare, automobile, manufacturing, home and enterprise, aviation and agriculture. As sensors become Internet connected, it also brings up the need for them to be configured and managed in an easy manner. The nature of these devices results in configuring, controlling and managing them especially challenging.

The applicability of sensors when Internet connected increases dramatically. However, the problem associated with configuring and controlling/managing them needs to be solved in order to make usage and deployment easier. Sensors do not have keyboards or a UI through which they can be configured. The IP stack and capabilities of many of these sensors is also very limited. Hence the problem to be solved is primarily about a protocol or method to configure such barebone sensors which are capable of connecting to the Internet.

3. Sensor communication models

This section presents the typical communication models between sensors and the Internet.

3.1. Local reporting, local consumption

In this scenario everything happens within a local/private network. Sensor sends data to the local server, consumer in the local network can access the data. Sensor can use local-network discovery methods to locate the server.

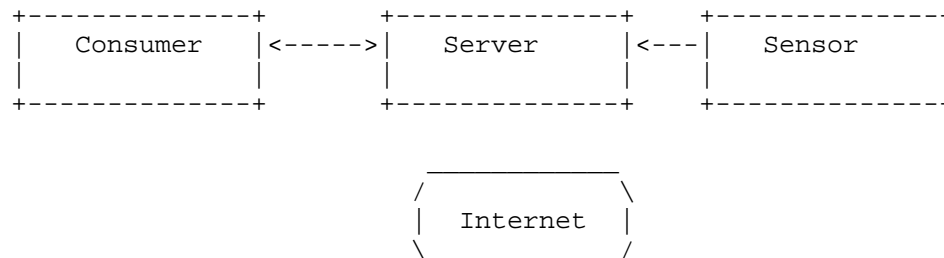


Figure 1: Local reporting, local consumption

3.2. Local reporting, remote consumption

In this scenario the sensor still sends data to a local server. However, the consumer can read the data from anywhere. The server needs to be reachable from the Internet (e.g. via a proxy or Firewall/NAT bindings).

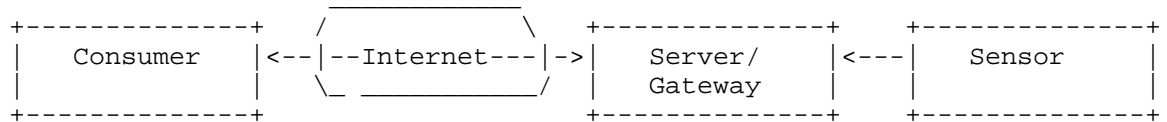


Figure 2: Local reporting, remote consumption

3.3. Local reporting, Internet service

In this scenario the sensor still sends data to a local server. The local server acts as a gateway and forwards data to an Internet service. The consumer can obtain data from the service.

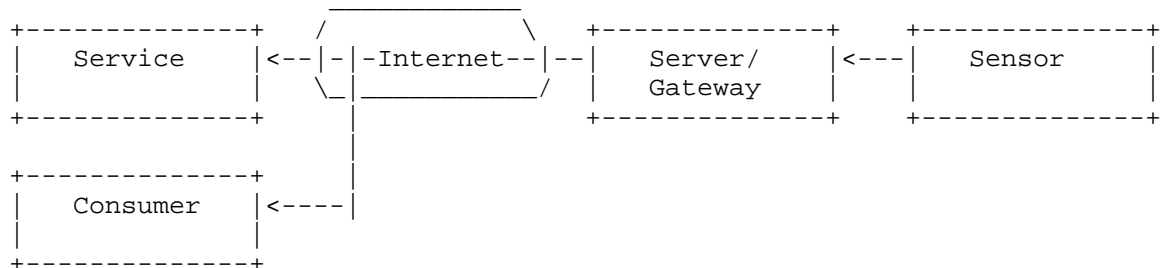


Figure 3: Local reporting, Internet service

3.4. Internet service

In this scenario the sensor communicates directly with the server, requiring configuration. The gateway is transparent and it's main role is to forward packets. The consumer obtains data from the service.

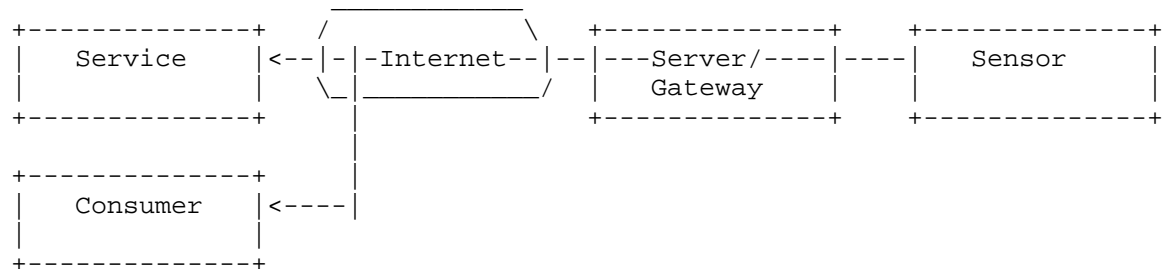


Figure 4: Internet service

3.5. Internet service with ability to control the sensor

This scenario is similar to the previous scenario with the difference that the consumer/controller can control the sensor via the service. The control can be synchronized with the communication pattern of the sensor or it can be asynchronous.

3.6. Internet service, direct sensor control

This scenario is similar to the previous scenario with the difference that control can happen directly via the gateway. The gateway can act as a proxy to the sensor.

4. Sensor configuration

The sensor runs a simple application that communicates for instance using the CoAP protocol. At minimum the application requires some basic configuration to operate properly. The configuration includes parameters such as the IP address, hostname or the URI of the server or gateway the application should send its requests to. In many cases the sensor also has to have the proper credentials to successfully communicate with the service.

If the sensor only communicates with the hosts in its local

environment, mechanisms such as well-known multicast addresses or DHCP might work for the sensor application to get its initial configuration. These correspond to the "local reporting" communication models described in the previous section. However, when the sensor application should communicate directly with a server or service across the Internet, with no relation to the access network the sensor is attached to, the situation becomes more challenging. For instance, a heart rate meter would need to be configured with the fitness service provider's domain name or URI, and the credentials of its current user, so that its data is sent to the right place with the correct privacy settings. This corresponds to the basic configuration of e-mail or SIP or XMPP accounts. The difference is that a sensor such as a heart rate belt does not have the user interface to input even these basic parameters.

One approach is to use another device with better user interface to assist with the configuration. Many devices such as home routers are currently configured in this manner. The routers run a web server and another host attached to the same LAN can access it with a web browser, usually using some obscure hard coded IP addresses to start with. Running even a simple web server and a set of web pages on it may however not be practical for a limited sensor. Instead, simpler mechanisms may be needed. For instance, the proper configuration might be manually input or automatically fetched to a device such as a PC or smartphone, where the sensor could discover and fetch it.

As the sensors are expected to use CoAP for their normal service communication, it would be sensible for the server to use the same protocol for fetching the configuration object. For this to work, there may be need to standardize these aspects:

1. How the sensor can discover that its default gateway or some other host in its local environment can offer it configuration for the correct type of service.
2. How the sensor fetches the configuration.
3. Format of the basic configuration: Server URI, credentials.

CoAP already offers the basic mechanisms for what is needed here, but small amount of additional standards development seems necessary to get all of this interoperable.

5. Resource and service discovery

Besides configuration, it is important to consider how the gateway or a consumer can identify names, states and capabilities of objects,

groups of objects and services (example: identify which sensors are attached to an HVAC system of a particular house)

One possibility is to use a protocol with multicast capability which sends out a query within the scope of a gateway or watcher and wait for responses from the sensors. The Sensors should wake up if they are in sleep mode and respond to such a query with a response which describes the sensors functionality and service provided.

6. IANA Considerations

This document does not require any IANA actions.

7. Security Considerations

Configuration and control of sensors have to be done in a secure manner. The ability to hijack a sensor by malicious entities is a threat and can be fairly critical depending on the type of sensor and application that it is being used in. Service discovery also needs to have security in terms of ensuring that the service is authentic and being offered by a valid entity. Various types of threats exist in an environment where sensors are Internet connected and these can vary depending on the scenario and method through which the sensor obtains connectivity.

8. Acknowledgements

9. Normative References

[I-D.ietf-6lowpan-btle]

Nieminen, J., Patil, B., Savolainen, T., Isomaki, M., Shelby, Z., and C. Gomez, "Transmission of IPv6 Packets over Bluetooth Low Energy", draft-ietf-6lowpan-btle-03 (work in progress), October 2011.

[I-D.ietf-6lowpan-hc]

Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams in Low Power and Lossy Networks (6LoWPAN)", draft-ietf-6lowpan-hc-15 (work in progress), February 2011.

[I-D.ietf-6lowpan-nd]

Shelby, Z., Chakrabarti, S., and E. Nordmark, "Neighbor Discovery Optimization for Low Power and Lossy Networks

(6LoWPAN)", draft-ietf-6lowpan-nd-18 (work in progress), October 2011.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2464] Crawford, M., "Transmission of IPv6 Packets over Ethernet Networks", RFC 2464, December 1998.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.
- [RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, September 2007.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC4994] Zeng, S., Volz, B., Kinnear, K., and J. Brzozowski, "DHCPv6 Relay Agent Echo Request Option", RFC 4994, September 2007.

Authors' Addresses

Johanna Nieminen (editor)
Nokia
Itaamerenkatu 11-13
FI-00180 Helsinki
Finland

Email: johanna.1.nieminen@nokia.com

Basavaraj Patil
Nokia
6021 Connection drive
Irving, TX 75039
USA

Email: basavaraj.patil@nokia.com

Teemu Savolainen
Nokia
Hermiankatu 12 D
FI-33720 Tampere
Finland

Email: teemu.savolainen@nokia.com

Markus Isomaki
Nokia
Keilalahdentie 2-4
FI-02150 Espoo
Finland

Email: markus.isomaki@nokia.com

Zach Shelby
Sensinode
Hallituskatu 13-17D
FI-90100 Oulu
Finland

Email: zach.shelby@sensinode.com

Carles Gomez
Universitat Politecnica de Catalunya/i2CAT
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Email: carlesgo@entel.upc.edu

CoRE
Internet-Draft
Intended status: Informational
Expires: April 14, 2012

A. Rahman, Ed.
InterDigital Communications, LLC
E. Dijk, Ed.
Philips Research
October 12, 2011

Group Communication for CoAP
draft-rahman-core-groupcomm-07

Abstract

This is a working document intended to trigger discussion and develop draft text for the CoAP protocol specification in the area of group communication. Engineering tradeoffs become more challenging in constrained environments, therefore group communication is considered within the context of adjacent topics that may impact or be impacted by design choices in the subject area. A solution based on IP multicast is proposed.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 14, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Conventions and Terminology	3
2. Introduction	3
2.1. Background	3
2.2. Problem Statement and Scope	4
3. Potential Solutions	5
3.1. Overview	5
3.2. Requirements	6
3.3. IP Multicast	8
3.3.1. Multicast Listener Discovery (MLD) and Multicast Router Discovery (MRD)	8
3.3.2. Group URIs and Multicast Addresses	9
3.3.3. Group Discovery	9
3.3.4. Group Resource Manipulation	10
3.3.5. IP Multicast Transmission Methods	11
3.3.6. Congestion Control	12
3.4. Overlay Multicast	13
3.5. CoAP Application Layer Group Management	13
3.6. CoAP Multicast and HTTP Unicast Interworking	16
3.7. CoAP-Observe for Group Communication	18
4. Recommended Solution	18
4.1. Overview	19
4.2. An Example Protocol Flow	19
4.3. Implementation in Target Network Topologies	22
4.3.1. Single LLN Topology	23
4.3.2. Single LLN with Backbone Topology	25
4.3.3. Multiple LLNs with Backbone Topology	27
4.3.4. LLN(s) with Multiple 6LBRs	27
4.3.5. Conclusions	27
4.4. HTTP/CoAP Interworking Aspects	28
4.5. Implementation Considerations	28
4.5.1. MLD Implementation on LLNs	28
4.5.2. 6LBR Implementation	29
4.5.3. Backbone IP Multicast Infrastructure	29
5. Security Considerations	30
6. IANA Considerations	31
7. Conclusions	31
8. Acknowledgements	31
9. References	32
9.1. Normative References	32
9.2. Informative References	33
Authors' Addresses	35

1. Conventions and 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 are definitions of specific terminology used in this draft.

Group Communication: A source node sends a message to more than one destination node, where all destinations are identified to belong to a specific group. The set of source nodes and destination nodes may consist of an arbitrary mix of constrained and non-constrained nodes. Sending methods may include serial unicast, multicast, or hybrid unicast-to-multicast solutions.

Multicast: Sending a message to multiple receiving nodes simultaneously. Typically, this is done as part of a group communication process. There are various options to implement multicast including layer 2 (Media Access Control) or layer 3 (IP) mechanisms.

IP Multicast: A specific multicast solution based on the use of IP multicast addresses as defined in "IANA Guidelines for IPv4 Multicast Address Assignments" [RFC5771] and "IP Version 6 Addressing Architecture" [RFC4291].

Low power and Lossy Network (LLN): LLNs are made up of constrained devices. These devices may be interconnected by a variety of links, such as IEEE 802.15.4, Bluetooth, WiFi, wired or low-power powerline communication links.

2. Introduction

2.1. Background

The CoRE working group is chartered to design and standardize a Constrained Application Protocol (CoAP) for resource constrained devices and networks [I-D.ietf-core-coap]. The requirements for CoAP are documented in [I-D.shelby-core-coap-req].

Constrained devices can be large in number, but highly correlated to each other. For example, all the light switches in a building may belong to one group and all the thermostats belong to another group. All the smart meters in the same region can belong to a group as well. Groups may be composed by function; for example, the group "all lights in building one" may consist of the groups "all lights on

floor one of building one", "all lights on floor two of building one", etc. Groups may also be configured or dynamically formed.

In this draft, we focus and expand discussions on the requirements pertaining to CoAP "group communication" and "multicast" support including:

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

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

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

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

2.2. Problem Statement and Scope

In this draft, we expand the scope from unreliable IP multicast in the current CoAP requirement to group communication, using either (reliable or unreliable) multicast or unicast or combinations thereof. We assume that all, or a substantial part of, devices participating in group communication are constrained devices (e.g. such as Low Power and Lossy Network (LLN) devices).

Machine-to-Machine (M2M) networks may contain groups of nodes that are highly correlated (e.g. by type or location). For example, all smart meters in a region may belong to one group, and all light switches in a building control system belong to another. Group communication mechanisms can improve efficiency and latency of communication and reduce bandwidth requirements for a given application.

In the following sections, we address the issues related to group communication in detail, with requirements, proposed solutions and analysis of their impact to the CoAP protocol and implementations.

3. Potential Solutions

3.1. Overview

The classic concept of group communications is that of a single source distributing content to multiple recipients that are all part of a group, as shown in the example sequence diagram in Figure 1. Also shown there is the pre-requisite step of forming the group before content can be distributed to it.

Group communication solutions have evolved from "bottom" to "top", i.e., from the network layer (IP multicast) to application layer group communication, also referred to as application layer multicast. A study published in 2005 [STUDY1] identified new solutions in the "middle" (referred to as overlay multicast) that utilize an infrastructure based on proxies.

Each of these classes of solutions may be compared [STUDY1] using metrics such as link stress and level of host complexity [STUDY2]. The results show for a realistic internet topology that IP Multicast is most resource-efficient, with the only downside being that it requires most effort to deploy in the infrastructure.

The approach adopted in this section is to begin with group communication requirements. This is followed by the solutions of IP multicast, an overlay multicast solution, and application layer group communication. Finally additional topics are covered such as group management and CoAP/HTTP proxies in group communication.

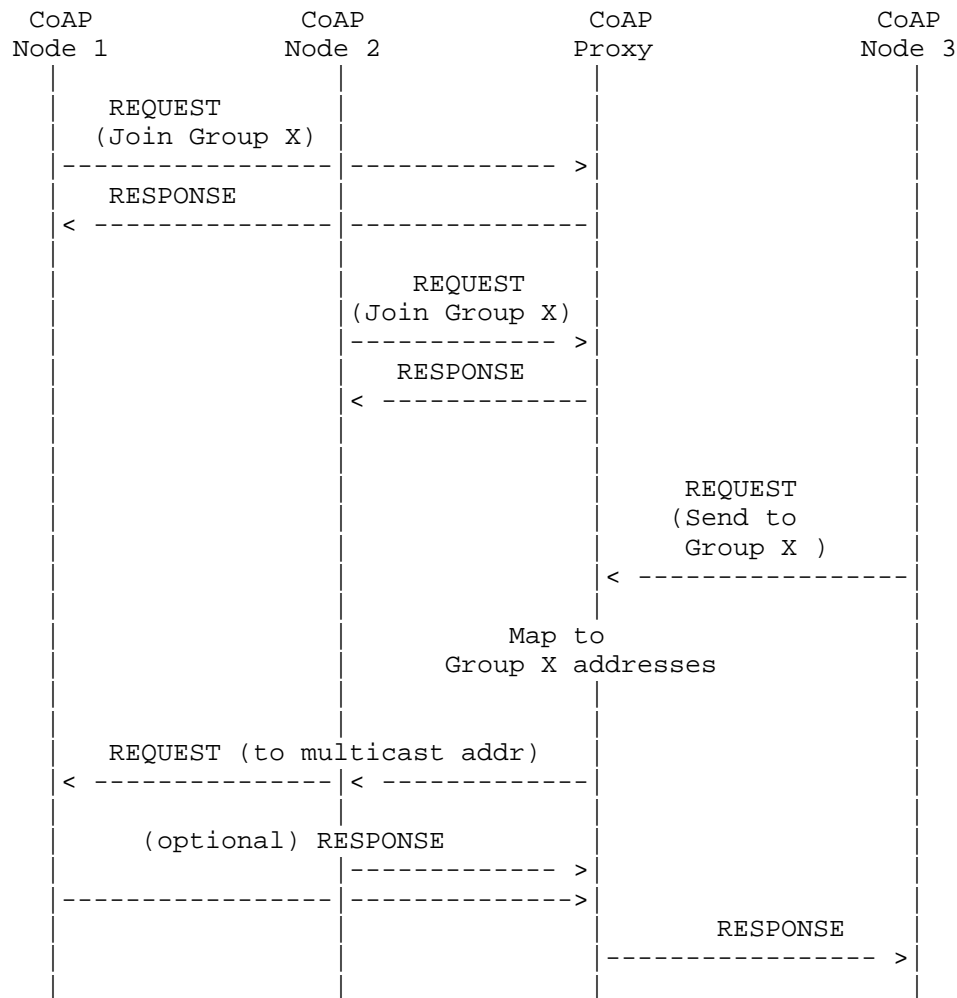


Figure 1: CoAP Group Communications

3.2. Requirements

Requirements that a group communication solution in CoRE should fulfill can be found in existing documents [RFC 5867] [draft-ietf-6lowpan-routing-requirements] [I-D.vanderstok-core-bc] [I-D.shelby-core-coap-req]. Below, a set of high-level requirements is listed that a group communication solution in CoRE should ideally fulfill. More precise requirements may depend on the chosen application (area).

A CoRE group communication solution should (ideally) offer:

- REQ 1: Optional Reliability: unreliable group communication, with preferably reliable group communication as an option.
- REQ 2: Efficiency: delivers messages more efficiently than a "serial unicast only" solution. Also, it should provide a right balance between group data traffic and control overhead.
- REQ 3: Low latency: deliver a message (preferably) as fast as possible.
- REQ 4: Synchrony: allows near-simultaneous modification of a resource on all devices in a group, providing to users a perceived effect of synchrony or simultaneity. It can be expressed as a time span D such that message m is delivered to all destinations in a time interval $[t, t+D]$ for arbitrary t .
- REQ 5: Ordering: message ordering in the reliable group communication mode.
- REQ 6: Security: see Section 5 for security requirements for group communication.
- REQ 7: Flexibility: support for one or many source(s), for dense and sparse networks, for high or low listener density, one or many group(s), and multi-group membership.
- REQ 8: Robust group management: includes functionality to join groups, leave groups, view group membership, and persistent group membership in failing node or sleeping node situations.
- REQ 9: Network layer independence: a solution should be specified independent from specific unicast and/or IP multicast routing protocols. It should support different routing protocols and implementations thereof.
- REQ 10: Minimal specification overhead: a group communication solution should preferably re-use existing/established (IETF) protocols that are suitable for LLN deployments, instead of defining new protocols from scratch.
- REQ 11: Minimal implementation overhead: e.g. a solution allows to re-use existing (software) components that are already present on constrained nodes such as (typical) 6LoWPAN/CoAP nodes.
- REQ 12: Mixed backbone/LLN topology support: a solution should work within a single LLN, and in combined LLN/backbone network topologies, including multi-LLN topologies. Both the senders and receivers of CoAP group messages may be attached to different network links or be part of different LLNs, possibly with routers or switches in between group members. In addition, different routing protocols may operate on the LLN and backbone networks. Preferably a solution also works with existing, common backbone IP infrastructure (e.g. switches or routers).

- REQ 13: CoAP Proxying support: a CoAP proxy can handle distribution of a message to a group on behalf of a (constrained) CoAP client.
- REQ 14: Suitable for operation on LLNs with constrained nodes.

3.3. IP Multicast

IP Multicast protocols have been evolving for decades, resulting in proposed standards such as Protocol Independent Multicast - Sparse Mode (PIM-SM) [RFC4601]. Yet, due to various technical and marketing reasons, IP Multicast is not widely deployed on the general Internet. However, IP Multicast is popular in specific deployments such as in enterprise networks (e.g. for video conferencing or general IP multicast PC applications within a single LAN broadcast domain) and carrier IPTV deployments. Therefore, the packet economy and minimal host complexity of IP multicast make it worth investigating for group communication in constrained environments.

3.3.1. Multicast Listener Discovery (MLD) and Multicast Router Discovery (MRD)

The Multicast Listener Discovery (MLD) protocol [RFC3810] (or its IPv4 pendant IGMP) is today the method of choice used by an (IP multicast enabled) IPv6 router to discover the presence of multicast listeners on directly attached links, and to discover which multicast addresses are of interest to those listening nodes. It was specifically designed to cope with fairly dynamic situations in which multicast listeners may join and leave at any time.

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

The Multicast Router Discovery (MRD) protocol [RFC4286] defines a way to discover multicast routers, for the purpose of using this information by IGMP/MLD snooping devices. However, it appears that this protocol is not as commonly implemented in existing products as MLD is.

3.3.2. Group URIs and Multicast Addresses

An approach to map group authorities onto IP multicast addresses using DNS was proposed in [I-D.vanderstok-core-bc]. Examples of group URI naming (and scoping) for a building control application are shown below. Group URIs MUST follow the approach of the URI structure defined in [RFC3986].

URI authority	Targeted group
all.bldg6...	"all nodes in building 6"
all.west.bldg6...	"all nodes in west wing, building 6"
all.floor1.west.bldg6...	"all nodes in floor 1, west wing, etc."
all.bu036.floor1.west.bldg6...	"all nodes in office bu036, floor1, etc."

The authority portion of the URI is used to identify a node (or group) and the resulting DNS name is bound to a unicast or multicast IP address. Each example group URI shown above can be mapped to a unique multicast IP address. This may be an address allocated according to [RFC3956], [RFC3306] or [RFC3307].

3.3.3. Group Discovery

CoAP defines a resource discovery capability but, in the absence of a standardized group communication infrastructure, it is limited to link-local scope IP multicast; examples may be found in [I-D.ietf-core-link-format]. A service discovery capability is required to extend discovery to other subnets and scale beyond a certain point, as originally proposed in [I-D.vanderstok-core-bc].

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

All CoAP nodes in a given subdomain may be enumerated by sending a query for PTR records named `_coap._udp` to the authoritative DNS server for that zone. A list of SRV records is returned. Each SRV record contains the port and host name (AAAA record) of a CoAP node. The IP address of the node is obtained by resolving the host name.

DNS-SD also specifies an optional TXT record, having the same name as the SRV record, which can contain "key=value" attributes. This can be used to store information about the device, e.g. schema=DALI, type=switch, group=lighting.bldg6, etc.

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

3.3.4. Group Resource Manipulation

At least two forms of group resource manipulation must be supported. The first is push (multicast PUT or MPUT for short) as e.g. "turn off all the lights simultaneously". Logically, this is similar to publishing a value to multiple subscribers. The second operation is pull (multicast GET or MGET), which is essential for discovery during commissioning and can be illustrated by the example "return all the resources on all CoAP servers advertized by their .well-known/core URI". MGET to an "all-nodes" or "all-CoAP-nodes" multicast IP address should perhaps be limited in scope to link-local multicast for scaling [TBD: and possibly for security reasons, e.g. DoS attacks].

Conceptually, the result of a multicast GET or PUT should be the same as if the client had unicast them serially (that is, a set of {URI, representation} tuples). Practically, there are major benefits to avoiding serial unicast in favor of a multicast CoAP GET/PUT solution:

- packet economy on constrained networks
- M2M resource discovery (solves the "chicken-and-egg" problem)
- apparent simultaneity of events (e.g. in lighting applications)
- average lower latency per event (e.g. in lighting applications)

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

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

A first solution in this respect is to couple groups to service

descriptions in DNS (using DNS-SD as in Section 3.3.3 and [I-D.vanderstok-core-bc]). A service description for a multicast group may have a TXT record in DNS defining a schema X (e.g. "schema=DALI"), which defines by service standard X (e.g. "DALI") which resources a node supporting X MUST have. Therefore a multicast source can safely refer to all resources with corresponding operations as prescribed by standard X. For port numbers (which can be found using DNS-SD also) the same holds. Alternatively, only the default CoAP port may be used in all requests.

A second solution is to impose the following restrictions, e.g. for groups not found using, or advertized in, DNS-SD:

- o All CoAP multicast requests MUST be sent to the well-known CoAP port.
- o All CoAP multicast requests SHOULD operate on /.well-known/core URIs

One question is whether the application (or middleboxes) need to be aware that a request is intended for a group. A separate scheme as proposed by [ID.goland-http-udp] might be useful (e.g. "corem" vs. "core"). To the extent that group membership might be implemented as a series of IP multicast, serial unicast, or some combination, having a distinct scheme for group operations might be a useful signal for a proxy receiving the request to look up the group membership and replicate serial unicasts as well as send multicast packets.

3.3.5. IP Multicast Transmission Methods

3.3.5.1. Serial unicast

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

3.3.5.2. Unreliable IP Multicast

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

3.3.5.3. Reliable IP Multicast

[TBD: This is a difficult problem. Need to investigate the benefits of repeating MGET and MPUT requests (saturation) to get "Pretty Good

Reliability". Use the same MID or a new MID for repeated requests? Carsten suggests the use of bloom filters to suppress duplicate responses.

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

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

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

There are various approaches to achieve reliability, such as

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

3.3.6. Congestion Control

CoAP requests may be multicast, resulting a multitude of replies from different nodes, potentially causing congestion. [I-D.eggert-core-congestion-control] suggests to conservatively control sending multicast requests.

CoAP already addresses the congestion problem to some extent by requiring all multicast CoAP requests to be Non-Confirmable. However, as responses to multicast requests (both MGET or MPUT) are required in CoAP, using CoAP multicast still may lead to congestion issues.

Various means can be implemented to prevent congestion.

[TBD: if an MGET or MPUT request leads to the sending of a CoAP response by servers, the servers should enforce a random delay within TIMEOUT before sending their responses. More investigation required.]

Currently in the CoAP protocol, a MAX_RETRANSMIT value set by default to 4 is used for retransmission of Confirmable messages. Since CoAP multicast messages are Non-Confirmable, no retransmissions will occur in CoAP, making the effective retransmission value 0.

3.4. Overlay Multicast

An alternative group communication solution (to IP Multicast) is an "overlay multicast" approach. We define an overlay multicast as one that utilizes an infrastructure based on proxies (rather than an IP router based IP multicast backbone) to deliver IP multicast packets to end devices. MLD (Section 3.3.1) has been selected as the basis for multicast support by the ROLL working group for the RPL routing protocol. Therefore, it is proposed that "IGMP/MLD Proxying" [RFC4605] be used as a basis for an overlay multicast solution for CoAP.

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

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

3.5. CoAP Application Layer Group Management

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

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

The following group management primitives are in general useful:

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

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

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

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

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

It is proposed that CoAP supports two Header Options for group "Join" and "Leave". These Options are Elective so they should be assigned an even number. Assuming the Type for "join" is x (value TBD), the Header Options are illustrated by the table in Figure 2:

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

Figure 2: CoAP Header Options for Group Management

Figure 3 illustrates how a node can join or leave a group using the Header Options in a CoAP message:

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

Figure 3: CoAP Message for Group Management

Header Fields for the above example:

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

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

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

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

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

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

CoAP defines a delta encoding for header options. The first delta is the "Type" for group join in this specific example. If the type for group join is x as illustrated in Figure 3, delta will be x. In the second header option, it is also a group join so the delta is 0. The third header option is a group leave so the delta is 2.

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

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

3.6. CoAP Multicast and HTTP Unicast Interworking

Within the constrained network, CoAP runs over UDP for which IP multicast is supported. In a non-constrained network (i.e. general Internet), HTTP over TCP is used for which IP multicast is not supported. Therefore a CoAP/HTTP Proxy node that supports group communication needs to have functionalities to support interworking of unicast and multicast. One possible way of operation of the Proxy is illustrated in Figure 4. Note that this topic is covered in more detail in [I-D.castellani-core-http-mapping].

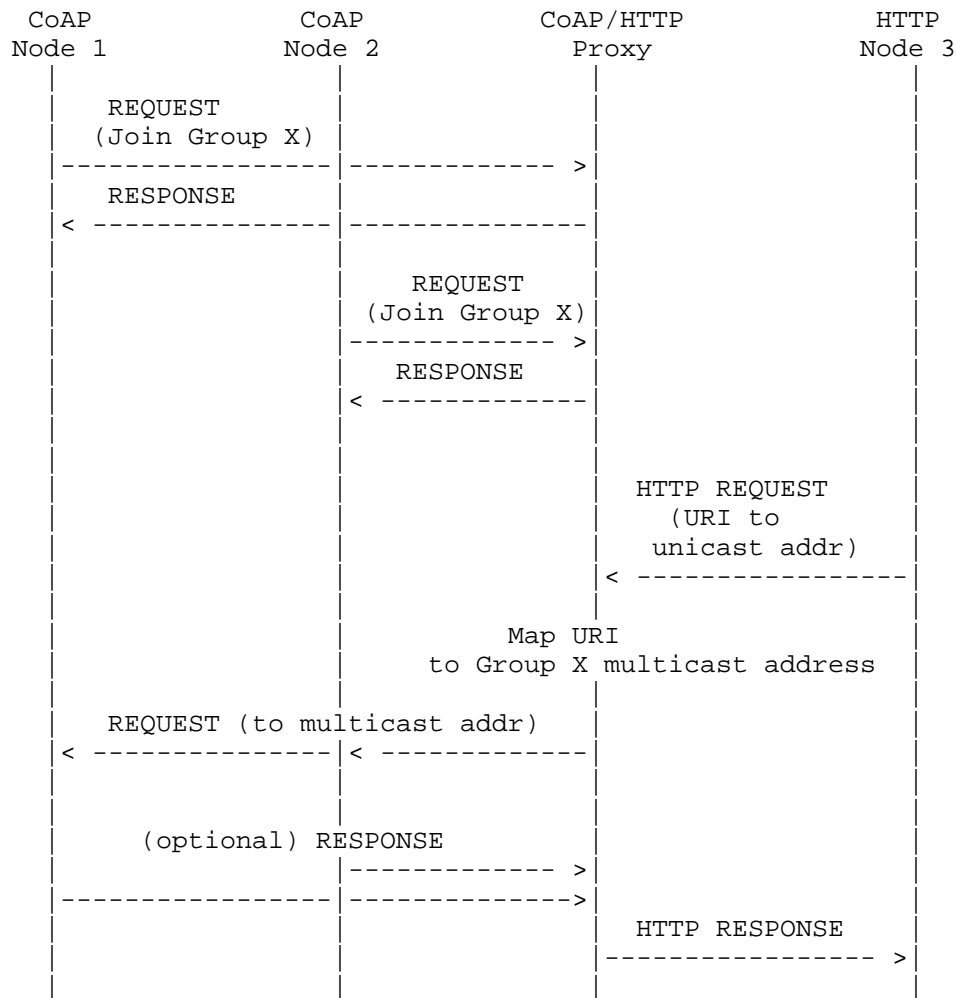


Figure 4: CoAP Multicast and HTTP Unicast Interworking

Note that Figure 4 illustrates the case of IP multicast as the underlying group communications mechanism. However the overlay multicast group communication (Section 3.4) or CoAP application group communication (Section 3.5) can be used as the underlying mechanism and the principles of the figure would still apply (i.e. CoAP proxy needs to do interworking between HTTP unicast and CoAP multicast).

A key point in Figure 4 is that the incoming HTTP Request (from node 3) will carry a URI (with the HTTP scheme) that resolves in the general Internet to the proxy node. At the proxy node, the URI will

then possibly be mapped (as detailed in [I-D.castellani-core-http-mapping]) and again resolved (with the CoAP scheme) to an IP multicast destination. This may be accomplished, for example, by using DNS-SD (Section 3.3.3). The proxy node will then IP multicast the CoAP Request (corresponding to the received HTTP Request) to the appropriate nodes (i.e. nodes 1 and 2).

In terms of the HTTP Response, Figure 4 illustrates that it will be generated by the proxy node based on aggregated responses of the CoAP nodes and sent back to the client in the general Internet that sent the HTTP Request (i.e. node 1). In [I-D.castellani-core-http-mapping] the HTTP Response that the Proxy may use to aggregate multiple CoAP responses is described in more detail. So in terms of overall operation, the CoAP proxy can be considered to be a "non-transparent" proxy according to [RFC2616]. Specifically, [RFC2616] states that a "non-transparent proxy is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction or anonymity filtering."

An alternative to the above is using a Forward Proxy. In this case, the CoAP request URI could be carried in the HTTP Request Line (as defined in [I-D.ietf-core-coap] Section 8) in a HTTP request sent to the IP address of the Proxy.

3.7. CoAP-Observe for Group Communication

The CoAP Observation extension [I-D.ietf-core-observe] can be directly used for group communication. A group then consists of a CoAP server hosting a specific resource, plus all CoAP clients observing that resource. The server is the only group member that can send a group message. It does this by modifying the state of a resource under observation and subsequently notifying its observers of the change. Serial unicast is used in this case for notifications.

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

4. Recommended Solution

4.1. Overview

We recommend that IP multicast as outlined in Section 3.3 be adopted as the base solution for CoAP Group Communication. This approach re-uses the IP multicast suite of protocols and can operate on both constrained and non-constrained network segments. The group communication can hence work regardless of the underlying networking technology. Still, this approach may require specifying or implementing additional IP Multicast functionality in an LLN, in a backbone network, or in both - this will be evaluated in more detail in this section.

4.2. An Example Protocol Flow

We first present an example use case to illustrate the overall steps in an IP Multicast based CoAP Group Communication solution. We assume the following network configuration for this example (see Figure 5):

- 1) A large room (Room-A) with three lights (Light-1, Light-2, Light-3) controlled by a Light Switch. The devices are organized into two 6LoWPAN subnets.
- 2) Light-1 and the Light Switch are connected to a router (Rtr-1) which is also a CoAP Proxy and a 6LoWPAN Border Router (6LBR).
- 3) Light-2 and the Light-3 are connected to another router (Rtr-2) which is also a CoAP Proxy and a 6LBR.
- 4) The routers are connected to a an IPv6 network backbone which is also multicast enabled. In the general case, this means the network backbone and 6LBRs support a PIM based multicast routing protocol, and MLD for forming groups. In a limited case, if the network backbone is one link, then the routers only have to support MLD-snooping for the example use case to work.

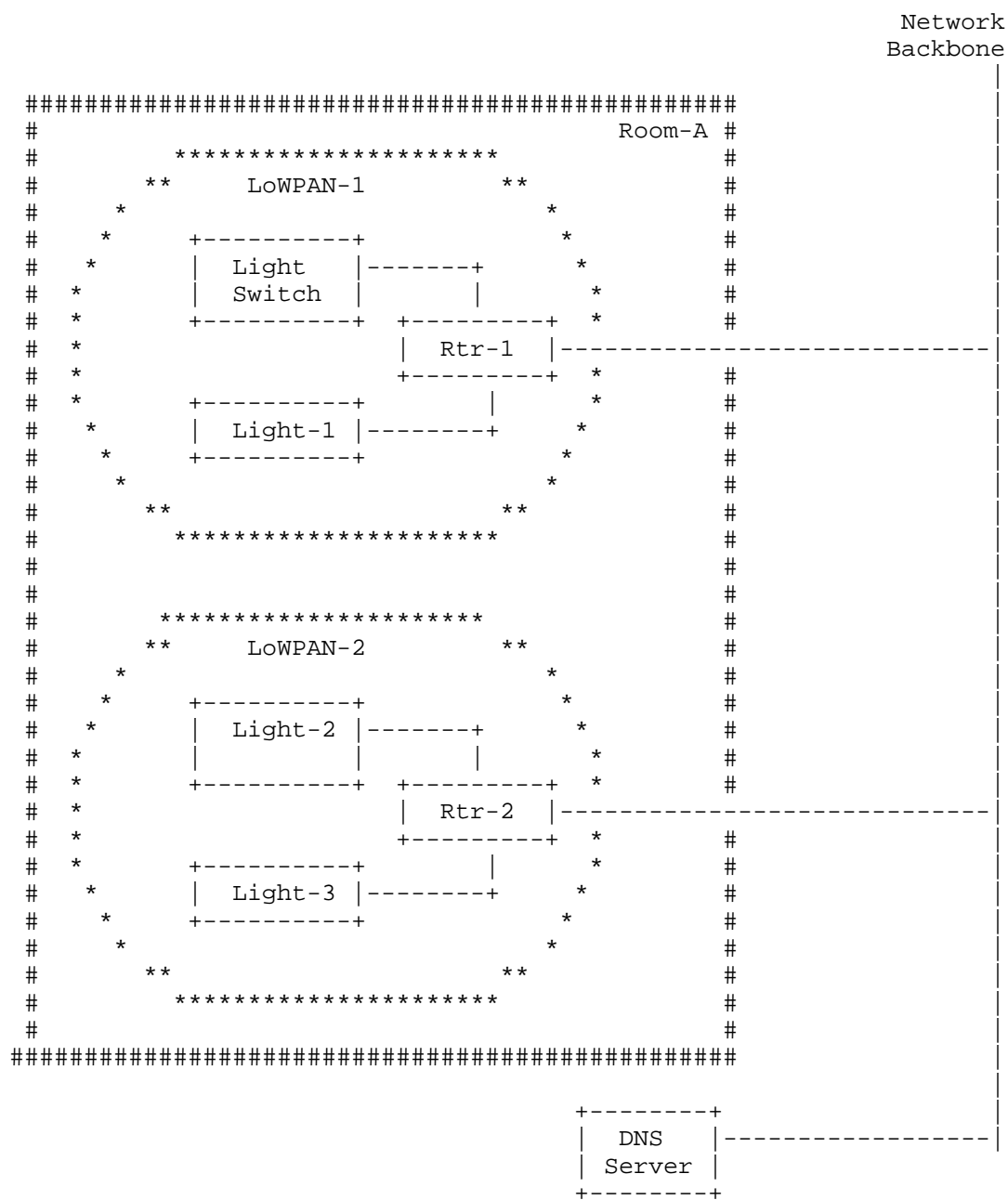
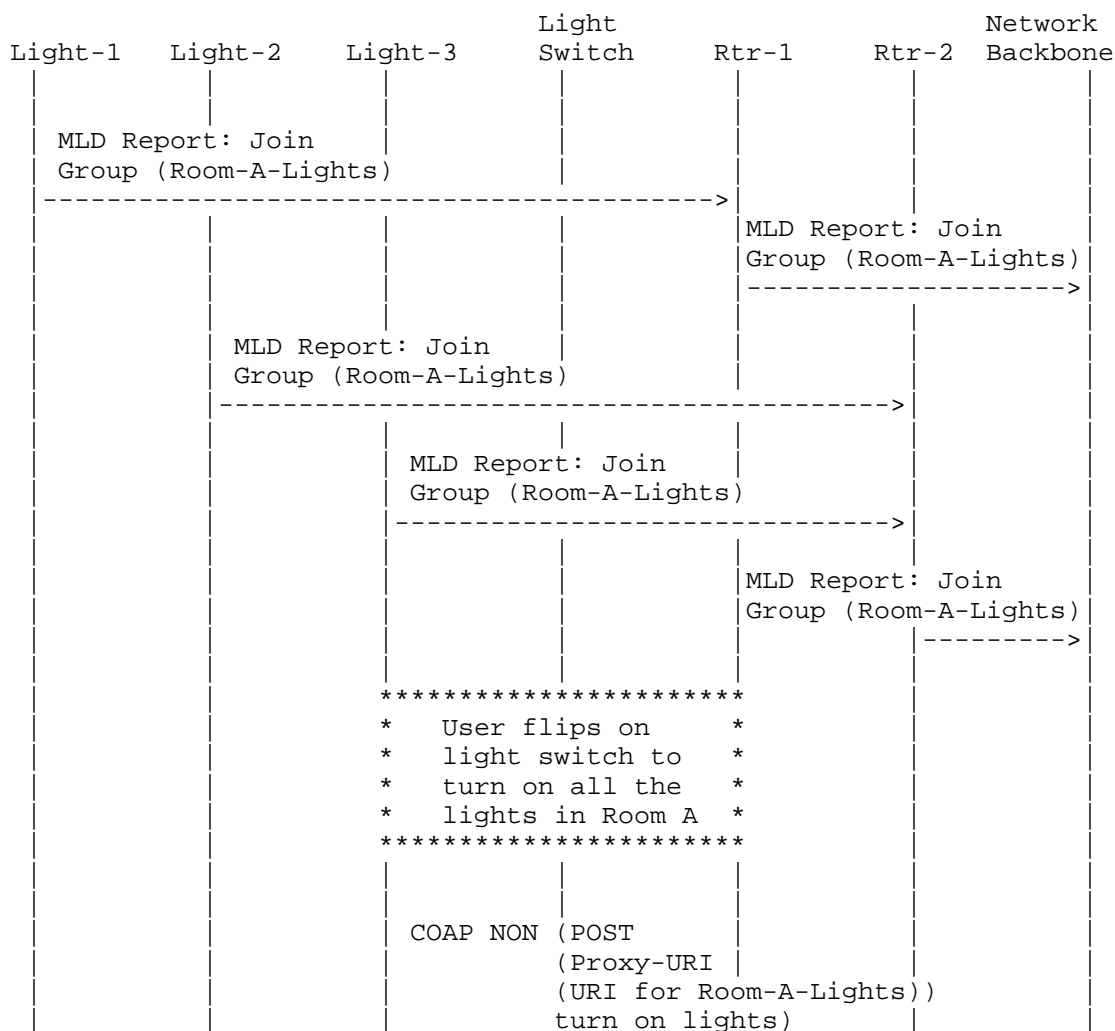


Figure 5: Network Topology of a Large Room (Room-A)

The corresponding protocol flow for an IP Multicast based CoAP Group Communication solution for the network shown in Figure 5 is shown in Figure 6. We assume the following steps occur before the illustrated flow:

1) Startup phase: 6LoWPANs are formed. IPv6 addresses assigned to all devices. The CoAP network is formed.

2) Commissioning phase (by applications): The IP multicast address of the group (Room-A-Lights) has been set in all the Lights. The URI of the group (Room-A-Lights) has been set in the Light Switch.



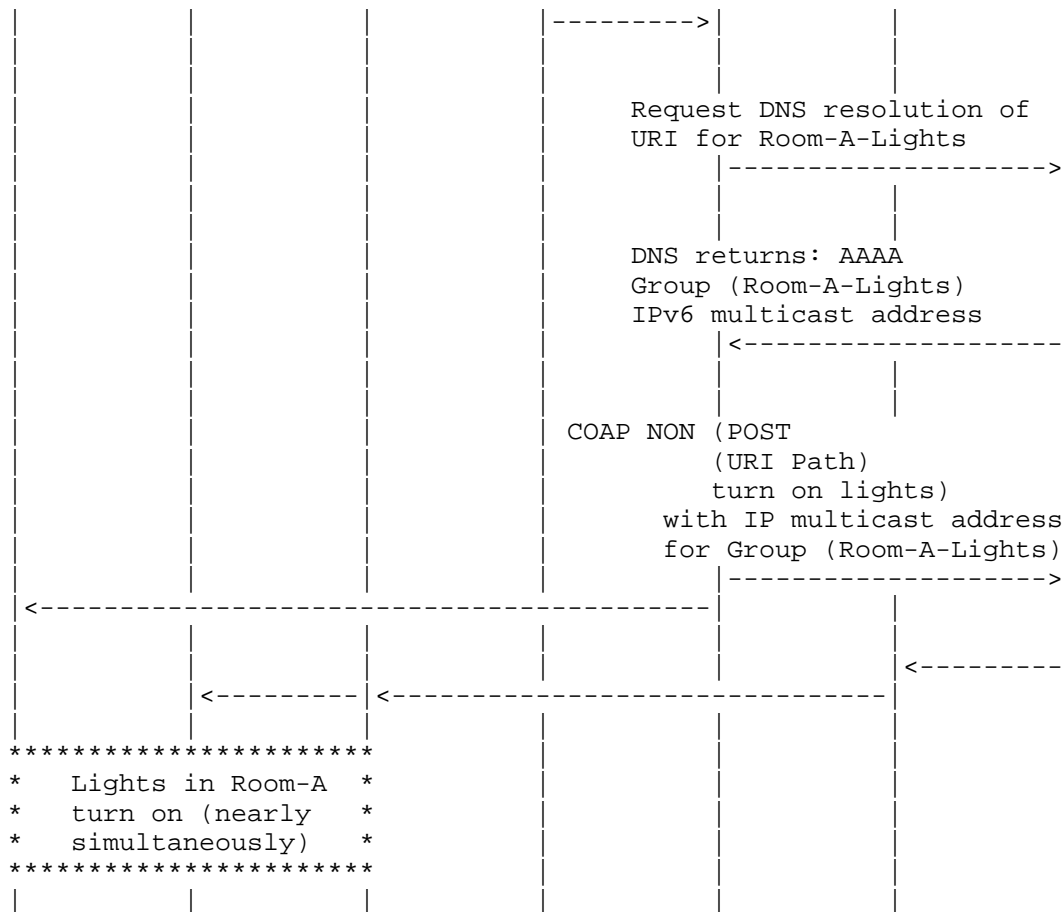


Figure 6: Turning on Lights in a Large Room (Room-A)

The indicated MLD Report messages are link-local multicast. In each LoWPAN, it is assumed that a multicast routing protocol in 6LRs will propagate the Join information over multiple hops to the 6LBR.

4.3. Implementation in Target Network Topologies

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

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

4.3.1. Single LLN Topology

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

4.3.1.1. Mesh-Under Multicast Routing

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

Because conceptually all nodes in the LLN are attached to a single link, there is in principle no need for nodes to announce their interest in multicast IP addresses via MLD (see Section 3.3.1). A multicast message to a specific IP destination, which is delivered to all 6LoWPAN nodes by the mesh routing algorithm, is accepted by the IP network layer of that node only if it is listening on that specific multicast IP address and port.

4.3.1.2. RPL Multicast Routing

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

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

The text in Section 12 of the RPL specification clearly implies that IP multicast packets are distributed using link-layer unicast transmissions, looking at the use of the word "copied" in this

section. Specifically in 6LoWPAN networks, this behavior conflicts with the requirement that IP multicast packets MUST be carried as link-layer 802.15.4 broadcast frames [RFC4944].

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

4.3.1.3. RPL Routers with Non-RPL Hosts

Now we consider the case that hosts exist in a RPL network that are not RPL-aware themselves, but use link-local RPL routers for their IP connectivity. Note that the current RPL specification [I-D.ietf-roll-rpl] considers this case to be out of scope. However, it was suggested on the ROLL mailing list that RPL could potentially be run with non-RPL-aware hosts but that it is simply not specified yet. Such non-RPL hosts can't advertize their IP multicast groups of interest via RPL DAO messages as defined above. Therefore in that case MLD can be used for such advertizements (State Change Report messages), with all or a subset of RPL routers acting in the role of MLD Routers as defined in [RFC3810]. However, as the MLD protocol is not designed specifically for LLNs it may be a burden for the constrained RPL router nodes to run the full MLD protocol. Alternatives are therefore proposed in Section 4.5.1.

4.3.1.4. Trickle Multicast Forwarding

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

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

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

discard multicast packets that are not of interest. Above properties seem to make Trickle especially efficient for cases where the multicast listener density is high and the number of distinct multicast groups relatively low.

4.3.1.5. Other Route-Over Methods

Other known IP multicast routing methods may be used, for example flooding or other to be defined methods suitable for LLNs. An important design consideration here is whether multicast listeners need to advertise their interest in specific multicast addresses, or not. If they do, MLD is a possible option but also protocol-specific means (as in RPL) is an option. See Section 4.5.1 for more efficient substitutes for MLD targeted towards a LLN context.

4.3.2. Single LLN with Backbone Topology

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

4.3.2.1. Mesh-Under Multicast Routing

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

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

To realize this "filter", nodes on the LLN may use MLD to announce their interest in specific multicast IP addresses to the 6LBR. One option is for the 6LBR to act in an MLD Router role on its LLN

interface. However, this may be too much of a "burden" for constrained nodes. Light-weight alternatives for MLD are discussed in Section 4.5.1.

4.3.2.2. RPL Multicast Routing

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

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

4.3.2.3. RPL Routers with Non-RPL Hosts

For the case that a RPL LLN contains non-RPL hosts, the solutions from the previous section can be used if in addition RPL routers implement MLD or "MLD like" functionality similar to as described in Section 4.3.1.3.

4.3.2.4. Trickle Multicast Forwarding

First, we consider the case of an IP multicast source node on the LLN (where all 6LRs support Trickle Multicast Forwarding) and IP multicast listeners that may be on the LLN and on the backbone. As Trickle will eventually deliver multicast packets also to a 6LBR, which acts as a Trickle Multicast router as well, the 6LBR can then forward onto the backbone in the ways described earlier in Section 4.3.2.1.

Second, for the case of an IP multicast source on the backbone and multicast listeners on both backbone and/or LLN, the 6LBR needs to forward multicast traffic from the backbone onto the LLN. Here, the aforementioned problem (Section 4.3.2.1) of potentially overloading the LLN with unwanted backbone IP multicast traffic appears again.

A possible solution to this is (again) to let multicast listeners advertize their interest using MLD as described in Section 4.3.2.1 or to use an MLD alternative suitable for LLNs as described in Section 4.5.1. However, following this approach requires possibly an

extension to Trickle Multicast Forwarding: the protocol should ensure that MLD-advertized information is somehow communicated to the 6LBR, possibly over multiple hops. MLD itself supports link-local communication only.

4.3.2.5. Other Route-Over Methods

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

4.3.3. Multiple LLNs with Backbone Topology

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

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

4.3.4. LLN(s) with Multiple 6LBRs

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

4.3.5. Conclusions

For all network topologies that were evaluated, CoAP group communication can be in principle supported with IP Multicast, making use of existing protocols. For the case of Trickle Multicast Forwarding, it appears that an addition to the protocol is required such that information about multicast listeners can be distributed towards the 6LBR. Opportunities were identified for an "MLD-like" or "MLD-lightweight" protocol specifically suitable for LLNs, which should interwork with regular MLD on the backbone network. Such MLD variants are further analyzed in Section 4.5.1.

4.4. HTTP/CoAP Interworking Aspects

The topic of HTTP unicast to CoAP multicast request proxying is treated in [I-D.castellani-core-http-mapping]. [TBD: only if needed more information will be added here in the future.]

4.5. Implementation Considerations

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

4.5.1. MLD Implementation on LLNs

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

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

A second option is to use MLD as-is but adapt the recommended parameter values such that operation on a LLN becomes more efficient.

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

A fourth option is now presented, which seems attractive in that it minimizes standardization, implementation and network communication overhead all at the same time. This option is to specify a new Multicast Listener Option (MLO) as an addition to the 6LoWPAN-ND [I-D.ietf-6lowpan-nd] protocol communication that is anyway ongoing between a 6LoWPAN host and router(s). This MLO is preferably designed to be maximally similar to the Address Registration Option (ARO), which minimizes the need for additional program code on constrained nodes. With an MLO, instead of registering a unicast IP address, a host "registers" its interest in a multicast IP address.

Unlike ARO, multiple MLO can be used in the same ND packet. A registration period is also defined just like in the ARO. MLO allows a host to persistently register as a listener to IP multicast traffic and to avoid the overhead of periodic multicast communication which is required for full MLD.

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

4.5.2. 6LBR Implementation

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

4.5.3. Backbone IP Multicast Infrastructure

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

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

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

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

capable router somewhere in the network, while for case 1) this may not be the case.

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

5. Security Considerations

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

Note that some of the requirements are marked optional. This means that, depending on the use case, these may be required or not. For this purpose each use case can be associated to a security profile as specified in [I-D.garcia-core-security]. The security profile prescribes what requirements should be taken into account for this profile. A mapping of these requirements to these profiles has not yet been done.

REQ1- Group communications data encryption: Important CoAP group communications shall be encrypted (using a group key) to preserve confidentiality. It shall also be possible to send CoAP group communications in the clear (i.e. unencrypted) for low value data.

REQ2- Group communications source data authentication: Important CoAP group communications shall be authenticated by verifying the source of the data (i.e. that it was generated by a given and trusted group member). It shall also be possible to send unauthenticated CoAP group communications for low value data.

REQ3- Group communications limited data authentication: Less important CoAP group communications shall be authenticated by simply verifying that it originated from one of the group members (i.e. without explicitly identifying the source node). This is a weaker requirement (but simpler to implement) than REQ2. It shall also be possible to send unauthenticated CoAP group communications for low value data.

REQ4- Group key management: There shall be a secure mechanism to

manage the cryptographic keys (e.g. generation and distribution) belonging to the group; the state (e.g. current membership) associated with the keys; and other security parameters.

REQ5- Use of Multicast IPsec: The CoAP protocol [I-D.ietf-core-coap] allows IPsec to be used as one option to secure CoAP. If IPsec is used as a way to security CoAP communications, then multicast IPsec [RFC5374] should be used for securing CoAP group communications.

REQ6- Independence from underlying routing security: CoAP group communication security shall not be tied to the security of underlying routing and distribution protocols such as PIM [RFC4601] and RPL [I-D.ietf-roll-rpl]. Insecure or inappropriate routing (including IP multicast routing) may cause loss of data to CoAP but will not affect the authenticity or secrecy of CoAP group communications.

REQ7- Interaction with HTTPS: The security scheme for CoAP group communications shall account for the fact that it may need to interact with HTTPS (Hypertext Transfer Protocol Secure) when a transaction involves a node in the general Internet (non-constrained network) communicating via a HTTP-CoAP proxy.

6. IANA Considerations

This document makes no request of IANA.

7. Conclusions

Three solutions for enabling CoAP group communications have been discussed.

Unreliable IP multicast as outlined in Section 3.3 is recommended to be adopted as the base solution for CoAP Group Communication on LLNs. This approach requires no standards changes to the IP multicast suite of protocols and it provides interoperability with IP multicast group communication on unconstrained backbone networks.

The proposals for group communication described in this draft should be considered for incorporation into the overall CoAP protocol specification.

8. Acknowledgements

Thanks to Peter Bigot, Carsten Bormann, Anders Brandt, Angelo

Castellani, Guang Lu, Salvatore Loreto, Kerry Lynn, Dale Seed, Zach Shelby, Peter van der Stok, and Juan Carlos Zuniga for their helpful comments and discussions that have helped shape this document.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3306] Haberman, B. and D. Thaler, "Unicast-Prefix-based IPv6 Multicast Addresses", RFC 3306, August 2002.
- [RFC3307] Haberman, B., "Allocation Guidelines for IPv6 Multicast Addresses", RFC 3307, August 2002.
- [RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, March 2004.
- [RFC3810] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.
- [RFC3956] Savola, P. and B. Haberman, "Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address", RFC 3956, November 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, April 2005.
- [RFC4286] Haberman, B. and J. Martin, "Multicast Router Discovery", RFC 4286, December 2005.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.
- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM):

Protocol Specification (Revised)", RFC 4601, August 2006.

- [RFC4605] Fenner, B., He, H., Haberman, B., and H. Sandick, "Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying")", RFC 4605, August 2006.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5374] Weis, B., Gross, G., and D. Ignjatic, "Multicast Extensions to the Security Architecture for the Internet Protocol", RFC 5374, November 2008.
- [RFC5771] Cotton, M., Vegoda, L., and D. Meyer, "IANA Guidelines for IPv4 Multicast Address Assignments", BCP 51, RFC 5771, March 2010.

9.2. Informative References

- [I-D.cheshire-dnsext-dns-sd]
Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-10 (work in progress), February 2011.
- [I-D.eggert-core-congestion-control]
Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.
- [I-D.ietf-6lowpan-hc]
Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams in Low Power and Lossy Networks (6LoWPAN)", draft-ietf-6lowpan-hc-15 (work in progress), February 2011.
- [I-D.ietf-6lowpan-nd]
Shelby, Z., Chakrabarti, S., and E. Nordmark, "Neighbor Discovery Optimization for Low Power and Lossy Networks (6LoWPAN)", draft-ietf-6lowpan-nd-17 (work in progress), June 2011.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), July 2011.

- [I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format",
draft-ietf-core-link-format-07 (work in progress),
July 2011.
- [I-D.ietf-core-observe]
Hartke, K. and Z. Shelby, "Observing Resources in CoAP",
draft-ietf-core-observe-02 (work in progress), March 2011.
- [I-D.shelby-core-coap-req]
Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R.
Kelsey, "CoAP Requirements and Features",
draft-shelby-core-coap-req-02 (work in progress),
October 2010.
- [I-D.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building
Control", draft-vanderstok-core-bc-04 (work in progress),
July 2011.
- [I-D.castellani-core-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and
E. Dijk, "Best practices for HTTP-CoAP mapping
implementation", draft-castellani-core-http-mapping-01
(work in progress), July 2011.
- [I-D.garcia-core-security]
Garcia-Morchon, O., Keoh, S., Kumar, S., Hummen, R., and
R. Struik, "Security Considerations in the IP-based
Internet of Things", draft-garcia-core-security-02 (work
in progress), July 2011.
- [I-D.ietf-roll-rpl]
Winter, T., Thubert, P., Brandt, A., Clausen, T., Hui, J.,
Kelsey, R., Levis, P., Pister, K., Struik, R., and J.
Vasseur, "RPL: IPv6 Routing Protocol for Low power and
Lossy Networks", draft-ietf-roll-rpl-19 (work in
progress), March 2011.
- [I-D.ietf-roll-trickle-mcast]
Hui, J. and R. Kelsey, "Multicast Forwarding Using
Trickle", draft-ietf-roll-trickle-mcast-00 (work in
progress), April 2011.
- [ID.goland-http-udp]
Goland, Y., "Multicast and Unicast UDP HTTP Messages",
1999,
<<http://tools.ietf.org/html/draft-goland-http-udp-01>>.

- [STUDY1] Lao, L., Cui, J., Gerla, M., and D. Maggiorini, "A Comparative Study of Multicast Protocols: Top, Bottom, or In the Middle?", 2005, <http://www.cs.ucla.edu/NRL/hpi/AggMC/papers/comparison_gi_2005.pdf>.
- [STUDY2] Banerjee, B. and B. Bhattacharjee, "A Comparative Study of Application Layer Multicast Protocols", 2001, <<http://wmedia.grnet.gr/P2PBackground/a-comparative-study-ofALM.pdf>>.

Authors' Addresses

Akbar Rahman (editor)
InterDigital Communications, LLC
Email: Akbar.Rahman@InterDigital.com

Esko Dijk (editor)
Philips Research
Email: esko.dijk@philips.com

Core
Internet-Draft
Intended status: Informational
Expires: May 3, 2012

B. Sarikaya
Huawei USA
Y. Ohba
Toshiba
R. Moskowitz
Verizon Business Systems
Z. Cao
China Mobile
R. Cragie
Pacific Gas and Electric
October 31, 2011

Security Bootstrapping of Resource-Constrained Devices
draft-sarikaya-core-sbootstrapping-03

Abstract

This document describes how to initially configure the network of resource constrained nodes securely, a.k.a., security bootstrapping. Bootstrapping architecture, communication channel and bootstrap security methods are described. System level objectives for security bootstrapping are stated followed by the protocols that can be used. Requirements on these protocols to be used as security bootstrapping protocols are identified.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 3, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	5
1.1. What is Bootstrapping?	5
1.2. Why IETF?	5
2. Bootstrapping Architecture	6
2.1. Areas of Bootstrapping	6
2.2. Architecture	7
3. Communications Channel	8
3.1. Supported Communication Channels	8
4. Bootstrap Security Method	9
4.1. None	9
4.2. Asymmetric with User Authentication, Followed by Symmetric	9
4.3. Asymmetric with Certificate Authority, Followed by Symmetric	9
4.4. Cryptographically Generated Address Based Address Ownership Verification	9
5. Bootstrapping Protocols	10
5.1. System Level Objectives	10
5.2. EAP Authentication Framework	11
5.3. PANA	13
5.4. HIP-DEX	14
5.5. 802.1X	15
6. Security Considerations	17
7. IANA Considerations	18
8. Contributors	18
9. Acknowledgements	18
10. References	18
10.1. Normative References	18
10.2. Informative References	19
Appendix A. Examples of Node Configuration	21
A.1. Smart Energy	22
A.1.1. Initial Meter Installation	22
A.1.2. Home Expansions	22
A.2. Consumer Products	22
A.2.1. Connecting DVD Remote to DVD Player	22
A.2.2. Adding a TV to a network with a DVD player and remote	22
A.2.3. Providing GPS Location Data	22
A.3. Commercial Building Automation	22
A.3.1. Light Installation	23
Appendix B. Example Exchanges	23
B.1. Smart Energy: Meter Manufacture	23
B.2. Smart Energy: Meter Installation	23
B.3. Smart Energy: Home Expansion	23
B.4. Consumer: Connecting DVD Remote to DVD Player	23
B.5. Consumer: Adding a TV to a network with a DVD player	

and remote	24
B.6. Consumer: Providing GPS Location Data	26
B.7. Commercial: Building Automation	26
Authors' Addresses	26

1. Introduction

Familiarity with constrained network types is assumed here. Documents produced in the 6LoWPAN, ROLL, and CoRE Working Groups (WGs) would be a useful reference for the reader. In particular RFC 4919 [RFC4919] from 6LoWPAN, RFC 5548 [RFC5548] and RFC 5673 [RFC5673] from ROLL, CoAP [I-D.ietf-core-coap] from CoRE, and a paper by Romer and Mattern [ROMER04]. Familiarity with application specific examples such as Zigbee or Smart Energy groups is assumed.

A summary of those will be presented, as far as network requirements are concerned. The general network requirements will be further concentrated into requirements surrounding only the bootstrapping issues.

A number of solutions which are currently in use will be presented. Requirements on each solution will be stated to enable their use as a security bootstrapping protocol.

1.1. What is Bootstrapping?

Node configuration is known as bootstrapping in this document. Bootstrapping is any processing required before the network can operate. Typically this will require a number of settings to be transferred between nodes at all layers. This could include anything from link-layer information (i.e., wireless channels, link-layer encryption keys) to application-layer information (i.e., network names, application encryption keys).

Bootstrapping is complete when settings have been securely transferred prior to normal operation in the network.

1.2. Why IETF?

The bootstrapping problem is not specific to any MAC or PHY. This problem exists across any two nodes which have no previous knowledge of each other. In particular, this problem is complicated when the nodes are resource-constrained and may not have an advanced user interface. The IETF is instrumental in defining standards which will be used by The Internet of Things. Ensuring these standards can be used across nodes and networks requires some form of bootstrapping which any node can use.

Existing standards will be used as much as possible in this document. The method proposed here should work across many different underlying layers. It could be used to allow two nodes on the same physical network to join at the physical layer, or allow two nodes on an incompatible physical network to join at the IPv6 layer.

2. Bootstrapping Architecture

2.1. Areas of Bootstrapping

In order to provide a flexible architecture, the bootstrapping method is split into five distinct areas and two distinct phases. The five areas are a 'user interface', 'bootstrap profile', 'security method', 'bootstrap protocol', and the 'communications channel'.

The phases are provisioning phase and bootstrapping phase. In the provisioning phase, statically configured parameters (e.g., certificates) needed for the bootstrapping phase is provisioned. In the bootstrapping phase, dynamically configured information is set up using the statically configured information provided in the provisioning phase.

The user interface provides both user input and user output. Simple nodes may only have a push-button and LED, more complex nodes may have a graphical display and keyboard. The user interface (which could be implemented as network management software graphical user interface running at the remote end) provides interaction between the user and bootstrapping methods. The user interface would be used during bootstrapping as an OOB channel. It may also be used to specify bootstrapping policies.

The user interface provides the interaction between the user and the bootstrap protocol. The user interface will vary depending on the capabilities of the node. Examples might include a push-button and LED on simple nodes, to full-blown graphical user interfaces. Note that a 'bootstrapping tool' used to initially deploy a network is just a special user interface. This allows a very uniform protocol in deployment and use of networks.

User interface is out-of-scope and will not be further discussed.

Two nodes communicate through some channel. For our purposes this is split into the 'control channel' and 'data channel'. The control channel is used for the bootstrap protocol, and the data channel is used during normal network operation. A node may support multiple control or data channels. When the control and data channels are the same, the bootstrapping is done In Band (IB). When the control and data channels are different, the bootstrapping is performed Out Of Band (OOB). An 802.15.4 network for instance would use an 802.15.4 control channel for IB bootstrapping, but a control channel of perhaps IrDA or USB for OOB bootstrapping.

The 'bootstrap profile', i.e. statically configured parameters during the provisioning phase, defines what information should be exchanged

during the process. A single node may run the protocol multiple times with different profiles. If the user wishes to associate a new lightswitch, the protocol is first run with the '802.15.4 Wireless Profile', through which it learns the channel and PAN-ID. The node then runs a 'Security Exchange Profile' to learn the needed encryption keys. Finally it runs a 'Lightswitch Association Profile' through which it learns which light to associate with.

An example of the 'bootstrap profile' attribute is the 'administrative domain name'. 'Bootstrap profiles' are required to be modified when the corresponding administrative domains are changed, a.k.a. recommissioning. In recommissioning, the domains are administratively repartitioned and nodes are therefore recommissioned.

The 'security method' defines supported security methods for bootstrapping. The supported security methods will depend on the control channel and bootstrap profile. In one node if the control channel is secure, then a simple clear-text security method is supported. For example when a physical connection between two nodes is used, the control channel is considered secure. However when the control channel is not secure, this clear-text security method is not supported. The 'bootstrap profile' additionally defines allowed security methods. Higher security nodes may outlaw ever performing a clear-text exchange, even if the control channel is deemed secure.

The 'bootstrap protocol' defines the actual messages exchanged during bootstrapping. The messages are used to transfer between nodes data, node information, and network state. The selected security method runs on top of the control channel, such as EAP-GPSK etc.

2.2. Architecture

Security bootstrapping architecture is structured in a hierarchy of nodes going from the least resource constraint to the most resource constraint. At the top there is a root node. The root node is called Coordinator or Trust Center in Zigbee and 6LowPAN Border Router (6LBR) in 6LowPAN ND.

At the next level there are interior Routers. Routers are able to run a routing protocol between other routers and the root. Routers are called 6LowPAN Routers (6BR) in 6LowPAN ND.

At the lowest level there are the nodes. The nodes do not run a routing protocol. They can connect to the nearest router over a single radio link. The nodes are called End Devices in Zigbee and hosts in 6LowPAN ND.

Routers first join the network as a node and go through security bootstrapping operations in order to create a Master Session Key (MSK). Next, routers execute routing protocol, e.g. [I-D.ietf-roll-rpl] specific steps to create session keys with their neighbors and to establish upstream and downstream next hop parents.

At each node hierarchy level described above, there are lower-layer and higher-layer protocols to bootstrap their ciphering keys, where the lower-layer refers to layers below IP layer including IEEE 802.15.4 MAC layer and LoWPAN adaptation layer and the higher-layer refers to IP layer and the above. In general, required bootstrapping procedures depend on the bootstrapping protocols to use. Section 5 describes the bootstrapping procedures where EAP (Extensible Authentication Protocol) [RFC3748] and other protocols are used as the bootstrapping protocols.

3. Communications Channel

The communications channel is the method used between two nodes to communicate. There are two main communication channels: the 'control' and 'data' channels. The control channel is used during bootstrapping, and the data channel is used during network operation.

3.1. Supported Communication Channels

There is no limit on what communications channels are supported. The following gives an example of several supported channels:

- o IEEE 802.15.4
- o Power-Line Communications
- o IrDA
- o RFID
- o Some simple physical link
- o Cellular
- o Ethernet
- o IPv6
- o Wi-Fi

Depending on the node's function, it may use different channels as

the data or control channel. Nodes may have multiple data and/or control channels as well.

4. Bootstrap Security Method

The bootstrap security method defines allowable security methods. A node may choose to support or use a subset of these methods. This is NOT the security architecture used for the application, but only the security used during bootstrapping. Typically some high-security method is used to generate a shared secret, which then switches to simpler symmetric encryption to secure the actual bootstrapping channel. The techniques negotiated should take advantage of hardware resources available, such as hardware encryption accelerators on an end node.

4.1. None

This is the simplest security method. No encryption or authentication is provided, messages are exchanged completely in clear-text. It is assumed some other layer provides security, such as a physical connection between devices.

4.2. Asymmetric with User Authentication, Followed by Symmetric

A Diffie-Hellman style key exchange is used to generate a shared secret. The authentication will be provided by the user, by confirming cryptographic signatures between two devices. With the shared secret generated through the DH, some symmetric encryption is used to secure the actual bootstrapping channel.

4.3. Asymmetric with Certificate Authority, Followed by Symmetric

Public key exchanges are used (aka: DH again), but with a Certificate Authority. Once a shared secret exists, symmetric encryption is used to secure the actual bootstrapping channel.

4.4. Cryptographically Generated Address Based Address Ownership Verification

A node may generate the global unique address using different techniques other than the stateless address autoconfiguration. For example, Cryptographically Generated Addresses (CGA) [RFC3972] is a type of global unique address that can be used to verify the address ownership. When the node uses CGA, it MUST execute SeND protocol [RFC3971]. In a 6LOWPAN network, a modified 6LOWPAN ND Protocol [I-D.ietf-6lowpan-nd] must be executed between the node and the edge router.

5. Bootstrapping Protocols

In this section we first present system level objectives that security bootstrapping protocols are expected to achieve. Next, we present EAP authentication framework and then describe three different protocols.

5.1. System Level Objectives

Authentication/ reauthentication: nodes joining the network MUST at the first place authenticate to the trust center. In order to achieve secure multi-hop routing, the node MUST authenticate to its upstream and downstream neighbors. A bootstrapping solution MUST support re-authentication of resource-constrained devices and re-keying of dynamically generated keys.

Data Confidentiality: the data communication between two endpoints MAY be encrypted using the derived key, avoiding being eavesdropped by a non-trusted third part.

Data Integrity: the data communication between two endpoints MUST NOT be altered by some intermediate nodes. The nodes should be able to detect the non-integral data.

Keys and key freshness: the keys used for data communication MUST have a lifetime, in order to keep their freshness. A bootstrapping solution MUST support both symmetric and asymmetric key authentication. If distribution of a key to be used for a resource-constrained device is required, a bootstrapping solution MUST support secure key distribution to prevent the key from eavesdropping, alternation and replay attacks.

Multi domain support: A bootstrapping solution MUST be able to allow resource-constrained devices that may be subscribed to different administrative domains to be connected to the same access network at the same time.

Multi domain support: A bootstrapping solution MUST be able to allow resource-constrained devices to be recommissioned. Recommissioning a device is defined to be (1) an resource-constrained device is administratively switched to a different domain, or (2) acting a new role with a different function set, or (3) both administrative domain and function set are modified.

Identities: A bootstrapping solution MUST be able to allow a resource-constrained device to use various types of identities used for authentication, including device identities, user identities or combinations of different types of identities. Also a bootstrapping

solution MUST be able to allow a resource-constrained device to change its identities used for authentication over time.

Authentication infrastructure: A bootstrapping solution MUST be able to operate with or without an authentication infrastructure.

5.2. EAP Authentication Framework

EAP is an authentication protocol that supports a number of authentication algorithms called EAP methods [RFC5247]. EAP messages can be transported in different layers: using 802.1X, EAP messages are carried in Layer 2 and using PANA in IP or Layer 3 between EAP peer and the authenticator. EAP messages between the authenticator and authentication server are carried using AAA protocols (RADIUS or Diameter). The associated AAA server address of each resource-constrained device is assigned by the domain administrator. In the recommissioning case, another AAA server is reassigned to devices by the domain administrator if the device is switched to another domain.

At the end of a successful EAP method execution a master session key (MSK) is generated at both the EAP peer and EAP server. Authenticator receives MSK from EAP server at the end of EAP method execution using key transport. MSK is used in deriving a session key between the node and the authenticator using a protocol called secure association protocol (SAP). Derivation of the session key terminates bootstrapping of a node.

Additional keying material derived between EAP client and server that is exported by the EAP method is called Extended Master Session Key (EMSK). EMSK is not used in session key derivation but it could be used for the needs of other applications in higher layer protocols.

In the architecture introduced in Section 2.2 the node or router is the peer and the root is the authenticator. When the supplicant and authenticator are one hop away the authenticator can be reached directly. However, this is rarely the case. In other cases the authenticator authenticates neighboring supplicants first. The router nodes that are authenticated become relay authenticators in the next phase and they relay authentication messages from the supplicants to the authenticator and vice versa. This continues until all nodes are authenticated.

EAP is a lock-step protocol, i.e. it executes in pairs of EAP-Request messages sent by the server and EAP-Response messages sent by the peer. At the end, the server indicates the status of authentication, usually by EAP-Success message which also carries the MSK. The first EAP-Request/Response pair is used for the server to request the identity and the peer to provide it. In the other pairs of EAP

exchanges EAP method is executed.

Several EAP methods have been standardized each for different purposes. To authenticate devices with certificates, EAP Transport Layer Security (TLS) v1.2 specified in [RFC5216] which supports certificate-based mutual authentication is used.

Zigbee Alliance's Smart Energy Profile 2.0 Application Protocol Specification [SE2.0] mandates each device to be factory programmed with a certificate. The certificate is bound to a unique network ID, e.g. the device's MAC address or EUI-64 address. During EAP-Identity exchange the EAP peer provides its EUI-64 address as an identity to EAP server. This enables the server to validate the device certificate.

There are three bootstrapping scenarios using EAP.

1. Use of EAP for bootstrapping link-layer security.

In this case, EAP is used for network access authentication to bootstrap link-layer ciphering. Security for higher-layer (i.e., IP layer and above) protocols is bootstrapped from an IB or OOB mechanism other than EAP.

2. Use of EAP for bootstrapping higher-layer security.

In this case, EAP is used as an OOB mechanism for higher-layer authentication to bootstrap ciphering keys for one or more higher-layer protocols independently of network access authentication. When bootstrapping Constrained Application Protocol (CoAP) security with DTLS protection, a PSK (Pre-Shared Key) credential in the combined usage of DTLS (Datagram Transport Layer Security) [RFC4347] and PSK mode of TLS [RFC4279] is derived from EAP key material and DTLS ciphering keys are generated as a result of a successful DTLS handshake. Similarly, when bootstrapping CoAP security with IPsec ESP protection, a PSK credential of IKEv2 [RFC5996] is derived from EAP key material and IPsec ESP ciphering keys are generated as a result of a successful IKEv2 handshake.

The ability to bootstrap multiple higher-layer protocols from a single execution of PANA authentication is important to save the computational resources for resource-constrained devices especially where public-key based authentication is used.

3. Use of EAP for bootstrapping both link-layer and higher-layer security.

This case is the combination of the other two cases, and the most optimized way for bootstrapping resource-constrained devices. This case is only applicable where both the network access authentication and the higher-layer authentication use the same authentication server with the same authentication credentials.

The second and third scenarios are generally referred to as Single Sign-On in Section 4.2.2.2 of [NISTIR7628VOL1], where the root keys for higher-layer protocols can be derived from EAP EMSK (Extended Master Session Key) as an USRK (Usage-Specific Root Key) [RFC5295].

5.3. PANA

PANA (Protocol for carrying Authentication for Network Access) [RFC5191] defines an EAP transport over UDP where a PANA Client (PaC) is an EAP peer and a PANA Authentication Agent (PAA) is an EAP authenticator.

PANA can achieve the authentication, key freshness and data confidentiality objectives of security bootstrapping.

Multi domain operation is intrinsically supported due to the use of EAP and AAA.

Even though PANA architecture consisting of PaC, PAA and AAA Server is generic enough to be used in security bootstrapping, the architecture introduced in Section 2.2 requires a new element called PANA Relay Element (PRE). PRE is needed to enable PANA messaging between a PaC and PAA because the two nodes cannot reach each other by means of regular IP routing since only a link-local IPv6 address can be used by PaC prior to the completion of a successful authentication.

PRE which is one hop away from PaC receives PANA messages and relays the message contents (payload) by encapsulating it in a message parameter called Attribute Value Pair (AVP). PRE also needs to send header contents such as PaC's IP address and UDP port number in a different AVP. PRE has IP routing established with PAA which could be several hops away. PAA sends its reply messages in which the payload is encapsulated in an AVP. It also adds the AVP containing PaC's IP address and UDP port number. PRE creates a link local PDU using these AVPs and sends it to PaC [I-D.ohba-pana-relay].

The requirements for the use of PANA as a bootstrapping protocol can be stated as follows:

- o A new entity called PANA Relay Element may be added to the PANA architecture. Behaviour of PANA Relay Element needs to be

defined. New AVPs needed for PANA Relay Element operation for properly relaying messages from the client to the authenticator and vice versa are required to be specified.

- o An extension to PANA to securely distribute keys from the PANA Authentication Agent to the PANA Client using AES Key Wrap with Padding algorithm needs to be defined. This is needed in order to use PANA for multicast group key distribution.

5.4. HIP-DEX

[RFC4423] introduces the Host Identity Protocol (HIP) where the Host Identity (HI) is a Cryptographic key (RSA, DSA, or ECC). A 128-bit length Host Identity Tag (HIT) is derived from the HI (hashed) and functions as an IPv6 address (/128 prefix) for applications. A four-packet Peer-to-Peer Host Identity Protocol Base EXchange (HIP BEX) establishes a security association (SA, similar to IKE), indexed by the HITs, but independent of the IP address. So HIP can be considered as a shim layer between the transport(TCP/UDP) and IP, providing authentication, data confidentiality, mobility in one basket.

As with IKE, HIP is typically used as a Key Management Protocol (KMP) for ESP. However, HIP is independent of IP and ESP and can be used for a KMP for any secure communication protocol at any level. Thus HIP can provide keying material for the MAC, IP, and Transport layers. HIP can be run directly over the MAC in an Ethernet Frame or within an Information Element in a MAC control frame. HIP can be run over UDP, traversing firewalls, and push keys over to Transport security protocols like SRTP or (D)TLS. Further, HIP could start on the MAC, then be enveloped over UDP after the first link.

The HIP-BEX involves many crypto primitives that are difficult to run on constrained nodes. HIP Diet Exchange (HIP-DEX) [I-D.moskowitz-hip-rg-dex] is a way to make HIP lightweight. Basically, HIP-DEX a variant of the HIP-BEX specifically designed to use as few crypto primitives as possible yet still provide the same class of security features as HIP-BEX.

HIP-DEX can be used for mutual authentication between two endpoints. After mutual authentication, the two endpoints establish a shared secret, which is fresh and fed into the encryption algorithm for data confidentiality. So HIP-DEX can achieve the authentication, key freshness and data confidentiality objectives of security bootstrapping.

When a node wants to authenticate to the network using HIP and Diet-HIP, it should be able to complete the HIP-BEX or HIP-DEX with the

trust anchor or some delegate. In HIP, it does not matter how many domains, and nodes can authenticate each other as long as they have the secret.

In the architecture introduced in Section 2.2 the node and router could be the HIP end-points. Or the router could be a HIP Rendezvous Server, with the node registering to the rendezvous server (RVS) [RFC5204] to be reachable by other nodes. Typically the initial interaction will have the node be the HIP DEX Initiator with the router being the Responder. Using the RVS function on a Router any node could be the Initiator with any other Responder node. As long as there is IP connectivity between the Initiator and Responder, they can be multiple hops away from each other. Alternatively if the first hop from the Initiator has knowledge of the IP address of the Responder, it can act as a MAC/IP gateway for HIP.

An important requirement for the HIP-DEX to work in the architecture, the initiator should be able to get the IP address of the responder or have a gateway function as a forwarder. The IP address can be obtained from a 3rd party source like DNS or Distributed Hash Table (DHT), from local configuration, or from an RVS.

5.5. 802.1X

IEEE 802.1X defines how EAP packets can be transported over in Layer 2, i.e. Ethernet frames [802.1x] by encapsulating EAP packets into EAP Over Lan (EAPOL) frames between EAP peer, called supplicant and the authenticator. EAPOL can also be used in 802.11 wireless links.

To enable IEEE 802.15.4 devices to use EAP authentication, EAP packets encapsulated in EAPOL frames can be carried as payload in 802.15.4 data frames [802.15.4]. EAPOL is well defined and widely used and it lends itself to be easily carried in 802.15.4 data frames. For this, Frame Type subfield of the Frame Control Field of IEEE 802.15.4 MAC header needs to be set to a special value to indicate the type of the payload, i.e. 802.1X encapsulated EAP packets. EAPOL packets are encoded following common EAPOL PDU structure defined in [802.1x] into the data payload field of 802.15.4 data frames.

Authentication proceeds as follows: authenticator authenticates the supplicants that are on the next hop first. This enables a secure link between the authenticator and these first-hop nodes. The architecture introduced in Section 2.2 requires a new entity called Relay Authenticator. First-hop nodes or router become Relay Authenticators in the next phase of authentication. Relay Authenticators tunnel EAPOL frames to the authenticator in the secure link established. This way all the supplicants are gradually

authenticated.

After the keys are established from a successful EAP method (such as PSK mode of TLS), the node runs neighbor discovery protocol to get an IPv6 address assigned [I-D.ietf-6lowpan-nd]. Data transfer can be secured using DTLS or IPSec. Keys derived from EAP TLS are used in either generating DTLS ciphering keys after a successful DTLS handshake or IPSec ESP ciphering keys after a successful IKEv2 handshake.

802.1X can achieve the authentication, key freshness and data confidentiality objectives of security bootstrapping.

Multi domain operation is intrinsically supported due to the use of EAP and AAA. In order to support a device recommissioning case whereby the device's administrative domain is modified, after a new AAA server address assigned and a successful 802.1X method execution, the old set of device 'name and password' MUST be overwritten into the device by a new set of 'name and password' that are assigned by the domain administrator. The device MUST be rebooted to execute EAP method again for authentication and a new master session key MUST be generated.

The requirements for the use of 802.1X defined EAPOL as a bootstrapping protocol can be stated as follows:

- o A special value in the Frame Type subfield of the Frame Control Field of IEEE 802.15.4 MAC header to indicate the type of the payload,
- o Link Layer Multicast Group addresses for 802.15.4 corresponding to EAPOL Group Address Assignments defined in Table 11.1 of [802.1x], especially to be used in EAPOL-Start packet.
- o Which MAC frames of beacon, data, acknowledgment and MAC command as defined in [802.15.4] with what security levels are mapped to controlled port, which MAC frames with what security levels are mapped to uncontrolled port and which MAC frames are never mapped to any of controlled/uncontrolled port (i.e., the payload of those frames are used by the MAC-layer itself and never used by upper layers).
- o A new entity called Relay Authenticator may be added to the 802.1x architecture. Behaviour of Relay Authenticator needs to be defined.

6. Security Considerations

When security bootstrapping resource constraint nodes is undertaken, several attacks are possible and security bootstrapping methods described in this document do not protect the nodes against such attacks. These attacks are similar to the ones described in [RFC3971] and mainly stem from unsecured link layer. Link layer must be secured on each node before the node can begin security bootstrapping.

If a bootstrapping protocol does not rely on a pre-shared key for peer authentication, it must rely on an online or offline third-party (e.g., an authentication server, a key distribution center in Kerberos, a certification authority in PKI, a private key generator in ID-based cryptography and so on) to prevent man-in-the-middle attacks during peer authentication. Depending on use cases, a resource-constrained device may not always have access to an online third-party for peer authentication.

Depending on use cases, a bootstrapping protocol may deal with authorization separately from authentication in terms of timing and signaling path. For example, two resource-constrained devices A and B may perform mutual authentication using authentication credentials provided by an offline third-party X whereas resource-constrained device A obtains authorization for running a particular application with resource-constrained device B from an online third-party Y before or after the authentication. In some use cases, authentication and authorization are tightly coupled, e.g., successful authentication also means successful authorization. A bootstrapping protocol supports various types of authentication and authorization or different bootstrapping protocols may be used for different types of authentication and authorization.

If authorization information includes cryptographic keys, a special care must be taken for dealing with the keys, e.g., guidelines for AAA-based key management are described in [RFC4962]. A recommissioning use case may require revocation and re-installation of authentication credentials (i.e., a certificate or a shared secret and identity information, etc.) to a large number of resource-constrained devices that are already deployed. Re-installation of authentication credentials must be as secure as the initial installation regardless of whether the re-installation is done manually or automatically.

If resource-constrained devices use a multicast group key for peer authentication or message authentication or encryption, the group key must be securely distributed to the current members of the group for both initial key distribution and key update. Protocols designed for

group key management such as GSAKMP [RFC4535], GDOI [RFC3547] and MIKEY [RFC3830] may be used for group key distribution. Alternatively, key wrap attributes for securely encapsulating group key may be defined in network access authentication protocols such as PANA [RFC5191] and EAP-TTLSv0 [RFC5281]. Those protocols use an end-to-end, point-to-point communication channel with a pair-wise security association between a key distribution center and each key recipient. Further considerations may be needed for more efficient group key management to support a large number of resource-constrained devices.

7. IANA Considerations

This memo includes no request to IANA.

8. Contributors

The people listed below have made significant text contributions to this document.

Colin O'Flynn

colin.oflynn@atmel.com

9. Acknowledgements

Thanks to Zach Shelby for editing, comments, and overall assistance. Special thanks also to Rene Struik, Carsten Borman, Gary Yang and Alper Yegin for their comments that helped us improve the writing.

10. References

10.1. Normative References

- [802.15.4] IEEE Std 802.15.4-2006, "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)", September 2006.
- [802.1x] IEEE Std 802.1X-2010, "IEEE 802.1X Port-Based Network Access Control", February 2010.
- [RF4CE] ZigBee Alliance, "Zigbee RF4CE Specification Version 1.00", March 2009.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.
- [RFC5191] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.
- [RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, March 2008.
- [RFC5548] Dohler, M., Watteyne, T., Winter, T., and D. Barthel, "Routing Requirements for Urban Low-Power and Lossy Networks", RFC 5548, May 2009.
- [RFC5673] Pister, K., Thubert, P., Dwars, S., and T. Phinney, "Industrial Routing Requirements in Low-Power and Lossy Networks", RFC 5673, October 2009.
- [ROMER04] Romer, K. and F. Mattern, "The design space of wireless sensor networks", IEEE Wireless Communications, vol. 11, no. 6, pp. 54-61, December 2004.
- [SE2.0] ZigBee Alliance, "Smart Energy Profile 2.0 Technical Requirements Document", April 2010.

10.2. Informative References

- [C1222] American National Standard, "Protocol Specification For Interfacing to Data Communication Networks", ANSI C12.22-2008, 2008.
- [I-D.ietf-6lowpan-nd] Shelby, Z., Chakrabarti, S., and E. Nordmark, "Neighbor Discovery Optimization for Low Power and Lossy Networks (6LoWPAN)", draft-ietf-6lowpan-nd-18 (work in progress), October 2011.
- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank,

"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-07 (work in progress), July 2011.

[I-D.ietf-roll-rpl]

Winter, T., Thubert, P., Brandt, A., Clausen, T., Hui, J.,
Kelsey, R., Levis, P., Pister, K., Struik, R., and J.
Vasseur, "RPL: IPv6 Routing Protocol for Low power and
Lossy Networks", draft-ietf-roll-rpl-19 (work in
progress), March 2011.

[I-D.moskowitz-hip-rg-dex]

Moskowitz, R., "HIP Diet EXchange (DEX)",
draft-moskowitz-hip-rg-dex-05 (work in progress),
March 2011.

[I-D.ohba-pana-keywrap]

Cragie, R., Duffy, P., Ohba, Y., and A. Yegin, "Protocol
for Carrying Authentication for Network Access (PANA)
Extension for Key Wrap", draft-ohba-pana-keywrap-04 (work
in progress), September 2011.

[I-D.ohba-pana-relay]

Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., and A.
Yegin, "Protocol for Carrying Authentication for Network
Access (PANA) Relay Element", draft-ohba-pana-relay-03
(work in progress), February 2011.

[NISTIR7628VOL1]

The Smart Grid Interoperability Panel - Cyber Security
Working Group, "Guidelines for Smart Grid Cyber Security:
Vol. 1, Smart Grid Cyber Security Strategy, Architecture,
and High-Level Requirements", NISTIR 7628, vol. 1, 2010.

[RFC3547] Baugher, M., Weis, B., Hardjono, T., and H. Harney, "The
Group Domain of Interpretation", RFC 3547, July 2003.

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K.
Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830,
August 2004.

[RFC3971] Arkko, J., Kempf, J., Zill, B., and P. Nikander, "SEcure
Neighbor Discovery (SEND)", RFC 3971, March 2005.

[RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)",
RFC 3972, March 2005.

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites
for Transport Layer Security (TLS)", RFC 4279,

December 2005.

- [RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.
- [RFC4423] Moskowitz, R. and P. Nikander, "Host Identity Protocol (HIP) Architecture", RFC 4423, May 2006.
- [RFC4535] Harney, H., Meth, U., Colegrove, A., and G. Gross, "GSAKMP: Group Secure Association Key Management Protocol", RFC 4535, June 2006.
- [RFC4962] Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, July 2007.
- [RFC5204] Laganier, J. and L. Eggert, "Host Identity Protocol (HIP) Rendezvous Extension", RFC 5204, April 2008.
- [RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.
- [RFC5281] Funk, P. and S. Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, August 2008.
- [RFC5295] Salowey, J., Dondeti, L., Narayanan, V., and M. Nakhjiri, "Specification for the Derivation of Root Keys from an Extended Master Session Key (EMSK)", RFC 5295, August 2008.
- [RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010.

Appendix A. Examples of Node Configuration

Before any detail on methods is explored, the following section will provide various examples this document could cover. Exact requirements will be brought forward in subsequent sections. For the reader's general understanding this section is placed to give an idea of an acceptable usage scenario.

A.1. Smart Energy

A.1.1. Initial Meter Installation

The meter is initially loaded with code and network keys through a physical interface at the factory. The meter is installed at a customers home, and configured by the installer through the backbone link (via GSM modem, etc). Both operations can be performed through methods defined herein.

A.1.2. Home Expansions

The user wishes to join a thermostat onto the network. They press a button on the thermostat, which enters join mode. They press a button on the smart meter, which allows nodes to join the network. The devices both have displays, so they display a certain number which the user verifies match on both devices. The thermostat has now securely joined the network.

A.2. Consumer Products

A.2.1. Connecting DVD Remote to DVD Player

The user pushes a join button on the DVD remote and DVD player. The devices find each other, and blink in unison to indicate to the user which two devices will join. The user presses the button to confirm this, and the two devices are now joined together.

A.2.2. Adding a TV to a network with a DVD player and remote

The user then presses the join button on the DVD player and TV. The devices again find each other and blink in unison, with the addition that the remote control also blinks to indicate it is present in the network.

A.2.3. Providing GPS Location Data

A user has a simple GPS receiver (that has no user interface) they wish to broadcast location data with. The user switches on their camera, and enters a PIN from the base of the GPS. The user can now view GPS information such as satellite health from their camera. In addition photos are automatically tagged with location information.

A.3. Commercial Building Automation

A.3.1. Light Installation

The electrician installs the light fixture. Each light has a barcode printed on it. They use a handheld barcode scanner tool, which acts as the commissioning tool. When they scan a barcode with the tool, the tool asks the electrician to enter some additional information such as light fixture location. The tool securely registers the light fixture on the network, along with setting parameters inside the light fixture.

Appendix B. Example Exchanges

The following details how the protocol handles certain conditions and situations through examples. Note that each example is a more detailed description of the examples in Appendix A.

B.1. Smart Energy: Meter Manufacture

B.2. Smart Energy: Meter Installation

B.3. Smart Energy: Home Expansion

B.4. Consumer: Connecting DVD Remote to DVD Player

Supported User Interface Profiles

Profile	DVD Player	Remote Control
none	Y	Y
simple	Y	Y
numerical	Y	N
alphanumeric	Y	N
Graphical	Y	N

Supported Bootstrap Transport Layers

Layer	DVD Player	Remote Control
Physical	Y	Y
802.15.4	Y	Y
IrDA	Y	N

Supported Security Methods

Method	DVD Player	Remote Control
None	Y	Y
EAP	Y	N
Asymmetric, User	Y	Y
Asymmetric, CA	Y	N

The DVD player and remote control have a number of ways in which they could be joined together. The remote control does not have any unique identifier printed on it, thus no pre-shared key can be identified. This leaves either an unsecure joining method, or some asymmetric security method.

The remote control has a button on it for 'join', as does the DVD player. The user pushes the button on the DVD player, and then pushes the button on the remote control. Based on the UI profile, this causes the following to occur:

- o DVD Player scans for existing network in advertise mode. Finding none, it starts a new network and that network enters advertise mode.
- o The DVD remote scans for a network, and then finds the DVD player's network.
- o The devices generate a shared secret (ie: Diffie-Hellman), and both blink their LED in a unique pattern based on this shared secret.
- o The user confirms both devices are blinking the same pattern, as both LEDs are blinking in unison.
- o The DVD player displays 'JOIN OK' on it's LCD panel.

B.5. Consumer: Adding a TV to a network with a DVD player and remote

This network will have three devices: a TV, a DVD Player, and a Remote Control. The user will run the bootstrap protocol between the TV and Remote Control in this example, although it could also be run between the TV and DVD player.

Supported User Interface Profiles

Profile	TV	Remote Control
none	Y	Y
simple	Y	Y
numerical	Y	N
alphanumeric	Y	N
Graphical	Y	N

Supported Bootstrap Transport Layers

Layer	TV	Remote Control
Physical	Y	Y
802.15.4	Y	Y
IrDA	Y	N

Supported Security Methods

Method	TV	Remote Control
None	Y	Y
EAP-GPSK	Y	N
Asymmetric, User	Y	Y
Asymmetric, CA	Y	N

The TV and remote control have a number of ways in which they could be joined together. The remote control does not have any unique identifier printed on it, thus no pre-shared key can be identified. This leaves either an unsecure joining method, or some asymmetric security method.

The remote control has a button on it for 'join', as does the TV. In this example two sequence will be considered: where the TV button is pressed first, and where the remote control button is pressed first.

If the TV join button is pressed first:

- o TV scans for existing networks in advertise mode. Finding none, it starts a new network and that network enters advertise mode.

- o The remote scans for a network, and then finds the TV's network.
- o The remote informs the TV it is on an existing network, and thus will require the TV to join this network.
- o The devices generate a shared secret, and both blink their LED in a unique pattern.
- o The DVD player in addition blinks, so the user is informed that if they confirm the join action the resulting network will have all three devices in it.
- o The user confirms both devices are blinking the same pattern, as both LEDs are blinking in unison.
- o The TV displays 'JOIN OK' onscreen, along with any information about the network it just joined.

If the remote control join button is pressed first:

- o Remote control scans for existing networks in advertise mode. Finding none, it advertises it's network.
- o The TV scans for a network, and then finds the remote control's network.
- o The devices generate a shared secret, and both blink their LED in a unique pattern.
- o The DVD player in addition blinks, so the user is informed that if they confirm the join action the resulting network will have all three devices in it.
- o The user confirms both devices are blinking the same pattern, as both LEDs are blinking in unison.
- o The TV displays 'JOIN OK' onscreen, along with any information about the network it just joined.

B.6. Consumer: Providing GPS Location Data

B.7. Commercial: Building Automation

Authors' Addresses

Behcet Sarikaya
Huawei USA
1700 Alma Dr. Suite 500
Plano, TX 75075

Email: sarikaya@ieee.org

Yoshihiro Ohba
Toshiba
Tokyo, Japan

Email: yoshihiro.ohba@toshiba.co.jp

Robert Moskowitz
Verizon Business Systems
15210 Sutherland
Oak Park, MI 48237

Email: rgm@labs.htt-consult.com

Zhen Cao
China Mobile
Beijing, China

Email: caozhen@chinamobile.com

Robert Cragie
Pacific Gas and Electric
89 Greenfield Crescent
Wakefield, UK WF4 4WA

Email: robert.cragie@gridmerge.com

CoRE
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2012

Z. Shelby
Sensinode
S. Krco
Ericsson
November 1, 2011

CoRE Resource Directory
draft-shelby-core-resource-directory-02

Abstract

In many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources. This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to registrar, maintain, lookup and remove resources descriptions. Furthermore, new link attributes useful in conjunction with an RD are defined.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 4, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
2. Terminology	3
3. Architecture and Use Cases	4
3.1. Use Case: Cellular M2M	5
3.2. Use Case: Home and Building Automation	6
4. Resource Directory Interfaces	6
4.1. Discovery	7
4.2. Registration	8
4.3. Update	10
4.4. Validation	12
4.5. Removal	13
4.6. Lookup	14
5. New Link-Format Attributes	16
5.1. Resource Instance 'ins' attribute	16
5.2. Export 'exp' attribute	16
6. Security Considerations	17
7. IANA Considerations	17
8. Acknowledgments	17
9. Changelog	17
10. References	18
10.1. Normative References	18
10.2. Informative References	18
Authors' Addresses	18

1. Introduction

The Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation [I-D.shelby-core-coap-req].

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC5988]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [I-D.ietf-core-link-format]. This specification however only describes how to discover resources from the web server that hosts them by requesting /.well-known/core. In many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to registrar, maintain, lookup and remove resources descriptions. Furthermore, new link attributes useful in conjunction with a Resource Directory are defined. Although the examples in this document show the use of these interfaces with CoAP [I-D.ietf-core-coap], they may be applied in an equivalent manner to HTTP [RFC2616].

2. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [I-D.ietf-core-link-format]. Readers should also be familiar with the terms and concepts discussed in [I-D.ietf-core-coap]. This specification makes use of the following additional terminology:

Resource Directory

An web entity that stores information about web resources and implements the REST interfaces defined in this specification for registration and lookup of those resources.

Domain

In the context of a Resource Directory, a domain is a logical grouping of end-points. All end-point within a domain MUST be unique.

End-point

An end-point (EP) is a term used to describe a web server or client in [I-D.ietf-core-coap]. In the context of this specification an end-point is used to describe a web server that registers resources to the Resource Directory. During registration the end-point is identified by a combination of the Host and Instance fields.

Host

In the context of this specification, a Host name can be given to the device that is registering.

Instance

In the context of this specification, the Instance is used when registering to differentiate between multiple web servers running on the same device.

3. Architecture and Use Cases

The resource directory architecture is shown in Figure 1. A Resource Directory (RD) is used as a repository for Web Links [RFC5988] about resources hosted on other web servers, which are called end-points (EP). An end-point is a web server associated with a port, thus a physical node may host one or more end-points. The RD implements a set of REST interfaces for end-points to register and maintain sets of Web Links (called resource directory entries), for the RD to validate entries, and for clients to lookup resources from the RD. End-points themselves can also act as clients. An RD can be logically segmented by the use of Domains. The domain an end-point is associated with can be defined by the RD, an outside entity or by the EP during registration.

End-points are assumed to proactively register and maintain resource directory entries on the RD, which are soft state and need to be periodically refreshed. An EP is provided with interfaces to register, update and remove a resource directory entry. Furthermore, a mechanism to discover a RD using the CoRE Link Format is defined.

It is also possible for an RD to proactively discover Web Links from EPs and add them as resource directory entries, or to validate existing resource directory entries. A lookup interface for discovering any of the Web Links held in the RD is provided using the CoRE Link Format.

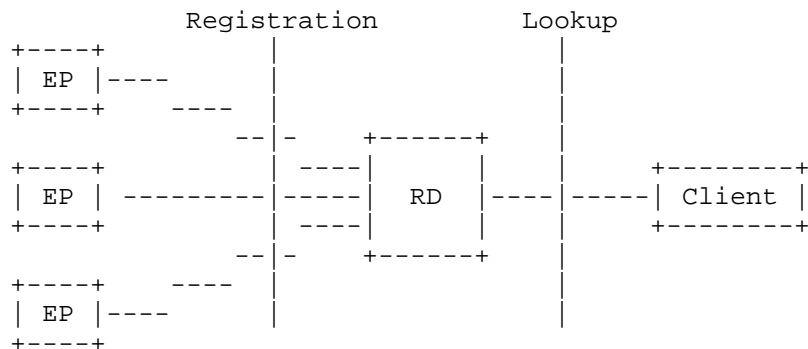


Figure 1: The resource directory architecture.

3.1. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users, i.e. machines. The machines are connected directly to a mobile network using appropriate embedded air interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. From the system design point of view, the ambition is to design horizontal solutions that can enable utilization of machines in different applications depending on their current availability and capabilities as well as application requirements, thus avoiding silo like solutions. One of the crucial enablers of such design is the ability to discover resources (machines - End Points) capable of providing required information at a given time or acting on instructions from the end users.

In a typical scenario, during a boot-up procedure (and periodically afterwards), the machines (EPs) register with a Resource Directory (for example EPs installed on vehicles enabling tracking of their position for the fleet management purposes and monitoring environment parameters) hosted by the mobile operator or somewhere else in the network, submitting a description of own capabilities. Due to the usual network configuration of mobile networks, the EPs attached to the mobile network do not have routable addresses. Therefore, a remote server is usually used to provide proxy access to the EPs.

The address of each (proxy) EP on this server is included in the resource description stored in the RD. The users, for example mobile applications for environment monitoring, contact the RD, look-up the EPs capable of providing information about the environment using appropriate set of tags, obtain information on how to contact them (URLs of the proxy server) and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide a set of credentials along with the appropriate tags to the RD to look-up for EPs deployed on the vehicles the application is responsible for.

3.2. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of M2M web services. The use of CoRE in home automation across multiple subnets is described in [I-D.brandt-coap-subnet-discovery] and in commercial building automation in [I-D.vanderstok-core-bc]. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery. Both home and building automation involve peer-to-peer interactions between end-points, and involve battery-powered sleeping devices.

The exporting of resource information to other discovery systems is also important in these automation applications. In home automation there is a need to interact with other consumer electronics, which may already support DNS-SD, and in building automation larger resource directories or DNS-SD covering multiple buildings.

4. Resource Directory Interfaces

This section defines the REST interfaces between an RD and end-points, and a lookup interface between an RD and clients. Although the examples throughout this section assume use of CoAP [I-D.ietf-core-coap], these REST interfaces can also be realized using HTTP [RFC2616]. An RD implementing this specification MUST support the discovery, registration, update, removal and lookup interfaces defined in this section and MAY support the validation interface. For the purpose of validation, an end-point implementing this specification SHOULD support Etag validation on /.well-known/core.

4.1. Discovery

Before an end-point can make use of an RD, it must first know its location and optionally the path of the RD root resource. There can be several mechanisms for discovering the RD including assuming a default location (e.g. on an Edge Router in a LoWPAN), by assigning an anycast address to the RD, using DHCP, or by discovering the RD using the CoRE Link Format. This section defines discovery of the RD using the well-known interface of the CoRE Link Format [I-D.ietf-core-link-format] as the required mechanism. It is however expected that RDs will also be discoverable via other methods depending on the deployment.

Discovery is performed by sending either a multicast or unicast GET request to `/.well-known/core` and including a Resource Type (rt) parameter [I-D.ietf-core-link-format] with the value "core-rd" in the query string. Upon success, the response will contain a payload with a link format entry for each RD discovered, with the URL indicating the root resource of the RD. When performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network (TBD if a specific multicast address should be defined for RDs).

An RD implementing this specification MUST support query filtering for the rt parameter as defined in [I-D.ietf-core-link-format].

The discovery interface is specified as follows:

Interaction: EP -> RD

Path: `/.well-known/core`

Method: GET

Content-Type: `application/link-format` (if any)

Parameters:

Resource Type (rt): MUST contain the value "core-rd"

Instance (ins): Used to differentiate between multiple RDs.

Success: 2.05 "Content" with an `application/link-format` payload containing a matching entry for the RD resource.

Failure: 2.05 "Content" (should be a "No Content" code?) with an empty payload is returned in case no matching entry is found for a unicast request.

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request"

The following example shows an end-point discovering an RD using this interface, thus learning that the base RD resource is at /rd. Note that it is up to the RD to choose its base RD resource.

End-point		RD
----- GET /.well-known/core?rt=core-rd ----->		
<----- 2.05 Content "</rd>; rt="core-rd" -----		

Req: GET coap://[ff02::1]/.well-known/core?rt=core-rd

Res: 2.05 Content
 </rd>;rt="core-rd";ins="Primary"

4.2. Registration

After discovering the location of an RD, an end-point MAY register its resources to the RD's registration interface. This interface accepts a POST from an end-point containing the list of resources to be added to the directory as the message payload in the CoRE Link Format along with query string parameters indicating the name of the end-point, an optional node identifier and the lifetime of the registration. The end-point name is formed by concatenating the Host and Instance parameters included with the registration. All parameters of the registration are optional. In the absense of a Host parameter, the RD will generate a unique one on behalf of the end-point. The RD then creates a new resource or updates an existing resource in the RD and returns its location. An end-point MUST use that location when refreshing registrations using this interface. End-point resources in the RD are kept active for the period indicated by the lifetime parameter. The end-point is responsible for refreshing the entry within this period using either the registration

or update interface.

The registration interface is specified as follows:

Interaction: EP -> RD

Path: /.well-known/core or /{rd-base}

Method: POST

Content-Type: application/link-format

Etag: The Etag option MAY be included to allow an RD to perform validation in the future.

Parameters:

Lifetime (lt): Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) SHOULD be assumed.

Host (h): The host identifier or name of the registering node. The maximum length of this parameter is 63 octets. This parameter is combined with the Instance parameter (if any) to form the end-point name. If not included, the RD MUST generate a unique Host name on behalf of the node.

Instance (ins): The instance of the end-point on this host, if there are more than one. The maximum length of this parameter is 63 octets. Optional.

Type (rt): The semantic type of the end-point. The maximum length of this parameter is 63 octets. Optional.

Domain (d): The domain to which this end-point belongs. The maximum length of this parameter is 63 octets. Optional.

Context (con): This parameter sets the scheme, address and port at which this server is available in the form scheme://host:port. Optional. In the absence of this parameter the scheme of the protocol, source IP address and source port used to register are assumed.

Success: 2.01 "Created". The Location header MUST be included with the new resource entry for the end-point. This Location SHOULD be an stable identifier generated by the RD as it is used for all subsequent operations on this registration (update, delete).

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows an end-point with the name "node1" registering two resources to an RD using this interface. The resulting location /rd/4521 is just an example of an RD generated key.

End-point		RD
---	POST /rd "</sensors..." ----->	
<--	2.01 Created Location: /rd/4521 -----	

```

Req: POST coap://rd.example.org/rd?h=node1&lt=1024
Etag: 0x3f
Payload:
</sensors/temp>;ct=41;rt="TemperatureC";if="sensor",
</sensors/light>;ct=41;rt="LightLux";if="sensor"

Res: 2.01 Created
Location: /rd/4521

```

4.3. Update

The update interface is used by an end-point to refresh or update its registration with an RD. To use the interface, the end-point sends a PUT request to the resource returned in the Location option in the response to the first registration. An update MAY contain registraion parameters or a payload in CoRE Link Format if there have been changes since the last registration or update. Parameters that have not changed SHOULD NOT be included in an update.

The update interface is specified as follows:

Interaction: EP -> RD

Path: Location returned by registration.

Method: PUT

Content-Type: application/link-format (if any)

Etag: The Etag option MAY be included to allow an RD to compare the existing entry and perform validation in the future.

Parameters:

Lifetime (lt): Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) SHOULD be assumed.

Host (h): The host identifier or name of the registering node. The maximum length of this parameter is 63 octets. This parameter is combined with the Instance parameter (if any) to form the end-point name. If not included, the RD MUST generate a unique Host name on behalf of the node.

Instance (ins): The instance of the end-point on this host, if there are more than one. The maximum length of this parameter is 63 octets. Optional.

Type (rt): The semantic type of the end-point. The maximum length of this parameter is 63 octets. Optional.

Domain (d): The domain to which this end-point belongs. The maximum length of this parameter is 63 octets. Optional.

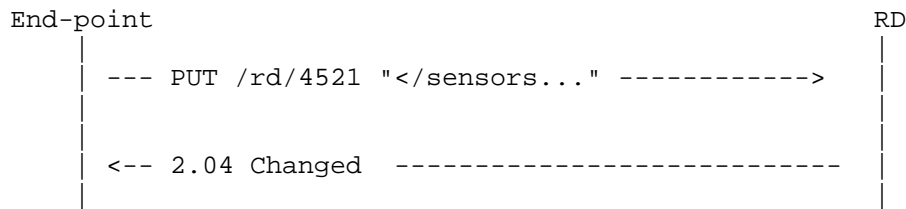
Context (con): This parameter sets the scheme, address and port at which this server is available in the form scheme://host:port. Optional. In the absence of this parameter the scheme of the protocol, source IP address and source port used to register are assumed.

Success: 2.04 "Changed" in case the resource and/or lifetime was successfully updated

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows an end-point updating a new set of resources to an RD using this interface.



Req: PUT /rd/4521

Etag: 0x40

Payload:

```
</sensors/temp/1>;ct=41;ins="Indoor";rt="TemperatureC";if="sensor",
</sensors/temp/2>;ct=41;ins="Outdoor";rt="TemperatureC";if="sensor",
</sensors/light>;ct=41;rt="LightLux";if="sensor"
```

Res: 2.04 Changed

4.4. Validation

In some cases, an RD may want to validate that it has the latest version of an end-point's resource. This can be performed with a GET on the well-known interface of the CoRE Link Format including the latest Etag stored for that end-point. For the purpose of validation, an end-point implementing this specification SHOULD support Etag validation on /.well-known/core.

The validation interface is specified as follows:

Interaction: RD -> EP

Path: /.well-known/core

Method: GET

Content-Type: application/link-format (if any)

Etag: The Etag option MUST be included

Parameters: None

Success: 2.03 "Valid" in case the Etag matches

Success: 2.05 "Content" in case the Etag does not match, the response MUST include the most recent resource representation and its corresponding Etag.

Failure: 4.00 "Bad Request". Malformed request.

The following examples shows a successful validation.

End-point	RD
<--- GET /.well-known/core Etag: 0x40 -----	
--- 2.03 Valid ----->	

Req: GET /.well-known/core
Etag: 0x40

Res: 2.03 Valid

4.5. Removal

Although RD entries have soft state and will eventually timeout after their lifetime, an end-point 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 end-point resource.

The removal interface is specified as follows:

Interaction: EP -> RD

Path: Location returned by registration.

Method: DELETE

Content-Type: None

Parameters: None

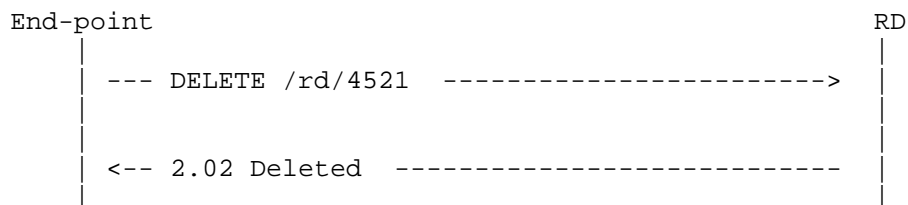
Success: 2.02 "Deleted" upon successful deletion

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following examples shows successful removal of the end-point from

the RD.



Req: DELETE /rd/4521

Res: 2.02 Deleted

4.6. Lookup

In order for an RD to be used for discovering resources registered with it, a lookup interface is provided. This lookup interface is provided 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).

The lookup interface is provided using the CoRE Link Format [I-D.ietf-core-link-format] resource discovery mechanism on the root RD resource (/rd in the examples). The scope of the discovery is controlled by the End-point (ep=) and Domain (d=) parameters. A lookup on the root RD resource /rd queries all resources on the RD, a lookup /rd?d=domain lists all resources in a domain and a lookup /rd?ep=end-point performs a lookup on resources associated with that end-point.

An RD SHOULD support the query filtering defined in [I-D.ietf-core-link-format] to allow for filtered lookups. To optimize the size of a lookup response, any non-wildcard attributes in the query string SHOULD NOT be included in the resulting links.

The lookup interface is specified as follows:

Interaction: Client -> RD

Path: /{rd-base}, e.g. /rd

Method: GET

Content-Type: application/link-format (if any)

Parameters:

End-point (ep): The end-point (concatenation of host and ins parameters used in registration) from which resources should be looked up.

Domain (d): The domain from which resources should be looked up. The maximum length of this parameter is 63 octets. Optional.

Filtering: CoRE Link Format attributes may be included to further filter the lookup.

Success: 2.05 "Content" with an application/link-format payload containing a matching entries for the lookup.

Failure: 4.04 "Not Found" in case no matching entry is found for a unicast request.

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows a client performing a lookup on an RD using this interface.

```
Client                                     RD
|                                         |
| ----- GET /rd?rt=Temperature -----> |
|                                         |
| <-- 2.05 Content "<coap://nodel/temp>;rt="Temperature" ---- |
|                                         |
```

Req: GET /rd?rt=Temperature

Res: 2.05 Content

```
<coap://nodel/temp>;rt="Temperature"
```

5. 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 [I-D.ietf-core-link-format]:

```
link-extension    = ( "ins" "=" quoted-string ) ; Max 63 octets
link-extension    = ( "exp" )
```

5.1. Resource Instance 'ins' attribute

The Resource Instance "ins" attribute is an identifier for this resource, which makes it possible to distinguish from other similar resources. This attribute is similar in use to the "Instance" portion of a DNS-SD record, and SHOULD be unique across resources with the same Resource Type attribute in the domain it is used. A Resource Instance might be a descriptive string like "Ceiling Light, Room 3", a short ID like "AF39" or a unique UUID or iNumber. This attribute is used by a Resource Directory to distinguish between multiple instances of the same resource type within a system.

This attribute MUST be no more than 63 octets in length. The resource identifier attribute MUST NOT appear more than once in a link description.

5.2. Export 'exp' attribute

The Export "exp" attribute is used as a flag to indicate that a link description MAY be exported by a resource directory to external directories.

The CoRE Link Format is used for many purposes between CoAP endpoints. Some are useful mainly locally, for example checking the observability of a resource before accessing it, determining the size of a resource, or traversing dynamic resource structures. However, other links are very useful to be exported to other directories, for example the entry point resource to a functional service.

6. Security Considerations

This document needs the same security considerations as described in Section 7 of [RFC5988] and Section 6 of [I-D.ietf-core-link-format]. The /.well-known/core resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [I-D.ietf-core-coap].

7. IANA Considerations

"core-rd" resource type needs to be registered if an appropriate registry is created.

"ins" and "exp" attributes need to be registered when a future Web Linking attribute is created.

8. Acknowledgments

Szymon Sasin, Carsten Bormann, Kerry Lynn, Peter van der Stok, Anders Brandt, Matthieu Vial and Linyi Tian have provided helpful comments, discussions and ideas to improve and shape this document. The authors would also like to thank their colleagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

9. Changelog

Changes from -01 to -02:

- o Added a terminology section.
- o Changed the including of an Etag in registration or update to a MAY.
- o Added the concept of an RD domain and a registration parameter for it.
- o Recommended the Location returned from a registration to be stable, allowing for end-point and domain information to be changed during updates.
- o Changed the lookup interface to accept end-point and domain as query string parameters to control the scope of a lookup.

10. References

10.1. Normative References

- [I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format",
draft-ietf-core-link-format-06 (work in progress),
June 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

10.2. Informative References

- [I-D.brandt-coap-subnet-discovery]
Brandt, A., "Discovery of CoAP servers across subnets",
draft-brandt-coap-subnet-discovery-00 (work in progress),
March 2011.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-06 (work in progress), May 2011.
- [I-D.shelby-core-coap-req]
Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R.
Kelsey, "CoAP Requirements and Features",
draft-shelby-core-coap-req-01 (work in progress),
April 2010.
- [I-D.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building
Control", draft-vanderstok-core-bc-03 (work in progress),
March 2011.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

Authors' Addresses

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach@sensinode.com

Srdjan Krco
Ericsson

Phone:
Email: srdjan.krco@ericsson.com

CoRE
Internet-Draft
Intended status: Informational
Expires: May 2, 2012

P. van der Stok
Philips Research
K. Lynn
Consultant
October 30, 2011

CoAP Utilization for Building Control
draft-vanderstok-core-bc-05

Abstract

This draft describes an example use of the RESTful CoAP protocol for building automation and control (BAC) applications such as HVAC and lighting. A few basic design assumptions are stated first, then URI structure is utilized to define group as well as unicast scope for RESTful operations.

This proposal supports the view that 1) service discovery is complementary to resource discovery and facilitates control network scaling, and 2) building control is likely to move in steps toward all-IP control networks based on the legacy efforts provided by DALI, LON, BACnet, ZigBee, and other standards.

The authority portion of the URI is used to identify a device (group) and the resulting DNS name is bound to a unicast (multicast) address. Group addressing has consequence for the naming convention of the resources of a device. Naming of URI is building or organization dependent, must be flexible, and SHOULD conform to some local convention. Naming of resources MUST be standardised preferable by a building control related organisation.

It is shown that DNS-based service discovery can be used to locate URIs on the scale necessary in large commercial BAC deployments. The relation of DNS-SD and a Resource Directory is discussed. Finally, a method is proposed for mapping URIs onto legacy BAC resources, e.g., to discover application-layer gateways, proxies, and their dependent services.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 2, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
1.1. Terminology	4
1.2. Motivation	4
2. URI structure	6
2.1. Scheme part	7
2.2. Authority part	7
2.3. Path part	8
3. Group Naming and Addressing	10
4. Discovery	12
4.1. Service discovery goals	12
4.2. DNS-Based Service Discovery	13
4.3. Browsing for Services	14
4.4. Resource vs Service Discovery	14
5. DNS record structure	15
5.1. DNS group example	18
5.2. Operational use of DNS-SD	19
5.3. Commissioning CoAP devices	20
5.3.1. DNS-SD server present	21
5.3.2. DNS-SD server not present	21
5.4. Proxy discovery	22
6. Legacy data Representations in CoAP	23
6.1. Network architectures	24
6.2. Discovery of legacy gateways	26
7. Conclusions	26
8. Security considerations	27
9. IANA considerations	28
10. Acknowledgements	28
11. Changelog	28
12. References	29
12.1. Normative References	29
12.2. Informative References	30
Authors' Addresses	32

1. Introduction

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 "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

In addition, the following conventions are used in this document:

A CoAP end-point, or server, is identified by a unique {IP address, port} tuple and characterised by a protocol. A server is completely specified by the authority part of a URI.

A device is the physical object that is connected to the network. A device may host one or more CoAP servers.

A service (in the service discovery sense) is a related set of resources on a CoAP server. A URI completely specifies the syntax of a service interface. Metadata describe the semantics of the service interface. The semantics may include the relation between service and the hardware connected to the device. A CoAP server may expose one or more services.

In examples below involving URIs, the authority is preceded by double slashes "://" and path is preceded by a single slash "/". The examples may make use of full or partial host names and the difference should be clear from the context.

1.2. Motivation

The CoAP protocol [I-D.ietf-core-coap] aims at providing a user application protocol architecture for a network of devices with a low resource provision such as memory, CPU capacity, and energy. In general, IT application manufacturers strive to provide the highest possible functionality and quality for a given price. In contrast, the building automation controls market is highly price sensitive and manufacturers tend to compete by delivering a given functionality and quality for the lowest price. In the first market a decreasing memory price leads to more software functionality, while in the second market it leads to a lower Bill of Material (BOM).

The vast majority of devices in a typical building control application is resource constrained, making the standardization of a lightweight application protocol like CoAP a necessary requirement for IP to penetrate the device market. The low energy consumption requirement of battery-less devices reinforces this approach. Low

resource budget implies low throughput and small packet size as for [IEEE.802.15.4]. Reduction of the packet size is obtained by using the header reduction of 6LoWPAN [RFC4944] and encouraging small payloads.

Several legacy building control standards (e.g. [BACnet], [DALI], [KNX], [LON], [ZigBee], etc.) have been developed based on years of accumulated knowledge and industry cooperation. These standards generally specify a data model, functional interfaces, packet formats, and sometimes a physical medium for data objects and function invocation. Many of these industry standards also specify proprietary transport protocols, necessitating expensive stateful gateways for these standards to interoperate. Many more recent building control network include IP-based standards for transport (at least to interconnect islands of functionality) and other functions such as naming and discovery. CoAP will be successful in the building control market to the extent that it can represent a given standard's data objects and provide functions, e.g. resource discovery, that these standards depend on.

From the above the wanted basic syntax properties can be summarized as:

- Generate small payloads.
- Compatible with legacy standards (e.g. LON, BACnet, DALI, ZigBee Device Objects).
- Service/resource discovery in agreement with legacy standards and naming conventions.

This submission defines an approach in which the payload contains messages with a syntax defined by legacy control standards. Accordingly, the syntax of the service/resource discovery messages encapsulates legacy control standard. The intention is a progressive approach to all-IP in building control. In a first stage standard IETF based protocols (e.g. CoAP, DNS-SD) are used for transport of control messages and discovery messages expressed in a legacy syntax. This approach enables the reuse of controllers based on the semantics of the chosen control standard. In a later stage a complete redesign of the controllers can be envisaged guided by the accumulated experience with all-IP control.

Two concepts, hierarchy and group, are of prime importance in building control, particularly in lighting and HVAC. Many control messages or events are multicast from one device to a group of devices (e.g. from a light switch to all lights in an area). The scope of a multicast command or discovery message determines the group of devices that is targeted. A group scope may be defined as link-local, as a tree maintained by an IP-multicast protocol, or an

overlay that corresponds to the logical structure of a building or campus and is independent of the underlying network structure. Techniques for group communication are discussed in [I-D.rahman-core-groupcomm].

As described in "Commercial Building Applications Requirements" [I-D.martocci-6lowapp-building-applications] it is typical practice to aggregate building control at the room, area, and supervisory levels. Furthermore, networks for different subsystems (lights, HVAC, etc.) or based on different legacy standards have historically been isolated from each other in so-called "silos". RESTful web services [Fielding] represent one possible way to expose functionality and normalize data representations between silos in order to facilitate higher order applications such as campus-wide energy management.

Consequently, additional group properties are:

- Devices may be part of one or more groups.
- Resources addressed by a group must be uniformly and consistently named across all targeted devices.

For clarity, this I-D limits itself to two types of applications: (1) M2M control applications running within a building area without any human intervention after commissioning of a given network segment and (2) maintenance oriented applications where data are collected from devices in several building areas by devices inside or outside the building, and humans may intervene to change control settings. This I-D compares commercial building solutions with solutions for the home.

2. URI structure

This I-D considers three elements of the URI: scheme, authority, and path, as defined in "Uniform Resource Identifier (URI): Generic Syntax" [RFC3986]. The authority is defined within the context of standard DNS host naming, while the path is valid in relation to a fully qualified domain name (FQDN) plus optional port (and protocol is implicit, based on scheme). An example based on [RFC3986] is: `foo://host.example.com:8042/over/there?name=ferret#nose`, where "foo" is the scheme, "host.example.com:8042" is the authority, "/over/there" is the path, "name=ferret" is the query, and "nose" is the fragment. Fragments are not supported in CoAP.

2.1. Scheme part

The CoAP URI scheme syntax is specified in section 6 of [I-D.ietf-core-coap] and is compatible with the "http" scheme specification [RFC2616]. The scheme is implicit from the perspective of the service, but it indicates the protocol used to access the service to potential clients.

2.2. Authority part

The authority part is either a literal IP address or a DNS name comprised of a local part, specifying an individual device or group of devices, and a global part specifying a (sub)domain that may reflect the logical hierarchical structure of the building control network. The result is said to be a fully qualified domain name (FQDN) which is globally unique down to the group or device level. An optional port number may be included in the authority following a single colon ":" if the service port is other than the default CoAP value. The authority resolves to a {IP-address, port} tuple. The IP-address may be either unicast or multicast. The authority therefore identifies an individual server or a named group of servers.

The CoAP spec [I-D.ietf-core-coap] states "When a CoAP server is hosted by a 6LoWPAN device, it SHOULD also support a port in the 61616-61631 compressed UDP port space defined in [RFC4944]." As shown below, DNS-SD [I-D.cheshire-dnsext-dns-sd] is a viable technique for discovering dynamic host and port assignments for a given service. However, the use of dynamic ports in URIs is likely to lead to brittle (non-durable) identifiers as there is no assurance that a CoAP server will consistently acquire the same dynamic port and different {IP-address, port} tuples conventionally represent different servers.

A building can be unambiguously addressed by its GPS coordinates or more functionally by its zip or postal code. For example the Dutch Internet provider, KPN, assigns to each subscriber a host name based on its postcode. Analogously, an example authority for a building may be given by: //bldg.zipcode-localnr.Country/ or more concretely an imaginary address in the Netherlands as: //bldg.5533BA-125a.nl/. The "bldg" prefix can specify the target device within the building. Arriving at the device identified by //bldg.5533BA-125a.nl, the receiving service can parse the path portion of the URI and perform the requested actions on the specified resource.

Buildings have a logical internal structure dependent on their size and function. This ranges from a single hall without any structure to a complex building with wings, floors, offices and possibly a

structure within individual rooms. The naming of the building control equipment and the actual control strategy are intimately linked to the building structure. It is therefore natural to name the equipment based on their location within the building. Consequently, the local part of the URI identifying a piece of equipment is expressed in the building structure. An example is: `//light-27.floor-1.west-wing...`

This proposal assumes a minimal level of cooperation between the IT and building management infrastructure, namely the ability of the former to delegate DNS subdomains to the latter. This allows the building controls installer to implement an appropriate naming scheme with the required granularity. For institutional real estate such as a college or corporate campus, the authority might be based on the organization's domain, e.g. `//device-or-group.floor.wing.bldg.campus.example.com/`. In cases where subdomain delegation is not an option, structure can still be represented in a "flat" namespace, subject to the 63 octet limit for a DNS label: `//group1-floor2-west-bldg3-campus.example.com`.

Most communication is device to device (M2M) within the building. Often a device needs to communicate to all devices of a given type within a given area of the building. For example a thermostat may access all radiator actuators in a zone. A light switch located at room 25b006 of floor one, expressed as: `//switch0.25b006.floor1.5533BA-125a.nl/`, might specify a command to `light1` within the same room with `//light1.25b006.floor1.5533BA-125a.nl/`. This approach can lead to rather verbose URI strings in the packet, contrary to the small packet assumption. The question arises as to whether the syntax of the authority part needs to be standardized for building control. Given the naming flexibility provided by DNS, authority names for building control are more the concern of the building owner or the installer than a standardization concern.

2.3. Path part

The path identifies the addressable attributes of the service at the highest possible granularity. A set of paths defines the syntax of the service invocation and constitutes the interface description of the service. Every network service attribute is completely identified by a URI scheme: `//authority/path`. In analogy, the path part of the URI specifies the resource of a given server. The naming of the services and their associated attributes are typically subjects for standardization. There is no widely accepted standard for uniformly naming building control services in a URI. A vigorous effort is undertaken by the oBIX working group of OASIS [oBIX], but its current impact is limited. There is also an open source point

naming effort underway called Project Haystack [HAYSTACK].

The path is constructed like a file system path name. It consists of a sequence of one or more name fields, with each field preceded with a slash, like /func1/subf2/final. The set of paths is structured as a tree. The last name in a name field sequence is called a leaf of the tree, and the authority is the root of the path tree of a given host. The semantics of a given sub-tree in the path tree is specified by the Interface Description (if=) attribute described in [I-D.ietf-core-link-format]. As for file systems some tree naming with associated semantics can be standardized such as the de facto PC standard directory "documents and settings" with the sub-directories "My documents", "usradmin", etc. When a given body, e.g. XXX, has defined a name structure and semantics for the path tree, we say that "if = XXX" when the path tree conforms to the name structure defined by XXX.

When a GET method with an URI like
"/t-sensor1.25b006.floor1.example.com/temperature" is sent, it represents an a priori understanding that the server with name t-sensor1 exists, provides a service of a given standard type (with associated semantics) (e.g. ZigBee temperature sensor), and that this standard type has the readable attribute: temperature. When commands are sent to a group of servers it MUST be the case that the targeted resource has the same path on all targeted servers. Therefore, it is necessary to establish at least a local uniform path naming convention to achieve this. One approach is to include the name of the standard, e.g. BACnet, as the first element in the path and then employ the standard's chosen data scheme (in the case of BACnet, /bacnet/device/object/property).

The organization responsible for defining a given industry standard XXX (e.g. BACnet, ZigBee, etc.) can register the /.well-known/XXX prefix and specify the allowable path-names for a server of a given type. The same body also defines the "if=XXX" attribute. This allows the standards development organization responsible for XXX to define the name space and resources associated with the prefix together with the associated semantics. The registered /.well-known/XXX URI effectively defines a standard object model, or schema, for services of the XXX application protocol. Manufacturers may optionally define proprietary resources that can be discovered dynamically using methods described below.

Although the authority part names need not always be transported, the path names MUST be transported in the CoAP packets. Therefore, path names SHOULD be as short as possible, even at the detriment of the clarity of the meaning of the path name.

3. Group Naming and Addressing

Within building control it is necessary to send the same command to a set of servers. Grouping allows to invoke the set of services with one application command to be executable within a specified time interval. Given a network configuration, the network operator needs to define an appropriate set of groups which can be mapped to the building areas. Knowledge about the hierarchical structure of the building areas may assist in defining a network architecture which encourages an efficient group communication implementation. IP-multicasting over the group is a possible approach for building control, although proxy-based methods may prove to be more appropriate in some deployments [I-D.rahman-core-groupcomm].

Example device groups become:

URI authority	Targeted group
//all.bldg6...	"all devices in building 6"
//all.west.bldg6...	"all devices in west wing, building 6"
//all.floor1.west.bldg6...	"all devices on floor 1, west wing, ..."
//all.bu036.floor1.west.bldg6...	"all devices in office bu036, ..."

The granularity of this example is for illustration rather than a recommendation. Experience will dictate the appropriate hierarchy for a given structure as well as the appropriate number of groups per subdomain. Note that in this example, the group name "all" is used to identify the group of all devices in each subdomain. In practice, "all" could name an address record in each of the DNS zones shown above and would bind to a different multicast address [RFC3596] in each zone. Highly granular multicast scopes are only practical using IPv6. The multicast address allocation strategy is beyond the scope of this I-D, but various alternatives have been proposed [RFC3306][RFC3307][RFC3956].

To illustrate the concept of multiple group names within a building, consider the definition, as done with [DALI], of scenes within the context of a floor or a single office. For example, the setting of all blue lights in office bu036 of floor 1 can be realized by multicasting a message to the group "//blue-lights.bu036.floor1". Each group is associated with a multicast IP address. Consequently, when the application specifies the sending of an "on" message to all blue lights in the office, the message is multicast to the associated IP address.

The binding of a group FQDN to a multicast address (i.e., creation of the AAAA record in the DNS zone server) happens during the

commissioning process. Resolution of the group name to a multicast address happens at restart of a device. A multicast address and associated group name in this context are assumed to be long-lived. It can happen that during operation the membership of the group changes (less or more lights) but its address is not altered and neither is its name. Group membership may be managed by a protocol such as Multicast Listener Discovery [RFC5790].

Similarly, a group can identify a set of resources of one server. For examples a device contains four I/O channels. The device hosts one server with four resources to access each of the four individual channels separately. Commonly, it is also required to access all four channels as one group. An additional path identifies the group of services. An example set of services and service-group is:

URI path	Targeted group
/IOchannel/1...	"channel 1 of the IO channel device "
/IOchannel/2...	"channel 2 of the IO channel device "
/IOchannel/3...	"channel 3 of the IO channel device "
/IOchannel/4...	"channel 4 of the IO channel device "
/IOchannel/...	"channel 1 to 4 of the IO channel device "

A group defines a set of servers possibly containing a set of resources. Grouping of the resources is provided by the device manufacturer. Grouping of the servers is supported by DNS and multicast protocols. The multicast address(es) identify the servers belonging to the group. A given server might belong to a number of groups. For example the server belonging to the "blue-lights" group in a given corridor might also belong to the groups: "whole building", "given wing", "given floor", "given corridor", and "lights in given corridor". From the perspective of a server, the main consequence of joining a group is it should accept packets for an additional IP address. The granularity of the domain names may have an impact on the complexity of the DNS infrastructure but not necessarily on the low-resource destinations or sources. Assuming that resolution of addresses only happens at device start-up, the complexity of the DNS server need not affect the responsiveness of the devices.

In summary, the authority portion of the URI resolves to an IP-address and port number, and identifies a server or group of servers. Authority naming is building or organization dependent, must be flexible, and does not require standardization efforts but SHOULD conform to some uniform convention. Path naming SHOULD conform to the naming convention of a standardization body.

4. Discovery

4.1. Service discovery goals

Service discovery in building control should rely on a minimal need for intervention by humans (or complete absence of humans) during system setup, bootstrapping, restart, configuration and daily operation. The goals for service discovery area:

Goal	Goal description
Return_instance	Return all instances of a given service type within a given domain
Group_instance	Group a set of instances within a group associated with a domain
Instance_resolution	Resolve the instance name to usable invocation information (e.g. IP address and port)
Group_resolution	resolve the group name to usable invocation information (e.g. IP address and port)

These goals are necessary to support the operation of commercial building control. Returning the instances results in a list of names. For building control these names can be any sequence of characters as long as for each service instance these names are unique within the domain. In [I-D.cheshire-dnsext-dns-sd] the office equipment in the IT domain is recommended to use understandable and human-readable names. The Home domain may have a need for human understandable names. This is not the case for the commercial building automation domain. However, uniqueness of the name is necessary for the application that needs to address the service in a consistent manner. Given the large number of devices in a building (several hundreds to thousands) scaling is an important aspect of the service discovery. A set of central DNS servers will provide the scalability. The expectation is that names need to be managed consistently by a central authority which can be supported by the DNS server. Tools will assist the installer and operator of the network to do the installation, configuration and maintenance of the network structure. Small devices will use the DNS server to learn the communication partners providing a given service within their domain and to resolve the IP addresses of the communication partners.

Within the home it is more important that the names convey the purpose of the service to the user reading the names and selecting his favored service instance. Non-unique names, although confusing, can probably be handled by the user of these names. Scalability is less of an issue because a smaller number of devices is implicated. The network in the home is probably more dynamic than its commercial counter-part, with many movements of devices and arrival or removal of devices.

Section 5 presents some examples of DNS structures to show how the choice of names influences the granularity of the discovery. In sections 5.1 and 5.3 a grouping example and a commissioning example, filling the DNS, are presented.

4.2. DNS-Based Service Discovery

DNS-Based Service Discovery (DNS-SD) defines a conventional way to configure DNS PTR, SRV, and TXT records to facilitate discovery of services within a subdomain, re-using the existing DNS infrastructure. This section gives a cursory overview of DNS-SD; see [I-D.cheshire-dnsext-dns-sd] for a complete description.

A DNS-SD service instance name is of the form
<Instance>.<ServiceType>.<Location>.

The Location part of the service name is identical to the DNS subdomain part of the authority in URIs that identify the resources of this server or group and may identify a building zone as in the examples above.

The ServiceType SHOULD have the form [_subtype._sub.]_type._proto (e.g. _temp._sub._bc._udp). The _proto identifier provides a transport protocol hint as required by the SRV record definition [RFC2782] and, in the case of CoAP, it is always "_udp". The _type identifier is determined by standards development organization (SDO) and MUST be registered with dns-sd.org [dns-sd] (e.g. _bc for building control). The SDO is then free to specify one or more _subtype identifiers, which must be unique for a given _type (e.g. _temp). The _subtype and _type labels are separated by the literal "_sub" label. The maximum length of the type and subtype fields is 14 octets, but shorter names are encouraged to reduce packet sizes.

A PTR record with the label "_type._proto" is defined for each server in a selected domain, and this record's value is set to the service instance name (which in turn identifies the SRV and TXT records for the CoAP server).

The Instance part of the service name may be changed during the commissioning process. It must be unique for a given ServiceType within the subdomain. The complete service name uniquely identifies an SRV and a TXT record in the DNS zone. The granularity of a service name MAY be at the group or server level, or it could represent a particular resource within a CoAP server. The SRV record contains the host (AAAAA record) name and port of the service. The path part of the URI MUST be placed in the TXT record (path=) when multiple resources belong to the same service.

4.3. Browsing for Services

Devices in a given Location with given ServiceType, `_type._proto`, may be enumerated by sending a DNS query for PTR records named `_type._proto` to the authoritative server for that zone associated with the Location. A list of instance names for SRV records matching that `<ServiceType>.<Location>` is returned. Each SRV record contains the host name and port of a CoAP server. The IP address of the device is obtained by resolving the host name. DNS-SD also specifies an optional TXT record, having the same name as the SRV record, which can contain "key=value" attributes. Apart from defining standardized resources identified by `if=XXX`, the XXX organization may also define the standard "key=value" pairs present in the TXT record, e.g. `type=switch`. By convention, the first pair is `txtver=<number>` so that different versions of the XXX schema may interoperate. For example: A query is sent to DNS-SD to return all DALI lamps within the domain `office5/mybuilding` and with ServiceType: `_lamp._sub._dali._udp`. DNS-SD returns the list of all SRV records and AAAA records of the devices within the domain providing the wanted service.

4.4. Resource vs Service Discovery

Service discovery is concerned with finding the IP address, port, protocol, and possibly path of a named service. Resource discovery is a fine-grained enumeration of resources (path-names) of a server. [I-D.ietf-core-link-format] specifies a resource discovery pattern, such that sending a confirmable GET message for the `/.well-known/core` resource returns a set of links available from the server. These links describe resources hosted on that server.

CoAP link format can be used to enumerate attributes and populate the DNS-SD database in a semi-automated fashion. CoAP resource descriptions can be imported into DNS-SD for exposure to service discovery as described in [I-D.lynn-core-discovery-mapping]. The values stored in the DNS-SD directory are extracted from the information stored in the resource directory associated with a set of CoAP hosts [I-D.shelby-core-resource-directory]. The resources describe how the services can be manipulated in detail and in concreto.

It is assumed that a resource directory exists per 6LoWPAN [RFC4944], possibly running on the edge router. The DNS-SD provides a larger scope by storing the info of all services over a set of interconnected 6LoWPANs. Where the resource directory is possibly completely adequate for home networks, handling of multiple resource directories can be quite cumbersome for the many 6LoWPANs envisaged for offices. However, during network configuration, the resource

directory can be used as long as the DNS is not yet accessible.

The DNS-SD approach is complementary to the more fine-grained resource discovery, fits better the concept of service by discovering servers with given properties. DNS-SD supports a hierarchical approach to the naming of the services as discussed in section 3. DNS-SD provides a directory structure that scales well with the network size as shown by its present-day operation.

5. DNS record structure

An example is presented which explains the Resource Record (RR) structure on the DNS server. This section follows the mapping specified in [I-D.lynn-core-discovery-mapping], which defines how to fill the DNS-SD records from the link extension values. Suppose the services are delivered by XXX building control devices. The example subtype- and context- names are assumed to be standardized by the XXX alliance. All devices are situated in one office with location office4.bldg8.example.com. The names in the examples are more verbose than recommended to make the examples more readable. The table presents the services provided in the office control network:

service	ServiceType	Number
illumination	_OnOff_light._sub._bc._udp	4
presence	_occup_sensor._sub._bc._udp	1
temperature	_temp_sensor._sub._bc._udp	1
shading	_shade_control._sub._bc._udp	1

In DNS PTR records with as label the ServiceType have as value service instance names. The unique Instance names identify the service instances. In the example, the names contain id-x, with x in natural numbers. The names are usually created at the factory floor and somehow attached to the product. The ServiceTypes have been suffixed with .04.b8 to represent office4 in building8. The same suffix is used as PTR label to enumerate all instance of a given service, or within a given domain.

_OnOff_light._sub._bc._udp.04.b8	PTR id-1._OnOff_light
bc._udp.04.b8	PTR id-1._OnOff_light
04.b8	PTR id-1._OnOff_light
_OnOff_light._sub._bc._udp.04.b8	PTR id-2._OnOff_light
bc._udp.04.b8	PTR id-2._OnOff_light
04.b8	PTR id-2._OnOff_light
_OnOff_light._sub._bc._udp.04.b8	PTR id-3._OnOff_light
bc._udp.04.b8	PTR id-3._OnOff_light
04.b8	PTR id-3._OnOff_light
_OnOff_light._sub._bc._udp.04.b8	PTR id-4._OnOff_light
bc._udp.04.b8	PTR id-4._OnOff_light
04.b8	PTR id-4._OnOff_light
_occup_sensor._sub._bc._udp.04.b8	PTR id-5._occup_sensor
bc._udp.04.b8	PTR id-5._occup_sensor
04.b8	PTR id-5._occup_sensor
_temp_sensor._sub._bc._udp.04.b8	PTR id-6._temp_sensor
bc._udp.04.b8	PTR id-6._temp_sensor
04.b8	PTR id-6._temp_sensor
_shade_control._sub._bc._udp.04.b8	PTR id-7._temp_sensor
bc._udp.04.b8	PTR id-7._temp_sensor
04.b8	PTR id-7._temp_sensor

In the above example the id-x identifiers without the subtype suffix would be discriminating enough.

Discovery can be done with the following results. A query with the following argument returns

query argument	result list
.04.8	id-1._OnOff_light

	id-7._temp_sensor
_bc._udp.04.b8	id-1._OnOff_light

	id-7._temp_sensor
_OnOff_light._sub._bc._udp.04.b8	id-1._OnOff_light

	id-4._OnOff_light
_occup_sensor._sub._bc._udp.04.b8	id-5._occup_sensor

When other offices are included in the database, the query argument 04.b8 selects those entries which are associated with office4 in building8 and rejects any others. The example shows clearly the query granularity that can be obtained and the care that must be exercised when defining the names of the ServiceTypes.

The service instances (value of PTR records) are the labels of the SRV, AAAA and TXT records describing the service instance. The SRV

record specifies the location (authority) and the port number. In the authority o4.b8 refers to office4 in building8. The AAAA record specifies the IP-address, while the TXT record specifies the subtype and the data representation of the legacy parser (if = ZigBee).

```

id-1._OnOff_light  SRV  light1.o4.b8.example.com Port-x
                   AAAA fdfd::1234
                   TXT  if=ZigBee
id-2._OnOff_light  SRV  light2.o4.b8.example.com Port-x
                   AAAA fdfd::1235
                   TXT  if=ZigBee
id-3._OnOff_light  SRV  light3.o4.b8.example.com Port-x
                   AAAA fdfd::1236
                   TXT  if=ZigBee
id-4._OnOff_light  SRV  light4.o4.b8.example.com Port-x
                   AAAA fdfd::1237
                   TXT  if=ZigBee
id-5._occup_sensor SRV  occup.o4.b8.example.com  Port-x
                   AAAA fdfd::1238
                   TXT  if=ZigBee
id-6._temp_sensor  SRV  temp.o4.b8.example.com   Port-x
                   AAAA fdfd::1239
                   TXT  if=ZigBee
id-7._shade_control SRV  shade.o4.b8.example.com Port-x
                   AAAA fdfd::1240
                   TXT  if=ZigBee

```

It is possible that the temperature sensor and occupancy sensor are delivered on one device. The consequence is that one device hosts two services. In the DNS table the four lights and the shade controller are unaffected. However, the PTR records with the occupancy and temperature sensor point to the same unique identifier id-8 that is suffixed with the name of the subtype. This example shows that the subtype suffix is needed to discriminate between the two service instances.

```

_occup_sensor._sub._bc._udp PTR id-8._occup_sensor
_temp_sensor._sub._bc._udp  PTR id-8._temp_sensor

```

Two SRV records with accompanying AAAA and TXT records describe the two servers, each providing one service, in more detail. The servers share the same IP address but are connected to different ports, and do have a different paths names. The TXT record is used to specify the path part with "path=".

```
id-8._occup_sensor SRV  occup.o4.b8.example.com Port-x
                    AAAA fdfd::1241
                    TXT  path=/os if=ZigBee
id-8._temp_sensor  SRV  temp.o4.b8.example.com  Port-y
                    AAAA fdfd::1241
                    TXT  path=/ts if=ZigBee
```

The path names /ts and /os are short names for temperature_sensor and occupancy_sensor respectively. Not all multi-function devices will use different ports for the individual functions. It is also quite common to use different IP interfaces with different IP addresses, reflected by the value of the AAAA records.

5.1. DNS group example

Another aspect is the grouping of servers. Where in the former section the names of the services are standardized names, this is less probable for the group names. Usually the group names are application specific or are standardized at the manufacturer. For example, assume that a group all_light.o4.b8.example.com is created which contains all four lights inside office4. The accompanying ServiceType can be defined as _all_light._sub._bc._udp. The ServiceType suffixed with 04.b8 points to a unique identifier defined as _all_light.04.b8, assuming that this is the only _all_light group within office 4 of building 8. The PTR record looks like:

```
_all_light._sub._bc._udp.04.b8 PTR _all_light.04.b8
```

It is assumed that the group all_light.o4.b8.example.com has received a multicast address: ffile::148. The accompanying SRV, AAAA, and TXT RR become:

```
_all_light.04.b8 SRV  all_light.o4.b8.example.com Port-z
                    AAAA ffile::148
                    TXT  if=ZigBee
```

When a multicast message is sent to a group, the path of the accessed resource must be strictly the same for all servers. The naming of the path is typically a responsibility for the standardisation organisations describing the command set for a given application area. However a constraint exists in the case of multi-function devices which host multiple resource of the same type. For example a device with three lamps with corresponding onoff attributes can be accessed via the three different paths:

```

/light/1/onoff
/light/2/onoff
/light/3/onoff

```

A unique path to the onoff resource of all instances of light on this device can be provided by /light/onoff. As this is logically the path to a single instance on a mono-function device. The corresponding unique paths for onoff to be used in the multicast message becomes /light/onoff. The corresponding resource records for a luminaire, named lml, in DNS become:

```

_light._sub._bc._udp.04.b8 PTR _all_light.04.b8
_light._sub._bc._udp.04.b8 PTR _light_1.04.b8
_light._sub._bc._udp.04.b8 PTR _light_2.04.b8
_light._sub._bc._udp.04.b8 PTR _light_3.04.b8
_all_light.04.b8 SRV all_light.o4.b8.example.com Port-x
AAAA ffile::148
TXT if=ZigBee path=/light
_light_1.04.b8 SRV lml.o4.b8.example.com Port-z
AAAA fdfd::1234
TXT if=ZigBee path=/light/1
_light_2.04.b8 SRV lml.o4.b8.example.com Port-z
AAAA fdfd::1234
TXT if=ZigBee path=/light/2
_light_3.04.b8 SRV lml.o4.b8.example.com Port-z
AAAA fdfd::1234
TXT if=ZigBee path=/light/3

```

The entries in DNS can be used to form groups with the light weight group management protocol and multicast listener discovery [RFC5790].

5.2. Operational use of DNS-SD

The populated DNS-SD server provides the necessary support for the applications to execute their control loops with minimum operator support. The operation of the office network can be split up in phases. In a first phase the network is commissioned, such that a relation is established between the IP address, the servicetype and the domain. The servicetype can be extracted from the link-format as described in [I-D.shelby-core-resource-directory]. After commissioning this information is stored in the DNS-SD files. In a second phase groups are formed and group names with their IP address are stored in the DNS-SD files. The IP multicast addresses are communicated to the members of the groups. In the third and final phase, applications query DNS-SD to find the IP addresses of the services within a given domain, and of the groups within a given domain.

In the home, a commissioning phase requiring the intervention of an installer (a "truck roll") is to be avoided if possible. Here the first phase consists of the booting up devices which insert their services resources to a link-format directory. The information from the resource directory can be inserted into DNS-SD or into xmDNS [I-D.lynn-dnsexst-site-mdns] when appropriate. In the second phase remote controllers or other hand-held devices can be used to discover the services of a given type, to group the services, and to store the group names into DNS-SD or xmDNS as appropriate. Pointing out the members of a group can be in any kind of manner from typing members in, selecting them from a browser list, etc.

5.3. Commissioning CoAP devices

For clarity it is assumed in this section that a device hosts one server. A device has received a unique device identifier at the production plant. Given the authority naming presented in section 2.2 the authority name represents the location of the host within the building.

Commissioning means the following three actions:

- Defining the URI (location)
- Assigning an IP address to the URI
- mapping the unique device identifier to the URI

Two cases of the office network are considered for commissioning: (1) no 6LBR and no DNS server connected, and (2) a 6LBR connects the office network to a DNS server.

When an architect has designed the building and described all light points, ventilators, heating- and cooling units, and sensors, it is necessary to identify all these devices spatially and functionally. Storing the triple <Instance>.<ServiceType>.<Location> into DNS-SD represents the commissioning process. The Instance is the unique identifier given to the device in the factory but which has no relation to its later location. The ServiceType together with the Location represent the spatial and functional aspects of the device as specified by the architect.

Design decision: A commissioning tool with access to the network is used for the commissioning phase.

For example, dependent on used technology and production process, the following situation (state) may exist in a host after physical installation of the devices and before commissioning:

- A given host is unaware of its Location.
- A given host knows its ServiceType and Instance. The Instance is also readable by bar code reader.
- The commissioning tool knows all Locations to which hosts need to be assigned.
- Each host has a (site-local) IP address.

Consider the commissioning process (1) with a central DNS-SD server and (2) without a central server using xmDNS. The commissioning processes described below are just examples and should not be taken as working procedures for commissioning devices in a building.

5.3.1. DNS-SD server present

The installer reads with a bar code reader, attached to the commissioning tool, the identifier of the device to commission. It is assumed that the tool can learn the IP address of the device with the given identifier. The tool displays on a screen the physical lay-out of the devices within the building. The installer selects, on the screen of the tool, the physical location of the chosen device. From the designated physical location the tool creates the URI of the selected device. The tool inserts the URI and the IP address into the DNS server. For example the light with URI `light1.o4.b8.example.com` is represented with an AAAA record:

```
light1.o4.b8.example.com AAAA fdfd::1234
```

The tool reads the service name and type from the device using resource information stored according to the link-format [I-D.ietf-core-link-format]. With this information the tool constructs the PTR, SRV and TXT records according to the example presented in section 5.

This is done for all devices within a given part of the building. After the commissioning process, all resources of each device have an URI and IP address which are stored in the central DNS-SD server. When devices are restarted, the DHCP server may allocate new IP addresses to the device and update the DNS server.

5.3.2. DNS-SD server not present

It is assumed that the building network is composed of independent network segments (possibly a single site) such that each device on a given segment can communicate directly with any other device on this segment. The segments are not connected to a 6LBR and have no access

to DNS or other servers. The installer knows these segments and has a list of devices for a given segment. In the tool the installer selects the names which belong to the given building segment. The selected names are converted to site-local authorities and stored in the tool. All devices are assumed to have selected a site-local IP address. Assume that every device has a unique barcode within the building and that the corresponding device knows the bar code number. The installer reads with a bar code reader, attached to the tool, the Instance name of the device to commission. The installer selects, on the screen of the tool, the physical location of the chosen device. The tool knows the authority of the selected device. The tool broadcasts the bar code number and authority to all connected devices. The device with the given barcode number, extends the authority with the path name of the resources. For each resource, the device multicasts the site-local IP-address and the site-local URI to the xmDNS servers in the connected devices. This concludes the commissioning of a network segment. All resources of each device have a site-local URI and a site-local IP address which are stored in the xmDNS servers.

5.4. Proxy discovery

Proxies will be used in CoAP networks for at least two major reasons: (1) http/coap proxy, and (2) proxy of service on battery-less device. The first proxy is probably implemented as forward proxy, while the latter is probably implemented as backward proxy. The battery-less device will at rare occasions (when it is not sleeping) and during installation answer the GET /.well-known/core request. The return data are used by the installation tool to make the proxy device return the same resource names on /.well-known/core as is returned by the sleeping device. An installation tool installs on the proxy all the resources of the sleeping device for which the proxy is assumed to answer. Consequently, the proxy is discovered as a multi-server host with as many path names as it proxies sleeping servers. The servers on sleeping devices should not be discoverable via DNS-SD. However, AAAA records are generated for the sleeping device host name. This host name is used by the proxy to subscribe to the "sporadic" services of the sleeping device. For example assume two sleeping devices, an occupancy sensor and a temperature sensor, and one proxy. Two service types are defined with PTR records in DNS-SD. The identifier id-1 of the proxy is used by the installation tool to define the Instances.

```
_occup_sensor._sub._bc._udp.04.b8 PTR id-1._occup_sensor
_temp_sensor._sub._bc._udp.04.b8 PTR id-1._temp_sensor
```

Two SRV records with accompanying AAAA and TXT records describe the two services in more detail. The services share the same IP address,

are connected to the same port, but do have different paths names. The TXT record is used to specify the path part with "path=".

```
id-1._occup_sensor      SRV  proxy.o4.b8.example.com Port-x
                        AAAA  fdfe:: 1241
                        TXT   path=/os if=ZigBee
id-1._temp_sensor       SRV  proxy.o4.b8.example.com Port-x
                        AAAA  fdfe:: 1241
                        TXT   path=/ts if=ZigBee
sl-ts.o4.b8.example.com AAAA  fdfe::1242
sl-os.o4.b8.example.com AAAA  fdfe::1243
```

The path names /ts and /os are short names for temperature_sensor and occupancy_sensor respectively and were taken over from link-format information contained in the sleeping devices. Two AAAA records are provided for the two sleeping devices. The proxy has used the domain names of the sleeping devices to subscribe to the publications of the two sleeping devices.

It is important to remark that there are now two services with the same resources present on two different devices: the sleeping device and its proxy. When a host invokes the /.well-known/core resource, it should be possible to distinguish between the proxy (to be invoked) and the sleeping device (not to be invoked). The distinction is necessary once the sleeping device is discoverable and the sleeping device is awake from time to time. It is suggested that the link-format syntax allows to make this distinction.

6. Legacy data Representations in CoAP

Before CoAP devices can come to market, manufacturers must agree that the type and resources of the device can be interpreted according to some generally recognized syntax. At this moment no such generally recognized syntax exists for CoAP devices. We do not expect an IETF working group to standardize such a syntax, and we are convinced that syntax standardization is the responsibility of industry standards organizations. Given the long history of building control, many groups have defined a data representation for building control devices for example BACnet, ZigBee, oBIX, LON, KNX, and many others. It is our belief that new representations will be defined and must coexist with the named legacy ones.

The CoAP protocol should transport any data representation, and certainly the legacy ones. It is expected that a CoAP client can handle one or more legacy representation. Given that a CoAP client can handle representation of standard XXX, this I-D proposes that such a CoAP device can communicate with legacy devices via a CoAP/

legacy gateway (router).

6.1. Network architectures

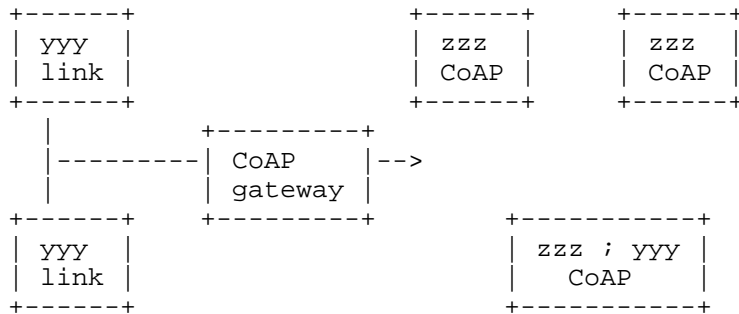


Figure 1: network with multiple representation standards

Figure 1 represents the network architecture which is expected for the purpose of this I-D. The CoAP gateway connects one link with two legacy devices -containing legacy data representation "yyy"- with the wireless CoAP network composed of three CoAP hosts. Two CoAP hosts contain the CoAP stack with a zzz representation and one host contains the CoAP stack with a zzz and an yyy representation. The yyy hosts can freely communicate according to the yyy link protocol over the yyy link. The zzz CoAP hosts, including the zzz;yyy host can freely exchange zzz data representations according to the CoAP protocol over the wireless 6LoWPAN network. The zzz;yyy host can send yyy data representations to the CoAP gateway which passes them on to the specified yyy legacy host. The yyy legacy device returns data to the requesting zzz;yyy CoAP host via the same gateway.

The CoAP hosts can address the legacy devices behind the gateway in at least 4 ways.

- All devices of legacy network YYY share the URI with the CoAP gateway. Every legacy device is a resource for the gateway as seen from the CoAP host. Consequently, the CoAP host sends the message to the IP address of the gateway and the gateway parses the URI-Path to determine the specified legacy device.
- All devices of legacy network YYY have IP addresses different from the IP address of the gateway. Consequently, a CoAP host sends the message to the IP address of the specified device. The routing protocol on the CoAP network makes the message arrive at the CoAP gateway. The gateway determines the specified legacy device from the destination IP address.

- All devices of legacy network YYY have different authorities. The authorities of the legacy device resolve to an IP address of the gateway. This means that the possibly lengthy authority names need to be transmitted. The gateway recognizes the authorities and maps authority to legacy device.
- All devices of legacy network YYY have different ports. This can be expressed in two ways (1) as :port in the URI, or (2) in the DNS-SD records. In the latter case the port is defined in the UDP header and is efficient in packet header size.

The major advantage of all four approaches is that the gateway only handles the URI or IP address and port number to select the destination legacy device independent of the type of legacy device and the contents of the legacy payload of the message. In Figure 1 the gateway connects to a single link. For example, this would be the case for DALI standard. Other legacy standards, like BACnet, LON, allow networks composed of multiple links.

An example of an invocation of a ZZZ service (See figure 2). The resource path /ZZZ identifies the parser of the ZZZ syntax. A 12 octet string completely describes the ZZZ command. The host is completely identified by the authority in the URI. The ZZZ parser on the host is identified by the port number in the UDP header (not shown).

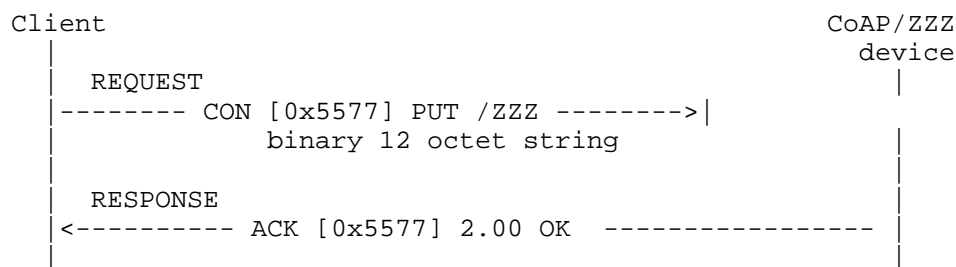


Figure 2: Sending a ZZZ command with CoAP to CoAP/ZZZ device

An example of an invocation of a DALI legacy device behind a gateway is given in figure 3. The resource path /DALI identifies the DALI parser. The application sets a value of 200 in the DALI device in the resource 256 defined by the DALI spec.

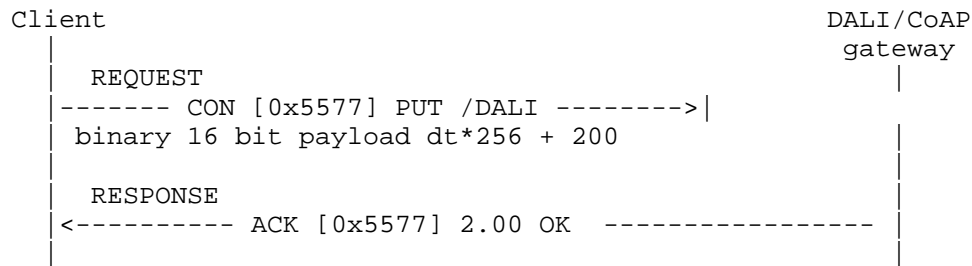


Figure 3: Sending a DALI setting with CoAP to CoAP/DALI gateway

6.2. Discovery of legacy gateways

Discovery of legacy gateways is not very different from discovery of proxies in section 5.4. the consequences for discovery are listed for the four modes of addressing legacy devices via a gateway of section 6.1.

- The gateway presents a list of resources representing the legacy devices. Discovery is done as for other CoAP devices.
- Each legacy device has a different IP address. The gateway must create entries in the DNS for as many legacy devices. The authority of the legacy device is the authority of the gateway with a ServiceType to be specified by the gateway.
- All devices of legacy network YYY have different authorities. In this case each legacy device has the same IP address as the gateway. The gateway must create entries in the DNS for as many legacy devices.
- All devices of legacy network YYY have different ports. The gateway must create entries in the DNS for as many legacy devices. Each entry has the authority of the gateway with a different ServiceType and a different port number.

7. Conclusions

This I-D explains how naming in building control is based on a hierarchical structure of the building areas. It is shown that DNS naming can be used to express this hierarchy in the authority portion of the URI, down to the group or device level. The hierarchical naming scheme need not be standardized, but rather can be designed to suit the application. However, it is recommended that the scheme be employed consistently throughout the delegated subdomain(s).

The authority portion of the URI is resolved by the client, using conventional DNS, into the unicast or multicast IP address of the targeted device(s). Taking advantage of the CoAP design [I-D.ietf-core-coap], the URI-Host option need not be transmitted in requests to origin servers and thus there is no performance penalty for using descriptive naming schemes. The CoAP design allows sending a short URI to distinguish between resources on a given device, resulting in very compact identifiers.

DNS-SD [I-D.cheshire-dnsext-dns-sd] can be used to scale up service discovery beyond the 6LoWPAN. DNS-SD can be used to enumerate instances of a given service type within a given sub-domain. This affords additional flexibility, such as the ability to discover dynamic port assignments for CoAP device, locate CoAP devices by subtype, or bind service names for particular CoAP URIs.

This I-D discusses the addressing, discovery and naming of legacy devices behind gateways. The discovery of backward proxies of sleeping devices is handled in a similar fashion.

A targeted resource is specified by the path portion of the URI. Again, this I-D does not mandate a universal naming standard for resources but uses examples to show how resources could be named for various legacy standards. An obvious requirement for resources that are accessed by multicast is that they MUST all share the same path. It is shown that it is possible to transport legacy commands (e.g. expressed in BACnet, LON, DALI, ZigBee, etc.) inside a CoAP message body. Entering ServiceTypes particular to a given standard necessitates that the standardization body declares the ServiceType to dns.org.

8. Security considerations

TBD: The detailed CoAP security analysis needs to encompass scenarios for building control applications.

Based on the programming model presented in this I-D, security scenarios for building control need to be stated. Appropriate methods to counteract the proposed threats may be based on the work done elsewhere, for example in the ZigBee over IP context.

Multicast messages are, by their nature, transmitted via UDP. Any privacy applied to such messages must be block oriented and based on group keys shared by all targeted devices. The CoRE security analysis must be broadened to include multicast scenarios.

9. IANA considerations

This I-D proposes that associations which standardize device representations (like BACnet, ZigBee, DALI,...) contact IANA to reserve the prefix /.well-known/XXX for the standard XXX.

10. Acknowledgements

This I-D has benefited from conversations with and comments from Andrew Tokmakoff, Emmanuel Frimout, Jamie Mc Cormack, Oscar Garcia, Dee Denteneer, Joop Talstra, Zach Shelby, Jerald Martocci, Anders Brandt, Matthieu Vial, Jerome Hamel, George Yianni, and Nicolas Riou.

11. Changelog

From bc-01 to bc-02

- Removed all references to multicast and multicast scope, given draft of rahman group communication.
- Adapted examples to CoAP-2 and core-link drafts.
- transport short URL for destination recognition.
- Elaborated legacy discovery under DNS-SD.

From bc-02 to bc-03

- Elaboration on gateways, commissioning and legacy networks.
- Recommendation to extend DNS-SD naming with sn, st, and ss attributes.

From bc-03 to bc-04

- moved core link extension sub-section to discovery mapping draft
- extended use of service type
- gave DNS record examples and worked out multifunction device
- added proxy discovery and legacy gateway discovery
- defined path tree and corresponding schema
- reviewed definition of group, device, server, service (interface),

resource, and attribute.

From bc-04 to bc-05

- extended and corrected examples for multi-function devices
- syntax more compatible with other resource discovery I-Ds
- abstract adapted
- more stringent use of the words server, end point, service and devices

12. References

12.1. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.
- [RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC3306] Haberman, B. and D. Thaler, "Unicast-Prefix-based IPv6 Multicast Addresses", RFC 3306, August 2002.
- [RFC3307] Haberman, B., "Allocation Guidelines for IPv6 Multicast Addresses", RFC 3307, August 2002.
- [RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", RFC 3596, October 2003.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

- [RFC3956] Savola, P. and B. Haberman, "Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address", RFC 3956, November 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC5790] Liu, H., Cao, W., and H. Asaeda, "Lightweight Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Version 2 (MLDv2) Protocols", RFC 5790, February 2010.

12.2. Informative References

- [I-D.cheshire-dnsext-dns-sd]
Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-10 (work in progress), February 2011.
- [I-D.cheshire-dnsext-multicastdns]
Cheshire, S. and M. Krochmal, "Multicast DNS", draft-cheshire-dnsext-multicastdns-14 (work in progress), February 2011.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), July 2011.
- [I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-07 (work in progress), July 2011.
- [I-D.martocci-6lowapp-building-applications]
Martocci, J., Schoofs, A., and P. Stok, "Commercial

Building Applications Requirements",
draft-martocci-6lowapp-building-applications-01 (work in
progress), July 2010.

[I-D.rahman-core-groupcomm]

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

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

Shelby, Z. and S. Krco, "CoRE Resource Directory",
draft-shelby-core-resource-directory-01 (work in
progress), September 2011.

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

Lynn, K. and Z. Shelby, "CoRE Link-Format to DNS-Based
Service Discovery Mapping",
draft-lynn-core-discovery-mapping-01 (work in progress),
July 2011.

[I-D.lynn-dnsexst-site-mdns]

Lynn, K. and D. Sturek, "Extended Multicast DNS",
draft-lynn-dnsexst-site-mdns-01 (work in progress),
March 2011.

[BACnet]

Bender, J. and M. Newman, "BACnet/IP",
Web <http://www.bacnet.org/Tutorial/BACnetIP/index.html>,
2000.

[ZigBee]

Tolle, G., "A UDP/IP Adaptation of the ZigBee Application
Protocol", draft-tolle-cap-00 (work in progress),
October 2008.

[LON]

"LONTalk protocol specification, version 3", 1994.

[DALI]

"DALI Manual", Web http://www.dali-ag.org/c/manual_gb.pdf,
2001.

[KNX]

Kastner, W., Neugschwandtner, G., and M. Koegler, "AN OPEN
APPROACH TO EIB/KNX SOFTWARE DEVELOPMENT", Web [http://
www.auto.tuwien.ac.at/~gneugsch/
fet05-openapproach-preprint.pdf](http://www.auto.tuwien.ac.at/~gneugsch/fet05-openapproach-preprint.pdf), 2005.

[IEEE.802.15.4]

"Information technology - Telecommunications and
information exchange between systems - Local and
metropolitan area networks - Specific requirements - Part
15.4: Wireless Medium Access Control (MAC) and Physical

Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)", IEEE Std 802.15.4-2006, June 2006,
<<http://standards.ieee.org/getieee802/802.15.html>>.

[HAYSTACK]

"Project Haystack", Web <http://project-haystack.org/>, 2011.

[oBIX]

Frank, B., Ed., "oBIX working group",
Web <http://www.obix.org>, 2006.

[Fielding]

Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures, Second Edition", Doctoral dissertation , University of California, Irvine ,
Web <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.html>, 2000.

[ZigBee-IP]

"ZigBee Smart Energy Profile 2.0 Application Protocol Specification", Draft ZigBee-11167, March 2011.

[dns-sd]

"dns-sd servicetype registration",
Web <http://www.dns-sd.org/ServiceTypes.html>, 2011.

Authors' Addresses

Peter van der Stok
Philips Research
High Tech Campus 34-1
Eindhoven, 5656 AA
The Netherlands

Email: peter.van.der.stok@philips.com

Kerry Lynn
Consultant

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 16, 2012

A. Yegin
Samsung
Z. Shelby
Sensinode
October 14, 2011

CoAP Security Options
draft-yegin-coap-security-options-00

Abstract

This document specifies a set of Constrained Application Protocol (CoAP) options that are used for providing data origin authentication, integrity and replay protection, and encryption for the CoAP messages.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 16, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Specification of Requirements	3
2. Details	3
2.1. CryptoInitiate Option	3
2.2. CryptoTerminate Option	5
2.3. CryptoEncap Option	6
2.3.1. AES-CCM (Counter with Cipher Block Chaining Message Authentication Code)	7
3. Security Considerations	8
4. IANA Considerations	8
5. Acknowledgments	8
6. Normative References	8
Authors' Addresses	9

1. Introduction

The CoAP base specifications identifies DTLS [RFC4347] and IPsec [RFC4303] as two of the methods that can be used for providing data origin authentication, integrity and replay protection, and encryption for the CoAP messages. This document defines an alternative method that operates at the CoAP layer. It is expected that a solution defined within the same layer as the base specification would provide an efficient alternative to DTLS and IPsec in terms of crypto context setup, packet size, and implementation overhead. How the CoAP client and server decide which one(s) of the DTLS, IPsec, and this method to use is outside the scope of this document.

Three new CoAP options are defined in this document. The CryptoInitiate Option is used for initializing a crypto context between the two end-points. It is used for setting up the crypto parameters that can be used with the subsequent CoAP messages. The CryptoEncap Option is used for securely delivering the CoAP messages using the crypto context between the end-points. This option can provide either data origin authentication, replay and integrity protection for the whole CoAP message, or encryption for the options and the payload of the message, or both depending on the crypto algorithm used by the crypto context. The CryptoTerminate Option is used for deleting the crypto context that is no longer needed by the end-points.

The crypto context is based on a symmetric secret key shared between the end-points. It is assumed that such a key is established a priori. Methods used for establishing such keys are outside the scope of this document.

1.1. Specification of Requirements

In this document, several words are used to signify the requirements of the specification. These words are often capitalized. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Details

2.1. CryptoInitiate Option

The CryptoInitiate Option (option no. TBD) is a Critical option that is used for setting up a crypto context between the CoAP client and server. Use of this option is not mandatory, as an out-of scope

method may substitute for its functionality.

When a CoAP client wants to setup a security context to be used with the CryptoEncap Option, it SHALL send a confirmable request message that includes a CryptoInitiate Option. This message MAY include other options and payload (i.e., piggybacked). The Code field of the message SHALL be set to 0 if no other options or payload are included.

The Option Value field of the CryptoInitiate Option contains a Context ID. This is an identifier value assigned to the context by the client. The Context ID space is shared between the client and the server. If an identifier is used for the initialization of a context by one of the end-points, the other end-point SHALL not use the same identifier for initializing another context. If the client re-uses an identifier that is already used with a context that was initialized by the same client, then the new request is used for re-initialization of that context. Otherwise, a new context with the provided identifier is be created. Note that these actions are executed only when the option is considered valid by the server as outlined below.

The Key Name parameter in the Option Value field contains an identifier for the shared secret key that will be used in the context. For example, "Device1-Device2-key#5".

The CryptoAlgos parameter in the Options Value field contains a list of one or more proposed crypto algorithms supported by the client. For example, AES-CCM, AES-CTR, etc.

When the server receives the request, it SHALL check the Option Value fields. If the Context ID is not already used for a context that was initialized by the server, and at least one of the proposed crypto algorithms is supported by the server, and the Key Name is recognized, then the server SHALL send back an acknowledgement message that contains CryptoInitiate Option whose Context ID is copied from the request, CryptoAlgos includes the selected crypto algorithm from the proposed list, and Key Name copied from the request.

If at least one of these checks fails, the server SHALL still send back a response with a CryptoInitiate Option. But this time the problematic parameters will include special reserved values to indicate the issue. If the Context ID is already used by the server, then the server's response SHALL carry the value of 0 in the Context ID field. If none of the proposed crypto algorithms are supported by the server, the CryptoAlgos field SHALL contain no algorithms. If the Key Name is not recognized by the server, the Key Name field

SHALL be set to "Unknown!".

A client MAY have more than one crypto context with the same server. It MAY negotiate them around the same time, or at different times. Each negotiation SHALL be carried over a dedicated request-response. In other words, there SHALL be at most one CryptoInitiate Option in a CoAP message.

The same context MAY be used by the both end-points whenever they are sending request and response messages. On the other hand, each end-point MAY also choose to initialize dedicated contexts to be used for the request-responses initiated by themselves.

Option Value field of the CryptoInitiate Option SHALL include the following parameters in the given order:

- Context ID: One-octet unsigned integer value. Value 0 is reserved to indicate invalid context ID, values 1-255 are valid.
- Key Name Length: One-octet unsigned integer value. Indicates the length of the Key Name field.
- Key Name: A non-NULL-terminated string. Value "Unknown!" is reserved for the server to indicate an unknown key name.
- CryptoAlgos: Zero or more one-octet unsigned integer values. Value 0, and 2-255 are reserved for future use. Value 1 indicates AES-CCM [NIST_SP800_38C]. Number of crypto algorithms carried in the option is determined by the difference between the option length and the space already consumed by the Context ID, Key Name Length, and Key Name parameters.

2.2. CryptoTerminate Option

CryptoTerminate Option (option no. TBD) is a Critical option that is used for deleting a crypto context shared between the CoAP client and server. Use of this option is not mandatory, as an out-of scope method may substitute for its functionality.

When a CoAP client wants to delete a security context with a server, it SHALL send a confirmable request message that includes a CryptoTerminate Option. This message MAY include other options and payload (i.e., piggybacked). The Code field of the message SHALL be set to 0 if no other options or payload are included.

Option Value field of the CryptoInitiate Option contains a Context ID identifying the context to be deleted. A client SHALL NOT attempt to delete a context that was not initialized by itself.

When the server receives a request carrying a CryptoTerminate Option, it SHALL check the Context ID. If the Context ID carries an identifier value for a context that was initialized by the client, then the server SHALL send an acknowledgment message carrying the exact same Option. Subsequently, the server SHALL delete the crypto context. The server MAY delay the deletion for handling the possibility of re-transmitted requests. If the Context ID does not match any context that was initialized by the client, then the server SHALL send the option with Context ID set to 0.

Option Value field of the CryptoTerminate Option SHALL include the following parameter:

- Context ID: One-octet unsigned integer value. Value 0 is reserved to indicate invalid context ID, values 1-255 are valid.

2.3. CryptoEncap Option

The CryptoEncap Option (option no. TBD) is a Critical option that is used for providing security for the messages carrying this option. It is used with a crypto context that is already initialized by the client. It is up-to the client to decide which CoAP messages would carry this option, and which crypto context is used with it. When this option is used with a request, the server SHALL also use this option and with the same crypto context in its response.

An appropriate crypto algorithm SHALL be negotiated for the crypto context whether this option is used for data origin authentication, integrity and replay protection, or for encryption, or both (e.g., AES-CCM).

Option Value of a CryptoEncap Option in a request message SHALL start with a Context ID parameter. Client SHALL include the desired crypto context's identifier value in this parameter.

If the CryptoEncap Option is used for data origin authentication, integrity and replay protection as indicated by the crypto algorithm, then it SHALL also include a Nonce value for freshness and a MAC (Message Authentication Code) value. Client SHALL ensure the same Nonce value is never re-used with the same crypto context. The exact format/content of the Nonce and MAC depends on the crypto algorithm being used. MAC value is computed using the complete CoAP message (including the CryptoEncap Option with its MAC value set to all zeroes), shared secret key associated with the crypto context, and the keyed hashed function indicated by the crypto algorithm.

If the CryptoEncap Option is used for encryption as indicated by the crypto algorithm, then it SHALL include OptionCount and EncryptedData

parameters. All other Options and the Payload SHALL be carried in EncryptedData parameter in encrypted form. The algorithm used for encryption is determined by the crypto algorithm. OptionCount indicates the number of Options carried in the EncryptedData.

When a server receives a CoAP message that includes a CryptoEncap Option, it SHALL first process this option. Server SHALL silently drop the message if the Context ID is unknown.

If a MAC parameter is used as indicated by the crypto algorithm, then the server SHALL compute its own MAC value using the complete CoAP message (including the CryptoEncap Option with its MAC value set to all zeroes), shared secret key associated with the crypto context, and the keyed hashed function indicated by the crypto algorithm. If the computed and received MAC values do not match, then the server SHALL silently drop the message.

If an EncryptedData parameter is used as indicated by the crypto algorithm, then the server SHALL decrypt the Options and the Payload using that parameter, the shared secret key associated with the crypto context, and the crypto algorithm. Following these procedures the server SHALL continue processing the CoAP message as usual [I-D.ietf-core-coap].

2.3.1. AES-CCM (Counter with Cipher Block Chaining Message Authentication Code)

When the negotiated crypto algorithm is AES-CCM (code 1), then the CryptoEncap Option is used for data origin authentication, integrity and replay protection, and encryption.

The formatting and counter generation function of AES-CCM as specified in Appendix A of [NIST_SP800_38C] is used with the following parameters:

n, octet length of Nonce, is 12.

t, octet length of MAC, is 16.

q, octet length of message length field is 3.

When the crypto algorithm is AES-CCM, then the Option Value field in CryptoEncap Option SHALL include the following parameters in the given order:

- Context ID: One-octet unsigned integer value.

- Nonce: 12-octet value.
- MAC: 16-octet value.
- OptionCount: One-octet unsigned integer value.
- EncryptedData: Variable-length value.

3. Security Considerations

It is possible to use this security methon in conjunction with the others, such as DTLS and IPsec.

Future specification can define support for additional crypto algorithms.

4. IANA Considerations

The following IANA actions are required by this specification:

- Assignment of a standard CoAP Option code TBD for CryptoInitiate Option
- Assignment of a standard CoAP Option code TBD for CryptoTerminate Option
- Assignment of a standard CoAP Option code TBD for CryptoEncap Option
- Creation of crypto algorithm identifier space for CryptoAlgos parameter.
- Assignment of an crypto algorithm identifier 1 for AES_CCM.

5. Acknowledgments

TBD

6. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-07 (work in progress), July 2011.

[NIST_SP800_38C]

Dworkin, M., "Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality", May 2004.

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

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

Authors' Addresses

Alper Yegin
Samsung
Istanbul
Turkey

Email: alper.yegin@yegin.org

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Email: zach@sensinode.com

