CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 8, 2011

C. Bormann
Universitaet Bremen TZI

March 7, 2011

CoRE Simple Server Discovery
draft-bormann-core-simple-server-discovery-00

Abstract

CoRE defines a mechanism for resource discovery based on Web linking. Many applications also need a simple form of discovery for the servers carrying these resources. This specification shows a simple way to extend the link-based resource discovery into a basic form of server discovery.

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 in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on September 8, 2011.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Terminology 3
2. Discovery Servers 4
3. CoAP Server Discovery 5
4. Finding a Candidate CoAP Server Discovery Server 6
5. IANA Considerations 7
6. Security Considerations 8
7. Acknowledgements 9
8. References 10
8.1. Normative References 10
8.2. Informative References 10
Author's Address 11

1. Introduction

CoRE defines a mechanism for resource discovery based on Web linking [RFC5988] [I-D.ietf-core-link-format]. Many applications also need a simple form of discovery for the servers carrying these resources.

More sophisticated CoRE server discovery mechanisms have been proposed [I-D.brandt-coap-subnet-discovery]. The present specification is not intended as a competing protocol but shows a very simple way to extend the link-based resource discovery into a basic form of server discovery. It is an open question whether different applications need different discovery solutions or whether there can be a "scalable" solution that covers both simple and complex scenarios.

The protocol as designed here has been prototyped in the SAHARA project at TZI in just a few lines of code. The current version -00 of this document serves as an initial draft that is intended to spark discussion. Not all aspects of the protocol as specified are part of the current prototype. We expect to update the specification both based on WG feedback and as we gain experience with the prototype.

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 RFC 2119. (See [RFC2119].)

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

The terminology from [I-D.ietf-core-coap] applies.

In addition:

CoAP Server Discovery: This protocol.

CoAP Server Discovery Server (CSDS): A server for this protocol, which interacts with other CoAP servers, collects resource discovery information from them and integrates it into larger resource discovery information sets.

Candidate CSDS: An IP address that might or might not be useful for conversion to a CSDS URI.

2. Discovery Servers

This specification defines a simple form of server discovery that makes use of CoAP Server Discovery Servers (CSDS), which are addressed simply using the CoAP protocol [I-D.ietf-core-coap].

The assumption is that there is a way to find one or more CSDSs (see Section 4 for a number of such ways). New CoAP servers that want to provide discoverable services can make themselves known at the CSDSs. CoAP clients can ask the CSDSs for a resource directory in the usual way, which will include both information about the discovery server's own resources and information about other servers that made themselves known to the discovery servers.

3. CoAP Server Discovery

Simple CoAP Server Discovery makes use of a simple mapping from a server's IP address to a default discovery URI: The default discovery URI is created from the 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].

A CoAP server that wants to make itself discoverable occasionally sends a POST request to the default discovery URI of any Candidate CSDS that it finds.

The body of the POST request is either

- empty, in which case the CoAP Server Discovery Server is encouraged by this POST request to perform GET requests at the requesting server's default discovery URI.

or

- a link-format document, which indicates the specific services that the requesting server wants to make known to the CSDS.

The CSDS integrates the information it received this way into its resource directory. It MAY make the information available to further CSDSs, if it can ensure that a loop does not form. The protocol used between CSDSs to ensure loop-free operation is outside the scope of this document.

4. Finding a Candidate CoAP Server Discovery Server

CoAP servers that want to contact a CSDS can obtain candidate IP addresses for such servers (Candidate CSDS) in a number of ways.

In a 6LoWPAN, good candidates can be taken from:

- specific static configuration (e.g., anycast addresses), if any,
- the ABRO option of 6LoWPAN-ND [I-D.ietf-6lowpan-nd],
- other ND options that happen to point to servers (such as RDNSS),
- DHCPv6 options that might be defined later.

In networks with more inexpensive use of multicast, the Candidate CSDS may be a well-known multicast address, i.e. CSDS are found by simply sending POST requests to that well-known multicast address (details TBD).

As some of these sources are just (more or less educated) guesses, CoAP servers MUST make use of any error messages to very strictly rate-limit requests to Candidate CSDSs that don’t work out. E.g., an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a CSDS.

5. IANA Considerations

This document has no actions for IANA.

6. Security Considerations

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

7. Acknowledgements

The concept for this document was inspired by Zach Shelby et al.’s CoAP discovery node that was available in various CoAP interop events. The current implementation was performed by the students of the SAHARA project, including Bengt Kohrt, Julian Kornberger, Henning Mueller, and Christian Thedieck. Philip Nguyen read an early draft of this document (but all errors are mine). Anders Brandt’s draft [I-D.brandt-coap-subnet-discovery] is a fine piece of work and certainly motivated me to finally write this up.

8. References

8.1. Normative References

[I-D.ietf-6lowpan-nd]
Shelby, Z., Chakrabarti, S., and E. Nordmark, "Neighbor Discovery Optimization for Low-power and Lossy Networks", draft-ietf-6lowpan-nd-15 (work in progress), December 2010.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-02 (work in progress), December 2010.

[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.

8.2. 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.

Author's Address

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

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

CoRE
Internet-Draft
Intended status: Experimental
Expires: September 8, 2011

A. Brandt
Sigma Designs
March 7, 2011

Discovery of CoAP servers across subnets
draft-brandt-coap-subnet-discovery-00

Abstract

The document describes the process of discovering CoAP servers distributed in multiple subnets in a non-specified topology. CoAP Discovery Gateways are used to discover one subnet from another. CoAP Discovery Gateways may provide caching to enable discovery of sleeping nodes in LLN environments. The solution scales to large installations since discovery is handled by the client in an incremental fashion.

Table of Contents

1. Introduction 3
1.1. Requirements Language 3
2. Requirements for Discovery services for very constrained nodes 3
2.1. Home Control network evolution - scenarios 4
2.1.1. Scenario A: Retail home control starter kit 4
2.1.2. Scenario B: Service provider home control offering . . 5
2.2. Discovery 6
2.3. Zero-Configuration 6
2.4. Multiple unique routable subnets 7
3. Building blocks of a CoAP Discovery infrastructure 7
3.1. Stand-alone CoAP Server 8
3.2. CoAP Discovery Gateway 8
3.3. Caching CoAP Discovery Gateway 8
3.4. CoAP Discovery Client 9
4. CoAP Discovery 10
4.1. Topology Discovery 11
4.1.1. Topology Path String definitions 11
4.1.2. ?n=DiscoveryGateway 13
4.1.3. Discovering Gateway interfaces - an example 13
4.2. Initial Topology Discovery 13
4.3. Incremental Topology Discovery 14
4.3.1. Multicast domain behind routers 14
4.4. CoAP Server Discovery 15
5. Examples 15
5.1. Discovery across CoAP Discovery Gateways 15
5.2. /topology path formats 17
6. IANA Considerations 19
7. Security Considerations 19
8. References 20
8.1. Normative References 20
8.2. Informative References 20
Appendix A. Open Issues 20
A.1. Large reports 20
A.2. M2M Filtering 20
A.3. Routable subnets 21
A.4. Compressing responses from caching CoAP Discovery Gateways 21
A.5. Integrated Battery support 21
Appendix B. Acknowledgements 22
Author’s Address 22

1. Introduction

This document describes the process of discovering CoAP servers distributed in multiple subnets in a non-specified topology. CoAP (Constrained object Application Protocol) Discovery Gateways are used to discover one subnet from another. CoAP Discovery Gateways may provide caching to enable discovery of sleeping nodes in Low-power and Lossy Network (LLN) environments. Caching CoAP Discovery Gateways may also have the purpose of preserving bandwidth in an LLN environment. No synchronization of databases is required. The solution scales to large installations since discovery is handled by the client in an incremental fashion.

CoAP servers may be running on a variety of physical layers; each implementing a subnet. A lamp module may be running over IEEE 802.15.4. A movement sensor may be running over Z-Wave.

The document presents requirements to a home control discovery solution and proposes a solution based on the CoAP link format [I-D.ietf-core-link-format].

CoAP Subnet Discovery Must support IPv6. An implementation MAY implement support for both IPv4 and IPv6 .

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. Requirements for Discovery services for very constrained nodes

The term constrained node covers a range of nodes that are constrained with regards to memory size, power consumption, packet size or CPU capabilities. This document covers nodes for applications within home control and building control. A Low-power Lossy Network (LLN) is used for communication.

The basic functionality of devices for home control and building control is the same: measure temperature, detect movement, dim light, operate locks, etc. The discovery, management and operation of networks is however somewhat different. This document addresses discovery requirements specific to the most constrained devices and outlines how proper solutions may address home control and building control technologies in the same way.

2.1. Home Control network evolution - scenarios

Home control systems may be constructed from consumer products acquired in a retail store. The products may come from multiple vendors. The user may have no knowledge of network management. The user may have no existing local network. If there is a local network, it may have no DHCP or DNS infrastructure. Devices must always work; the consumer price level does not leave much margin for support hotlines.

The following scenarios assume one common application protocol. Multi-protocol support may be achieved via application gateways; potentially with CoAP as the common language. This is out of scope of this document.

2.1.1. Scenario A: Retail home control starter kit

This scenario outlines how LLN nodes may be used in the most simple network configurations and how such simple networks may grow from there. In such simple networks the 6LoWPAN border router is reduced to a conceptual function. The remote control acts as a coordinating master node on the link layer as well as a 6LoWPAN border router. The remote control enters a sleep state when it is not in use.

1. Starter kit bring-up

   A user starts with an RF remote control and three RF plug-in modules. The remote control assigns unique link-layer addresses and ULA IP addresses [RFC4193] to modules during network bootstrapping so that the remote control and the plug-in modules are in the same IP network. ULA IP addresses are needed as multi-hop routing may be used inside the LLN network. Prefixes and header compression CIDs have infinite lifetime as there is no listening border router.

   Remote control buttons are mapped to (groups of) IP addresses of the plug-in modules. The setup mode may be activated via a special key sequence in the remote control.

2. Adding sensors

   The user adds another plug-in module and a movement sensor to the network. The remote control is still used as a coordinating master node. The IP address of the target module is stored in the movement sensor. When the sensor detects a movement, it sends an "ON" command to the plug-in module.

3. Adding home control to the smart phone

   The user adds a border router to interface the LLN network to the LAN (and WiFi) of the house.
   The user connects a smartphone to the WiFi network. A home control smartphone app performs home control resource discovery. A list of LLN nodes allows the user to configure smartphone widgets for scene control of lamps; e.g. "Watch TV" or "Doing the dishes".

2.1.2. Scenario B: Service provider home control offering

In this scenario, a consumer receives a pre-bundled kit from a service provider:

- A combined internet access router and LLN border router with WiFi support
- Three plug-in modules for installation in the home
- A personal web profile on the service provider web portal
- A smartphone app for control via WiFi

  Remote access credentials, LLN prefix and other parameters are preconfigured by the service provider.

1. Setting up the kit

   The LLN border router manages link-layer node properties as well as prefix assignment, etc. A web page is used to add the three plug-in modules to the LLN. The smartphone app controls the plug-in modules via WiFi. Routable addresses allow the app to reach the modules from the WiFi subnet.

2. Adding other technologies

   The original starter kit was a wireless LLN. The user connects a power-line LLN border router to the LAN, thus forming a backbone network for the two LLN border routers. The user includes power-line based plug-in modules with the new power-line LLN border router.
   Each individual border router manages local ULA prefixes and header compression CIDs.

3. Re-discovering the network

   The smartphone app rediscovers home control resources in both

LLNs and the backbone network. The lists of home control resources allow the user to configure smartphone widgets for scene control of lamps in both subnets.
The border routers runs a routing protocol on the backbone to allow IP packets to traverse from one subnet into the other without any manual configuration of routers.

2.2. Discovery

Discovery serves multiple purposes in a home control network:

1. When the home control network has grown from the original starter kit to a substantial number of nodes, the owner may get a desire for centralized management of the network. The management tool needs a way to request information from each node in the network. Typically, nodes will carry no visible identification at this time. If possible, non-functioning nodes should also be identified. Once nodes are identified, the user may locate the nodes physically and assign naming and location information, e.g. "bedroom, ceiling light" or "living room, drapes".

2. When setting up a remote control, the user may want to browse all drape controllers in the entire network - both in the wireless LLN and in the powerline LLN. The wireless drape controllers may be battery powered and sleeping most of the time. The powerline drape controllers may be in a dedicated powerline LLN subnet behind several border routers. It is a challenge to discover nodes in other subnets. A multicast-based discovery protocol like mDNS cannot have its messages forwarded by routers since it uses link-local addressing. Even if mDNS messages could be forwarded, some LLNs featuring multi-hop routing do only support multicast in a very slow and inefficient fashion. Assuming multicast messages could be distributed over a LLN, sleeping nodes would not be able to detect discovery requests.

A solution is required which

- Does not flood the LLN with unnecessary discovery traffic
- Does not require multicasting in LLNs
- Does allow the discovery of sleeping nodes

2.3. Zero-Configuration

One major property of home control networks is the absence of a professional installer. When making consumer products, one cannot make any assumptions about the qualifications of the user. Simple

operation is a requirement. As an example, a user may associate a light module by activating a setup sequence on a wall switch and then pushing a button on the light module. The user never sees any data. Most users actually do not care about the IP address of the light module.

No border router, DHCP server or DNS server is present in a starter kit network. Modules that were initially purchased and configured with a remote control may one day be used in a far more advanced installation with global routable prefixes and hierarchical DNS naming.

2.4. Multiple unique routable subnets

Home control domains may be composed of devices communicating via one or more border routers, e.g power-line to wireless, optionally via a backbone network. A wireless home control LLN may in itself contain routers to do multihop forwarding within the home control LLN. The two subnets may host different MAC/PHYs as well as different routing protocols. The user can extend the home control starter kit with another subnet without having to re-install the original subnet. The construction of unique local subnet prefixes is described in[RFC4193].

3. Building blocks of a CoAP Discovery infrastructure

CoAP defines a protocol for exchange of requests and commands between constrained nodes. Already described in [I-D.ietf-core-coap], the CoAP Server is a host capable of responding to CoAP messages. CoAP Servers may be classified into the following sub-types:

- Stand-alone CoAP Server
- CoAP Discovery Gateway
- Caching CoAP Discovery Gateway

These are described in the following sections. Finally, the CoAP Discovery Client is described.

CoAP Discovery Gateways MAY advertise legacy devices along with native CoAP servers. CoAP Discovery Gateways MAY provide access to legacy services via CoAP request and control messages. Discovery of diverse resource types enables a migration path from legacy technologies towards an all-CoAP infrastructure.

3.1. Stand-alone CoAP Server

A stand-alone CoAP Server does not provide access to other CoAP Servers; physical or logical. Typically a stand-alone CoAP Server is able to perform some action, e.g. measuring a temperature or turning on light.
A CoAP Server reports periodically to the Discovery Gateway via the 6LoWPAN ND address registration process, e.g. once every hour. Battery operated CoAP servers may run out of battery. Light modules may become defect. The reporting allows the Discovery gateway to monitor the availability of CoAP Servers.

3.2. CoAP Discovery Gateway

A CoAP Discovery Gateway is a CoAP enabled router interconnecting different subnets, e.g. a LLN Border Router. The subnets may host stand-alone CoAP Servers as well as other CoAP Discovery Gateways. Each CoAP Discovery Gateway interface MUST respond to the CoAP Discovery request "GET /.well-known/core?n=DiscoveryGateway". When queried, the gateway MUST report all other interfaces maintained by the discovery gateway.

The Discovery Gateway MAY maintain a list of CoAP Servers that recently stopped sending address registrations. How a CoAP Discovery Gateway is to advertize such CoAP Servers is TBD.

3.3. Caching CoAP Discovery Gateway

A Caching CoAP Discovery Gateway performs caching of discovery information on behalf of other nodes in a given subnet. A discovery gateway interface MUST respond to the CoAP Discovery request "GET /.well-known/core?n=DiscoveryGateway". When queried, the discovery gateway MUST report all other interfaces maintained by the discovery gateway. The gateway MUST indicate if respective interfaces are of the type "CoAP Discovery Gateway" or "Caching CoAP Discovery Gateway". This information MAY be be ignored by a discovery client.

CoAP Servers reporting to a Caching CoAP Discovery Gateway MAY respond to CoAP Discovery requests. A Caching CoAP Discovery Gateway MUST intercept discovery requests and respond on behalf of CoAP Servers. This allows sleeping nodes to be discovered, saves LLN bandwidth and allows the Caching CoAP Discovery Gateway to maintain connectivity state information like "online/offline/unstable" per CoAP server.

The Caching CoAP Discovery Gateway MAY maintain a list of CoAP Servers that recently stopped sending address registrations. How a CoAP Discovery Gateway is to advertize such CoAP Servers is TBD.

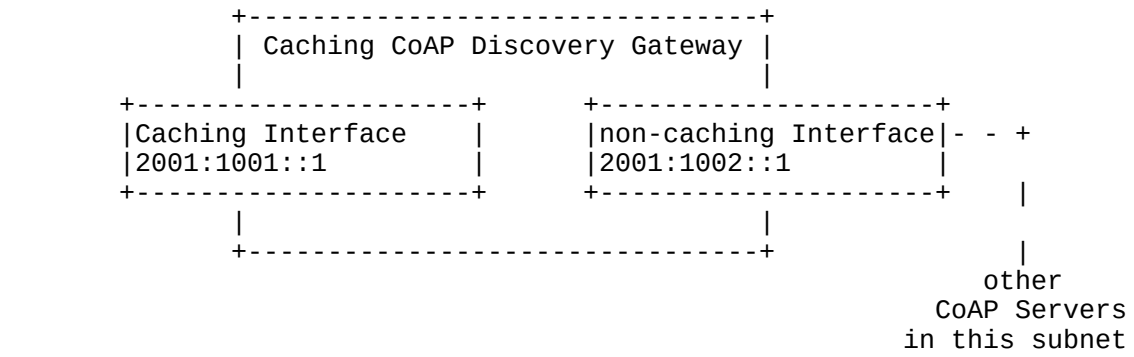


Figure 1: Caching CoAP Discovery Gateway

A Caching CoAP Discovery Gateway MAY report that it is a (non-caching) CoAP Discovery Gateway on some interfaces; refer to Section 3.2 .

The device type "Caching CoAP Discovery Gateway" only indicates that discovery information is cached. Caching of real-time data from CoAP servers is out of scope of this document.

A Caching CoAP Discovery Gateway SHOULD NOT interfere with any other traffic than Discovery Requests for Discovery Gateways or the capabilities of CoAP Servers which report to the CoAP Discovery Gateway, e.g. "Device type = Thermostat". A Caching CoAP Discovery Gateway MUST NOT cache any changing parameters.

3.4. CoAP Discovery Client

A CoAP Discovery Client uses the CoAP Discovery request "GET /.well-known/core?n=DiscoveryGateway" to initiate a discovery session. The target for the discovery request depends on the properties of the actual network. Two cases apply:

1. Client is in a LLN domain with no multicast support

   The initial request is sent to the Authoritative Border Router of that LLN.

2. Client is in a link-local subnet domain with multicast support (like Ethernet)

   The initial request is transmitted to the link-local "all-routers" multicast address.

If discovery requests cause CoAP Discovery Gateways to announce other CoAP Discovery Gateways in other subnets, additional discovery requests are directed to those CoAP Discovery Gateways.

A Discovery Client may run in remote controls, smart phone apps or central management systems for home automation or building control.

The discovery process depends on the presence of a CoAP discovery gateway in the subnet of the discovery client. Since CoAP subnet discovery uses normal CoAP messages, link-local discovery works "out of the box" in link-local enabled environments. Refer to [I-D.ietf-core-link-format].

4. CoAP Discovery

CoAP Discovery is a hierarchical process and involves two phases: CoAP Topology Discovery and CoAP Server Discovery.

The purpose of the Topology Discovery phase is to establish a snapshot of the available CoAP Discovery Gateways.

CoAP messages are used to discover CoAP Discovery Gateways in a hierarchical fashion. Having completed the topology discovery phase, a CoAP client may initiate discovery of particular CoAP server resources, e.g. light dimmers, or a more general wildcard discovery may be done by the client; building a complete database. The same request MUST be sent to each discovered CoAP Discovery Gateway interface in a sequential fashion.

The information may be presented, e.g. in lists of different device types. CoAP subnet discovery enables access to end nodes in multiple subnets without any manual configuration of routers. The topology discovery process may return information on Caching CoAP Discovery Gateways. Caching CoAP Discovery Gateways allow the discovery of sleeping and defective nodes but require that CoAP clients implement 6lowPAN-ND address registration [I-D.ietf-6lowpan-nd]with the Authoritative Border Router. Management and naming related issues of CoAP servers in building control are discussed in [I-D.vanderstok-core-bc].

CoAP Discovery depends on the ability to traverse subnets. Thus, all CoAP Servers MUST have routable IPv6 addresses; either in global prefixes or according to ULA principles. The border router is typically the physical device implementing the CoAP Discovery Gateway function in a LLN. This specification collapses the address of the default gateway (border router) and the discovery gateway in order to limit the number of IP addresses that LLN nodes have to manage - and

to avoid having to distribute the address of the CoAP Discovery Gateway in LLNs. RAs already convey the IP address of the default gateway.

4.1. Topology Discovery

A client may be anywhere in the topology when initiating the Topology Discovery. Any topology may be traversed (if allowed by firewall policies in border routers).

The discovery process allows a client to discover CoAP servers according to the classification described in Section 3.

4.1.1. Topology Path String definitions

A CoAP Discovery Gateway or caching CoAP Discovery Gateway MUST support the following CoAP messages:

- GET /.well-known/core?n=DiscoveryGateway
- GET /.well-known/core/topology
- GET /.well-known/core/topology/device
- GET /.well-known/core/topology/interfaces
- GET /.well-known/core/topology/servers

4.1.1.1. /topology/device

A CoAP Discovery Gateway MUST report one of the device types listed in Figure 2.

Device type	Label	Interpretation
1	Discovery Gateway	This is a CoAP Discovery Gateway interface
2	Discovery Gateway, caching	This CoAP Discovery Gateway interface is caching

Figure 2: CoAP Discovery Device Types

The report formats are described in the following sections.

4.1.1.2. /topology/interfaces

A CoAP Discovery Gateway MUST return the structure

```
[interface address_1]
...
[interface address_n]
```

in response to a query for /topology/interfaces.

The processing of a discovery request depends on the receiving interface:

If the request is targeting the gateway interface that physically received the request, the response contains all subnet interfaces of the discovery gateway.

If the request is targeting another gateway interface than the gateway interface that physically received the request, the response contains all discovery gateways known to the subnet of the targeted interface.

4.1.1.3. /topology/servers

A Caching CoAP Discovery Gateway MUST return the structure

```
[server address_1]
...
[server address_n]
```

in response to a query for /topology/servers.

If the request is targeting the gateway interface that physically received the request, the response contains the identity of known CoAP Servers that report to this Caching CoAP Discovery Gateway interface.

If the request is targeting another gateway interface than the gateway interface that physically received the request, a (non-caching) CoAP Discovery Gateway interface in a subnet supporting multicast MUST issue a multicast request for all CoAP Servers in response to a query for /topology/servers. The individual CoAP Server responses are returned directly to the requesting discovery Client.

4.1.2. ?n=DiscoveryGateway

A CoAP Discovery Gateway MUST report the path /topology in response to ?n=DiscoveryGateway. A CoAP Discovery Gateway MAY report other paths as well.

4.1.3. Discovering Gateway interfaces - an example

The query string ?n=DiscoveryGateway is used for discovering topology information in a CoAP enabled infrastructure.

```
[Client sends multicast request to WiFi subnet]
| CoAP GET /.well-known/core?n=DiscoveryGateway

[LLN Border Router responds]
| 200 OK
|
| core://2001:.../topology/device;ct=0;n="DiscoveryGateway"

[Client sends unicast request to the responding CoAP Discovery Gateway]
[(LLN border router)]
| CoAP GET /.well-known/core/topology/interfaces

[LLN Border Router responds]
| 200 OK
|
| 2001:0db8:85a3:beef:0000:8a2e:0370:7334
| 2001:0db8:85a3:babe:0000:8a2e:0370:4321
```

It is seen that the initial Discovery Gateway request only returns a single string: "/topology/device/...". Since the path "/topology/interfaces" is mandatory for CoAP Discovery Gateways, the client may request this structure as soon as it has detected the CoAP Discovery Gateway.

4.2. Initial Topology Discovery

A Topology Discovery operation is initiated by a CoAP client with the CoAP message GET /.well-known/core?n=DiscoveryGateway in one of the following ways.

1. If a client resides in a multicast enabled environment (like Ethernet or WiFi) the client issues a multicast message (as described in [I-D.ietf-core-link-format]) to the "all nodes" address. All on-link CoAP Discovery Gateways MUST respond to the GET message by returning a list of other interfaces of the respective CoAP Discovery Gateways. In order to avoid collisions, the responding CoAP Discovery Gateways MUST insert a 0..MAX_RA_DELAY_TIME [I-D.ietf-6lowpan-nd] random delay before

responding.

2. If a discovery client resides in a LLN environment (like IEEE 802.15.4 or Z-Wave) the client issues a unicast message to the border router of the LLN. The default border router of the LLN MUST respond to a CoAP Discovery request by returning a list of other interfaces of that particular CoAP Discovery Gateway.

The discovery client builds a list of reported subnets that it has to discover. Duplicates MUST be omitted.

4.3. Incremental Topology Discovery

The client holds a list of reported discovery gateway subnets that it has to discover; either from the Initial Topology Discovery or from a previous Topology Discovery.

For each subnet interface, the client sends a unicast GET /.well-known/core?n=DiscoveryGateway message to the interface. In its default configuration, a CoAP Discovery Gateway MUST return the address of each remote interface. A CoAP Discovery Gateway MAY be configured to return URIs for identification. CoAP Discovery MUST support zero-configuration environments like home control where no DNS server can be assumed.

4.3.1. Multicast domain behind routers

If a discovery client initiates discovery from a LLN environment, it may reach a backbone router interface residing in a multicast enabled network domain such as Ethernet. When a CoAP Discovery Gateway receives a unicast discovery request for a multicast enabled network interface via another discovery gateway interface, that CoAP Discovery Gateway interface MUST forward the discovery request in a multicast message for the "all nodes" multicast address.

The discovery client MUST ignore a reported Discovery Gateway interface if that interface is already in the list of known Discovery Gateway interfaces. This is to prevent loops.

A discovery client MAY perform hierarchical discovery by using the general /.well-known/core path. This combines the topology and server discovery phases. The downside is that a client may receive large amounts of data for each individual discovery message. This may be a problem for memory constrained nodes. By discovering the gateway topology first and using filtered server discovery, a client may achieve significant reductions in received data.

4.4. CoAP Server Discovery

CoAP Server Discovery builds on network information revealed during topology discovery. Each discovered subnet must be discovered individually. As an example, if a client connected to a backbone has discovered two LLNs behind two border routers, the client must perform CoAP Server discovery in the backbone (on-link subnet) as well as each of the two LLN interfaces.

5. Examples

5.1. Discovery across CoAP Discovery Gateways

Consider the following network environment: The client is located in LAN2. The discovery process has to traverse CoAP Discovery Gateway GW4 and LAN1 to locate CoAP Discovery Gateways GW1, GW2 and GW3. CoAP Discovery Gateways 1, 2 and 3 also have to be traversed. When no more new CoAP Discovery Gateways are discovered, discovery for CoAP Servers can be initiated.

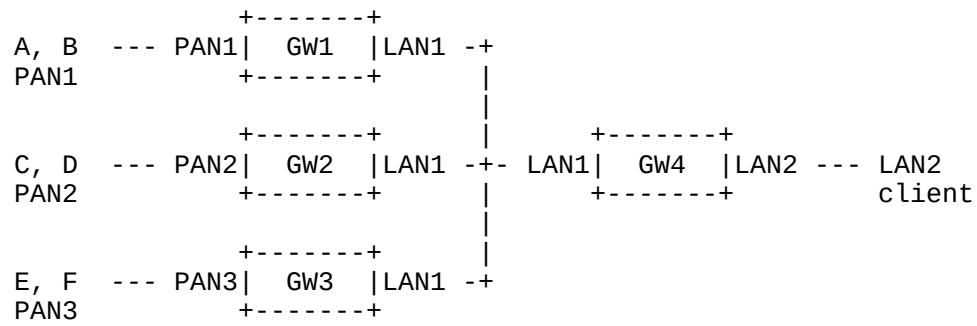


Figure 3: Discovery across CoAP Discovery Gateways

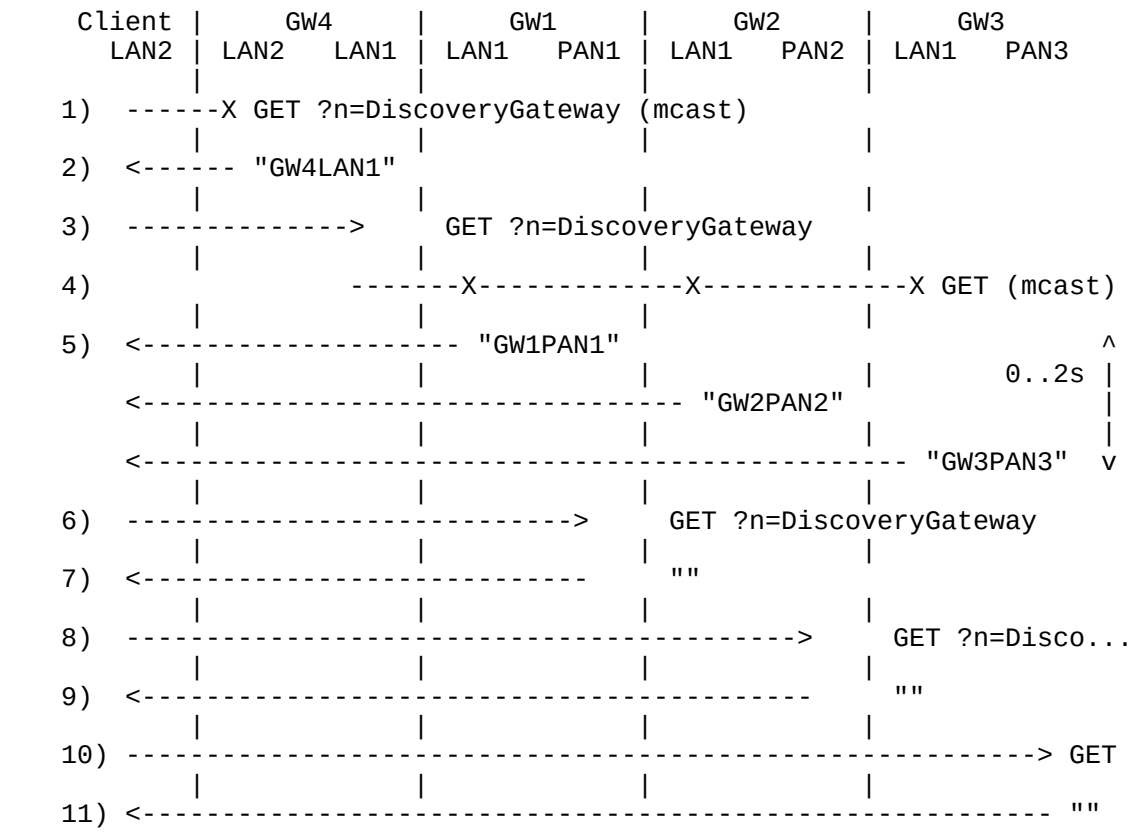


Figure 4: CoAP Discovery Process

1. The client sends out a multicast "GET /.well-known/core?n=DiscoveryGateway"

2. GW4 reports its other interfaces (GW4LAN1).

3. The client sends "GET /.well-known/core?n=DiscoveryGateway" to GW4LAN1

4. GW4LAN1 is not caching and received request in unicast => "GET /.well-known/core?n=DiscoveryGateway" is forwarded as multicast

5. Reports received from GW1LAN1, GW2LAN1 and GW3LAN1 within 2 seconds.

6. The client sends "GET /.well-known/core?n=DiscoveryGateway" to GW1PAN1

7. GW1PAN1 reports that no other gateways were found in PAN1

8. The client sends "GET /.well-known/core?n=DiscoveryGateway" to GW2PAN2

9. GW1PAN1 reports that no other gateways were found in PAN2

10. The client sends "GET /.well-known/core?n=DiscoveryGateway" to GW3PAN3

11. GW1PAN1 reports that no other gateways were found in PAN3

The client now has the following list of CoAP Discovery Gateway interfaces in unique subnets:

1. (mcast in on-link subnet)

2. GW4LAN1

3. GW1PAN1

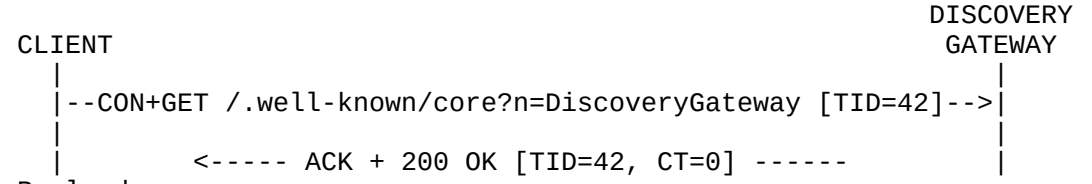
4. GW2PAN2

5. GW3PAN3

The client may now issue searches for other CoAP Servers by sending the request "GET /.well-known/core" to each CoAP Discovery Gateway in the list.

5.2. /topology path formats

Figure 5 shows an example of a client sending a request for /.well-known/core/topology?n=DiscoveryGateway. The discovery gateway sends back a response with the matching resources in the payload.



Payload:
<core://.../topology/device>;sh="/??";ct=0;n="DiscoveryGateway"

Figure 5: Looking for CoAP Discovery Gateways

Figure 6 shows an example of a client sending a request for /.well-known/core/topology/interfaces. The discovery gateway sends back a response with the actual interfaces provided by the CoAP Discovery

Gateway. A CoAP Discovery Gateway MUST implement the path /topology/interfaces.

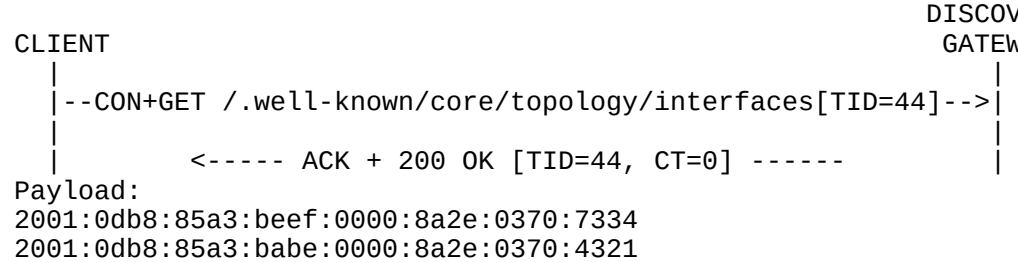


Figure 6: Using the "/topology/interfaces" path

Figure 7 shows an example of a client sending a request for /.well-known/core/topology/interfaces to a caching CoAP Discovery Gateway. The discovery gateway sends back a response with the actual interfaces provided by the CoAP Discovery Gateway.

Subsequently, the client may sending a request for /.well-known/core/topology/servers to get a list CoAP servers known by the Caching CoAP Discovery Gateway. This list includes sleeping and FLN nodes.

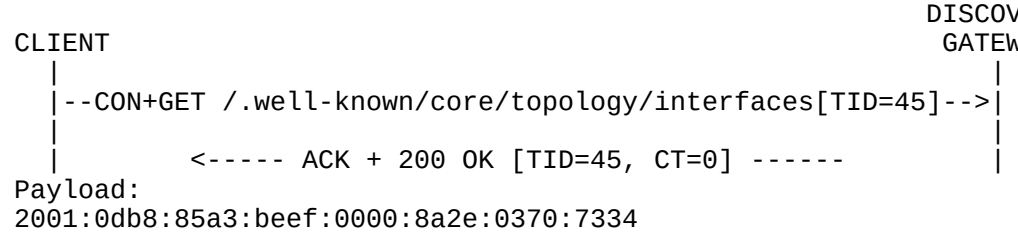


Figure 7: Receiving addresses from a caching CoAP Discovery Gateway

Figure 8 shows an example of a client sending a request for /.well-known/core/topology/servers to a caching CoAP Discovery Gateway. The discovery gateway sends back a response with the CoAP Servers known by the Caching CoAP Discovery Gateway.

In this case the Caching CoAP Discovery Gateway reports legacy devices which do not necessarily speak CoAP. The client may need to implement multiprotocol support in order to communicate to the devices.

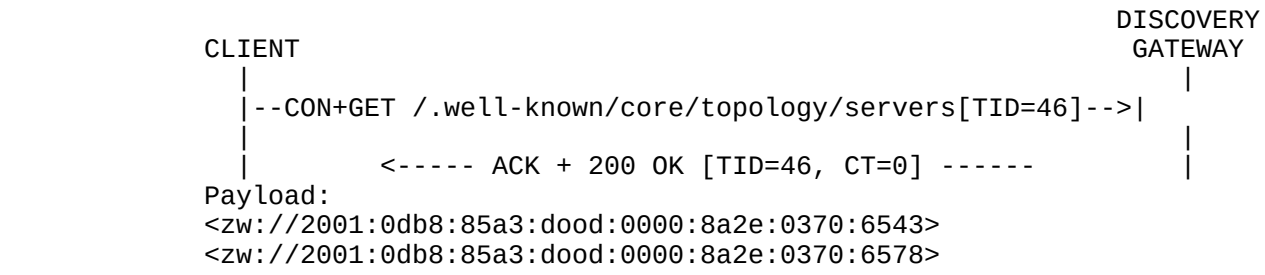


Figure 8: Advertising legacy devices via CoAP Discovery

A more advanced CoAP application gateway may provide translation between legacy protocols and CoAP. This is out of scope of this document.

6. IANA Considerations

This document has no actions for IANA.

7. Security Considerations

If a CoAP Discovery Gateway receives a generalized CoAP GET /.well-known/core message and that interface resides in a multicast enabled environment such as Ethernet, the CoAP Discovery Gateway forwards that CoAP request in an "all nodes" multicast message. In response, the CoAP Discovery Gateway potentially receives messages from a number of CoAP Discovery Gateways connected to that link. Forwarding all responses back to the requesting client in individual messages MAY be used for an amplification attack.

Coap discovery is not intended for Internet-wide operation. An internet access router SHOULD NOT forward CoAP messages to or from the Internet domain unless there is a specific application need for doing so. CoAP Discovery depends on a secure perimeter. So does many of the LLN nodes which this discovery mechanism is targeting.

Triggering an amplification attack requires that an attacker has access to the LAN or has control over LLN nodes. If the LLN implements link-layer security, an attacker cannot simply inject a wireless packet. If, however, one is within radio range of a LLN, a modified microwave oven may be a more efficient jamming tool than an amplification attack.

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.

[RFC4193] "Unique Local IPv6 Unicast Addresses", October 2005.

8.2. Informative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-02 (work in progress), December 2010.

[I-D.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-02 (work in progress), October 2010.

[I-D.ietf-6lowpan-nd]
Shelby, Z., "Neighbor Discovery Optimization for Low-power and Lossy Networks".

[RFC2080] Malkin, G. and R. Minnear, "RIPng for IPv6", RFC 2080, January 1997.

Appendix A. Open Issues

A.1. Large reports

The CoAP Block transfer mode MUST be implemented in order to support large reports, e.g. from a caching CoAP Discovery Gateway responding on behalf of 100’s of CoAP Servers in an LLN.

A.2. M2M Filtering

The document describes the use of ?n=DiscoveryGateway for detecting CoAP Discovery Gateways.

It should be considered if dedicated codes or keywords should be assigned. M2M applications will benefit from shorter codes and avoid the ambiguity of the free text allowed for ’?n=’.

A.3. Routable subnets

CoAP Discovery requires that all CoAP subnets are all reachable from a given subnet. Some application spaces, e.g. DIY home control, may be set up as off-the-shelf boxes with auto-assigned IPv6 subnets [RFC4193] with no route entries to other subnets.

RIPng [RFC2080] is considered sufficient for a home control application space. Larger installations for building control and the like are expected to be managed networks.

The following requirements could ensure successful plug-and-play behavior when combining Border Routers with CoAP Discovery Gateways from different vendors:

- A CoAP Discovery Gateway MUST support RIPng
- RIPng SHOULD be enabled in all CoAP Discovery Gateways
- A CoAP Discovery Gateway MAY implement other routing protocols

A.4. Compressing responses from caching CoAP Discovery Gateways

A caching CoAP Discovery Gateway SHOULD omit leading bytes of each reported address if the addresses are all in the same subnet served by the CoAP Discovery Gateway. If omitting leading bytes, a responding CoAP Discovery Gateway MUST provide the prefix information that was omitted from the reported addresses.

If no prefix is specified the interface identifiers MUST be full IP addresses or URIs.

A Caching CoAP Discovery Gateway MAY return more than one response message. If compression is used all addresses in a response message MUST belong to the same subnet prefix.

A.5. Integrated Battery support

A caching CoAP Discovery Gateway MAY be integrated into a LLN border router. This allows for tight integration of support services for sleeping nodes.

The Caching CoAP Discovery Server allows sleeping nodes to be discovered. The border router may implement mailbox delivery services for sleeping nodes. The border router may return "Destination Responding Slowly" ICMP messages to IP hosts sending to a sleeping node. The purpose of the ICMP message is to prevent IP applications from resending messages because they are not receiving

application acks.

A distributed routing protocol MAY distribute the mailbox services. This is out of scope of this specification.

Appendix B. Acknowledgements

Special thanks to Klaus Hartke, Peter Bigot, Peter van der Stok, Kerry Lynn and Zach Shelby for substantial contributions to the ideas and text in the document.

Author's Address

Anders Brandt
Sigma Designs
Emdrupvej 26A, 1.
Copenhagen O DK-2100
DENMARK

Phone: +4529609501
Email: abr@sdesigns.dk

CoRE Working Group A. Castellani
Internet-Draft M. Gheda
Intended status: Informational University of Padova
Expires: September 15, 2011 March 14, 2011

CoAP overhead: protocol analysis and reduction proposals
draft-castellani-core-coap-overhead-01.txt

Abstract

This draft aims at providing an analysis of the current overhead present in CoAP and at proposing alternative formats leading to an even more compact procotol.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 15, 2011.

Table of Contents

1. Introduction 3
2. Overhead analysis 3
2.1. Message ID and Token 3
2.1.1. Limit Token usage 4
2.2. Uri-* Options 4
2.3. Overhead in CoAP examples 4
3. Transport-related optimization 6
3.1. Action and Action Code 7
3.2. CON messages with ACK piggybacked (CONACK) 8
3.3. Multi-datagram transfers 9
3.3.1. Link-MTU Option 10
3.3.2. Example 11
4. URI optimization 11
5. Acknowledgements 12
6. Normative References 12
Authors' Addresses 12

1. Introduction

Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] design is focused on defining a REST protocol for constrained devices.

CoAP's strength is mainly to be a light binary REST protocol mappable to HTTP and suitable for M2M communication.

The design of such a protocol requires a lot of effort to effectively reduce the required overhead to a minimum, still leaving the resulting protocol simple enought for severely limited devices to handle it.

This document aims at highlighting possible aspects of the protocol design not fully optimized. To the best of the authors's knowledge, discussion about the outlined topics has not yet happened in the WG.

The document is organized as follows:
- Section 2 briefly analyzes the protocol overhead;
- Section 3 discusses alternative header formats and transport-related optimizations;
- Section 4 describes possible URI encodings to reduce overhead (TBD).

2. Overhead analysis

A synthetic description of the fields with higher impact on overhead and an example follow:
- Section 2.1 briefly discusses the Message ID field and Token Option;
- Section 2.2 outlines the overhead related Uri-* options (TBD);
- Finally, Section 2.3 analyzes the overhead allocation of a sample CoAP exchange.

2.1. Message ID and Token

In [I-D.ietf-core-coap] two different fields are available to perform session matching when concurrent sessions are present between the endpoints:
- Message ID: defined in Sec. 3.1 of [I-D.ietf-core-coap]
- Token: defined in Sec. 5.10.1 of [I-D.ietf-core-coap]

The Message ID field is also intended to handle datagram duplication.

The use of the aforementioned fields introduces a total overhead ranging between 2 and 11 bytes per message.

As a side note, dynamic memory allocation is usually avoided on contrained devices. The Token Option as currently defined requires implementors on statically allocating server devices to fully reserve 8 bytes per request for Token independently on the size chosen by the client.

When Token option is not present, session matching rule for non-empty messages is simple. It is sufficient to check that source and destination address and port.

Session matching rule is more complex when Token option can be used. If an endpoint supports the Token option, the following session matching rules MUST be added:

- also the Token MUST be checked for every non-empty message, because messages with a different Token belong to a different session;
- to understand to which session belongs an empty ACK or RST message (has no Token), the MID field in the received packet has to be used to search which session sent a CON message with that MID.

Session matching logic can be significantly simplified by adding the Token option to empty messages.

2.1.1. Limit Token usage

The Token option as defined in CoAP is needed to successfully match requests to deferred responses when a client has opened more than one concurrent session to the same server using the same UDP source port.

A client can always choose to open concurrent sessions to the same server using different UDP source ports. Clients SHOULD avoid sending Token option using dynamic source port assignment.

Constrained server implementation MAY not support Token option, in order to limit memory consumption and session matching complexity. Clients accessing these servers, in order to perform concurrent requests to them, are required to send concurrent requests using different source ports for each concurrent request to them.

2.2. Uri-* Options

TBD

2.3. Overhead in CoAP examples

An overhead analysis of the example present in Section 2.2 of [I-D.ietf-core-coap] and for convenience reported hereafter in Figure 1 follows.

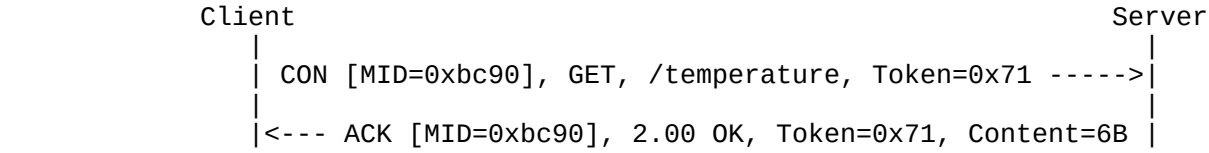


Figure 1: CoAP example

Table 1 and Table 2 analyze how the request/response bytes are allocated to the different fields. Req1/Res1 refer to the exact example shown in Figure 1. Req2/Res2 are referred to the same message exchange however with smaller Uri "/t" and 2B Content; the size of Uri/Content are under the control of the system developers and are thus expected to be reduced when needed.

Moreover as pointed out in Section 2.1.1, clients can always avoid using Token option by using different source UDP ports for concurrent requests.

Field(s)	Req1 (Ratio)	Req2 (Ratio)	Req3 (Ratio)
Base	2B (0.11)	2B (0.25)	2B (0.33)
MID and Token	4B (0.22)	4B (0.50)	2B (0.33)
URI	12B (0.67)	2B (0.25)	2B (0.33)
Total	18B (1.00)	8B (1.00)	6B (1.00)

Table 1: Request size analysis

Field(s)	Res1 (Ratio)	Res2 (Ratio)	Res3 (Ratio)
Base	2B (0.17)	2B (0.25)	2B (0.33)
MID and Token	4B (0.33)	4B (0.50)	2B (0.33)
Content	6B (0.50)	2B (0.25)	2B (0.33)
Total	12B (1.00)	8B (1.00)	6B (1.00)

Table 2: Response size analysis

In Req1/Res1 results, it can be seen that Uri/Content have high impact on the total message size, however, when optimizing their size by using a shorter Uri and a more efficient content representation, their impact can be substantially reduced by system developers using CoAP (Res2/Req2).

If CoAP implementation avoids using Token option, the resulting CoAP packets are even more compact (Req3/Res3).

3. Transport-related optimization

The following section discusses a proposal to reduce even more the impact of overhead, and adding the following features to CoAP "message"-layer:

- multi-datagram message support;
- partial reordering of unconfirmed messages.

Figure 2 shows a proposal of modification of the current CoAP header format.

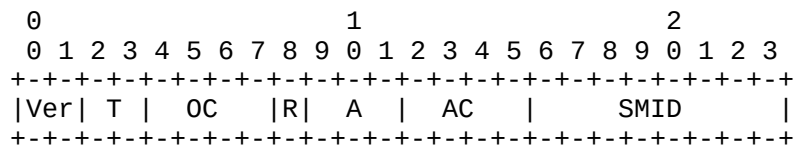


Figure 2: Header format w/ sequence number

The definition of the fields already introduced in Figure 6 of [I-D.ietf-core-coap] is still valid.

New fields introduced in Figure 2 are defined as follows:

Reserved field (R): 1-bit unsigned integer. This field is reserved for future use and MUST always be set to 0. If R field is set to 1, a RST message SHOULD be sent and the message MUST be dropped.

Action (A): 3-bit unsigned integer. Indicates whether the message is empty (0) or is carrying a response (1-6) or a request (7). More details about the value of this field can be found in Table 3

Action Code (AC): 4-bit unsigned integer. Indicates more details about the kind of request (method), response (specific status) carried in the message. If the Action is an empty message, AC field usage is reserved for future use.

Sequential Message ID (SMID): 8-bit unsigned integer. Used to detect duplication and to match Acknowledgment/Reset to a previous message of type Confirmable.

SMID is a 8-bit unsigned integer that MUST be initialized on the first message of a session. SMID is a global parameter of the

session shared between the endpoints. Each endpoint emitting a non-empty message MUST echo SMID increased by 1, otherwise for empty messages SMID value is echoed as is.

When Token option is used multiple different sessions MAY exist with the same destination address and port. To successfully handle SMID across multiple sessions, the Token option MUST be echoed back also in every empty message.

3.1. Action and Action Code

The Action field contains synthetic information about message semantics. By leveraging the information contained in the Action field (see Table 3), CoAP implementors can scalably choose the level of support to implement in each Action class.

Example: On extremely constrained implementations, the Action Code can be read only for requests or for 2xx, 4xx, 5xx responses. For other actions the effects of the message do not depend upon the specific code received.

Value	Action
0	Empty message
1	1xx Informational response
2	2xx Successful response
3	3xx Redirection response
4	4xx Client Error response
5	5xx Server Error response
6	CoAP-specific response
7	Request

Table 3: CoAP Action values

When the action is a request, the code field will contain the details about the method used to access the resource. The actual definition of the Action Code values to GET, POST, PUT and DELETE methods is out-of-scope of this document, however it can be easily obtained if the described compressed header is believed to be suitable to the CoAP protocol specification.

As before, the actual mapping between Action Code and specific response codes is out-of-scope of this document, but an option for a simple assignment could be to use "direct YY mapping", that is deriving the value directly from YY of every response code (X.YY) as defined in Table 6 of [I-D.ietf-core-coap].

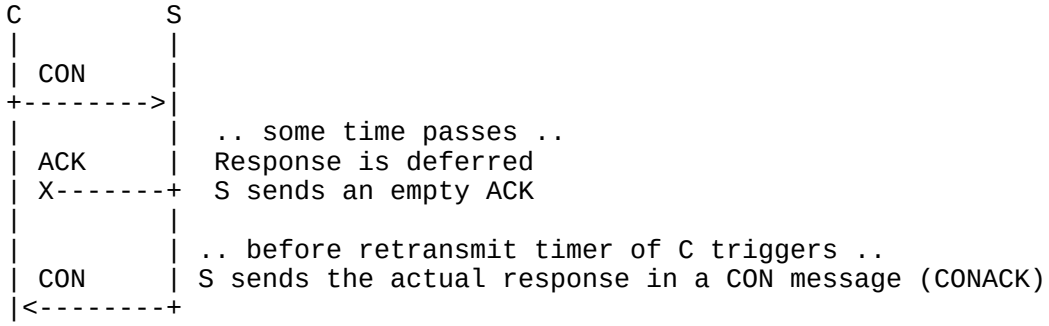
"Direct YY mapping" example: 5.02 Bad Gateway can be mapped to A=5 and AC=2.

The response codes currently defined in CoAP are all mappable using "direct YY mapping".

Even looking in [RFC2616] or in the more recent [I-D.ietf-httpbis-p2-semantics], all HTTP status codes except two 4xx status codes (416, 417) are mappable using "direct YY mapping". Moreover 416 and 417 status codes are intended for responses to complex request statements, currently out-of-scope from the CoRE charter and otherwise manageable using different AC mappings.

3.2. CON messages with ACK piggybacked (CONACK)

Assume the following situation happens:



In order to simplify client logic and to reduce overhead, the most simple action C can do is to stop the retransmit timer and assume that this CON message piggybacks the lost acknowledge.

Otherwise the client should keep the retransmit timer active, and when the timer triggers, the initial request message (CON) has to be retransmitted. The server itself MUST keep track of both active CON messages, to be able to retransmit an empty ACK when received the same request again and to be able to retransmit the response if no ACK is received.

A simple solution to this problem is introducing a new logical message type (CONACK): a CON message sent after receiving CON message. A received CONACK message piggybacks the acknowledge to a previously sent CON message in the same session. Before accepting this CON message, datagram duplication MUST be checked as usual.

A consequence of using CONACKs is an added benefit to the protocol itself, because a server can always decide to emit a CONACK after a CON request, if it wants to receive an acknowledge for any sent response.

3.3. Multi-datagram transfers

Limiting the CoAP message size to the datagram MTU can lead to some issues and limit its range of application, as discussed in [I-D.ietf-core-block].

[I-D.ietf-core-block] defines how to transfer large resource representations, by introducing the Block Option and defining a request format to get a specific part of a large resource. The Block option allows to transfer response content fragmented in parts of agreed size. Using this option, the client is enabled to request a specific part of the resource representation, and the resulting exchange is similar to subsequent HTTP Range requests (see Sec. 14.35 in [RFC2616]).

Using this approach when dealing with large resources with non-static representations lead to an increased complexity in the upper layers, due to the necessary controls and buffering required to avoid transferring multiple parts belonging to different representations. A more traditional ordered data transfer technique handles more easily this situation, and can be built inside the CoAP "messages"-layer depicted in Figure 1 of [I-D.ietf-core-coap].

In the following part of this section, an alternative method for transferring segmented messages is described.

Endpoints willing to support multi-datagram transfers MAY support the following variant. The reserved field R shown in Figure 2 can be redefined as follows:

More datagrams pending (M): 1-bit boolean value. Indicates that the current message is formed by other datagrams too, the endpoint receiving this message MUST wait to receive all the datagrams before handling it.

If the message does not fit into a single datagram, the M bit MUST be set. After receiving an ACK message confirming the reception of each data segment, the subsequent part fitting in a datagram should be sent. If the sender times out waiting for an ACK, it retransmits the same data segment.

Multi-datagram messages SHOULD be sent only using CON messages in order to perform congestion control of any kind.

The SMID field present in the variant shown in Figure 2 is suitable for multi-datagram tranfers too. SMID field MUST be increased at every subsequent data segment sent, ACK messages MUST have SMID value matching the SMID of the acknowledged message. The presence of this field is useful to avoid confusing a new ACK with a network duplicated one.

When a message is received containing the field M set to 0, the last content segment is attached to the message, which is then passed to the CoAP "request/response"-layer and handled.

Thanks to the half-duplex nature of a REST conversation, multi-datagram requests and responses can be tranferred using the same method: after sending all the request datagrams, the server handles the request and can send all the datagrams forming the response using this technique. The same SMID value is initially chosen and increased by the client to send the request, and is then increased by the server when sending back the response.

Action, Action Code and Options related to the message SHOULD be sent only in the first datagram. To reduce overhead, the Options SHOULD be removed from subsequent datagrams; to avoid confusion and redundancy, Action and Action Code fields MAY be set to zero, the presence of content in the message allows to distinguish subsequent segments from an empty message.

3.3.1. Link-MTU Option

As noted in [I-D.ietf-core-block], the fragmentation/reassembly process loads the lower layers. To avoid this problem multi-datagram transfer SHOULD transfer data segments fitting in a single link layer frame.

Type	C/E	Name	Data type	Length	Default
TBD	E	Link-MTU	uint	1-2 B	TBD

To address situations where the sending endpoint does not know the frame limits of the receiver, its size can be proactively shared attaching a Link-MTU option to requests expecting big responses. To react even more promptly, this option can be piggybacked over an empty ACK aknowledging a fragmented message.

3.3.2. Example

Figure 3 shows an interaction between a CoAP Client and Server, where both are transferring content using multi-datagram transfers.

Right after the client has completed the multi-datagram request, the server starts replying with a multi-datagram response.

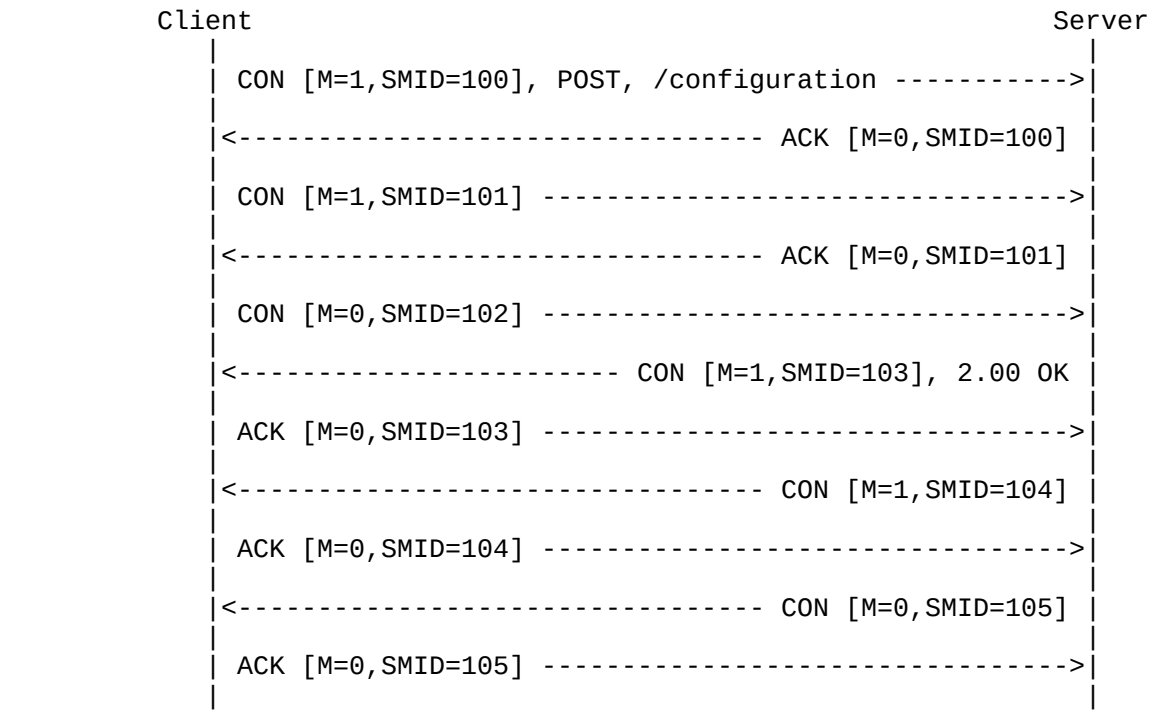


Figure 3: Multi-datagram messaging example

Figure 3 also shows an application of CONACK. The CON message sent by the server with SMID=103 (CONACK) piggybacks an acknowledge to the CON message sent by the client with SMID=102.

4. URI optimization

Further discussion about URI encoding using two optional compressed URI options will follow in a planned revision of the document.

Refer to the following message to get an idea of the encodings that

will be discussed:

http://www.ietf.org/mail-archive/web/core/current/msg00315.html

Discussion about the proposed encodings was partly done on the ML, however a more complete proposal may be useful to better evaluate the subject.

5. Acknowledgements

Special thanks to Nicola Bui and Michele Zorzi for their support and contributions.

6. Normative 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.

[I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[I-D.ietf-httpbis-p2-semantics] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., Berners-Lee, T., and J. Reschke, "HTTP/1.1, part 2: Message Semantics", draft-ietf-httpbis-p2-semantics-12 (work in progress), October 2010.

[I-D.ietf-core-block] Shelby, Z. and C. Bormann, "Blockwise transfers in CoAP", draft-ietf-core-block-01 (work in progress), January 2011.

Authors' Addresses

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

Email: angelo@castellani.net

Mattia Gheda
University of Padova
Via Gradenigo 6/B
Padova 35131
Italy

Email: ghedamat@dei.unipd.it

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 15, 2011

A. Castellani
University of Padova
S. Loreto
Ericsson
March 14, 2011

Best Practice to map HTTP to COAP and viceversa
draft-castellani-core-http-coap-mapping-01.txt

Abstract

This draft aims at being a simple guide to the use of CoAP REST interface, to show how it can be mapped to and from HTTP, and at being a base reference documentation for CoAP/HTTP proxy implementors.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 15, 2011.

Table of Contents

1. Introduction 3
2. HTTP-CoAP 3
2.1. URI 4
2.2. Proxy 4
2.2.1. HC proxy discovery using DNS-SD 6
2.3. Mapping 7
2.4. Multiplexing CoAP responses 10
2.4.1. Establishing a CoAP subscription 12
3. CoAP-HTTP 14
4. Security Considerations 14
5. IANA Considerations 14
6. Acknowledgements 14
7. References 15
7.1. Normative References 15
7.2. Informative References 15
Authors' Addresses 16

1. Introduction

Since implementing on constrained devices the full HyperText Transfer Protocol (HTTP) [RFC2616] is believed to be operationally and computationally too complex, especially in an M2M communication environment, resources available on constrained nodes are expected to be served using CoAP [I-D.ietf-core-coap].

> "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." Section 2 [I-D.ietf-core-coap]

These days the information is increasingly converging on the Web, thus an easy CoAP interoperability with HTTP is a paramount feature for CoAP. Indeed leveraging on both the easy CoAP/HTTP translation and the common usage of URI(s) to identify resources, it will become extremely simple to integrate constrained nodes in the Web.

The internetworking described in this document between CoAP and HTTP is mainly based on three points:

- the URI does not change between CoAP and HTTP, the scheme identifies the protocol;
- HTTP/CoAP mapping is performed by a proxy, both HTTP/CoAP endpoints can be not aware that a mapping is happening;
- using a named URI authority and DNS can be useful for the mapping.

The proxy itself does not require any particular knowledge about the constrained network topology, devices contained, nor about the content of data exchanged.

2. HTTP-CoAP

HTTP-CoAP mapping spans across several protocol layers:
- HTTP is mapped to CoAP
- TCP is used on the HTTP side, while CoAP uses UDP transport

In addition to this 6LoWPAN adaptation layer addresses a similar networking scenario, thus a convertion between IPv4/IPv6 to 6LoWPAN MAY be present as well.

2.1. URI

Any resource available in CoAP can be accessed using HTTP at the same URI, except for the scheme. The scheme represents the protocol used by the endpoint to access the resource.

The CoAP resource "//node.coap.something.net/foo" can be accessed using CoAP at the URI "coap://node.coap.something.net/foo", and using HTTP at the URI "http://node.coap.something.net/foo". When the resource is accessed using HTTP, the mapping from HTTP to CoAP is performed by a proxy

The usage of the same URI to access a resource, independently if it is accessed by a CoAP client within the same constrained network or by a HTTP client outside the constrained network, reduces the complexity of a proxy performing the mapping.

OPEN ISSUE: discuss the DNS usage resolving the URI.

2.2. Proxy

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

Usually regular HTTP proxies are same-protocol proxies, because can map from HTTP to HTTP. CoAP same-protocol proxies are intermediaries for CoAP to CoAP exchanges, however the discussion about that entities is out-of-scope of this document.

At least two different kinds of HC proxies may exist:

- One-way cross-protocol proxy (1-way proxy): It can translate from a client of a protocol to a server of another protocol but not viceversa.
- Two-way (or bidirectional) cross-protocol proxy (2-way proxy): It can translate from a client of both protocols to a server of the other protocol.

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

Forward proxy (F): It 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): It is 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 be known in advance by the proxy (ZCC: zero client configuration).

Transparent (or Intercepting) proxy (I): This proxy can intercept any origin protocol request (HTTP or CoAP) and map it 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 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.

In the most common scenario the HC proxy is expected to be server-side and deployed at the edge of the constrained network. The arguments supporting this assumption are the following:

TCP/UDP: Translation between HTTP and CoAP requires also a TCP to UDP mapping; UDP performance over the Internet may not be adequate, UDP should be dropped as soon as possible to minimize the number of required retransmissions and overall reliability.

Multicast: To enable access to local-multicast in the constrained network, the HC proxy may require a network interface directly attached to the constrained network.

Caching: Efficient caching requires that all the CoAP traffic is intercepted by the same proxy, network edge is a strategical placement for this need.

Security: HTTPS sessions should be terminated as near as possible to the CoAP server.

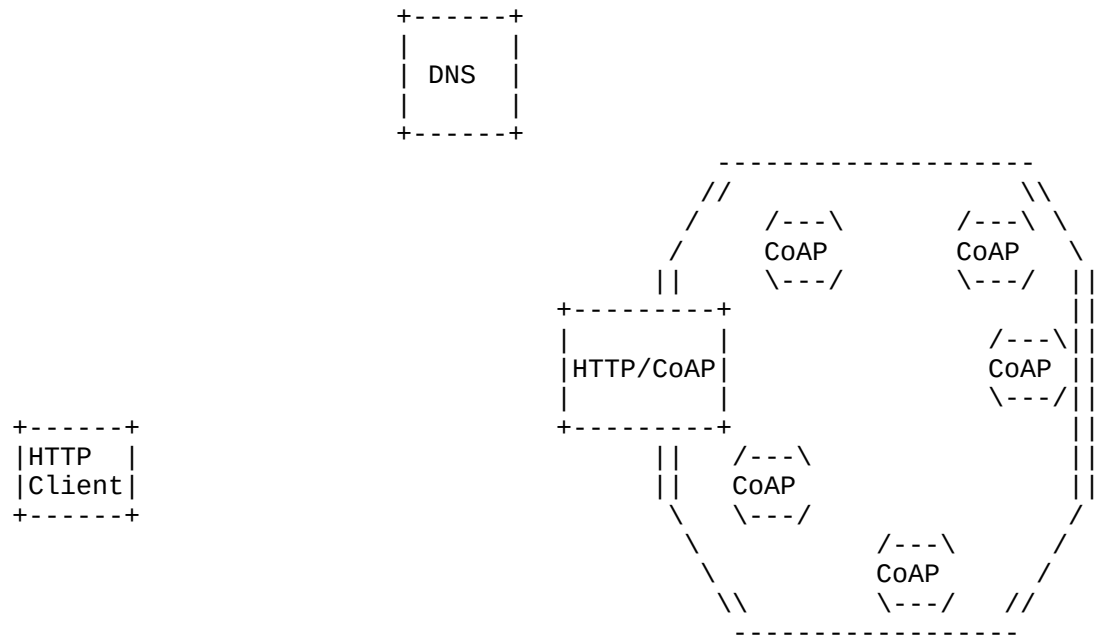


Table 1 shows some interesting HC proxy scenarios, and quickly marks the advantages related to each scenario.

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

Table 1: Interesting HC proxy deployments

The following open questions are left open in Table 1:
1. Are CoAP security modes adequate for Internet-wide operation?
2. Are reverse proxy setups scalable?

2.2.1. HC proxy discovery using DNS-SD

DNS-SD can be used by an HTTP client to discover the HC proxy in authority for a specific domain [I-D.jennings-http-srv].

An HTTP client wants access a resource that it knows being identified by the following URI:

//node.coap.something.net/foo

To find the address of the HC proxy, the HTTP client will look up the following SRV record:

_http._tcp.node.coap.something.net

The DNS will contain the following record:

```
      _http._tcp.node.coap.something.net  IN  SRV     0  1  80  hc-proxy.somet
hing.net
        hc-proxy.something.net  IN  A  192.168.0.1  ;  the address of the HC pr
oxy
```

The client will pass the request to the HC proxy that will translate it in a CoAP request. The CoAP side of the proxy will lookup the DNS in order to find the actual constrained device in authority for that URI.

2.3. Mapping

CoAP offers a subset of HTTP features in terms of methods, statuses and options supported; thus some HTTP request MAY NOT be mappable to CoAP.

In particular CoAP lacks the following methods defined in HTTP: OPTIONS, HEAD, TRACE and CONNECT.

An HC proxy receiving an HTTP request with a method not supported in CoAP MUST immediately drop handling the request and MUST send a response with status "405 Method Not Allowed" to the HTTP client.

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].

The mapping of conditional HTTP requests is defined in Section 8.2 of [I-D.ietf-core-coap].

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

If an empty CoAP ACK is received, the actual CoAP response is

deferred. As described in CoAP specification the ACK is transparent to the HTTP client.

No upper bound is defined for a server to provide that response, thus for long delays the HTTP client or any other proxy in between MAY timeout, further considerations are available in Section 7.1.4 of [I-D.ietf-httpbis-p1-messaging].

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.

Safe or non-idempotent requests MAY timeout. How the HC proxy should handle this situation?

The HC proxy MUST define an internal timeout for each CoAP request pending, because the CoAP server MAY silently die before completing the request. This timeout SHOULD be as high as possible.

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

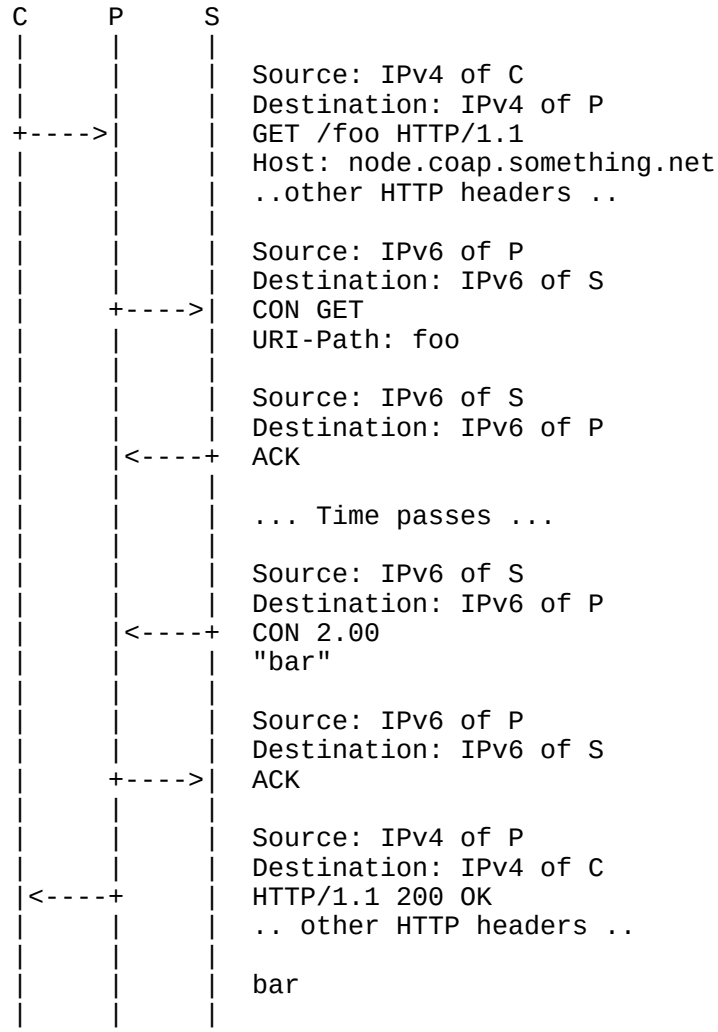


Figure 2: HTTP/IPv4 to CoAP/IPv6 mapping

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. Thus IPv6 connectivity is not required at the HTTP client when accessing a CoAP server over IPv6 only, which is a typically expected use case.

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

When the HTTP client has native IPv6 support, a convenient deployment choice should be to use an HC intecepting proxy. Thus the proxy MUST be located in the IPv6 network path between the client and the server, thus near to the server itself in order to support any Internet client.

2.4. Multiplexing CoAP responses

Defining the mapping of some advanced CoAP features to HTTP (i.e. multicast, observe) must address the need to asynchronously deliver multiple responses to the same HTTP request.

Some HTTP features are useful to succesfully represent these particular sessions.

Using Multipart media type is a suitable solution to deliver multiple CoAP responses within a single HTTP response.

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.

An HC proxy may prefer to transfer each CoAP response immediately after its reception. Responses can be immediately transferred in "chunks" of an HTTP chunked Transfer-Encoding session, without knowing in advance the total number of responses and with arbitrary delay between them.

A detailed discussion on the use of chunked Transfer-Encoding to stream data over HTTP can be found in [I-D.loreto-http-bidirectional]. 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].

When responses are coming from different sources, i.e. multicast, details about the actual source of each CoAP response SHOULD be provided. Source information can be represented in HTTP using a Link option described in [RFC5988] using "via" relation type.

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). Discussion related to group communication in CoAP can be found in [I-D.rahman-core-groupcomm].

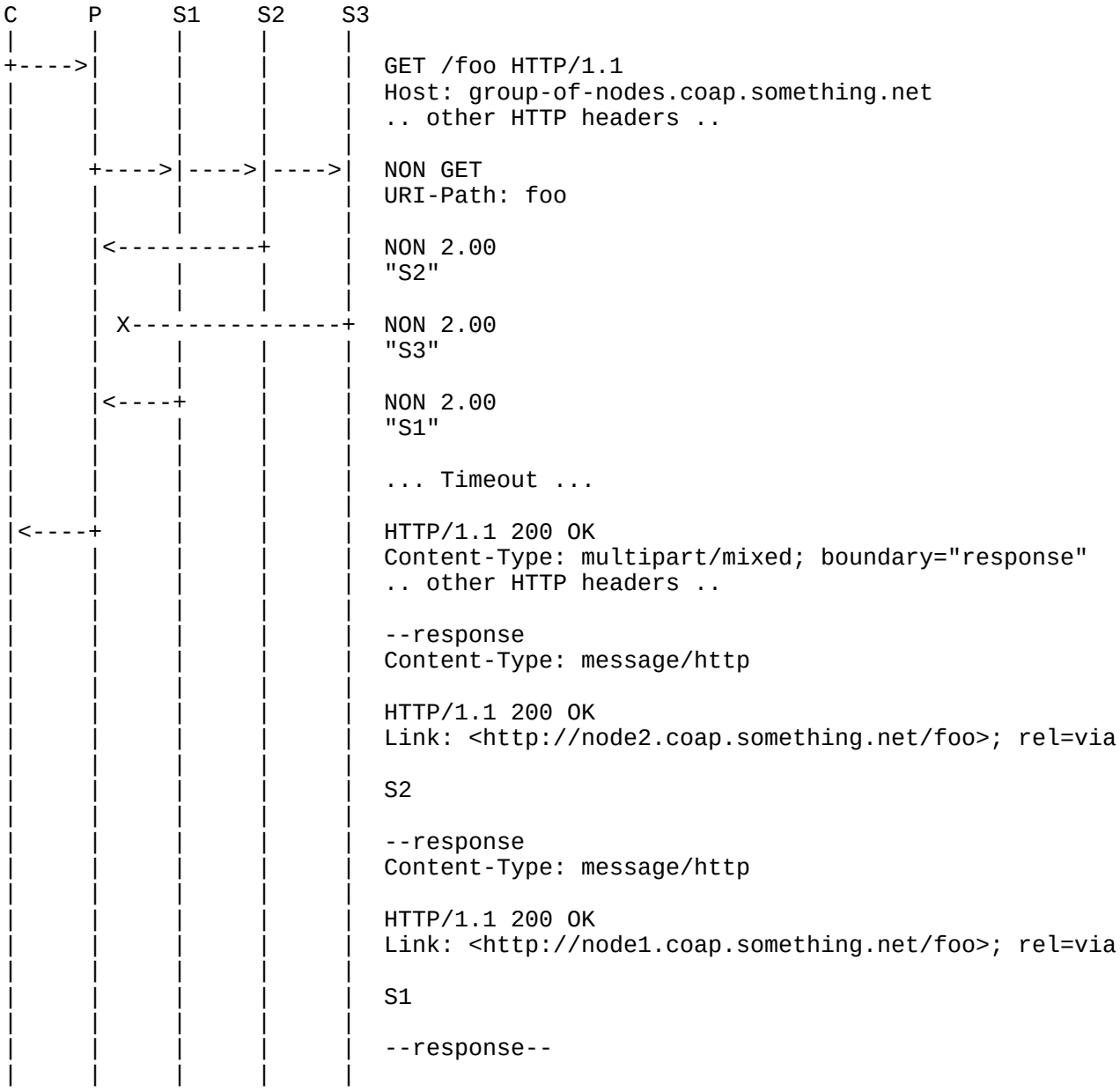


Figure 3: Unicast HTTP to multicast CoAP mapping

The mapping proposed in the above diagram does not make any assumption in how multicasting is done on the constrained network.

If IPv6 multicast support is present in the constrained network, an AAAA record containing the IPv6 multicast group will start multicast operation at the proxy. Otherwise the authority part of the URI is used by the HC proxy to match with a locally defined group of nodes.

In order to minimize the delay in delivering the responses (e.g. HTTP client can incrementally process the responses, HC proxy can reduce internal buffering), each CoAP response can be immediately streamed using HTTP chunked Transfer-Encoding. This encoding was not shown in order to simplify Figure 3, an example showing immediate delivery of CoAP responses is provided in Figure 4 (observe session).

2.4.1. Establishing a CoAP subscription

Using an exchange similar to the one shown in Figure 3, a CoAP observe session can be directly established by a willing HTTP client. Observe mechanism is specified in [I-D.ietf-core-observe].

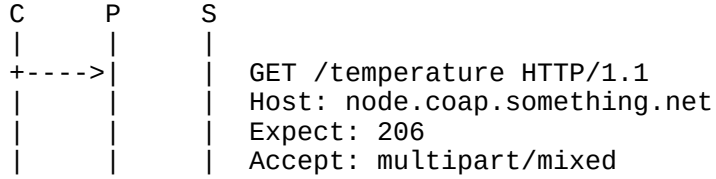
An HTTP client willing to establish a subscription to the "/temperature" resource of a CoAP server SHOULD send an HTTP request with Expect header set to "206" and Accept header set to "multipart/mixed".

The Lifetime of the subscription itself SHALL be sent defining the subscription interval using "Date:" header as starting time and "If-Modified-Since:" as ending time. The HC proxy can compute Lifetime option by using that HTTP headers.

Due to the asynchronous nature of this exchange, the HC proxy willing to accept establishing a subscription SHOULD send an HTTP response with status "206 Partial Content", Content-Type "multipart/mixed" and Transfer-Encoding "chunked".

Each CoAP response will be delivered in a different HTTP chunk until the subscription lifetime expires, when the subscription has expired the HTTP session MUST be closed.

If the HC proxy does not support this exchange or is not willing to establish this session, it SHOULD fail with status "417 Expectation failed".



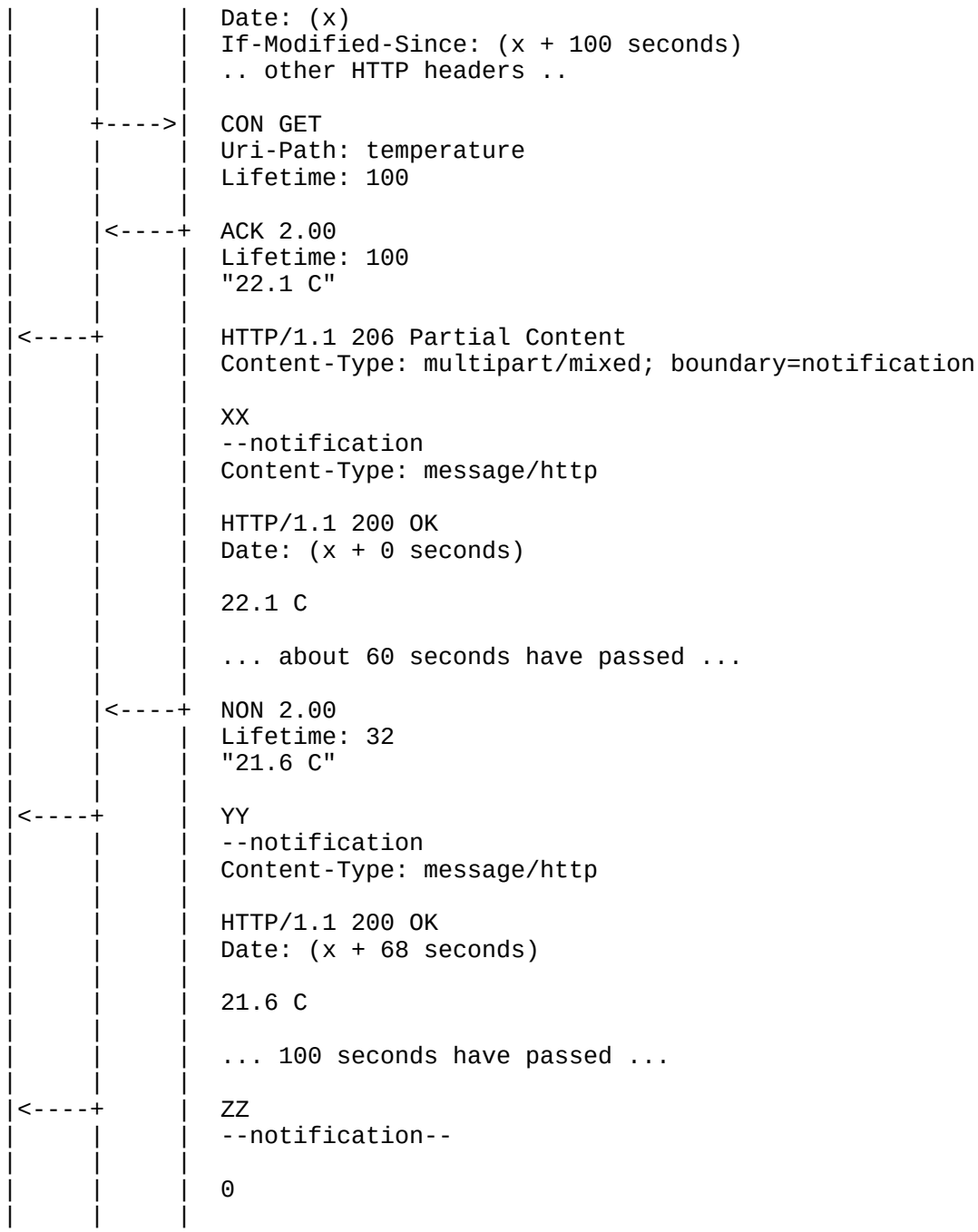

Date: (x)
If-Modified-Since: (x + 100 seconds)
.. other HTTP headers ..
CON GET
Uri-Path: temperature
Lifetime: 100
ACK 2.00
Lifetime: 100
"22.1 C"
HTTP/1.1 206 Partial Content
Content-Type: multipart/mixed; boundary=notification
XX
--notification
Content-Type: message/http
HTTP/1.1 200 OK
Date: (x + 0 seconds)
22.1 C
... about 60 seconds have passed ...
NON 2.00
Lifetime: 32
"21.6 C"
YY
--notification
Content-Type: message/http
HTTP/1.1 200 OK
Date: (x + 68 seconds)
21.6 C
... 100 seconds have passed ...
ZZ
--notification--
0

Figure 4: HTTP subscription to a CoAP resource

When an HTTP client performs direct subscriptions to CoAP servers using this method, the HC proxy has to keep for a possibly long time state information about the observe session and an open HTTP/TCP session to the client.

Soft state required by the various involved protocols (HTTP/TCP, CoAP/UDP) leads to scalability issues when an high number of direct subscriptions are established using the same HC proxy.

Moreover the HC proxy has an active role in the subscription process, thus if crashed or rebooted the subscription to the CoAP node will be lost.

HTTP clients in the real world usually implement notification mechanisms over HTTP using a technique called "Long Polling", an extensive description of this technique is available in Section 2 of [I-D.loreto-http-bidirectional]. A mapping using a "Long Polling" may be identified and can be preferred for longer sessions of observe.

3. CoAP-HTTP

TBD

4. Security Considerations

TBD

5. IANA Considerations

This document does not require any actions by the IANA.

6. Acknowledgements

Special thanks to Nicola Bui and Michele Zorzi for their support and for the various contributions to this document.

Thanks to Kerry Lynn, Akbar Rahman, Peter van der Stok and Anders Brandt for the extensive fruitful discussion about URI mapping, DNS and multicast group communication useful for writing various sections of this document.

Thanks to Brian Frank for its support and its feedback about the content.

7. References

7.1. Normative 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.

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

[I-D.ietf-httpbis-p1-messaging] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., Berners-Lee, T., and J. Reschke, "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-12 (work in progress), October 2010.

[I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[I-D.ietf-core-observe] Hartke, K. and Z. Shelby, "Observing Resources in CoAP", draft-ietf-core-observe-01 (work in progress), February 2011.

7.2. Informative References

[I-D.loreto-http-bidirectional] 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", draft-loreto-http-bidirectional-07 (work in progress), January 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.

[I-D.jennings-http-srv] Jennings, C., "DNS SRV Records for HTTP",

draft-jennings-http-srv-05 (work in progress), March 2009.

[I-D.rahman-core-groupcomm]
Rahman, A., "Group Communication for CoAP",
draft-rahman-core-groupcomm-04 (work in progress),
March 2011.

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

CoRE
Internet-Draft
Intended status: Informational
Expires: September 15, 2011

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

March 14, 2011

Security Considerations in the IP-based Internet of Things
draft-garcia-core-security-01

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on September 15, 2011.

Copyright Notice

Table of Contents

1. Conventions and Terminology Used in this Document 3
2. Introduction 3
3. The Thing Lifecycle and Architectural Considerations 4
3.1. Security Aspects 5
4. State of the Art 8
4.1. IP-based Security Solutions 8
4.2. Wireless Sensor Network Security and Beyond 10
5. Challenges for a Secure Internet of Things 11
5.1. Constraints and Heterogeneous Communication 11
5.1.1. Tight Resource Constraints 11
5.1.2. Denial-of-Service Resistance 13
5.1.3. Protocol Translation and End-to-End Security 13
5.2. Bootstrapping of a Security Domain 15
5.2.1. Distributed vs. Centralized Architecture and Operation 15
5.2.2. Bootstrapping a thing's identity and keying materials 16
5.2.3. Privacy-aware Identification 17
5.3. Operation 18
5.3.1. End-to-End Security 18
5.3.2. Group Membership and Security 19
5.3.3. Mobility and IP Network Dynamics 19
6. Next Steps towards a Flexible and Secure Internet of Things . 20
7. Security Considerations 22
8. IANA Considerations 22
9. Acknowledgements 23
10. Normative References 23
Authors' Addresses 26

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 summarizes concrete steps towards a secure Internet of Things. Section 7 includes final remarks and conclusions.

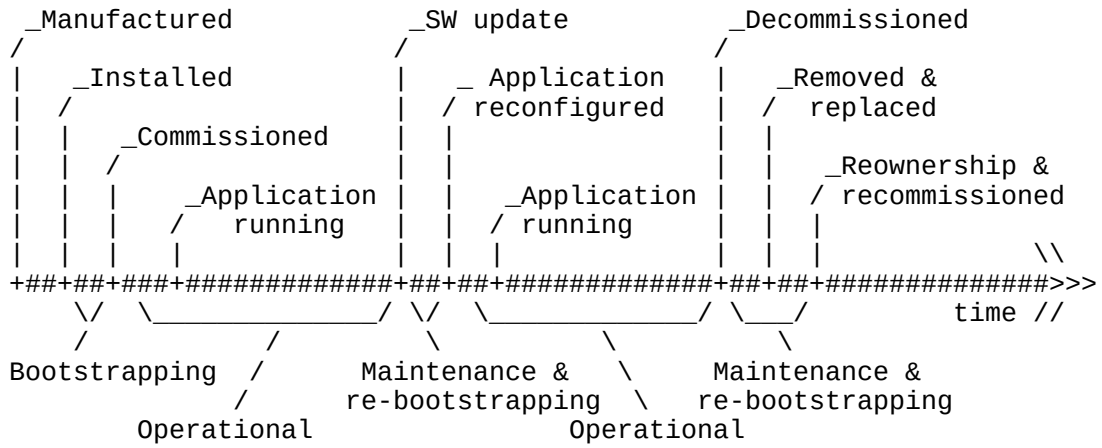
3. The Thing Lifecycle and Architectural Considerations

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. 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:

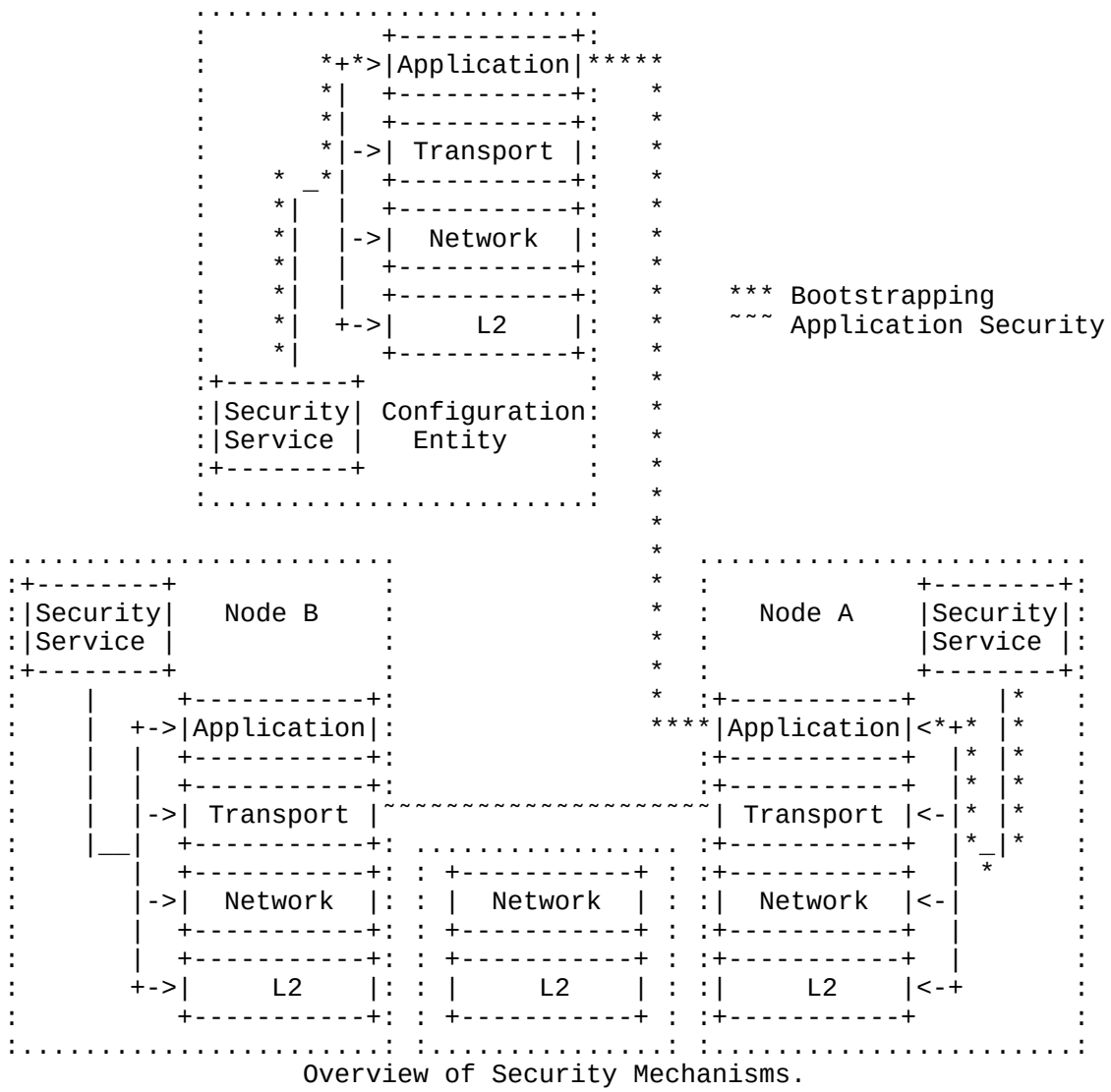
1 The security architecture refers to the system elements involved in the management of the security relationships between things and the way these security interactions are handled (e.g., centralized or distributed) during the lifecycle of a thing.

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.



Overview of Security Mechanisms.

Figure 2

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.

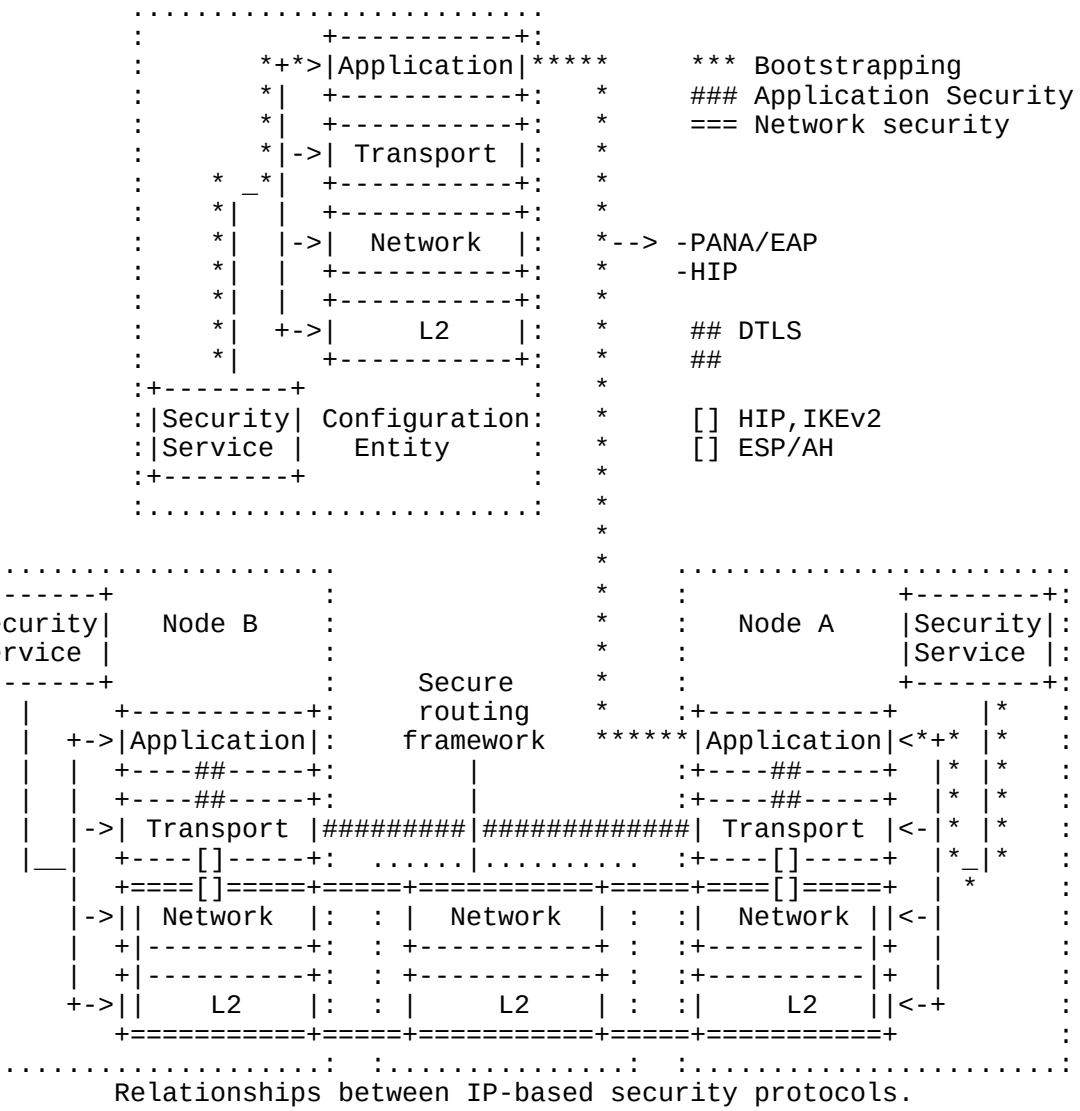
4. State of the Art

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.



Relationships between IP-based security protocols.

Figure 3

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

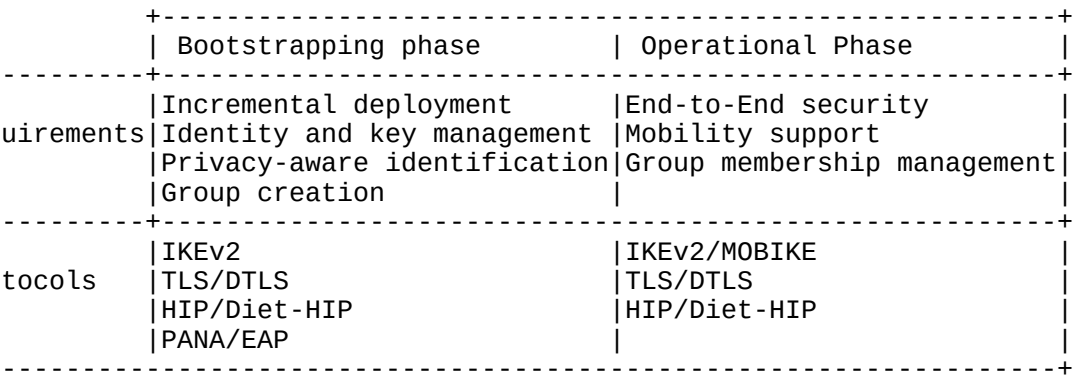
In this section, we take a closer look at the various security challenges in the operational and technical features of the IoT and then discuss how existing Internet security protocols cope with these technical and conceptual challenges through the lifecycle of a thing. Table 1 summarizes which requirements 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.



Relationships between IP-based security protocols.

Figure 4

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. This involves le

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 or a combination. 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-O'Flynn] 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-O'Flynn]. 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, and may force installers to conduct site surveys that include measurement of communication range and signal strength prior to deciding on device placement and conducting the installation itself.

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 e xists. 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][RFC3830]. 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. Next Steps towards a Flexible and Secure Internet of Things

As evident from the discussions of the lifecycle of a thing and the IP security challenges in the Internet of Things, 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 scenario 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 subnetwork may occur over time (if only to facilitate smooth device repair/replacement without the need for a hard "system reboot" or to realize ownership transfer). 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.

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.

7. Security Considerations

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

8. IANA Considerations

This document contains no request to IANA.

9. Acknowledgements

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

10. Normative 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.

[ID-CoAP] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-05 (work in progress), 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), Feb 2011.

[ID-HIP] Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-04 (work in progress), Jan 2011.

[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 (HIPv2)", draft-ietf-hip-rfc5201-bis-05 (work in progress), Mar 2011.

[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-O'Flynn] 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, Mar 2009.

[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.

[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.

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, Aug 2004.

[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, Dec 2005.

[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.

[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.

[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, Apr 2008.

[RFC5238] Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol version 1.2", RFC 5246, Aug 2008.

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

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

[WG-6LoWPAN] "IETF 6LoWPAN Working Group", Web https://datatracker.ietf.org/wg/6lowpan/charter/, 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

Rene Hummen
RWTH Aachen University
Templergraben 55
Aachen, 52056
Germany

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

Rene Struik
Struik Security Consultancy
Toronto, ON
Canada

Email: rstruik.ext@gmail.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 8, 2011

K. Hartke
Universitaet Bremen TZI

March 7, 2011

Accessing HTTP resources over CoAP
draft-hartke-core-coap-http-00

Abstract

The CoAP/HTTP proxying mechanism defined in this document enables Constrained Application Protocol (CoAP) clients to access a Hypertext Transfer Protocol (HTTP) resource by delegating the task of retrieving resource representations and manipulating resources to a CoAP intermediary.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 8, 2011.

Table of Contents

1. Introduction 3
2. CoAP/HTTP Proxying 3
2.1. GET 4
2.2. PUT 4
2.3. DELETE 5
2.4. POST 5
3. Implementation Considerations 5
3.1. Payloads and Media Types 6
3.2. Max-Age and ETag Options 6
3.3. Block-wise Transfers 6
3.4. HTTP Status Codes 1xx and 3xx 7
3.5. Example 7
4. Security Considerations 9
5. IANA Considerations 10
6. Acknowledgements 10
7. Normative References 10
Author's Address 10

1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] allows clients to specify an arbitrary, absolute Uniform Resource Identifier (URI) [RFC3986] in requests to CoAP proxies. This document defines what it means to perform a CoAP request on a Hypertext Transfer Protocol (HTTP) URI [RFC2616].

The resulting CoAP/HTTP proxying mechanism enables CoAP clients to access HTTP resources over CoAP by delegating the task of retrieving resource representations and manipulating resources to a CoAP intermediary. This relieves the client from the burden of implementing any HTTP functionality itself.

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/HTTP Proxying

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 (see Section 3).

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 MUST 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 timeframe, a 5.04 (Gateway Timeout) response MUST be returned; if a result can be obtained but is not understood, a 5.02 (Bad Gateway) response MUST be returned.

2.1. GET

The GET method requests the proxy to return a representation of the HTTP resource identified by the request URI.

Upon success, a 2.00 (OK) 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 options:

Lifetime: The request MAY include a Lifetime Option [I-D.ietf-core-observe]. This requests the proxy to notify the client of any changes to the state of the target HTTP resource. If the proxy is unable to detect changes to the resource or is unwilling to establish an observation relationship, the proxy MUST silently ignore the option. Otherwise, the proxy establishes an observation relationship as described in [I-D.ietf-core-observe] and, whenever the state of the HTTP resource changes, sends a 2.00 (OK) notification response to the client.

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 2.00 (OK) response with an entity tag in the requested set otherwise. (See also Section 3.2.)

Block: The request MAY include a Block Option [I-D.ietf-core-block], specifying the block the client is interested in. This requests the proxy to send only the requested block in a 2.00 (OK) response instead of the whole representation. (See also Section 3.3.)

2.2. 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.

A client can influence the processing of a PUT request by including the following options:

Block: The request MAY include a Block Option, indicating a block-wise transfer as described in [I-D.ietf-core-block]. (See also Section 3.3.)

2.3. 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.

No further options are defined to enable the client to influence the processing of a DELETE request.

2.4. 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.

A client can influence the processing of a PUT request by including the following options:

Block: The request MAY include a Block Option, indicating a block-wise transfer as described in [I-D.ietf-core-block]. (See also Section 3.3.)

3. Implementation Considerations

The CoAP/HTTP proxying requirements described in Section 2 deliberately make no restrictions on how a proxy implementation actually satisfies a request to produce the expected response. Obviously, these requirements would not be very useful if they could not be implemented. Also, some of the proxies will run on nodes that exhibit some constraints and may use networks that also are constrained.

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 non-normative section gives some hints for implementing the translation.

3.1. 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".

3.2. 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.

3.3. Block-wise Transfers

The proxy can optimize the processing of many requests. For example, when a GET request includes a Block Option, the proxy can use HTTP Range Requests to retrieve only the requested block(s) of the HTTP resource from the origin server instead of retrieving the whole resource.

Section 2 makes no restrictions on whether the proxy attempts to re-assemble all block-wise PUT requests into a single HTTP PUT request or attempts to push each request forward by making use of the HTTP Content-Range header field (which is not widely implemented for PUT outside WebDAV). In either case, the handling chosen MUST be reflected in the More bit sent back in the CoAP response as specified in [I-D.ietf-core-block].

For POST, similar considerations as with PUT apply, except that HTTP

Content-Range is even less widely implemented with HTTP POST.

3.4. 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.

3.5. Example

The example in Figure 1 shows a possible 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 UTF-8 charset, calculates the Max-Age Option from the Expires header field, and derives an entity-tag from the ETag header field.

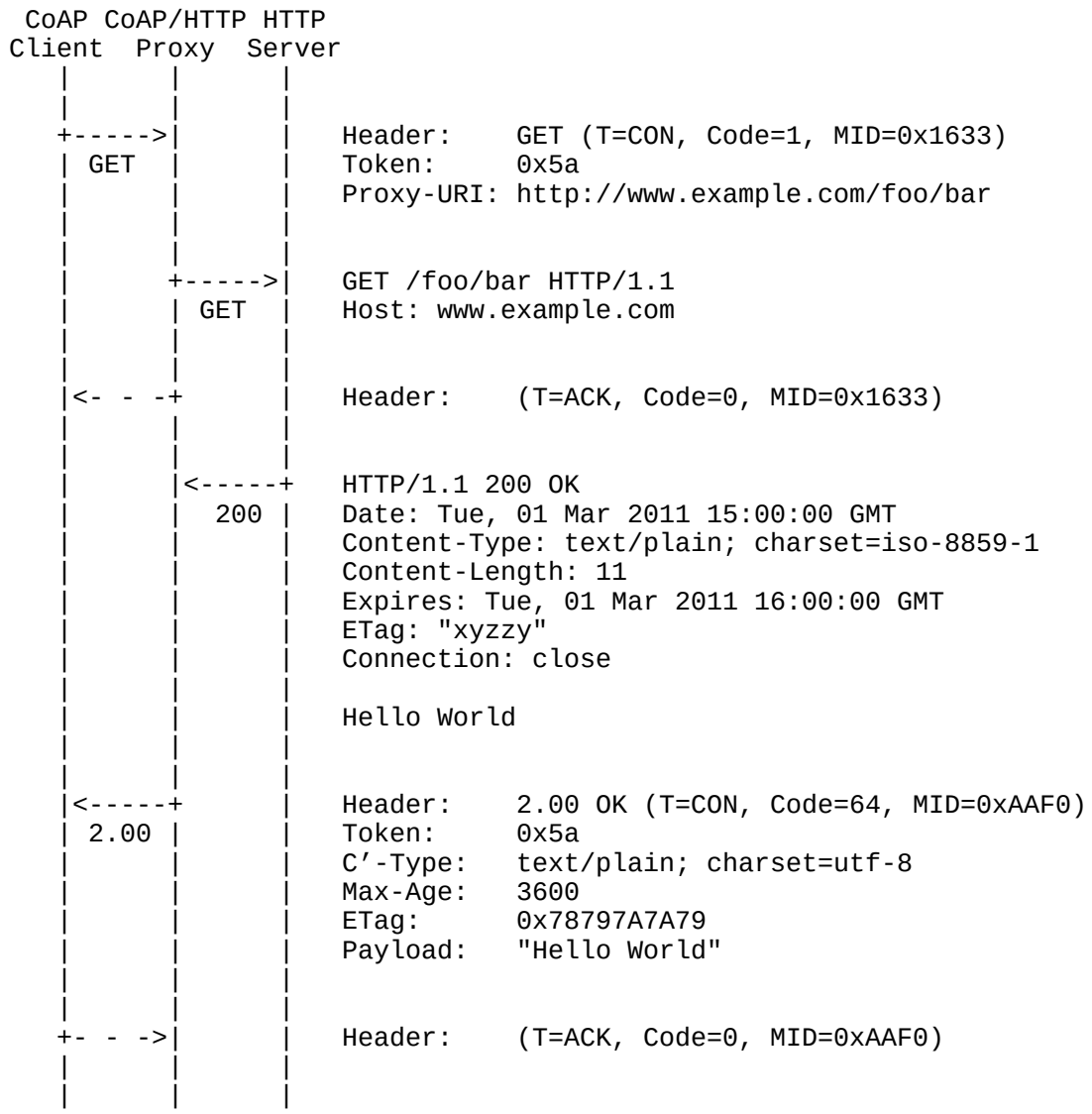


Figure 1: A basic GET request

The example in Figure 2 builds on the previous example and shows a possible 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.

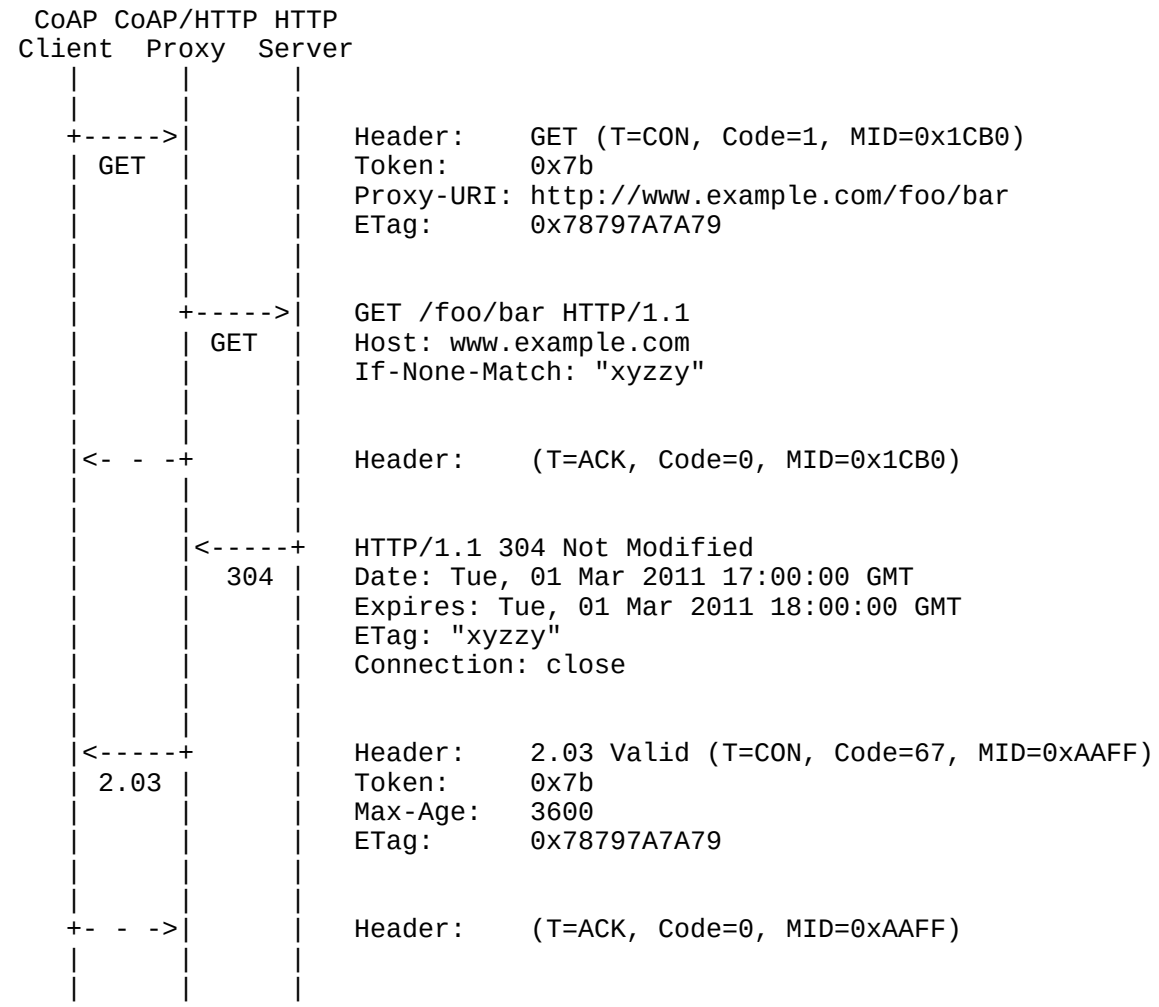


Figure 2: A GET request with an ETag Option

4. Security Considerations

When CoAP NoSec mode is used, a CoAP to HTTP proxy provides a way for a UDP sender to initiate an HTTP transfer, possibly using a fake source address, making the original request untraceable in general. (I.e. the usual minimal protection afforded to HTTP proxies by the TCP three-way handshake does not apply.) CoAP/HTTP proxies MUST be used in a way that they don't attract attackers that want to exploit this, e.g., by requiring communication security and/or by running them in a protected network.

The security considerations of section 15.7 of RFC 2616 apply and MUST be heeded. (TODO: Add further text about the perils of proxies, e.g., cache poisoning.)

5. IANA Considerations

This document does not require IANA action.

6. Acknowledgements

Some text for Section 3 and Section 4 was contributed by Carsten Bormann.

7. Normative References

[I-D.ietf-core-block]
Shelby, Z. and C. Bormann, "Blockwise transfers in CoAP", draft-ietf-core-block-01 (work in progress), January 2011.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[I-D.ietf-core-observe]
Hartke, K. and Z. Shelby, "Observing Resources in CoAP", draft-ietf-core-observe-01 (work in progress), February 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.

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

Author's Address

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 15, 2011

Z. Shelby, Ed.
Sensinode
C. Bormann
Universitaet Bremen TZI
March 14, 2011

Blockwise transfers in CoAP
draft-ietf-core-block-02

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 option provides a minimal way to transfer larger representations in a block-wise fashion.

Table of Contents

1. Introduction 3
2. Block-wise transfers 5
2.1. The Block Option 5
2.2. Using the Block Option 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
Authors' Addresses 22

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:

- reducing implementation complexity in order to minimize code size,
- 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,
- 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 CoAP option to enable _block-wise_ access to resource representations. The Block option provides 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 Block option to CoAP that can be used for block-wise transfers. Benefits of using this option

include:

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

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

Implementation of the 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 should not be fixed by the protocol. On the other hand, implementation should be as simple as possible. The Block option therefore supports a small range of power-of-two block sizes, from 2^4 (16) to 2^11 (2048) bytes. One of these eight values can be encoded in three bits (0 for 2^4 to 7 for 2^11 bytes), which we call the "SZX" (size exponent); the actual block size is then "1 << (SZX + 4)".

When a representation is larger than can be comfortably transferred in a single UDP datagram, the Block option can be used to indicate a block-wise transfer. Block 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 (M or "more" bit). The option value divided by sixteen is the number of the block currently being transferred, starting from zero, i.e., the current transfer is about the "size" bytes starting at byte "block number << (SZX + 4)". The default value of the Block Option 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.

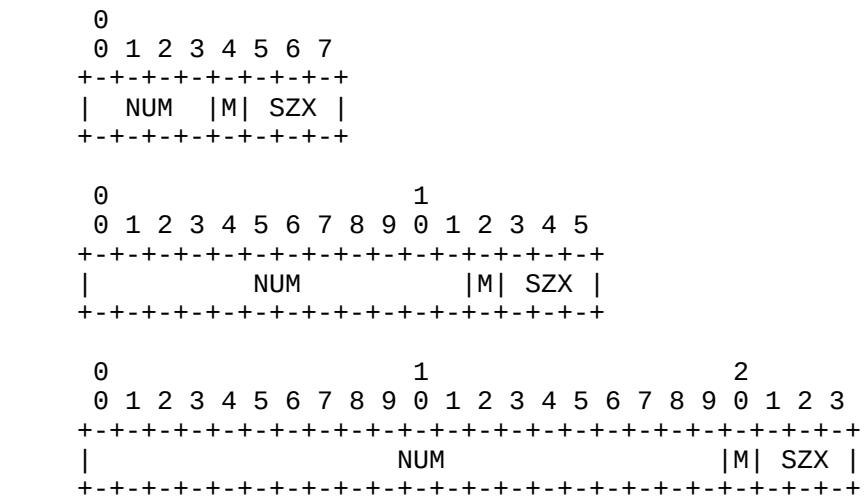


Figure 1: Block option

(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 indicating the block number being requested or provided. Block number 0 indicates the first block of a representation.

M: More Flag. 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 the block option is used in a request to retrieve a specific block number, the M bit MUST be sent as zero and ignored on reception.

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). As there are three bits available for SZX, the minimum block size is 2^(0+4) = 16 and the maximum is 2^(7+4) = 2048.

The Block option is used in one of three roles:

o In the request for a GET, the Block option gives the block number requested and suggests a block size (block number 0) or echoes the block size of previous blocks received (block numbers other than 0).

- In the response for a GET or in the request for a PUT or POST, the Block option describes what block number is contained in the payload, 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).

- In the response for a PUT or POST, the Block option indicates what block number is being acknowledged. 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 PUT/POST atomically (i.e., acts only upon reception of the last block). 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 this sequence of block-wise transfers). Finally, the block size given in such a Block 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 PUT/POST requests in the transfer sequence.

2.2. Using the Block Option

Using the Block option, 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 GET is answered with a response carrying a Block option with the M bit set, the requester may retrieve additional blocks of the resource representation by sending requests with a Block option giving the block number desired. In such a Block option, the M bit MUST be sent as zero and ignored on reception.

To influence the block size used in response to a GET request, the requester uses the Block option, giving the desired size, a block number of zero and an M bit of zero. A server SHOULD 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 Block option.

Once the Block option is used by the requester, all GET requests in a single 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 Block option in a second request after a first request without a Block 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 GET and proceed to use it in subsequent GETs; 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 Block 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 Block 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. When reassembling the representation from the blocks being exchanged, the reassembler 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 PUT or POST transfer, the Block option refers to the body in the request, i.e., there is no way to perform a block-wise retrieval of the body of the response. Servers that do need to supply large bodies in response to PUT/POST SHOULD therefore be employing mechanisms such as providing a location for a resource that can be used in a GET to obtain that information.

In a PUT or POST transfer response, the block size given in the Block option indicates the block size preference of the server for this resource. Obviously, at this point the first block has already been transferred 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 PUT or POST transfer 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,

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 PUT or POST transfer that it would intend to implement in an atomic fashion.

If multiple concurrently proceeding block-wise 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 block number (NUM), more bit (M), and block size exponent (2^(SZX+4)) by slashes. E.g., a block option value of 33 would be shown as 2/0/32, or a block option value of 59 would be shown as 3/1/128.

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

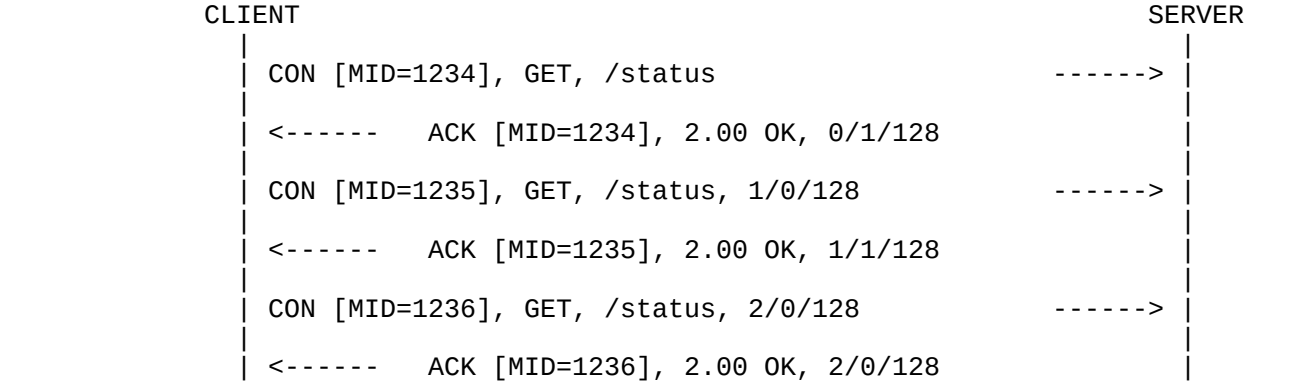


Figure 2: Simple blockwise GET

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

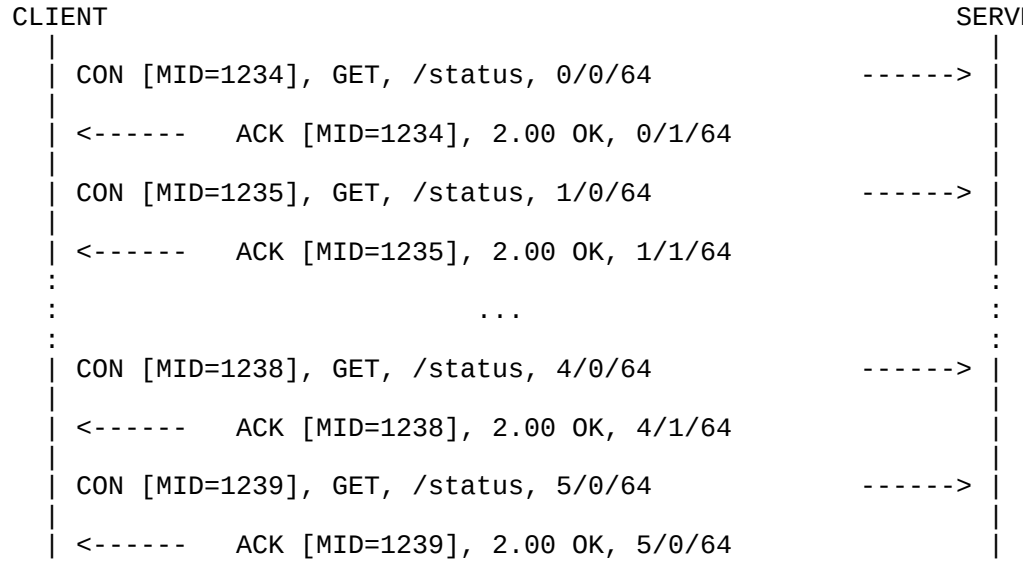


Figure 3: Blockwise GET with early negotiation

In the third example (Figure 4), the client is surprised by the need for a blockwise transfer, and unhappy with the size chosen unilaterally by the server. As it did not send a size proposal initially, the negotiation only influences the size from the second message exchange. 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.

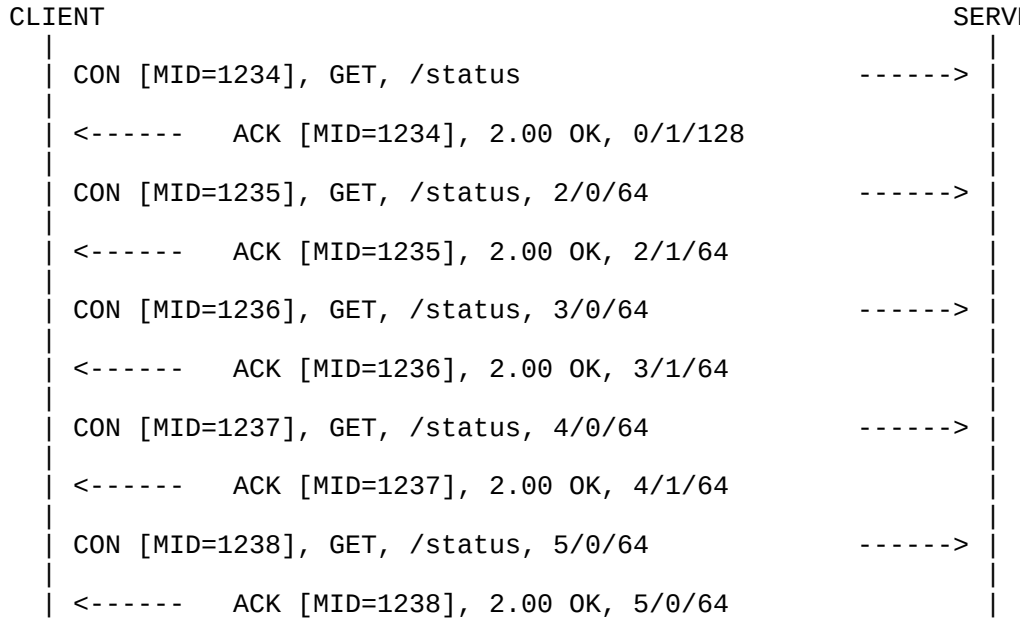


Figure 4: Blockwise GET with late negotiation

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).

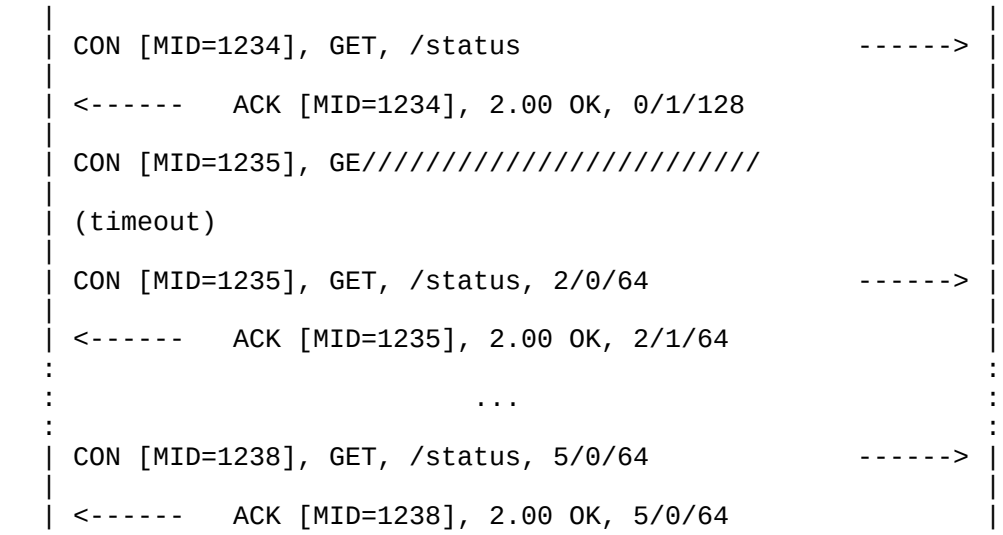


Figure 5: Blockwise GET with late negotiation and lost CON

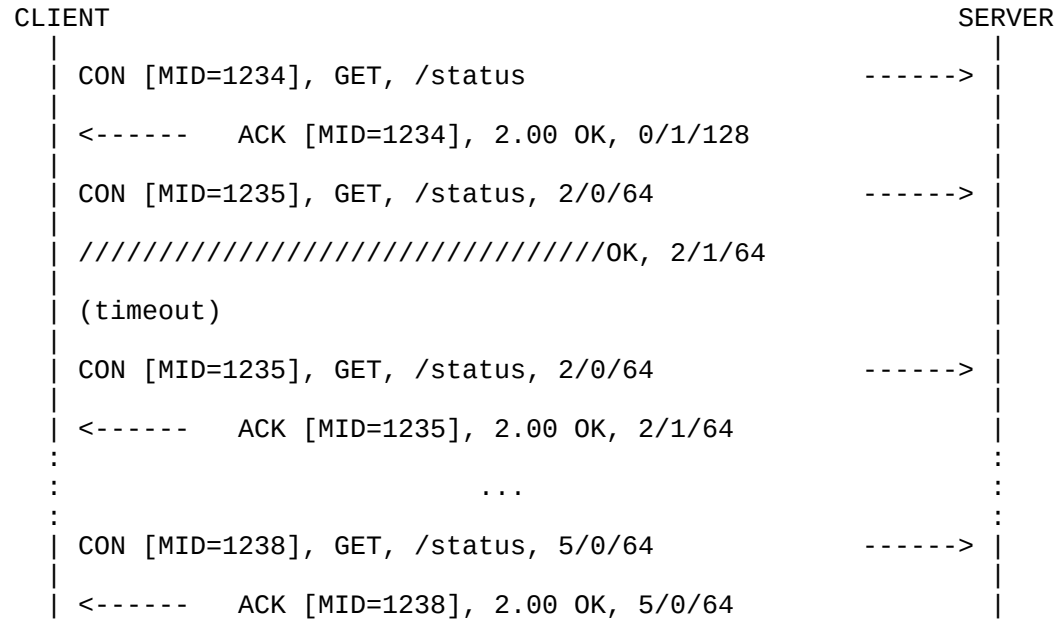


Figure 6: Blockwise GET with late negotiation and lost ACK

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 block option are provisional; only the final response tells the client that the PUT succeeded.

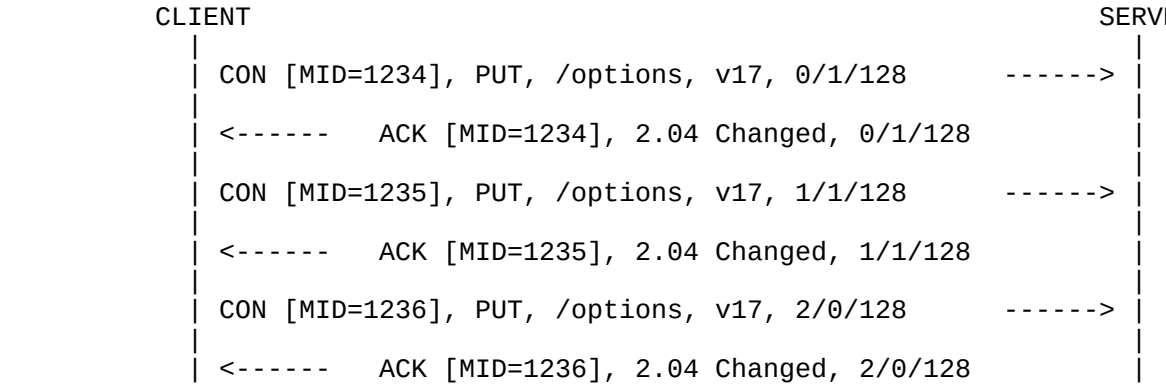


Figure 7: Simple atomic blockwise PUT

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.

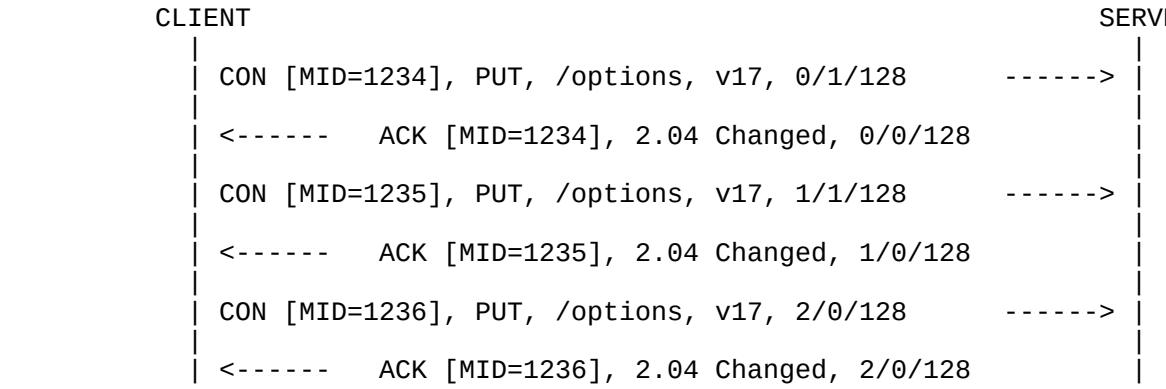


Figure 8: Simple stateless blockwise PUT

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.

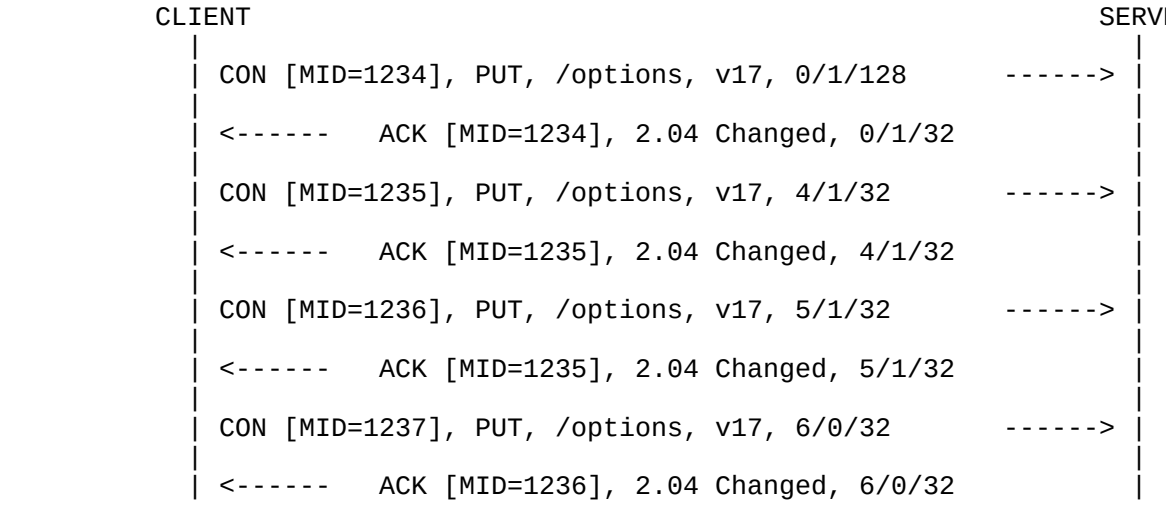


Figure 9: Simple atomic blockwise PUT with negotiation

3.1. HTTP Mapping Considerations

In this subsection, we give some brief examples for the influence the Block Option 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.

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
13	Block	[RFCXXXX]

Table 1: 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 2: 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 option 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 accreted 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.

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-05 (work in progress), March 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.

Authors' Addresses

Zach Shelby (editor)
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

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

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 15, 2011

Z. Shelby
Sensinode
K. Hartke
C. Bormann
Universitaet Bremen TZI
B. Frank
SkyFoundry

March 14, 2011

Constrained Application Protocol (CoAP)
draft-ietf-core-coap-05

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.

Table of Contents

1. Introduction 5
1.1. Features 5
1.2. Terminology 6
2. Constrained Application Protocol 7
2.1. Messaging Model 8
2.2. Request/Response Model 9
2.3. Intermediaries and Caching 11
2.4. Resource Discovery 11
3. Message Syntax 12
3.1. Message Format 12
3.1.1. Message Size Implementation Considerations 13
3.2. Option Format 14
4. Message Semantics 15
4.1. Reliable Messages 16
4.2. Unreliable Messages 17
4.3. Message Types 17
4.3.1. Confirmable (CON) 18
4.3.2. Non-Confirmable (NON) 18
4.3.3. Acknowledgement (ACK) 18
4.3.4. Reset (RST) 18
4.4. Multicast 19
4.5. Congestion Control 19
5. Request/Response Semantics 19
5.1. Requests 19
5.2. Responses 20
5.2.1. Piggy-backed 21
5.2.2. Separate 21
5.2.3. Non-Confirmable 22
5.3. Request/Response Matching 22
5.4. Options 23
5.4.1. Critical/Elective 23
5.4.2. Length 24

5.4.3. Default Values 24
5.4.4. Repeating Options 24
5.4.5. Option Numbers 24
5.5. Payload 25
5.6. Caching 25
5.6.1. Freshness Model 26
5.6.2. Validation Model 26
5.7. Proxying 27
5.8. Method Definitions 28
5.8.1. GET 28
5.8.2. POST 28
5.8.3. PUT 29
5.8.4. DELETE 29
5.9. Response Code Definitions 30
5.9.1. Success 2.xx 30
5.9.2. Client Error 4.xx 31
5.9.3. Server Error 5.xx 32
5.10. Option Definitions 33
5.10.1. Token 33
5.10.2. Uri-Host, Uri-Port, Uri-Path and Uri-Query 34
5.10.3. Proxy-Uri 35
5.10.4. Content-Type 35
5.10.5. Max-Age 35
5.10.6. ETag 36
5.10.7. Location-Path and Location-Query 36
6. CoAP URIs 36
6.1. URI Scheme Syntax 37
6.2. Normalization and Comparison Rules 37
6.3. Parsing URIs 38
6.4. Constructing URIs 39
7. Finding and Addressing CoAP End-Points 40
7.1. Resource Discovery 40
7.2. Default Port 40
7.3. Multiplexing DTLS and CoAP 41
7.3.1. Future-Proofing the Multiplexing 41
8. HTTP Mapping 42
8.1. CoAP-HTTP Mapping 43
8.2. HTTP-CoAP Mapping 47
9. Protocol Constants 49
10. Security Considerations 49
10.1. Securing CoAP with IPsec 50
10.2. Securing CoAP with DTLS 51
10.2.1. SharedKey and MultiKey Modes 52
10.2.2. Certificate Mode 52
10.3. Threat analysis and protocol limitations 53
10.3.1. Protocol Parsing, Processing URIs 53
10.3.2. Proxying and Caching 53
10.3.3. Risk of amplification 54

10.3.4. Cross-Protocol Attacks 55
11. IANA Considerations 56
11.1. CoAP Code Registry 57
11.1.1. Method Codes 57
11.1.2. Response Codes 58
11.2. Option Number Registry 59
11.3. Media Type Registry 60
11.4. URI Scheme Registration 62
11.5. Service Name and Port Number Registration 63
12. Acknowledgements 63
13. References 64
13.1. Normative References 64
13.2. Informative References 66
Appendix A. Integer Option Value Format 67
Appendix B. Examples 68
Appendix C. URI Examples 74
Appendix D. Changelog 76
Authors' Addresses 80

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:

- Constrained web protocol fulfilling M2M requirements.
- 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.
- UDP binding with reliable unicast and best-effort multicast support.
- Asynchronous message exchanges.
- Low header overhead and parsing complexity.

- URI and Content-type support.
- Simple proxy and caching capabilities.
- Optional resource discovery.

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).

Intermediary
: There are two common forms of intermediary: proxy and reverse proxy. In some cases, a single intermediary 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 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 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).

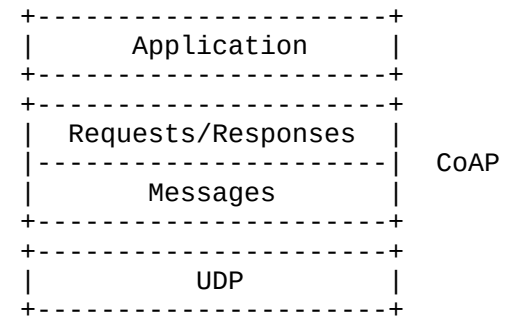


Figure 1: Abstract layering of CoAP

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); 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).

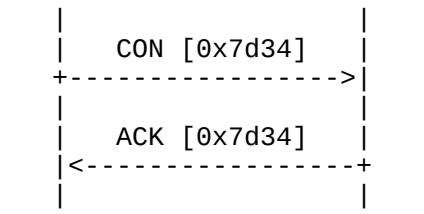


Figure 2: Reliable message delivery

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 (Figure 3).

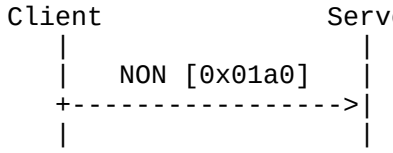


Figure 3: Unreliable message delivery

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 4.4 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.

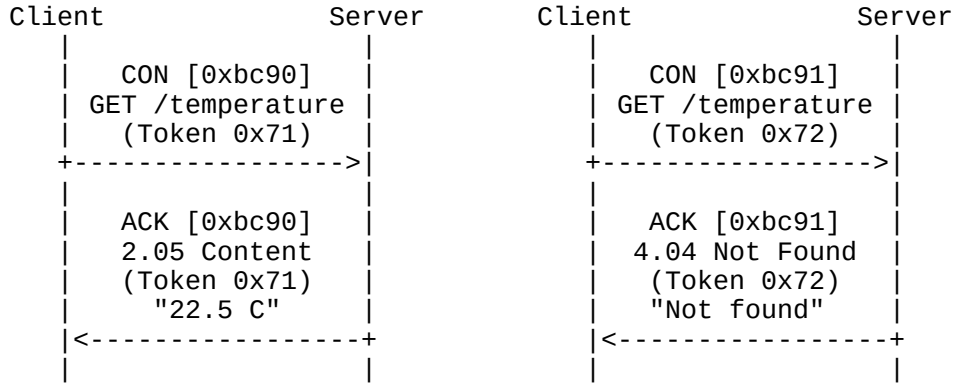


Figure 4: Two GET requests with piggy-backed responses, one successful, one not found

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.

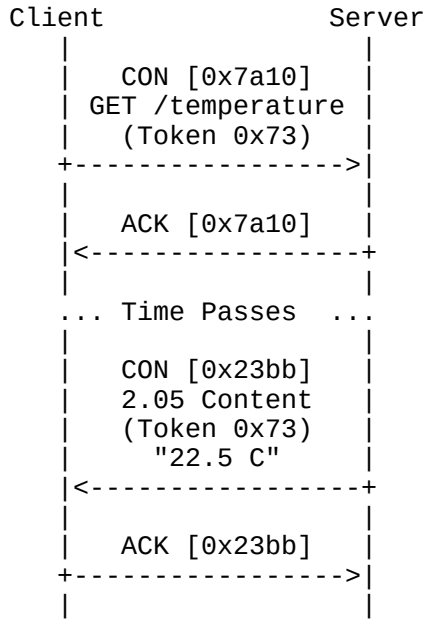


Figure 5: A GET request with a separate response

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 10.2). 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.

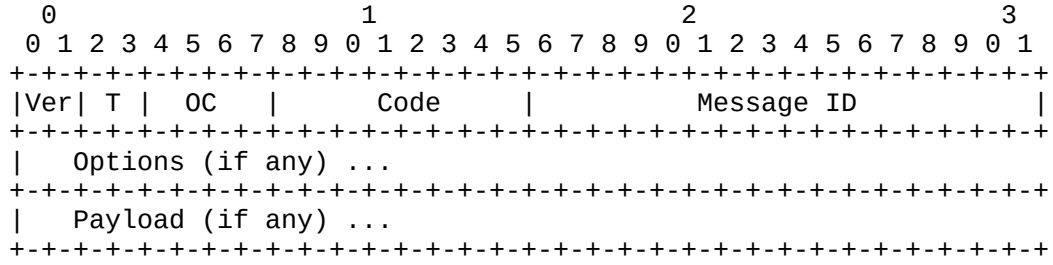


Figure 6: Message Format

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 (see also Section 7.3.1).

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.

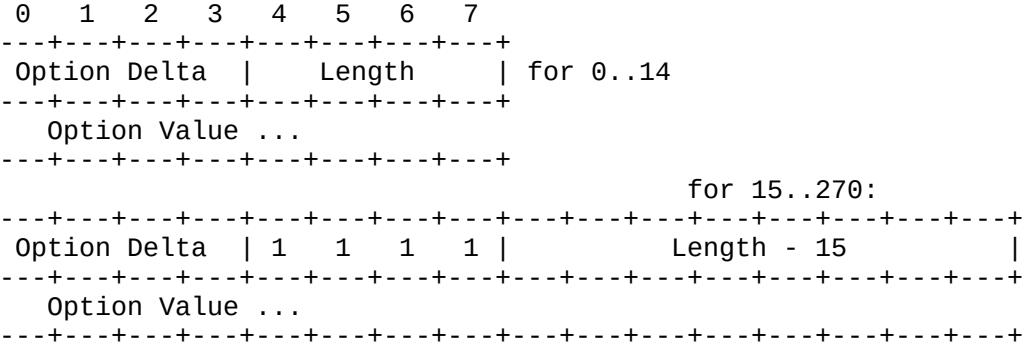


Figure 7: Option Format

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].

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:

- Simple stop-and-wait retransmission reliability with exponential back-off for "confirmable" messages.
- Duplicate detection for both "confirmable" and "non-confirmable" messages.
- 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 RESPONSE_TIMEOUT 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 cancelled 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. 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. 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. The initial variable value SHOULD be randomized. The same Message ID MUST NOT be re-used within the potential retransmission window, calculated as RESPONSE_TIMEOUT * (2 ^ 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) 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.)

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 or be rejected by the recipient. If a recipient lacks context to process the message properly, the message MUST be silently ignored.

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) 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 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 signalled 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 those bytes MUST be ignored by any recipient.

4.3.1. 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.

4.3.2. 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.

4.3.3. 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).

4.3.4. 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.

4.4. Multicast

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].

4.5. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.1. Further congestion control optimizations are being considered and tested for CoAP [I-D.eggert-core-congestion-control].

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 can be 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).

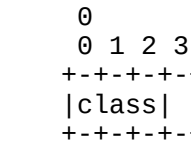


Figure 8: Structure of a Response Code

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 8). 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.

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.

(Note that, as the underlying datagram transport may not be sequence-preserving, the confirmable message carrying the response may actually arrive before or after the acknowledgement message for the request.)

For 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. 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:

- Content-Type
- ETag
- Location-Path
- Location-Query
- Max-Age
- Proxy-Uri
- Token
- Uri-Host
- Uri-Path
- Uri-Port
- Uri-Query

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 end-point is handled:

- Upon reception, unrecognized options of class "elective" MUST be silently ignored.

- 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).

- Unrecognized options of class "critical" that occur in a confirmable response SHOULD cause the response to be rejected with a reset message.

- Unrecognized options of class "critical" that occur in a non-confirmable message MUST cause the message 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. A default value of "text/plain; charset=utf-8" 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 5.10.5). The Max-Age Option indicates that the response is to be considered not fresh after its age is greater than the specified number of seconds.

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: proxy-max-age = original-max-age - 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 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.

If the request passes through a cache that has one or more stored responses for the request URI, those stored responses SHOULD be marked as stale.

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.

If the request passes through a cache that has one or more stored responses for the request URI, those stored responses SHOULD be marked as stale.

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.

If the request passes through a cache and the request URI identifies one or more currently stored responses, those entries SHOULD be marked as stale.

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.

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 location 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.

This response is not cacheable.

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.

This response is not cacheable.

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 "Accept" header field.

5.9.2.7. 4.13 Request Entity Too Large

Like HTTP 413 "Request Entity Too Large".

5.9.2.8. 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	1-2 B	0
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)
15	Critical	Uri-Query	string	1-270 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 token is intended for use as a client-local identifier for differentiating between concurrent requests. A client SHOULD generate tokens in a way that tokens currently in use for a given source/destination pair are unique. An end-point receiving a token MUST treat it as opaque and make no assumptions about its format.

A default value of a zero-length token is assumed in the absence of the option.

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 (except for Uri-Query) 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 6.3. 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:

- the Uri-Host Option specifies the Internet host of the resource being requested,
- the Uri-Port Option specifies the port number of the resource,
- each Uri-Path Option specifies one segment of the absolute path to the resource, and
- the Uri-Query Option specifies the query.

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 Uri-Path Option can contain any character sequence. No percent-encoding is performed. The value 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 6.4. 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, Uri-Port and Uri-Query MUST NOT occur more than once; Uri-Path 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). A default value of 0 (meaning "text/plain; charset=utf-8") is assumed in the absence of the option.

This option is "critical". It MUST NOT occur more than once.

5.10.5. 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.

5.10.6. 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.

5.10.7. 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 treated as stale.

Both options are "elective". Location-Path MAY occur one or more times. Location-Query MUST NOT occur more than once.

6. CoAP URIs

CoAP uses the "coap" URI scheme 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 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. CoAP URIs can thus be compared to the "http" URI scheme.

6.1. URI Scheme Syntax

The syntax of the "coap" URI scheme is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", and "query", "segment", "IP-literal", "IPv4address" and "reg-name" are adopted from [RFC3986].

```
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 [IANA_TBD_PORT] 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. The query serves to further parametrize the resource, often in the form of "key=value" pairs.

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. Normalization and Comparison Rules

Since the "coap" scheme conforms to the URI generic syntax, URIs of this scheme are normalized and compared according to the algorithm defined in [RFC3986], Section 6.

If the port is equal to the default port [IANA_TBD_PORT], the normal form is to elide the port component. 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:[IANA_TBD_PORT]/~sensors/temp.xml

coap://EXAMPLE.com/%7Esensors/temp.xml

coap://EXAMPLE.com:/%7esensors/temp.xml

6.3. Parsing URIs

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", 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 each percent-encoding ("%" followed by two hexadecimal digits) to the corresponding byte.

   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 [IANA_TBD_PORT].

7. If /port/ does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be /port/.

8. If the value of the <path> component of /url/ is empty or consists of a single slash character (U+002F SOLIDUS "/"), then move to the next step.

   Otherwise, for each segment in the <path> component, include a Uri-Path Option and let that option's value be the segment (not including the delimiting slash characters) after converting each percent-encoding ("%" followed by two hexadecimal digits) to the corresponding byte.

9. If /url/ has a <query> component, then include a Uri-Query Option and let that option's value be the value of the <query> component (not including the delimiting question mark). (Note that, in contrast to the other components, percent-encodings stay intact in the Uri-Query option.)

Note that these rules completely resolve any percent-encoding except in a reg-name and in a query.

6.4. Constructing URIs

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. 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 [IANA_TBD_PORT], 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 character (:) or U+0040 COMMERCIAL AT (@), to its percent-encoded form.

7. If /resource name/ is the empty string, set it to a single character U+002F SOLIDUS (/).

8. Append /resource name/ to /url/.

9. If the request includes a Uri-Query Option, append a single U+003F QUESTION MARK character (?) to /url/, followed by the option's value.

10. Return /url/.

Note that these steps have been designed to lead to a URI in normal form (see Section 6.2).

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].

7.2. Default Port

The CoAP default port number [IANA_TBD_PORT] MUST be supported by a server for resource discovery and SHOULD be supported for providing access to other resources. In addition other end-points 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].

7.3. Multiplexing DTLS and CoAP

The CoAP encoding has been chosen to enable demultiplexing of two kinds of packets that arrive on a single UDP port:

- CoAP messages directly sent within UDP
- DTLS 1.1 or 1.2 messages (which might contain CoAP messages) on UDP

Possibly less importantly, a distinction can also be made between these two and:

- STUN messages on UDP

This demultiplexing is possible because DTLS 1.1 or 1.2 UDP payloads begin with a byte out of:

```
enum {
  change_cipher_spec(20), alert(21), handshake(22),
  application_data(23), (255)
} ContentType;
```

Figure 9: TLS ContentType

i.e. 0x14 to 0x17 hex [RFC4347]. In a CoAP message, such an initial byte would be decoded as a CoAP version 0, which is not in use.

7.3.1. Future-Proofing the Multiplexing

To maintain this property, future versions of CoAP will not use version number 0. Note that future versions of DTLS might theoretically start to use "ContentType" values that fall into the range of 64 to 127; CoAP implementations would then not be able to reliably multiplex these new kinds of DTLS datagrams with CoAP datagrams on the same UDP port. To maintain transparency for this case, an initial byte of 0x11 (17 decimal) is inserted on transmission and discarded upon reception; the rest of the datagram is interpreted as the DTLS message. 0x11 MUST NOT be followed by 0x14 to 0x17 hex, i.e. the DTLS messages defined by DTLS 1.1 and 1.2 are always sent unescaped. Datagrams starting with 0x11 and then 0x14 to 0x17 MUST be discarded.

Similarly, STUN messages begin with 00mmmmmc binary (MSBs) [RFC5389] and so far happen to use an encoding for mmmmmc that also enables

this initial byte to be distinguished from valid DTLS messages. Again, future versions of CoAP will need to avoid using version number 0. STUN messages are most likely to begin with 0x00 and 0x01. All other STUN messages MUST be escaped with an initial 0x10 byte (16 decimal). 0x10 MUST NOT be followed by 0x00 or 0x01 hex, i.e. the more likely STUN messages are always sent unescaped.

Future versions of CoAP could potentially make changes to the CoAP header structure that are not backwards compatible to the current version. In order to allow demultiplexing those packets that adhere to the present version of CoAP from those using the future version, the new version may want to increase the CoAP version number in the header (fixed at "1" in the present specification) and/or make other changes in the initial byte and/or the escaping rules. Whatever these changes may be, their objective will be to enable seamless interworking of existing and new protocol implementations to enable an orderly transition to the new version.

Note that the escaping rules defined in this section are insurance for the future; they need no additional code in implementations that do not implement STUN or DTLS or implement only the versions current at the time of writing. For easy reference, Table 2 summarizes the rules upon reception.

initial byte	disposition	interpretation
0x00 or 0x01	keep	STUN
0x10	remove	STUN
0x11	remove	DTLS
0x14 to 0x17	keep	DTLS
0x40 to 0x7F	keep	CoAP
all others		(invalid)

Table 2: Interpretation of initial byte when multiplexing

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 [RFC3920] or SIP [RFC3264], the definition of these mappings is out of scope of this specification.

This section discusses two ways of mapping:

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" URI in a CoAP request to a CoAP-HTTP proxy, or by sending a CoAP request to a reverse proxy that maps CoAP to HTTP.

HTTP-CoAP Mapping: Enables HTTP clients to access resources on CoAP servers through an intermediary. This is initiated by specifying a "coap" URI in the Request-Line of an HTTP request to an HTTP-CoAP proxy, or by sending an HTTP request to a reverse proxy that maps HTTP to CoAP.

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.

8.1. CoAP-HTTP Mapping

The mapping of CoAP to HTTP is a relatively straightforward conversion of the CoAP method or response code, content-type and options to the corresponding HTTP feature. The payload is carried in an equivalent way by both protocols.

In a similar manner to CoAP-CoAP proxying, the CoAP-HTTP proxy MAY perform caching of HTTP responses. If no caching is performed, a CoAP GET request that specifies an entity-tag in an ETag Option SHOULD be mapped to a conditional HTTP request that includes the entity-tag in the "If-None-Match" request-header field. If the entity-tag matches the entity-tag of the representation, the HTTP server responds with an HTTP 304 (Not Modified) response which SHOULD be mapped to a CoAP 2.03 (Valid) response with the ETag Option reflecting the response’s "ETag" response-header field. The mapping of max-age is straightforward.

HTTP entity-tags consist of characters in a subset of the US-ASCII character set, which can be carried directly in a CoAP ETag Option. Weak entity-tags are not supported by this mapping. However, an entity-tag may not fit within the CoAP ETag Option. In this case, the proxy MAY map the entity-tag to a shorter unique byte sequence and keep state, or MAY silently ignore the "ETag" response-header when mapping an HTTP response to CoAP (so the CoAP client will never send a CoAP GET request with an ETag Option).

Provisional responses (HTTP Status Codes 1xx), and responses indicating that further action needs to be taken (HTTP Status Codes

3xx), SHOULD cause the proxy to complete the request, e.g., by following the redirects. If the proxy is unable to complete the request, it SHOULD respond with a CoAP 5.02 (Bad Gateway) error.

HTTP responses are mapped to CoAP responses as follows:

HTTP Status Code	CoAP Response Code	Notes
100 Continue		2
101 Switching Protocols		2
200 OK		3
201 Created	2.01 Created	
202 Accepted		4
203 Non-Authoritative Information		4
204 No Content		6
205 Reset Content		4
206 Partial Content		2
300 Multiple Choices		2
301 Moved Permanently		2
302 Found		2
303 See Other		2
304 Not Modified	2.03 Valid	7
305 Use Proxy		2
306 (Unused)	5.02 Bad Gateway	1
307 Temporary Redirect		2
400 Bad Request	4.00 Bad Request	
401 Unauthorized	4.01 Unauthorized	5
402 Payment Required	4.00 Bad Request	1
403 Forbidden	4.03 Forbidden	
404 Not Found	4.04 Not Found	
405 Method Not Allowed	4.05 Method Not Allowed	8
406 Not Acceptable	4.00 Bad Request	1
407 Proxy Authentication Required	4.00 Bad Request	1
408 Request Timeout	4.00 Bad Request	1
409 Conflict	4.00 Bad Request	1
410 Gone	4.00 Bad Request	1
411 Length Required	4.00 Bad Request	1
412 Precondition Failed	4.00 Bad Request	1
413 Request Entity Too Large	4.13 Request Entity Too Large	
414 URI Too Long	4.00 Bad Request	1
415 Unsupported Media Type	4.15 Unsupported Media Type	
416 Requested Range Not Satisfiable	4.00 Bad Request	1
417 Expectation Failed	4.00 Bad Request	1
500 Internal Server Error	5.00 Internal Server Error	
501 Not Implemented	5.01 Not Implemented	
502 Bad Gateway	5.02 Bad Gateway	
503 Service Unavailable	5.03 Service Unavailable	9

504 Gateway Timeout 505 HTTP Version Not Supported	5.04 Gateway Timeout	2

Table 3: CoAP-HTTP Mapping

Notes:

1. There is no equivalent CoAP response.

2. The proxy should perform the action implied by the response code (e.g., by following redirects); i.e. this response is never forwarded to the CoAP client. If the proxy is unable or unwilling to perform this function, the CoAP response code 5.02 (Bad Gateway) can be returned.

3. The CoAP response code depends on the request method. For a GET request, the response code SHOULD be 2.05 (Content). For a POST, PUT or DELETE request, the mapping is only partial: response entities are ignored, and the response code depends on the method as defined in Section 5.8.

4. (The mapping for these rarely-used status codes is not defined in this specification.)

5. The HTTP "WWW-Authenticate" response-header field has no equivalent option in CoAP and is either processed by the proxy by performing an additional request or silently dropped.

6. The CoAP response code depends on the request method. For a POST or PUT request, the response code SHOULD be 2.04 (Changed); for a DELETE request, 2.02 (Deleted).

7. Since a CoAP request with ETag Option is mapped to a conditional HTTP GET request with a "If-None-Match" request-header field, any HTTP 304 (Not Modified) response will confirm that the ETag is valid. Except for the max-age directive of the Cache-Control header field, any additional headers in the HTTP Not Modified response are not carried through to the CoAP client, though.

8. The HTTP "Accept" response-header field has no equivalent option in CoAP and is silently dropped.

9. The HTTP "Retry-After" response-header field has no equivalent option in CoAP, although it may be used to find a value for the Max-Age Option.

8.2. HTTP-CoAP Mapping

The mapping of HTTP to CoAP requires checking for methods, response codes and options that are not supported by CoAP. A proxy SHOULD attempt to map options, response codes and content-types to a suitable alternative if possible. Otherwise the unsupported feature SHOULD be silently dropped if possible, or an appropriate error code generated otherwise.

Mapping MAY include performing payload conversion (e.g., from EXI to XML), the definition of which is out of this document’s scope.

Only those Conditional HTTP requests can be mapped to CoAP requests that have method GET and include a "If-None-Match" request-header field. The "If-Match", "If-Modified-Since" and "If-Unmodified-Since" request-header fields are not supported on the CoAP side, but could be implemented locally by a caching proxy. A HTTP-CoAP proxy SHOULD map ETags generated by a CoAP server to HTTP-friendly ETags by using Base64 [RFC4648].

A proxy SHOULD respond with a HTTP 502 (Bad Gateway) error to HTTP requests which can not be successfully mapped to CoAP.

A proxy SHOULD employ a cache to limit traffic on the constrained network.

CoAP responses are mapped to HTTP responses as follows:

CoAP Response Code	HTTP Status Code	Notes
2.01 Created	201 Created	
2.02 Deleted	204 No Content	
2.03 Valid	304 Not Modified	1
2.04 Changed	204 No Content	
2.05 Content	200 OK	
4.00 Bad Request	400 Bad Request	
4.01 Unauthorized	400 Bad Request	2
4.02 Bad Option	400 Bad Request	
4.03 Forbidden	403 Forbidden	
4.04 Not Found	404 Not Found	
4.05 Method Not Allowed	405 Method Not Allowed	3
4.13 Request Entity Too Large	413 Request Entity Too Large	
4.15 Unsupported Media Type	415 Unsupported Media Type	
5.00 Internal Server Error	500 Internal Server Error	
5.01 Not Implemented	501 Not Implemented	
5.02 Bad Gateway	502 Bad Gateway	
5.03 Service Unavailable	503 Service Unavailable	4
5.04 Gateway Timeout	504 Gateway Timeout	
5.05 Proxying Not Supported	502 Bad Gateway	

Table 4: HTTP-CoAP Mapping

Notes:

1. A CoAP 2.03 (Valid) response only (1) confirms that the request ETag is valid and (2) provides a new Max-Age value. HTTP 304 (Not Modified) also updates some header fields of a stored response. A non-caching proxy may not have enough information to fill in the required values in the HTTP 304 (Not Modified) response, so it may not be advisable to provoke the 2.03 (Valid) response by forwarding an ETag. A caching proxy will fill the information out of the cache.

2. There is no equivalent HTTP status code.

3. CoAP does not provide enough information to compute a value for the required "Allow" response-header field. If this violation of [RFC2616] cannot be tolerated, the proxy should instead send a 4.00 (Bad Request) response.

4. The value of the "Retry-After" response-header field is the value of the Max-Age Option.

9. Protocol Constants

This section defines the relevant protocol constants defined in this document:

RESPONSE_TIMEOUT 2 seconds

MAX_RETRANSMIT 4

10. Security Considerations

This section describes mechanisms that can be used to secure CoAP and analyzes the possible threats to the protocol and its limitations. Security bootstrapping (authenticating nodes and setting up keys) in constrained environments is considered in [I-D.oflynn-core-bootstrapping].

During the bootstrap and enrollment phases, a CoAP device is provided with the security information that it needs, including keying materials. How this is done is out of scope for this specification but a couple of ways of doing this are described in [I-D.oflynn-core-bootstrapping]. At the end of the enrollment and bootstrap, the device will be in one of four security modes with the following information for the given mode:

NoSec: There is no protocol level security.

SharedKey: There is one shared key between all the nodes that this CoAP node needs to communicate with.

MultiKey: There is a list of shared keys and each key includes a list of which nodes it can be used to communicate with. At the extreme there may be one key for each node this CoAP node needs to communicate with.

Certificate: The device has an asymmetric key pair with a X.509 [RFC5280] certificate that binds it to its Authority Name and is signed by a some common trust root. The device also has a list of root trust anchors that can be used for validating a certificate. There may be an optional shared key that all the nodes that communicate have access to.

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.

In the "NoSec" mode, the system simply sends the packets over normal UDP over IP. 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 10.3.4 for an additional complication with this approach. The other three security modes can be achieved with IPsec or DTLS. 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 IPsec

One mechanism to secure CoAP in constrained environments is the IPsec Encapsulating Security Payload (ESP) [RFC4303]. 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]; 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 per-packet overhead of approximately 10 bytes, 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.

10.2. Securing CoAP with DTLS

Just as HTTP may be secured using Transport Layer Security (TLS) over TCP, CoAP may be secured using Datagram TLS (DTLS) [RFC4347] over UDP. This section gives a quick overview of how to secure CoAP with DTLS, along with the minimal 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 the proposed TLS_PSK_WITH_AES_128_CCM_8 [I-D.mcgrew-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 with certificates 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.

10.2.1. SharedKey and MultiKey Modes

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 SHOULD support the mandatory to implement cipher suite TLS_PSK_WITH_AES_128_CBC_SHA as specified in [RFC4279]; once TLS_PSK_WITH_AES_128_CCM_8 as specified in [I-D.mcgrew-tls-aes-ccm] (or related cipher suites specified in [I-D.mcgrew-tls-aes-ccm-ecc]) in conjunction with [I-D.ietf-tls-rfc4347-bis] becomes available, this may be easier to implement on certain contemporary chipsets.

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).

10.2.2. Certificate Mode

As with IPsec, DTLS should be configured with a cipher suite compatible with any possible hardware engine on the node, for example AES-CBC in the case of IEEE 802.15.4. Implementations SHOULD support the mandatory to implement cipher suite TLS_RSA_WITH_AES_128_CBC_SHA as specified in [RFC5246].

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 are 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.

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.

10.3. 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 15.2 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 and 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 network 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. 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:

- the attacker sends a message to a CoAP end point with a fake source address,
- the CoAP end point replies with a message to the given source address,
- 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 endpoint 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 endpoint. Unfortunately, the first two bytes of a DNS reply are an ID that can be chosen by the attacker, which map into the interesting part of the CoAP header, and the next two bytes are then interpreted as CoAP’s Message ID (i.e., any value is

acceptable). The DNS count words may be interpreted as multiple instances of a (non-existent, but elective) CoAP option 0. The echoed query finally may be manufactured by the attacker to achieve a desired effect on the CoAP endpoint; the response added by the server (if any) might then just be interpreted as added payload.

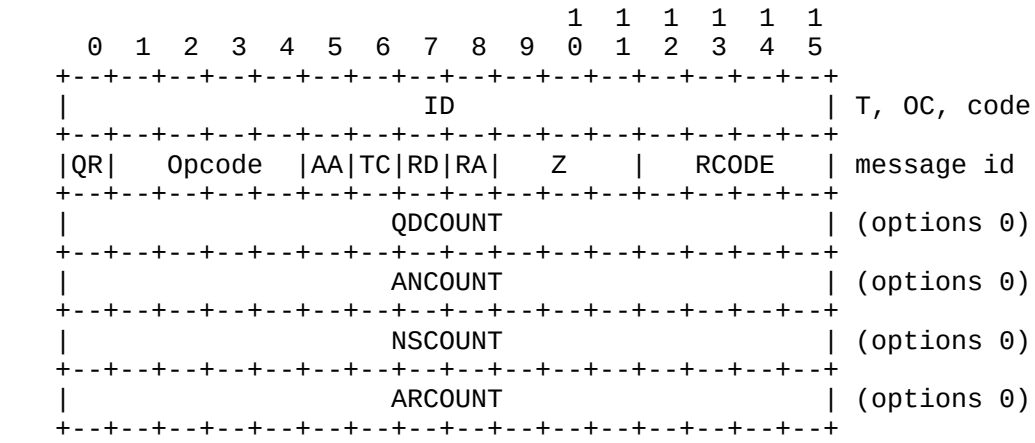


Figure 10: DNS Header vs. CoAP Message

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 4.3).
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 5: 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:

- The response codes the method returns in the success case.
- Whether the method is idempotent, safe, or both.
- Whether the request causes a cache to mark responses stored for the request URI as stale.

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]
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 6: 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:

- The methods the response code applies to.
- Whether payload is required, optional or not allowed.
- 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.
- 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.
- Whether the response is cacheable according to the freshness model.
- Whether the response is validatable according to the validation model.
- Whether the response causes a cache to mark responses stored for the request URI as stale.

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]
15	Uri-Query	[RFCXXXX]

Table 7: 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:

- The meaning of the option in a request.
- The meaning of the option in a response.
- Whether the option is critical of elective, as determined by the option number.
- The format and length of the option’s value.
- Whether the option must occur at most once or whether it can occur multiple times.
- 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 types means semantically.

Initial entries in this registry are as follows:

Media type	Id.	Reference
text/plain; charset=utf-8	0	
text/xml; charset=utf-8	1	
text/csv; charset=utf-8	2	
text/html; charset=utf-8	3	
application/link-format	40	[I-D.ietf-core-link-format]
application/xml	41	
application/octet-stream	42	
application/rdf+xml	43	
application/soap+xml	44	
application/atom+xml	45	
application/xmpp+xml	46	
application/exi	47	[EXIMIME]
application/fastinfoset	48	
application/soap+fastinfoset	49	
application/json	50	
application/x-obix-binary	51	[OBIX1.1]

Table 8: CoAP Media Types

The identifiers between 201 and 255 inclusive are reserved for Private Use. The identifiers between 256 and 65535 inclusive are Reserved for future use. All other identifiers are Unassigned.

Because the name space is so small, the IANA policy for future additions to this registry is "Expert Review" 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. 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. Correct examples from Table 8 include application/link-format, application/atom+xml and

application/x-obix-binary. For example, a Smart Energy application payload carried as XML would 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. 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.

This document requests the assignment of 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

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 Beroset, Gilman Tolle, Robby Simpson, Colin O'Flynn, Eric Rescorla, Matthieu Vial, Linyi Tian, Kerry Lynn, Dale Seed, Akbar Rahman 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

[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.

[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.

[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.

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

[RFC4835] Manral, V., "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 4835, April 2007.

[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.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 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.

[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] Shelby, Z. and C. Bormann, "Blockwise transfers in CoAP", draft-ietf-core-block-01 (work in progress), January 2011.

[I-D.ietf-core-link-format] Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-02 (work in progress), December 2010.

[I-D.ietf-tls-rfc4347-bis] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security version 1.2", draft-ietf-tls-rfc4347-bis-04 (work in progress), July 2010.

[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.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.

[I-D.mcgrew-tls-aes-ccm-ecc] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-

CCM ECC Cipher Suites for TLS",
draft-mcgrew-tls-aes-ccm-ecc-01 (work in progress),
January 2011.

[I-D.oflynn-core-bootstrapping]
Sarikaya, B., Ohba, Y., Cao, Z., and R. Cragie, "Security Bootstrapping of Resource-Constrained Devices", draft-oflynn-core-bootstrapping-03 (work in progress), November 2010.

[OBIX1.1] "OBIX Version 1.1", June 2010, <http://www.oasis-open.org/committees/download.php/38212/oBIX-1-1-spec-wd06.pdf>.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

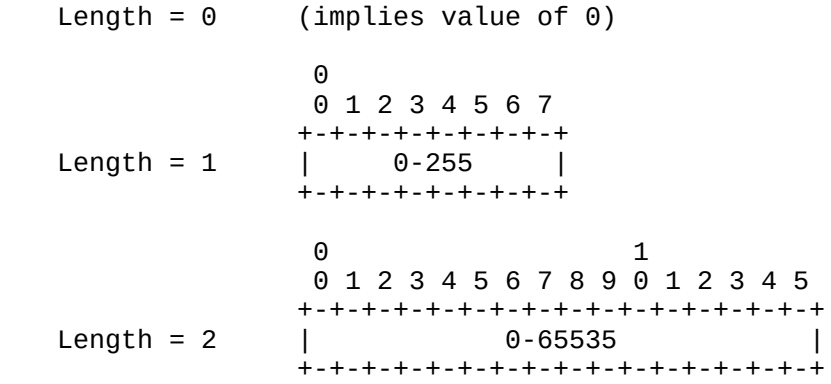
[RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.

[RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

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 in Figure 11.



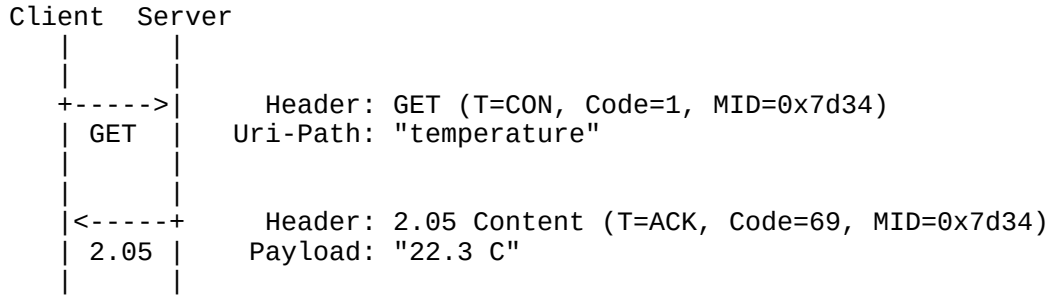
Length = 3 is 24 bits, Length = 4 is 32 bits etc.

Figure 11: Variable-length unsigned integer format

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 12 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.



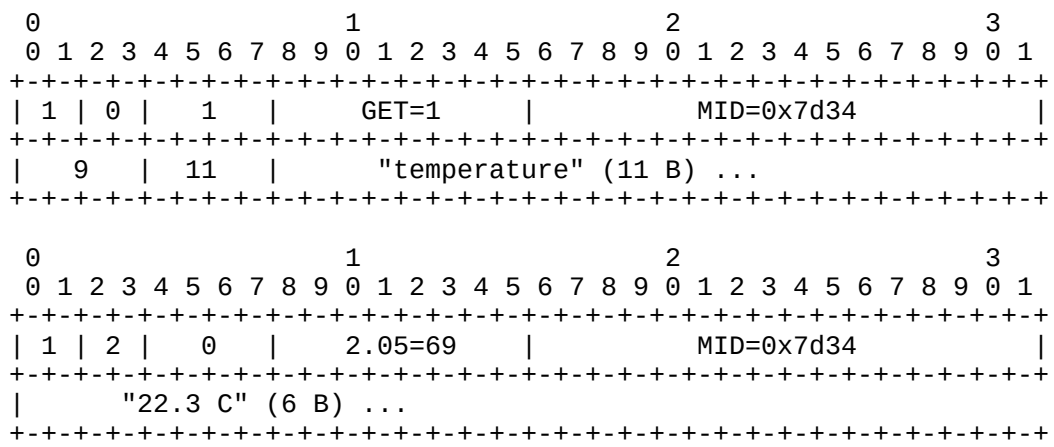


Figure 12: Confirmable request; piggy-backed response

Figure 13 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.

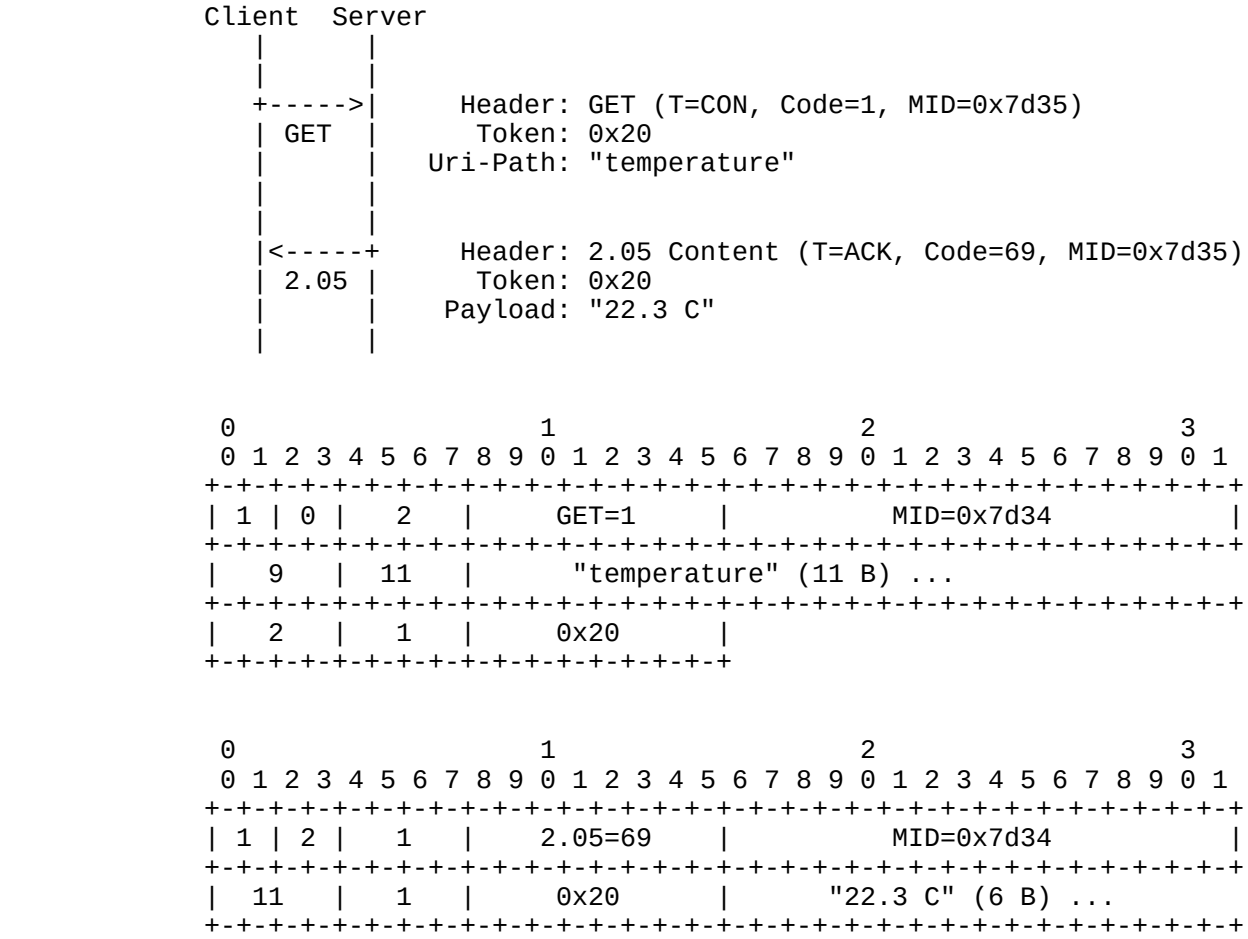


Figure 13: Confirmable request; piggy-backed response

In Figure 14, 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.

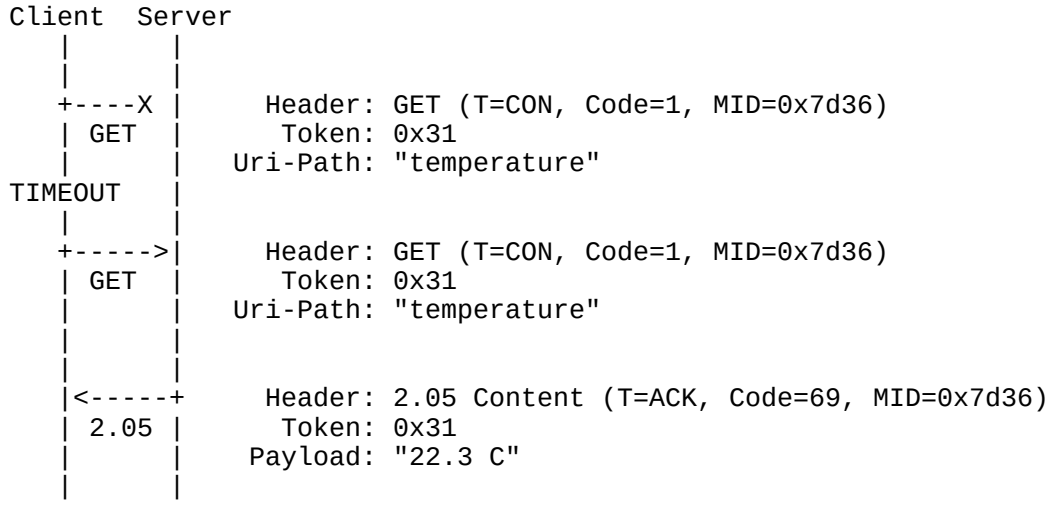


Figure 14: Confirmable request (retransmitted); piggy-backed response

In Figure 15, the first Acknowledgement message from the server to the client is lost. After RESPONSE_TIMEOUT seconds, the client retransmits the request.

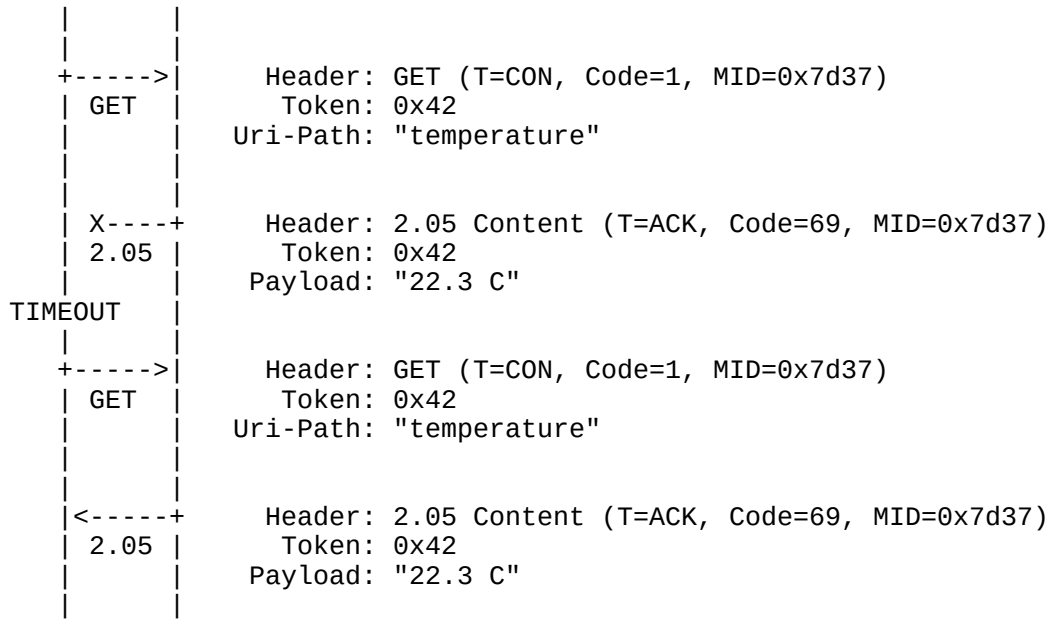


Figure 15: Confirmable request; piggy-backed response (retransmitted)

In Figure 16, 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.

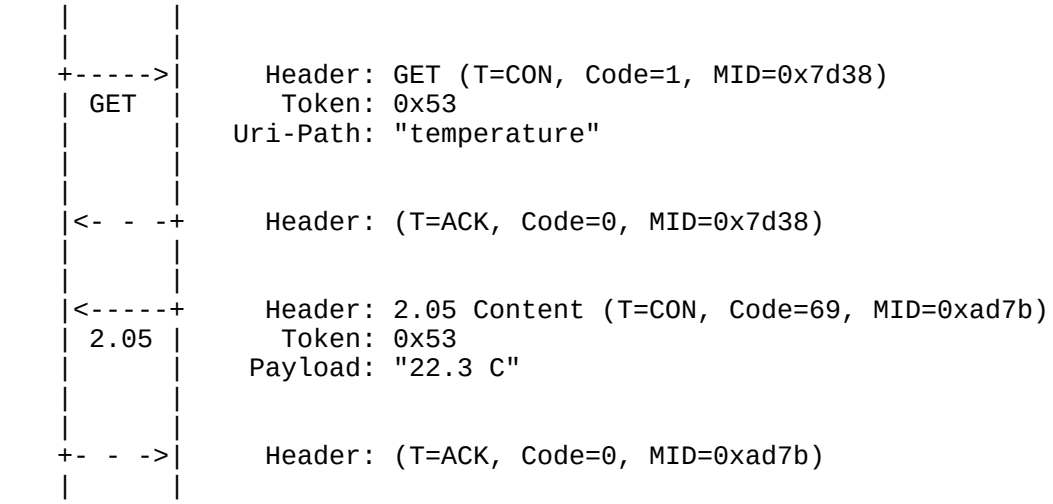


Figure 16: Confirmable request; separate response

Figure 17 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.

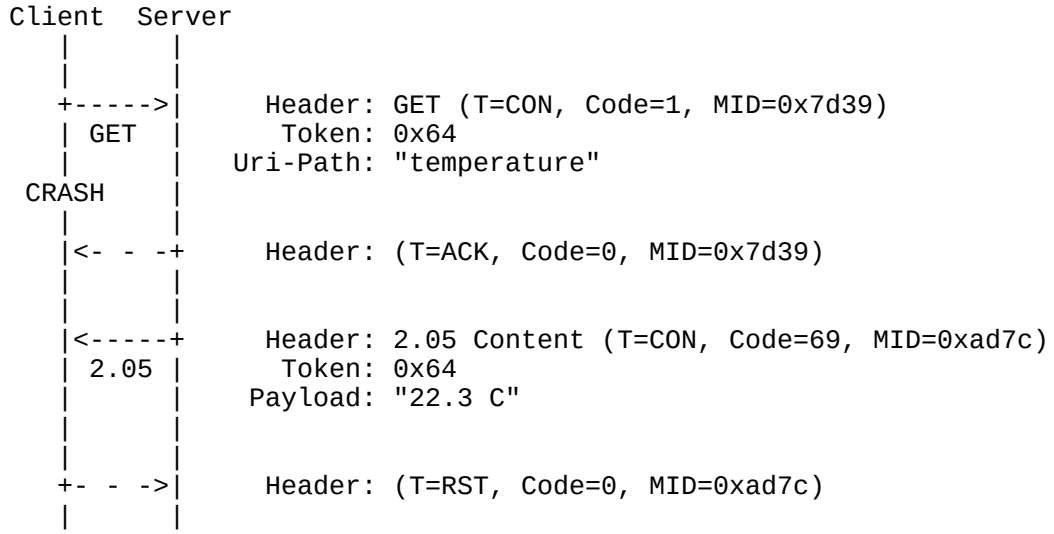


Figure 17: Confirmable request; separate response (unexpected)

Figure 18 shows a basic GET request where the request and the response are non-confirmable, so both may be lost without notice.

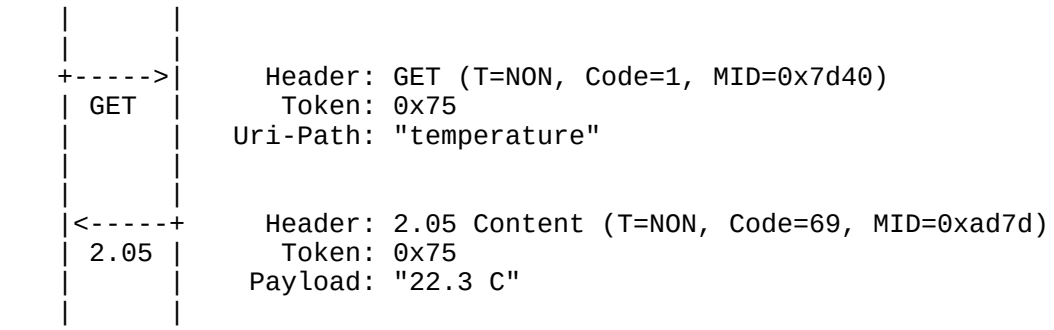


Figure 18: Non-confirmable request; Non-confirmable response

In Figure 19, 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.

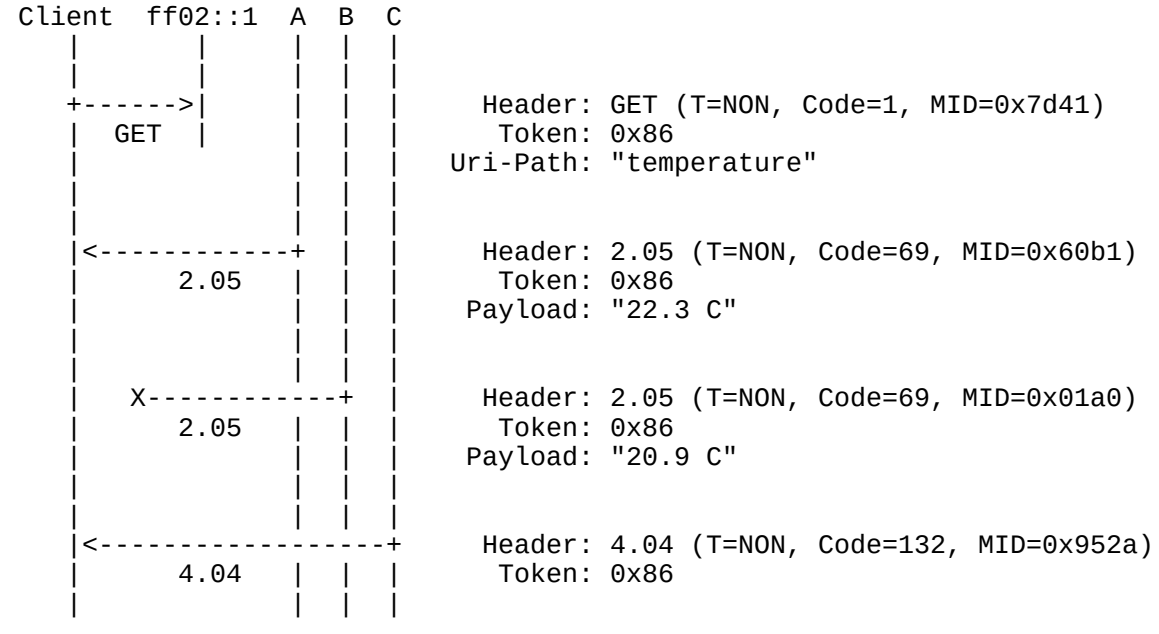


Figure 19: Non-confirmable request (multicast); Non-confirmable response

Appendix C. URI Examples

The following examples demonstrate different sets of Uri options, and the result after constructing an URI from them.

- coap://[2001:db8::2:1]/

  Destination IP Address = [2001:db8::2:1]

  Destination UDP Port = [IANA_TBD_PORT]

- coap://example.net/

  Destination IP Address = [2001:db8::2:1]

  Destination UDP Port = [IANA_TBD_PORT]

  Uri-Host = "example.net"

- coap://example.net/.well-known/core

Destination IP Address = [2001:db8::2:1]

Destination UDP Port = [IANA_TBD_PORT]

Uri-Host = "example.net"

Uri-Path = ".well-known"

Uri-Path = "core"

- 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 = [IANA_TBD_PORT]

  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

- coap://198.51.100.1:61616//%2F//?%2F%2F

  Destination IP Address = 198.51.100.1

  Destination UDP Port = 61616

  Uri-Path = ""

  Uri-Path = "/"

  Uri-Path = ""

  Uri-Path = ""

  Uri-Query = "%2F%2F"

- coap://[2001:db8::2:1]/sensors/temp

  Destination IP Address = [::1]

  Destination UDP Port = 61616

  Uri-Host = "[2001:db8::2:1]"

```
Uri-Port = [IANA_TBD_PORT]

Uri-Path = "sensors"

Uri-Path = "temp"
```

Appendix D. Changelog

Changed from ietf-04 to ietf-05:

- Renamed Immediate into Piggy-backed and Deferred into Separate -- should finally end the confusion on what this is about.
- GET requests now return a 2.05 (Content) response instead of 2.00 (OK) response (#104).
- Added text to allow 2.02 (Deleted) responses in reply to POST requests (#105).
- Improved message deduplication rules (#106).
- Section added on message size implementation considerations (#103).
- Clarification made on human readable error payloads (#109).
- Definition of CoAP methods improved (#108).
- Max-Age removed from requests (#107).
- Clarified uniqueness of tokens (#112).
- Location-Query Option added (#113).
- ETag length set to 1-8 bytes (#123).
- Clarified relation between elective/critical and option numbers (#110).
- Defined when to update Version header field (#111).
- URI scheme registration improved (#102).
- Added review guidelines for new CoAP codes and numbers.

Changes from ietf-03 to ietf-04:

- Major document reorganization (#51, #63, #71, #81).
- Max-age length set to 0-4 bytes (#30).
- Added variable unsigned integer definition (#31).
- Clarification made on human readable error payloads (#50).
- Definition of POST improved (#52).
- Token length changed to 0-8 bytes (#53).
- Section added on multiplexing CoAP, DTLS and STUN (#56).
- Added cross-protocol attack considerations (#61).
- Used new Immediate/Deferred response definitions (#73).
- Improved request/response matching rules (#74).
- Removed unnecessary media types and added recommendations for their use in M2M (#76).
- Response codes changed to base 32 coding, new Y.XX naming (#77).
- References updated as per AD review (#79).
- IANA section completed (#80).
- Proxy-Uri option added to diambiguate between proxy and non-proxy requests (#82).
- Added text on critical options in cached states (#83).
- HTTP mapping sections improved (#88).
- Added text on reverse proxies (#72).
- Some security text on multicast added (#54).
- Trust model text added to introduction (#58, #60).
- AES-CCM vs. AES-CCB text added (#55).
- Text added about device capabilities (#59).
- DTLS section improvements (#87).

- Caching semantics aligned with RFC2616 (#78).
- Uri-Path option split into multiple path segments.
- MAX_RETRANSMIT changed to 4 to adjust for RESPONSE_TIME = 2.

Changes from ietf-02 to ietf-03:

- Token Option and related use in asynchronous requests added (#25).
- CoAP specific error codes added (#26).
- Erroring out on unknown critical options changed to a MUST (#27).
- Uri-Query option added.
- Terminology and definitions of URIs improved.
- Security section completed (#22).

Changes from ietf-01 to ietf-02:

- Sending an error on a critical option clarified (#18).
- Clarification on behavior of PUT and idempotent operations (#19).
- Use of Uri-Authority clarified along with server processing rules; Uri-Scheme option removed (#20, #23).
- Resource discovery section removed to a separate CoRE Link Format draft (#21).
- Initial security section outline added.

Changes from ietf-00 to ietf-01:

- New cleaner transaction message model and header (#5).
- Removed subscription while being designed (#1).
- Section 2 re-written (#3).
- Text added about use of short URIs (#4).
- Improved header option scheme (#5, #14).
- Date option removed whiled being designed (#6).

- New text for CoAP default port (#7).
- Completed proxying section (#8).
- Completed resource discovery section (#9).
- Completed HTTP mapping section (#10).
- Several new examples added (#11).
- URI split into 3 options (#12).
- MIME type defined for link-format (#13, #16).
- New text on maximum message size (#15).
- Location Option added.

Changes from shelby-01 to ietf-00:

- Removed the TCP binding section, left open for the future.
- Fixed a bug in the example.
- Marked current Sub/Notify as (Experimental) while under WG discussion.
- Fixed maximum datagram size to 1280 for both IPv4 and IPv6 (for CoAP-CoAP proxying to work).
- Temporarily removed the Magic Byte header as TCP is no longer included as a binding.
- Removed the Uri-code Option as different URI encoding schemes are being discussed.
- Changed the rel= field to desc= for resource discovery.
- Changed the maximum message size to 1024 bytes to allow for IP/UDP headers.
- Made the URI slash optimization and method impotence MUSTs
- Minor editing and bug fixing.

Changes from shelby-00 to shelby-01:

- Unified the message header and added a notify message type.
- Renamed methods with HTTP names and removed the NOTIFY method.
- Added a number of options field to the header.
- Combines the Option Type and Length into an 8-bit field.
- Added the magic byte header.
- Added new ETag option.
- Added new Date option.
- Added new Subscription option.
- Completed the HTTP Code - CoAP Code mapping table appendix.
- Completed the Content-type Identifier appendix and tables.
- Added more simplifications for URI support.
- Initial subscription and discovery sections.
- 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: December 30, 2013

Z. Shelby
Sensinode
K. Hartke
C. Bormann
Universitaet Bremen TZI

June 28, 2013

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

Abstract

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

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

Status of This Memo

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

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

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

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

Table of Contents

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

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

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

Appendix C. Changelog 107
Authors' Addresses 117

1. Introduction

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

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

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

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

1.1. Features

CoAP has the following main features:

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

- Simple proxy and caching capabilities.

- 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.

- Security binding to Datagram Transport Layer Security (DTLS) [RFC6347].

1.2. Terminology

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2. Constrained Application Protocol

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

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

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

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

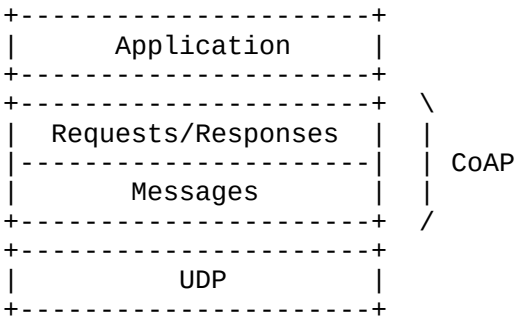


Figure 1: Abstract layering of CoAP

2.1. Messaging Model

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

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

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

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

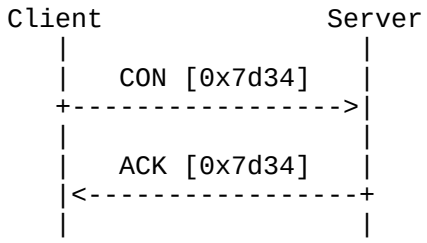


Figure 2: Reliable message transmission

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

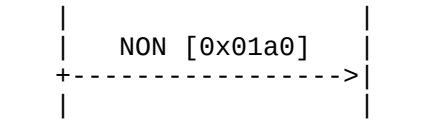


Figure 3: Unreliable message transmission

See Section 4 for details of CoAP messages.

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

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

2.2. Request/Response Model

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

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

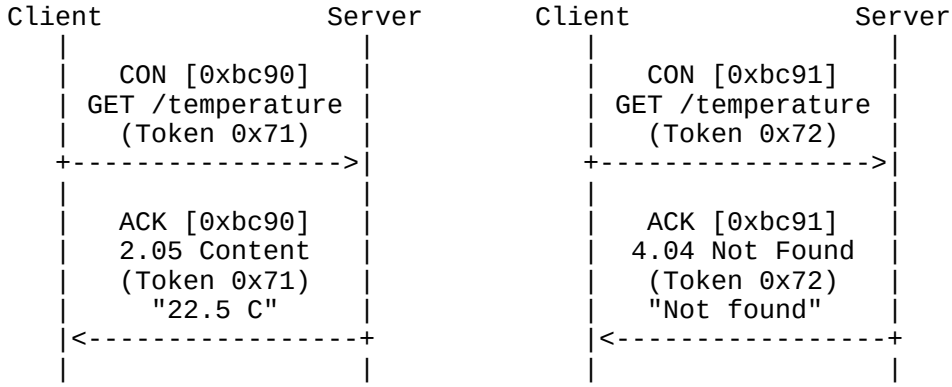


Figure 4: Two GET requests with piggy-backed responses

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

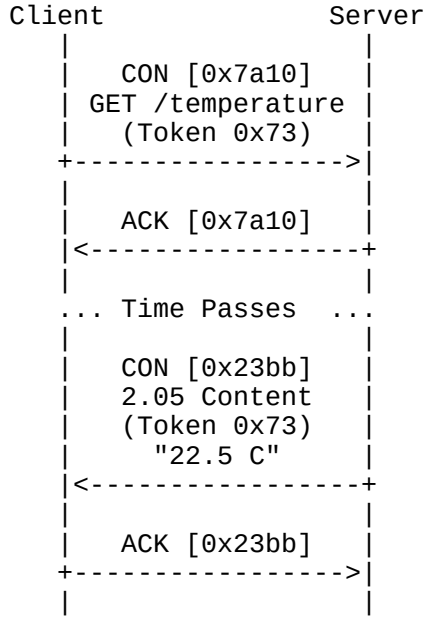
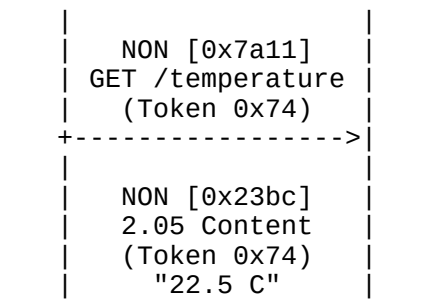


Figure 5: A GET request with a separate response

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



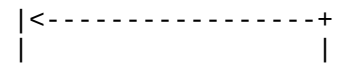


Figure 6: A NON request and response

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

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

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

2.3. Intermediaries and Caching

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

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

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

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

2.4. Resource Discovery

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

3. Message Format

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

CoAP messages are encoded in a simple binary format. The message format starts with a fixed-size 4-byte header. This is followed by a variable-length Token value which can be between 0 and 8 bytes long. Following the Token value comes a sequence of zero or more CoAP Options in Type-Length-Value (TLV) format, optionally followed by a payload which takes up the rest of the datagram.

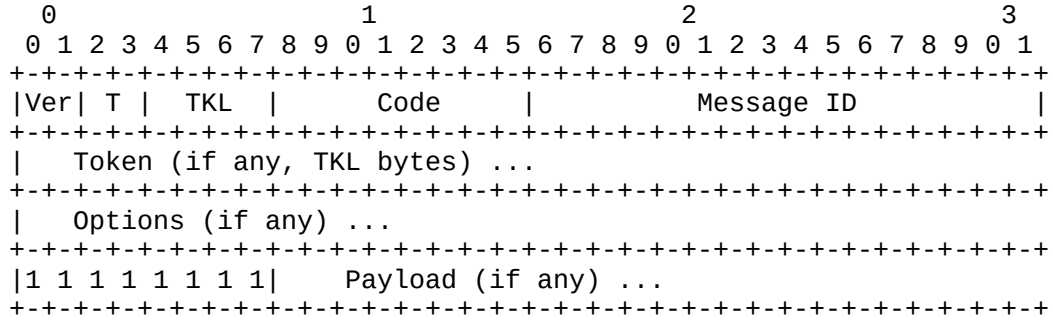


Figure 7: Message Format

The fields in the header are defined as follows:

Version (Ver): 2-bit unsigned integer. Indicates the CoAP version number. Implementations of this specification MUST set this field to 1 (01 binary). Other values are reserved for future versions. Messages with unknown version numbers MUST be silently ignored.

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

Token Length (TKL): 4-bit unsigned integer. Indicates the length of the variable-length Token field (0-8 bytes). Lengths 9-15 are reserved, MUST NOT be sent, and MUST be processed as a message format error.

Code: 8-bit unsigned integer, split into a 3-bit class (most significant bits) and a 5-bit detail (least significant bits), documented as c.dd where c is a digit from 0 to 7 for the 3-bit subfield and dd are two digits from 00 to 31 for the 5-bit subfield. The class can indicate a request (0), a success response (2), a client error response (4), or a server error response (5). (All other class values are reserved.) As a special case, Code 0.00 indicates an Empty message. In case of a request, the Code field indicates the Request Method; in case of a response a Response Code. Possible values are maintained in the CoAP Code Registries (Section 12.1). The semantics of requests and responses are defined in Section 5.

Message ID: 16-bit unsigned integer in network byte order. Used for the detection of message duplication, and to match messages of type Acknowledgement/Reset to messages of type Confirmable/Non-confirmable. The rules for generating a Message ID and matching messages are defined in Section 4.

The header is followed by the Token value, which may be 0 to 8 bytes, as given by the Token Length field. The Token value is used to correlate requests and responses. The rules for generating a Token and correlating requests and responses are defined in Section 5.3.1.

Header and Token are followed by zero or more Options (Section 3.1). An Option can be followed by the end of the message, by another Option, or by the Payload Marker and the payload.

Following the header, token, and options, if any, comes the optional payload. If present and of non-zero length, it is prefixed by a fixed, one-byte Payload Marker (0xFF) which indicates the end of options and the start of the payload. The payload data extends from after the marker to the end of the UDP datagram, i.e., the Payload Length is calculated from the datagram size. The absence of the Payload Marker denotes a zero-length payload. The presence of a marker followed by a zero-length payload MUST be processed as a message format error.

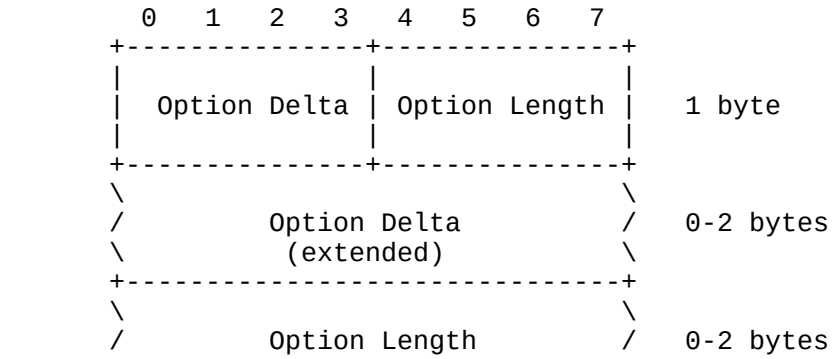
Implementation Note: The byte value 0xFF may also occur within an option length or value, so simple byte-wise scanning for 0xFF is not a viable technique for finding the payload marker. The byte 0xFF has the meaning of a payload marker only where the beginning of another option could occur.

3.1. Option Format

CoAP defines a number of options which can be included in a message. Each option instance in a message specifies the Option Number of the defined CoAP option, the length of the Option Value and the Option Value itself.

Instead of specifying the Option Number directly, the instances MUST appear in order of their Option Numbers and a delta encoding is used between them: The Option Number for each instance is calculated as the sum of its delta and the Option Number of the preceding instance in the message. For the first instance in a message, a preceding option instance with Option Number zero is assumed. Multiple instances of the same option can be included by using a delta of zero.

Option Numbers are maintained in the CoAP Option Number Registry (Section 12.2). See Section 5.4 for the semantics of the options defined in this document.



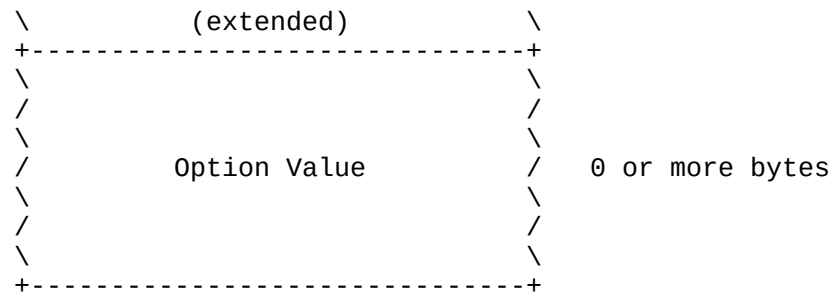


Figure 8: Option Format

The fields in an option are defined as follows:

Option Delta: 4-bit unsigned integer. A value between 0 and 12 indicates the Option Delta. Three values are reserved for special constructs:

13: An 8-bit unsigned integer follows the initial byte and indicates the Option Delta minus 13.

14: A 16-bit unsigned integer in network byte order follows the initial byte and indicates the Option Delta minus 269.

15: Reserved for the Payload Marker. If the field is set to this value but the entire byte is not the payload marker, this MUST be processed as a message format error.

The resulting Option Delta is used as the difference between the Option Number of this option and that of the previous option (or zero for the first option). In other words, the Option Number is calculated by simply summing the Option Delta values of this and all previous options before it.

Option Length: 4-bit unsigned integer. A value between 0 and 12 indicates the length of the Option Value, in bytes. Three values are reserved for special constructs:

13: An 8-bit unsigned integer precedes the Option Value and indicates the Option Length minus 13.

14: A 16-bit unsigned integer in network byte order precedes the Option Value and indicates the Option Length minus 269.

15: Reserved for future use. If the field is set to this value, it MUST be processed as a message format error.

Value: A sequence of exactly Option Length bytes. The length and format of the Option Value depend on the respective option, which MAY define variable length values. See Section 3.2 for the formats used in this document; options defined in other documents MAY make use of other option value formats.

3.2. Option Value Formats

The options defined in this document make use of the following option value formats.

empty: A zero-length sequence of bytes.

opaque: An opaque sequence of bytes.

uint: A non-negative integer which is represented in network byte order using the number of bytes given by the Option Length field.

An option definition may specify a range of permissible numbers of bytes; if it has a choice, a sender SHOULD represent the integer with as few bytes as possible, i.e., without leading zero bytes. For example, the number 0 is represented with an empty option value (a zero-length sequence of bytes), and the number 1 by a single byte with the numerical value of 1 (bit combination 00000001 in most significant bit first notation). A recipient MUST be prepared to process values with leading zero bytes.

Implementation Note: The exceptional behavior permitted for the sender is intended for highly constrained, templated implementations (e.g., hardware implementations) that use fixed size options in the templates.

string: A Unicode string which is encoded using UTF-8 [RFC3629] in Net-Unicode form [RFC5198].

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

4. Message Transmission

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

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

- Simple stop-and-wait retransmission reliability with exponential back-off for Confirmable messages.

- Duplicate detection for both Confirmable and Non-confirmable messages.

4.1. Messages and Endpoints

A CoAP endpoint is the source or destination of a CoAP message. The specific definition of an endpoint depends on the transport being used for CoAP. For the transports defined in this specification, the endpoint is identified depending on the security mode used (see Section 9): With no security, the endpoint is solely identified by an IP address and a UDP port number. With other security modes, the endpoint is identified as defined by the security mode.

There are different types of messages. The type of a message is specified by the Type field of the CoAP Header.

Separate from the message type, a message may carry a request, a response, or be Empty. This is signaled by the Request/Response Code field in the CoAP Header and is relevant to the request/response model. Possible values for the field are maintained in the CoAP Code Registries (Section 12.1).

An Empty message has the Code field set to 0.00. The Token Length field MUST be set to 0 and bytes of data MUST NOT be present after the Message ID field. If there are any bytes, they MUST be processed as a message format error.

4.2. Messages Transmitted Reliably

The reliable transmission of a message is initiated by marking the message as Confirmable in the CoAP header. A Confirmable message always carries either a request or response, unless it is used only to elicit a Reset message in which case it is Empty. A recipient

MUST acknowledge a Confirmable message with an Acknowledgement message or, if it lacks context to process the message properly (including the case where the message is Empty, uses a code with a reserved class (1, 6 or 7), or has a message format error), MUST reject it; rejecting a Confirmable message is effected by sending a matching Reset message and otherwise ignoring it. The Acknowledgement message MUST echo the Message ID of the Confirmable message, and MUST carry a response or be Empty (see Section 5.2.1 and Section 5.2.2). The Reset message MUST echo the Message ID of the Confirmable message, and MUST be Empty. Rejecting an Acknowledgement or Reset message (including the case where the Acknowledgement carries a request or a code with a reserved class, or the Reset message is not Empty) is effected by silently ignoring it. More generally, recipients of Acknowledgement and Reset messages MUST NOT respond with either Acknowledgement or Reset messages.

The sender retransmits the Confirmable message at exponentially increasing intervals, until it receives an acknowledgement (or Reset message), or runs out of attempts.

Retransmission is controlled by two things that a CoAP endpoint MUST keep track of for each Confirmable message it sends while waiting for an acknowledgement (or reset): a timeout and a retransmission counter. For a new Confirmable message, the initial timeout is set to a random duration (often not an integral number of seconds) between ACK_TIMEOUT and (ACK_TIMEOUT * ACK_RANDOM_FACTOR) (see Section 4.8), and the retransmission counter is set to 0. When the timeout is triggered and the retransmission counter is less than MAX_RETRANSMIT, the message is retransmitted, the retransmission counter is incremented, and the timeout is doubled. If the retransmission counter reaches MAX_RETRANSMIT on a timeout, or if the endpoint receives a Reset message, then the attempt to transmit the message is canceled and the application process informed of failure. On the other hand, if the endpoint receives an acknowledgement in time, transmission is considered successful.

This specification makes no strong requirements on the accuracy of the clocks used to implement the above binary exponential backoff algorithm. In particular, an endpoint may be late for a specific retransmission due to its sleep schedule, and maybe catch up on the next one. However, the minimum spacing before another retransmission is ACK_TIMEOUT, and the entire sequence of (re-)transmissions MUST stay in the envelope of MAX_TRANSMIT_SPAN (see Section 4.8.2), even if that means a sender may miss an opportunity to transmit.

A CoAP endpoint that sent a Confirmable message MAY give up in attempting to obtain an ACK even before the MAX_RETRANSMIT counter value is reached: E.g., the application has canceled the request as

it no longer needs a response, or there is some other indication that the CON message did arrive. In particular, a CoAP request message may have elicited a separate response, in which case it is clear to the requester that only the ACK was lost and a retransmission of the request would serve no purpose. However, a responder MUST NOT in turn rely on this cross-layer behavior from a requester, i.e. it MUST retain the state to create the ACK for the request, if needed, even if a Confirmable response was already acknowledged by the requester.

Another reason for giving up retransmission MAY be the receipt of ICMP errors. If it is desired to take account of ICMP errors, to mitigate potential spoofing attacks, implementations SHOULD take care to check the information about the original datagram in the ICMP message, including port numbers and CoAP header information such as message type and code, Message ID, and Token; if this is not possible due to limitations of the UDP service API, ICMP errors SHOULD be ignored. Packet Too Big errors [RFC4443] ("fragmentation needed and DF set" for IPv4 [RFC0792]) cannot properly occur and SHOULD be ignored if the implementation note in Section 4.6 is followed; otherwise, they SHOULD feed into a path MTU discovery algorithm [RFC4821]. Source Quench and Time Exceeded ICMP messages SHOULD be ignored. Host, network, port or protocol unreachable errors, or parameter problem errors MAY, after appropriate vetting, be used to inform the application of a failure in sending.

4.3. Messages Transmitted Without Reliability

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

As a more lightweight alternative, a message can be transmitted less reliably by marking the message as Non-confirmable. A Non-confirmable message always carries either a request or response and MUST NOT be Empty. A Non-confirmable message MUST NOT be acknowledged by the recipient. If a recipient lacks context to process the message properly (including the case where the message is Empty, uses a code with a reserved class (1, 6 or 7), or has a message format error), it MUST reject the message; rejecting a Non-confirmable message MAY involve sending a matching Reset message, and apart from the Reset message the rejected message MUST be silently ignored.

At the CoAP level, there is no way for the sender to detect if a Non-confirmable message was received or not. A sender MAY choose to transmit multiple copies of a Non-confirmable message within

MAX_TRANSMIT_SPAN (limited by the provisions of Section 4.7, in particular by PROBING_RATE if no response is received), or the network may duplicate the message in transit. To enable the receiver to act only once on the message, Non-confirmable messages specify a Message ID as well. (This Message ID is drawn from the same number space as the Message IDs for Confirmable messages.)

Summarizing Section 4.2 and Section 4.3, the four message types can be used as in Table 1. "*" means that the combination is not used in normal operation, but only to elicit a Reset message ("CoAP ping").

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

Table 1: Usage of message types

4.4. Message Correlation

An Acknowledgement or Reset message is related to a Confirmable message or Non-confirmable message by means of a Message ID along with additional address information of the corresponding endpoint. The Message ID is a 16-bit unsigned integer that is generated by the sender of a Confirmable or Non-confirmable message and included in the CoAP header. The Message ID MUST be echoed in the Acknowledgement or Reset message by the recipient.

The same Message ID MUST NOT be re-used (in communicating with the same endpoint) within the EXCHANGE_LIFETIME (Section 4.8.2).

Implementation Note: Several implementation strategies can be employed for generating Message IDs. In the simplest case a CoAP endpoint generates Message IDs by keeping a single Message ID variable, which is changed each time a new Confirmable or Non-confirmable message is sent regardless of the destination address or port. Endpoints dealing with large numbers of transactions could keep multiple Message ID variables, for example per prefix or destination address (note that some receiving endpoints may not be able to distinguish unicast and multicast packets addressed to it, so endpoints generating Message IDs need to make sure these do not overlap). It is strongly recommended that the initial value of the variable (e.g., on startup) be randomized, in order to make successful off-path attacks on the protocol less likely.

For an Acknowledgement or Reset message to match a Confirmable or Non-confirmable message, the Message ID and source endpoint of the Acknowledgement or Reset message MUST match the Message ID and destination endpoint of the Confirmable or Non-confirmable message.

4.5. Message Deduplication

A recipient might receive the same Confirmable message (as indicated by the Message ID and source endpoint) multiple times within the EXCHANGE_LIFETIME (Section 4.8.2), for example, when its Acknowledgement went missing or didn't reach the original sender before the first timeout. The recipient SHOULD acknowledge each duplicate copy of a Confirmable message using the same Acknowledgement or Reset message, but SHOULD process any request or response in the message only once. This rule MAY be relaxed in case the Confirmable message transports a request that is idempotent (see Section 5.1) or can be handled in an idempotent fashion. Examples for relaxed message deduplication:

- A server might relax the requirement to answer all retransmissions of an idempotent request with the same response (Section 4.2), so that it does not have to maintain state for Message IDs. For example, an implementation might want to process duplicate transmissions of a GET, PUT or DELETE request as separate requests if the effort incurred by duplicate processing is less expensive than keeping track of previous responses would be.

- A constrained server might even want to relax this requirement for certain non-idempotent requests if the application semantics make this trade-off favorable. For example, if the result of a POST request is just the creation of some short-lived state at the server, it may be less expensive to incur this effort multiple times for a request than keeping track of whether a previous transmission of the same request already was processed.

A recipient might receive the same Non-confirmable message (as indicated by the Message ID and source endpoint) multiple times within NON_LIFETIME (Section 4.8.2). As a general rule that MAY be relaxed based on the specific semantics of a message, the recipient SHOULD silently ignore any duplicated Non-confirmable message, and SHOULD process any request or response in the message only once.

4.6. Message Size

While specific link layers make it beneficial to keep CoAP messages small enough to fit into their link layer packets (see Section 1), this is a matter of implementation quality. The CoAP specification itself provides only an upper bound to the message size. Messages

larger than an IP packet result in undesirable packet fragmentation. A CoAP message, appropriately encapsulated, SHOULD fit within a single IP packet (i.e., avoid IP fragmentation) and (by fitting into one UDP payload) obviously needs to fit within a single IP datagram. If the Path MTU is not known for a destination, an IP MTU of 1280 bytes SHOULD be assumed; if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size.

Implementation Note: CoAP's choice of message size parameters works well with IPv6 and with most of today's IPv4 paths. (However, with IPv4, it is harder to absolutely ensure that there is no IP fragmentation. If IPv4 support on unusual networks is a consideration, implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes; worse, the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload. Implementations extremely focused on this problem set might also set the IPv4 DF bit and perform some form of path MTU discovery [RFC4821]; this should generally be unnecessary in most realistic use cases for CoAP, however.) A more important kind of fragmentation in many constrained networks is that on the adaptation layer (e.g., 6LoWPAN L2 packets are limited to 127 bytes including various overheads); this may motivate implementations to be frugal in their packet sizes and to move to block-wise transfers [I-D.ietf-core-block] when approaching three-digit message sizes.

Message sizes are also of considerable importance to implementations on constrained nodes. Many implementations will need to allocate a buffer for incoming messages. If an implementation is too constrained to allow for allocating the above-mentioned upper bound, it could apply the following implementation strategy for messages not using DTLS security: Implementations receiving a datagram into a buffer that is too small are usually able to determine if the trailing portion of a datagram was discarded and to retrieve the initial portion. So, if not all of the payload, at least the CoAP header and options are likely to fit within the buffer. A server can thus fully interpret a request and return a 4.13 (Request Entity Too Large, see Section 5.9.2.9) response code if the payload was truncated. A client sending an idempotent request and receiving a response larger than would fit in the buffer can repeat the request with a suitable value for the Block Option [I-D.ietf-core-block].

4.7. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.2.

In order not to cause congestion, Clients (including proxies) MUST strictly limit the number of simultaneous outstanding interactions that they maintain to a given server (including proxies) to NSTART. An outstanding interaction is either a CON for which an ACK has not yet been received but is still expected (message layer) or a request for which neither a response nor an Acknowledgment message has yet been received but is still expected (which may both occur at the same time, counting as one outstanding interaction). The default value of NSTART for this specification is 1.

Further congestion control optimizations and considerations are expected in the future, which may for example provide automatic initialization of the CoAP transmission parameters defined in Section 4.8, and thus may allow a value for NSTART greater than one.

A client stops expecting a response to a Confirmable request for which no acknowledgment message was received, after EXCHANGE_LIFETIME. The specific algorithm by which a client stops to "expect" a response to a Confirmable request that was acknowledged, or to a Non-confirmable request, is not defined. Unless this is modified by additional congestion control optimizations, it MUST be chosen in such a way that an endpoint does not exceed an average data rate of PROBING_RATE in sending to another endpoint that does not respond.

Note: CoAP places the onus of congestion control mostly on the clients. However, clients may malfunction or actually be attackers, e.g. to perform amplification attacks (Section 11.3). To limit the damage (to the network and to its own energy resources), a server SHOULD implement some rate limiting for its response transmission based on reasonable assumptions about application requirements. This is most helpful if the rate limit can be made effective for the misbehaving endpoints, only.

4.8. Transmission Parameters

Message transmission is controlled by the following parameters:

name	default value
ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1

| DEFAULT_LEISURE | 5 seconds |
| PROBING_RATE | 1 Byte/second |

Table 2: CoAP Protocol Parameters

4.8.1. Changing The Parameters

The values for ACK_TIMEOUT, ACK_RANDOM_FACTOR, MAX_RETRANSMIT, NSTART, DEFAULT_LEISURE (Section 8.2), and PROBING_RATE may be configured to values specific to the application environment (including dynamically adjusted values), however the configuration method is out of scope of this document. It is RECOMMENDED that an application environment use consistent values for these parameters; the specific effects of operating with inconsistent values in an application environment are outside the scope of the present specification.

The transmission parameters have been chosen to achieve a behavior in the presence of congestion that is safe in the Internet. If a configuration desires to use different values, the onus is on the configuration to ensure these congestion control properties are not violated. In particular, a decrease of ACK_TIMEOUT below 1 second would violate the guidelines of [RFC5405]. ([I-D.allman-tcpm-rto-consider] provides some additional background.) CoAP was designed to enable implementations that do not maintain round-trip-time (RTT) measurements. However, where it is desired to decrease the ACK_TIMEOUT significantly or increase NSTART, this can only be done safely when maintaining such measurements. Configurations MUST NOT decrease ACK_TIMEOUT or increase NSTART without using mechanisms that ensure congestion control safety, either defined in the configuration or in future standards documents.

ACK_RANDOM_FACTOR MUST NOT be decreased below 1.0, and it SHOULD have a value that is sufficiently different from 1.0 to provide some protection from synchronization effects.

MAX_RETRANSMIT can be freely adjusted, but a too small value will reduce the probability that a Confirmable message is actually received, while a larger value than given here will require further adjustments in the time values (see Section 4.8.2).

If the choice of transmission parameters leads to an increase of derived time values (see Section 4.8.2), the configuration mechanism MUST ensure the adjusted value is also available to all the endpoints that these adjusted values are to be used to communicate with.

4.8.2. Time Values derived from Transmission Parameters

The combination of ACK_TIMEOUT, ACK_RANDOM_FACTOR and MAX_RETRANSMIT influences the timing of retransmissions, which in turn influences how long certain information items need to be kept by an implementation. To be able to unambiguously reference these derived time values, we give them names as follows:

- MAX_TRANSMIT_SPAN is the maximum time from the first transmission of a Confirmable message to its last retransmission. For the default transmission parameters, the value is (2+4+8+16)*1.5 = 45 seconds, or more generally:

  ```
  ACK_TIMEOUT * ((2 ** MAX_RETRANSMIT) - 1) * ACK_RANDOM_FACTOR
  ```

- MAX_TRANSMIT_WAIT is the maximum time from the first transmission of a Confirmable message to the time when the sender gives up on receiving an acknowledgement or reset. For the default transmission parameters, the value is (2+4+8+16+32)*1.5 = 93 seconds, or more generally:

  ```
  ACK_TIMEOUT * ((2 ** (MAX_RETRANSMIT + 1)) - 1) *
  ACK_RANDOM_FACTOR
  ```

In addition, some assumptions need to be made on the characteristics of the network and the nodes.

- MAX_LATENCY is the maximum time a datagram is expected to take from the start of its transmission to the completion of its reception. This constant is related to the MSL (Maximum Segment Lifetime) of [RFC0793], which is "arbitrarily defined to be 2 minutes" ([RFC0793] glossary, page 81). Note that this is not necessarily smaller than MAX_TRANSMIT_WAIT, as MAX_LATENCY is not intended to describe a situation when the protocol works well, but the worst case situation against which the protocol has to guard. We, also arbitrarily, define MAX_LATENCY to be 100 seconds. Apart from being reasonably realistic for the bulk of configurations as well as close to the historic choice for TCP, this value also allows Message ID lifetime timers to be represented in 8 bits (when measured in seconds). In these calculations, there is no assumption that the direction of the transmission is irrelevant (i.e. that the network is symmetric), just that the same value can reasonably be used as a maximum value for both directions. If that is not the case, the following calculations become only slightly more complex.

- PROCESSING_DELAY is the time a node takes to turn around a Confirmable message into an acknowledgement. We assume the node

will attempt to send an ACK before having the sender time out, so as a conservative assumption we set it equal to ACK_TIMEOUT.

- MAX_RTT is the maximum round-trip time, or:

  (2 * MAX_LATENCY) + PROCESSING_DELAY

From these values, we can derive the following values relevant to the protocol operation:

- EXCHANGE_LIFETIME is the time from starting to send a Confirmable message to the time when an acknowledgement is no longer expected, i.e. message layer information about the message exchange can be purged. EXCHANGE_LIFETIME includes a MAX_TRANSMIT_SPAN, a MAX_LATENCY forward, PROCESSING_DELAY, and a MAX_LATENCY for the way back. Note that there is no need to consider MAX_TRANSMIT_WAIT if the configuration is chosen such that the last waiting period (ACK_TIMEOUT * (2 ** MAX_RETRANSMIT) or the difference between MAX_TRANSMIT_SPAN and MAX_TRANSMIT_WAIT) is less than MAX_LATENCY -- which is a likely choice, as MAX_LATENCY is a worst case value unlikely to be met in the real world. In this case, EXCHANGE_LIFETIME simplifies to:

  MAX_TRANSMIT_SPAN + (2 * MAX_LATENCY) + PROCESSING_DELAY

  or 247 seconds with the default transmission parameters.

- NON_LIFETIME is the time from sending a Non-confirmable message to the time its Message ID can be safely reused. If multiple transmission of a NON message is not used, its value is MAX_LATENCY, or 100 seconds. However, a CoAP sender might send a NON message multiple times, in particular for multicast applications. While the period of re-use is not bounded by the specification, an expectation of reliable detection of duplication at the receiver is in the timescales of MAX_TRANSMIT_SPAN. Therefore, for this purpose, it is safer to use the value:

  MAX_TRANSMIT_SPAN + MAX_LATENCY

  or 145 seconds with the default transmission parameters; however, an implementation that just wants to use a single timeout value for retiring Message IDs can safely use the larger value for EXCHANGE_LIFETIME.

Table 3 summarizes the derived parameters introduced in this subsection with their default values.

name	default value
MAX_TRANSMIT_SPAN	45 s
MAX_TRANSMIT_WAIT	93 s
MAX_LATENCY	100 s
PROCESSING_DELAY	2 s
MAX_RTT	202 s
EXCHANGE_LIFETIME	247 s
NON_LIFETIME	145 s

Table 3: Derived Protocol Parameters

5. Request/Response Semantics

CoAP operates under a similar request/response model as HTTP: a CoAP endpoint in the role of a "client" sends one or more CoAP requests to a "server", which services the requests by sending CoAP responses. Unlike HTTP, requests and responses are not sent over a previously established connection, but exchanged asynchronously over CoAP messages.

5.1. Requests

A CoAP request consists of the method to be applied to the resource, the identifier of the resource, a payload and Internet media type (if any), and optional meta-data about the request.

CoAP supports the basic methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP. They have the same properties of safe (only retrieval) and idempotent (you can invoke it multiple times with the same effects) as HTTP (see Section 9.1 of [RFC2616]). The GET method is safe, therefore it MUST NOT take any other action on a resource other than retrieval. The GET, PUT and DELETE methods MUST be performed in such a way that they are idempotent. POST is not idempotent, because its effect is determined by the origin server and dependent on the target resource; it usually results in a new resource being created or the target resource being updated.

A request is initiated by setting the Code field in the CoAP header of a Confirmable or a Non-confirmable message to a Method Code and including request information.

The methods used in requests are described in detail in Section 5.8.

5.2. Responses

After receiving and interpreting a request, a server responds with a CoAP response, which is matched to the request by means of a client-generated token (Section 5.3, note that this is different from the Message ID that matches a Confirmable message to its Acknowledgement).

A response is identified by the Code field in the CoAP header being set to a Response Code. Similar to the HTTP Status Code, the CoAP Response Code indicates the result of the attempt to understand and satisfy the request. These codes are fully defined in Section 5.9. The Response Code numbers to be set in the Code field of the CoAP header are maintained in the CoAP Response Code Registry (Section 12.1.2).

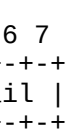


Figure 9: Structure of a Response Code

The upper three bits of the 8-bit Response Code number define the class of response. The lower five bits do not have any categorization role; they give additional detail to the overall class (Figure 9).

As a human readable notation for specifications and protocol diagnostics, CoAP code numbers including the response code are documented in the format "c.dd", where "c" is the class in decimal, and "dd" is the detail as a two-digit decimal. For example, "Forbidden" is written as 4.03 -- indicating an 8-bit code value of hexadecimal 0x83 (4*0x20+3) or decimal 131 (4*32+3).

There are 3 classes of response codes:

2 - Success: The request was successfully received, understood, and accepted.

4 - Client Error: The request contains bad syntax or cannot be fulfilled.

5 - Server Error: The server failed to fulfill an apparently valid request.

The response codes are designed to be extensible: Response Codes in the Client Error and Server Error class that are unrecognized by an endpoint are treated as being equivalent to the generic Response Code

of that class (4.00 and 5.00, respectively). However, there is no generic Response Code indicating success, so a Response Code in the Success class that is unrecognized by an endpoint can only be used to determine that the request was successful without any further details.

The possible response codes are described in detail in Section 5.9.

Responses can be sent in multiple ways, which are defined in the following subsections.

5.2.1. Piggy-backed

In the most basic case, the response is carried directly in the Acknowledgement message that acknowledges the request (which requires that the request was carried in a Confirmable message). This is called a "Piggy-backed" Response.

The response is returned in the Acknowledgement message independent of whether the response indicates success or failure. In effect, the response is piggy-backed on the Acknowledgement message, and no separate message is required to return the response.

Implementation Note: The protocol leaves the decision whether to piggy-back a response or not (i.e., send a separate response) to the server. The client MUST be prepared to receive either. On the quality of implementation level, there is a strong expectation that servers will implement code to piggy-back whenever possible -- saving resources in the network and both at the client and at the server.

5.2.2. Separate

It may not be possible to return a piggy-backed response in all cases. For example, a server might need longer to obtain the representation of the resource requested than it can wait sending back the Acknowledgement message, without risking the client to repeatedly retransmit the request message (see also the discussion of PROCESSING_DELAY in Section 4.8.2). The Response to a request carried in a Non-confirmable message is always sent separately (as there is no Acknowledgement message).

One way to implement this in a server is to initiate the attempt to obtain the resource representation and, while that is in progress, time out an acknowledgement timer. A server may also immediately send an acknowledgement knowing in advance that there will be no piggy-backed response. In both cases, the acknowledgement effectively is a promise that the request will be acted upon later.

When the server finally has obtained the resource representation, it sends the response. When it is desired that this message is not lost, it is sent as a Confirmable message from the server to the client and answered by the client with an Acknowledgement, echoing the new Message ID chosen by the server. (It may also be sent as a Non-confirmable message; see Section 5.2.3.)

When the server chooses to use a separate response, it sends the Acknowledgement to the Confirmable request as an Empty message. Once the server sends back an Empty Acknowledgement, it MUST NOT send back the response in another Acknowledgement, even if the client retransmits another identical request. If a retransmitted request is received (perhaps because the original Acknowledgement was delayed), another Empty Acknowledgement is sent and any response MUST be sent as a separate response.

If the server then sends a Confirmable response, the client's Acknowledgement to that response MUST also be an Empty message (one that carries neither a request nor a response). The server MUST stop retransmitting its response on any matching Acknowledgement (silently ignoring any response code or payload) or Reset message.

Implementation Notes: Note that, as the underlying datagram transport may not be sequence-preserving, the Confirmable message carrying the response may actually arrive before or after the Acknowledgement message for the request; for the purposes of terminating the retransmission sequence, this also serves as an acknowledgement. Note also that, while the CoAP protocol itself does not make any specific demands here, there is an expectation that the response will come within a time frame that is reasonable from an application point of view; as there is no underlying transport protocol that could be instructed to run a keep-alive mechanism, the requester may want to set up a timeout that is unrelated to CoAP's retransmission timers in case the server is destroyed or otherwise unable to send the response.)

5.2.3. Non-confirmable

If the request message is Non-confirmable, then the response SHOULD be returned in a Non-confirmable message as well. However, an endpoint MUST be prepared to receive a Non-confirmable response (preceded or followed by an Empty Acknowledgement message) in reply to a Confirmable request, or a Confirmable response in reply to a Non-confirmable request.

5.3. Request/Response Matching

Regardless of how a response is sent, it is matched to the request by means of a token that is included by the client in the request, along with additional address information of the corresponding endpoint.

5.3.1. Token

The Token is used to match a response with a request. The token value is a sequence of 0 to 8 bytes. (Note that every message carries a token, even if it is of zero length.) Every request carries a client-generated token, which the server MUST echo in any resulting response without modification.

A token is intended for use as a client-local identifier for differentiating between concurrent requests (see Section 5.3); it could have been called a "request ID".

The client SHOULD generate tokens in such a way that tokens currently in use for a given source/destination endpoint pair are unique. (Note that a client implementation can use the same token for any request if it uses a different endpoint each time, e.g. a different source port number.) An empty token value is appropriate e.g. when no other tokens are in use to a destination, or when requests are made serially per destination and receive piggy-backed responses. There are however multiple possible implementation strategies to fulfill this.

A client sending a request without using transport layer security (Section 9) SHOULD use a non-trivial, randomized token to guard against spoofing of responses (Section 11.4). This protective use of tokens is the reason they are allowed to be up to 8 bytes in size. The actual size of the random component to be used for the Token depends on the security requirements of the client and the level of threat posed by spoofing of responses. A client that is connected to the general Internet SHOULD use at least 32 bits of randomness; keeping in mind that not being directly connected to the Internet is not necessarily sufficient protection against spoofing. (Note that the Message ID adds little in protection as it is usually sequentially assigned, i.e. guessable, and can be circumvented by spoofing a separate response.) Clients that want to optimize the Token length may further want to detect the level of ongoing attacks (e.g., by tallying recent Token mismatches in incoming messages) and adjust the Token length upwards appropriately. [RFC4086] discusses randomness requirements for security.

An endpoint receiving a token it did not generate MUST treat it as opaque and make no assumptions about its content or structure.

5.3.2. Request/Response Matching Rules

The exact rules for matching a response to a request are as follows:

1. The source endpoint of the response MUST be the same as the destination endpoint of the original request.

2. In a piggy-backed response, both the Message ID of the Confirmable request and the Acknowledgement, and the token of the response and original request MUST match. In a separate response, just the token of the response and original request MUST match.

In case a message carrying a response is unexpected (the client is not waiting for a response from the identified endpoint, at the endpoint addressed, and/or with the given token), the response is rejected (Section 4.2, Section 4.3).

Implementation Note: A client that receives a response in a CON message may want to clean up the message state right after sending the ACK. If that ACK is lost and the server retransmits the CON, the client may no longer have any state to correlate this response to, making the retransmission an unexpected message; the client will likely send a Reset message so it does not receive any more retransmissions. This behavior is normal and not an indication of an error. (Clients that are not aggressively optimized in their state memory usage will still have message state that will identify the second CON as a retransmission. Clients that actually expect more messages from the server [I-D.ietf-core-observe] will have to keep state in any case.)

5.4. Options

Both requests and responses may include a list of one or more options. For example, the URI in a request is transported in several options, and meta-data that would be carried in an HTTP header in HTTP is supplied as options as well.

CoAP defines a single set of options that are used in both requests and responses:

- Content-Format
- ETag
- Location-Path
- Location-Query

- Max-Age
- Proxy-Uri
- Proxy-Scheme
- Uri-Host
- Uri-Path
- Uri-Port
- Uri-Query
- Accept
- If-Match
- If-None-Match
- Size1

The semantics of these options along with their properties are defined in detail in Section 5.10.

Not all options are defined for use with all methods and response codes. The possible options for methods and response codes are defined in Section 5.8 and Section 5.9 respectively. In case an option is not defined for a method or response code, it MUST NOT be included by a sender and MUST be treated like an unrecognized option by a recipient.

5.4.1. Critical/Elective

Options fall into one of two classes: "critical" or "elective". The difference between these is how an option unrecognized by an endpoint is handled:

- Upon reception, unrecognized options of class "elective" MUST be silently ignored.
- Unrecognized options of class "critical" that occur in a Confirmable request MUST cause the return of a 4.02 (Bad Option) response. This response SHOULD include a diagnostic payload describing the unrecognized option(s) (see Section 5.5.2).

- Unrecognized options of class "critical" that occur in a Confirmable response, or piggy-backed in an Acknowledgement, MUST cause the response to be rejected (Section 4.2).

- Unrecognized options of class "critical" that occur in a Non-confirmable message MUST cause the message to be rejected (Section 4.3).

Note that, whether critical or elective, an option is never "mandatory" (it is always optional): These rules are defined in order to enable implementations to stop processing options they do not understand or implement.

Critical/Elective rules apply to non-proxying endpoints. A proxy processes options based on Unsafe/Safe-to-Forward classes as defined in Section 5.7.

5.4.2. Proxy Unsafe/Safe-to-Forward and NoCacheKey

In addition to an option being marked as Critical or Elective, options are also classified based on how a proxy is to deal with the option if it does not recognize it. For this purpose, an option can either be considered Unsafe to Forward (UnSafe is set) or Safe-to-Forward (UnSafe is clear).

In addition, for an option that is marked Safe-to-Forward, the option number indicates whether it is intended to be part of the Cache-Key (Section 5.6) in a request or not; if some of the NoCacheKey bits are 0, it is, if all NoCacheKey bits are 1, it is not (see Section 5.4.6).

Note: The Cache-Key indication is relevant only for proxies that do not implement the given option as a request option and instead rely on the Unsafe/Safe-to-Forward indication only. E.g., for ETag, actually using the request option as a part of the Cache-Key is grossly inefficient, but it is the best thing one can do if ETag is not implemented by a proxy, as the response is going to differ based on the presence of the request option. A more useful proxy that does implement the ETag request option is not using ETag as a part of the Cache-Key.

NoCacheKey is indicated in three bits so that only one out of eight codepoints is qualified as NoCacheKey, assuming this is the less likely case.

Proxy behavior with regard to these classes is defined in Section 5.7.

5.4.3. Length

Option values are defined to have a specific length, often in the form of an upper and lower bound. If the length of an option value in a request is outside the defined range, that option MUST be treated like an unrecognized option (see Section 5.4.1).

5.4.4. Default Values

Options may be defined to have a default value. If the value of option is intended to be this default value, the option SHOULD NOT be included in the message. If the option is not present, the default value MUST be assumed.

Where a critical option has a default value, this is chosen in such a way that the absence of the option in a message can be processed properly both by implementations unaware of the critical option and by implementations that interpret this absence as the presence of the default value for the option.

5.4.5. Repeatable Options

The definition of some options specifies that those options are repeatable. An option that is repeatable MAY be included one or more times in a message. An option that is not repeatable MUST NOT be included more than once in a message.

If a message includes an option with more occurrences than the option is defined for, each supernumerary option occurrence that appears subsequently in the message MUST be treated like an unrecognized option (see Section 5.4.1).

5.4.6. Option Numbers

An Option is identified by an option number, which also provides some additional semantics information: e.g., odd numbers indicate a critical option, while even numbers indicate an elective option. Note that this is not just a convention, it is a feature of the protocol: Whether an option is elective or critical is entirely determined by whether its option number is even or odd.

More generally speaking, an Option number is constructed with a bit mask to indicate if an option is Critical/Elective, Unsafe/Safe-to-Forward and in the case of Safe-to-Forward, also a Cache-Key indication as shown by the following figure. In the following text, the bit mask is expressed as a single byte that is applied to the least significant byte of the option number in unsigned integer representation. When bit 7 (the least significant bit) is 1, an

option is Critical (and likewise Elective when 0). When bit 6 is 1, an option is Unsafe (and likewise Safe-to-Forward when 0). When bit 6 is 0, i.e., the option is not Unsafe, it is not a Cache-Key (NoCacheKey) if and only if bits 3-5 are all set to 1; all other bit combinations mean that it indeed is a Cache-Key. These classes of options are explained in the next sections.

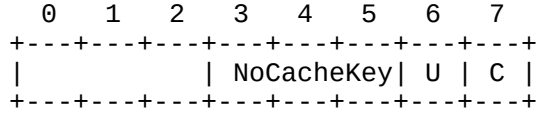


Figure 10: Option Number Mask (Least Significant Byte)

An endpoint may use an equivalent of the C code in Figure 11 to derive the characteristics of an option number "onum".

```
Critical = (onum & 1);
UnSafe = (onum & 2);
NoCacheKey = ((onum & 0x1e) == 0x1c);
```

Figure 11: Determining Characteristics from an Option Number

The option numbers for the options defined in this document are listed in the CoAP Option Number Registry (Section 12.2).

5.5. Payloads and Representations

Both requests and responses may include a payload, depending on the method or response code respectively. If a method or response code is not defined to have a payload, then a sender MUST NOT include one, and a recipient MUST ignore it.

5.5.1. Representation

The payload of requests or of responses indicating success is typically a representation of a resource ("resource representation") or the result of the requested action ("action result"). Its format is specified by the Internet media type and content coding given by the Content-Format Option. In the absence of this option, no default value is assumed and the format will need to be inferred by the application (e.g., from the application context). Payload "sniffing" SHOULD only be attempted if no content type is given.

Implementation Note: On a quality of implementation level, there is a strong expectation that a Content-Format indication will be provided with resource representations whenever possible. This is not a "SHOULD"-level requirement solely because it is not a

protocol requirement, and it also would be difficult to outline exactly in what cases this expectation can be violated.

For responses indicating a client or server error, the payload is considered a representation of the result of the requested action only if a Content-Format Option is given. In the absence of this option, the payload is a Diagnostic Payload (Section 5.5.2).

5.5.2. Diagnostic Payload

If no Content-Format option is given, the payload of responses indicating a client or server error is a brief human-readable diagnostic message, explaining the error situation. This diagnostic message MUST be encoded using UTF-8 [RFC3629], more specifically using Net-Unicode form [RFC5198].

The message is similar to the Reason-Phrase on an HTTP status line. It is not intended for end-users but for software engineers that during debugging need to interpret it in the context of the present, English-language specification; therefore no mechanism for language tagging is needed or provided. In contrast to what is usual in HTTP, the payload SHOULD be empty if there is no additional information beyond the response code.

5.5.3. Selected Representation

Not all responses carry a payload that provides a representation of the resource addressed by the request. It is, however, sometimes useful to be able to refer to such a representation in relation to a response, independent of whether it actually was enclosed.

We use the term "selected representation" to refer to the current representation of a target resource that would have been selected in a successful response if the corresponding request had used the method GET and excluded any conditional request options (Section 5.10.8).

Certain response options provide metadata about the selected representation, which might differ from the representation included in the message for responses to some state-changing methods. Of the response options defined in this specification, only the ETag response option (Section 5.10.6) is defined as selected representation metadata.

5.5.4. Content Negotiation

A server may be able to supply a representation for a resource in one of multiple representation formats. Without further information from

the client, it will provide the representation in the format it prefers.

By using the Accept Option (Section 5.10.4) in a request, the client can indicate which content-format it prefers to receive.

5.6. Caching

CoAP endpoints MAY cache responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests.

The goal of caching in CoAP is to reuse a prior response message to satisfy a current request. In some cases, a stored response can be reused without the need for a network request, reducing latency and network round-trips; a "freshness" mechanism is used for this purpose (see Section 5.6.1). Even when a new request is required, it is often possible to reuse the payload of a prior response to satisfy the request, thereby reducing network bandwidth usage; a "validation" mechanism is used for this purpose (see Section 5.6.2).

Unlike HTTP, the cacheability of CoAP responses does not depend on the request method, but the Response Code. The cacheability of each Response Code is defined along the Response Code definitions in Section 5.9. Response Codes that indicate success and are unrecognized by an endpoint MUST NOT be cached.

For a presented request, a CoAP endpoint MUST NOT use a stored response, unless:

- the presented request method and that used to obtain the stored response match,
- all options match between those in the presented request and those of the request used to obtain the stored response (which includes the request URI), except that there is no need for a match of any request options marked as NoCacheKey (Section 5.4) or recognized by the Cache and fully interpreted with respect to its specified cache behavior (such as the ETag request option, Section 5.10.6, see also Section 5.4.2), and
- the stored response is either fresh or successfully validated as defined below.

The set of request options that is used for matching the cache entry is also collectively referred to as the "Cache-Key". For URI schemes other than coap and coaps, matching of those options that constitute the request URI may be performed under rules specific to the URI scheme.

5.6.1. Freshness Model

When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.

The mechanism for determining freshness is for an origin server to provide an explicit expiration time in the future, using the Max-Age Option (see Section 5.10.5). The Max-Age Option indicates that the response is to be considered not fresh after its age is greater than the specified number of seconds.

The Max-Age Option defaults to a value of 60. Thus, if it is not present in a cacheable response, then the response is considered not fresh after its age is greater than 60 seconds. If an origin server wishes to prevent caching, it MUST explicitly include a Max-Age Option with a value of zero seconds.

If a client has a fresh stored response and makes a new request matching the request for that stored response, the new response invalidates the old response.

5.6.2. Validation Model

When an endpoint has one or more stored responses for a GET request, but cannot use any of them (e.g., because they are not fresh), it can use the ETag Option (Section 5.10.6) in the GET request to give the origin server an opportunity to both select a stored response to be used, and to update its freshness. This process is known as "validating" or "revalidating" the stored response.

When sending such a request, the endpoint SHOULD add an ETag Option specifying the entity-tag of each stored response that is applicable.

A 2.03 (Valid) response indicates the stored response identified by the entity-tag given in the response’s ETag Option can be reused, after updating it as described in Section 5.9.1.3.

Any other response code indicates that none of the stored responses nominated in the request is suitable. Instead, the response SHOULD be used to satisfy the request and MAY replace the stored response.

5.7. Proxying

A proxy is a CoAP endpoint that can be tasked by CoAP clients to perform requests on their behalf. This may be useful, for example, when the request could otherwise not be made, or to service the response from a cache in order to reduce response time and network bandwidth or energy consumption.

In an overall architecture for a Constrained RESTful Environment, proxies can serve quite different purposes. Proxies can be explicitly selected by clients, a role that we term "forward-proxy". Proxies can also be inserted to stand in for origin servers, a role that we term "reverse-proxy". Orthogonal to this distinction, a proxy can map from a CoAP request to a CoAP request (CoAP-to-CoAP proxy) or translate from or to a different protocol ("cross-proxy"). Full definitions of these terms are provided in Section 1.2.

Notes: The terminology in this specification has been selected to be culturally compatible with the terminology used in the wider Web application environments, without necessarily matching it in every detail (which may not even be relevant to Constrained RESTful Environments). Not too much semantics should be ascribed to the components of the terms (such as "forward", "reverse", or "cross").

HTTP proxies, besides acting as HTTP proxies, often offer a transport protocol proxying function ("CONNECT") to enable end-to-end transport layer security through the proxy. No such function is defined for CoAP-to-CoAP proxies in this specification, as forwarding of UDP packets is unlikely to be of much value in Constrained RESTful environments. See also Section 10.2.7 for the cross-proxy case.

When a client uses a proxy to make a request that will use a secure URI scheme (e.g., coaps or https), the request towards the proxy SHOULD be sent using DTLS security except where equivalent lower layer security is used for the leg between the client and the proxy.

5.7.1. Proxy Operation

A proxy generally needs a way to determine potential request parameters for a request to a destination based on the request it received. This way is fully specified for a forward-proxy, but may depend on the specific configuration for a reverse-proxy. In particular, the client of a reverse-proxy generally does not indicate a locator for the destination, necessitating some form of namespace translation in the reverse-proxy. However, some aspects of the operation of proxies are common to all its forms.

If a proxy does not employ a cache, then it simply forwards the translated request to the determined destination. Otherwise, if it does employ a cache but does not have a stored response that matches the translated request and is considered fresh, then it needs to refresh its cache according to Section 5.6. For options in the request that the proxy recognizes, it knows whether the option is intended to act as part of the key used in looking up the cached value or not. E.g., since requests for different Uri-Path values address different resources, Uri-Path values are always part of the Cache-Key, while, e.g., Token values are never part of the Cache-Key. For options that the proxy does not recognize but that are marked Safe-to-Forward in the option number, the option also indicates whether it is to be included in the Cache-Key (NoCacheKey is not all set) or not (NoCacheKey is all set). (Options that are unrecognized and marked Unsafe lead to 4.02 Bad Option.)

If the request to the destination times out, then a 5.04 (Gateway Timeout) response MUST be returned. If the request to the destination returns a response that cannot be processed by the proxy (e.g, due to unrecognized critical options, message format errors), then a 5.02 (Bad Gateway) response MUST be returned. Otherwise, the proxy returns the response to the client.

If a response is generated out of a cache, the generated (or implied) Max-Age Option MUST NOT extend the max-age originally set by the server, considering the time the resource representation spent in the cache. E.g., the Max-Age Option could be adjusted by the proxy for each response using the formula:

```
proxy-max-age = original-max-age - cache-age
```

For example if a request is made to a proxied resource that was refreshed 20 seconds ago and had an original Max-Age of 60 seconds, then that resource’s proxied max-age is now 40 seconds. Considering potential network delays on the way from the origin server, a proxy should be conservative in the max-age values offered.

All options present in a proxy request MUST be processed at the proxy. Unsafe options in a request that are not recognized by the proxy MUST lead to a 4.02 (Bad Option) response being returned by the proxy. A CoAP-to-CoAP proxy MUST forward to the origin server all Safe-to-Forward options that it does not recognize. Similarly, Unsafe options in a response that are not recognized by the CoAP-to-CoAP proxy server MUST lead to a 5.02 (Bad Gateway) response. Again, Safe-to-Forward options that are not recognized MUST be forwarded.

Additional considerations for cross-protocol proxying between CoAP and HTTP are discussed in Section 10.

5.7.2. Forward-Proxies

CoAP distinguishes between requests made (as if) to an origin server and a request made through a forward-proxy. CoAP requests to a forward-proxy are made as normal Confirmable or Non-confirmable requests to the forward-proxy endpoint, but specify the request URI in a different way: The request URI in a proxy request is specified as a string in the Proxy-Uri Option (see Section 5.10.2), while the request URI in a request to an origin server is split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options (see Section 5.10.1); alternatively the URI in a proxy request can be assembled from a Proxy-Scheme option and the split options mentioned.

When a proxy request is made to an endpoint and the endpoint is unwilling or unable to act as proxy for the request URI, it MUST return a 5.05 (Proxying Not Supported) response. If the authority (host and port) is recognized as identifying the proxy endpoint itself (see Section 5.10.2), then the request MUST be treated as a local (non-proxied) request.

Unless a proxy is configured to forward the proxy request to another proxy, it MUST translate the request as follows: The scheme of the request URI defines the outgoing protocol and its details (e.g., CoAP is used over UDP for the "coap" scheme and over DTLS for the "coaps" scheme.) For a CoAP-to-CoAP proxy, the origin server’s IP address and port are determined by the authority component of the request URI, and the request URI is decoded and split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options. This consumes the Proxy-Uri or Proxy-Scheme option, which is therefore not forwarded to the origin server.

5.7.3. Reverse-Proxies

Reverse-proxies do not make use of the Proxy-Uri or Proxy-Scheme options, but need to determine the destination (next hop) of a request from information in the request and information in their configuration. E.g., a reverse-proxy might offer various resources the existence of which it has learned through resource discovery as if they were its own resources. The reverse-proxy is free to build a namespace for the URIs that identify these resources. A reverse-proxy may also build a namespace that gives the client more control over where the request goes, e.g. by embedding host identifiers and port numbers into the URI path of the resources offered.

In processing the response, a reverse-proxy has to be careful that ETag option values from different sources are not mixed up on one resource offered to its clients. In many cases, the ETag can be forwarded unchanged. If the mapping from a resource offered by the

reverse-proxy to resources offered by its various origin servers is not unique, the reverse-proxy may need to generate a new ETag, making sure the semantics of this option are properly preserved.

5.8. Method Definitions

In this section each method is defined along with its behavior. A request with an unrecognized or unsupported Method Code MUST generate a 4.05 (Method Not Allowed) piggy-backed response.

5.8.1. GET

The GET method retrieves a representation for the information that currently corresponds to the resource identified by the request URI. If the request includes an Accept Option, that indicates the preferred content-format of a response. If the request includes an ETag Option, the GET method requests that ETag be validated and that the representation be transferred only if validation failed. Upon success a 2.05 (Content) or 2.03 (Valid) response code SHOULD be present in the response.

The GET method is safe and idempotent.

5.8.2. POST

The POST method requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and dependent on the target resource. It usually results in a new resource being created or the target resource being updated.

If a resource has been created on the server, the response returned by the server SHOULD have a 2.01 (Created) response code and SHOULD include the URI of the new resource in a sequence of one or more Location-Path and/or Location-Query Options (Section 5.10.7). If the POST succeeds but does not result in a new resource being created on the server, the response SHOULD have a 2.04 (Changed) response code. If the POST succeeds and results in the target resource being deleted, the response SHOULD have a 2.02 (Deleted) response code.

POST is neither safe nor idempotent.

5.8.3. PUT

The PUT method requests that the resource identified by the request URI be updated or created with the enclosed representation. The representation format is specified by the media type and content coding given in the Content-Format Option, if provided.

If a resource exists at the request URI the enclosed representation SHOULD be considered a modified version of that resource, and a 2.04 (Changed) response code SHOULD be returned. If no resource exists then the server MAY create a new resource with that URI, resulting in a 2.01 (Created) response code. If the resource could not be created or modified, then an appropriate error response code SHOULD be sent.

Further restrictions to a PUT can be made by including the If-Match (see Section 5.10.8.1) or If-None-Match (see Section 5.10.8.2) options in the request.

PUT is not safe, but is idempotent.

5.8.4. DELETE

The DELETE method requests that the resource identified by the request URI be deleted. A 2.02 (Deleted) response code SHOULD be used on success or in case the resource did not exist before the request.

DELETE is not safe, but is idempotent.

5.9. Response Code Definitions

Each response code is described below, including any options required in the response. Where appropriate, some of the codes will be specified in regards to related response codes in HTTP [RFC2616]; this does not mean that any such relationship modifies the HTTP mapping specified in Section 10.

5.9.1. Success 2.xx

This class of status code indicates that the clients request was successfully received, understood, and accepted.

5.9.1.1. 2.01 Created

Like HTTP 201 "Created", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

If the response includes one or more Location-Path and/or Location-Query Options, the values of these options specify the location at which the resource was created. Otherwise, the resource was created at the request URI. A cache receiving this response MUST mark any stored response for the created resource as not fresh.

This response is not cacheable.

5.9.1.2. 2.02 Deleted

Like HTTP 204 "No Content", but only used in response to requests that cause the resource to cease being available, such as DELETE and in certain circumstances POST. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache MUST mark any stored response for the deleted resource as not fresh.

5.9.1.3. 2.03 Valid

Related to HTTP 304 "Not Modified", but only used to indicate that the response identified by the entity-tag identified by the included ETag Option is valid. Accordingly, the response MUST include an ETag Option, and MUST NOT include a payload.

When a cache that recognizes and processes the ETag response option receives a 2.03 (Valid) response, it MUST update the stored response with the value of the Max-Age Option included in the response (explicitly, or implicitly as a default value; see also Section 5.6.2). For each type of Safe-to-Forward option present in the response, the (possibly empty) set of options of this type that are present in the stored response MUST be replaced with the set of options of this type in the response received. (Unsafe options may trigger similar option specific processing as defined by the option.)

5.9.1.4. 2.04 Changed

Like HTTP 204 "No Content", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache MUST mark any stored response for the changed resource as not fresh.

5.9.1.5. 2.05 Content

Like HTTP 200 "OK", but only used in response to GET requests.

The payload returned with the response is a representation of the target resource.

This response is cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1) and (if present) the ETag Option for validation (see Section 5.6.2).

5.9.2. Client Error 4.xx

This class of response code is intended for cases in which the client seems to have erred. These response codes are applicable to any request method.

The server SHOULD include a diagnostic payload under the conditions detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

5.9.2.1. 4.00 Bad Request

Like HTTP 400 "Bad Request".

5.9.2.2. 4.01 Unauthorized

The client is not authorized to perform the requested action. The client SHOULD NOT repeat the request without first improving its authentication status to the server. Which specific mechanism can be used for this is outside this document's scope; see also Section 9.

5.9.2.3. 4.02 Bad Option

The request could not be understood by the server due to one or more unrecognized or malformed options. The client SHOULD NOT repeat the request without modification.

5.9.2.4. 4.03 Forbidden

Like HTTP 403 "Forbidden".

5.9.2.5. 4.04 Not Found

Like HTTP 404 "Not Found".

5.9.2.6. 4.05 Method Not Allowed

Like HTTP 405 "Method Not Allowed", but with no parallel to the "Allow" header field.

5.9.2.7. 4.06 Not Acceptable

Like HTTP 406 "Not Acceptable", but with no response entity.

5.9.2.8. 4.12 Precondition Failed

Like HTTP 412 "Precondition Failed".

5.9.2.9. 4.13 Request Entity Too Large

Like HTTP 413 "Request Entity Too Large".

The response SHOULD include a Size1 Option (Section 5.10.9) to indicate the maximum size of request entity the server is able and willing to handle, unless the server is not in a position to make this information available.

5.9.2.10. 4.15 Unsupported Content-Format

Like HTTP 415 "Unsupported Media Type".

5.9.3. Server Error 5.xx

This class of response code indicates cases in which the server is aware that it has erred or is incapable of performing the request. These response codes are applicable to any request method.

The server SHOULD include a diagnostic payload under the conditions detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

5.9.3.1. 5.00 Internal Server Error

Like HTTP 500 "Internal Server Error".

5.9.3.2. 5.01 Not Implemented

Like HTTP 501 "Not Implemented".

5.9.3.3. 5.02 Bad Gateway

Like HTTP 502 "Bad Gateway".

5.9.3.4. 5.03 Service Unavailable

Like HTTP 503 "Service Unavailable", but using the Max-Age Option in place of the "Retry-After" header field to indicate the number of seconds after which to retry.

5.9.3.5. 5.04 Gateway Timeout

Like HTTP 504 "Gateway Timeout".

5.9.3.6. 5.05 Proxying Not Supported

The server is unable or unwilling to act as a forward-proxy for the URI specified in the Proxy-Uri Option or using Proxy-Scheme (see Section 5.10.2).

5.10. Option Definitions

The individual CoAP options are summarized in Table 4 and explained in the subsections of this section.

In this table, the C, U, and N columns indicate the properties, Critical, UnSafe, and NoCacheKey, respectively. Since NoCacheKey only has a meaning for options that are Safe-to-Forward (not marked Unsafe), the column is filled with a dash for UnSafe options. (The present specification does not define any NoCacheKey options, but the format of the table is intended to be useful for additional specifications.)

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 4: Options

5.10.1. Uri-Host, Uri-Port, Uri-Path and Uri-Query

The Uri-Host, Uri-Port, Uri-Path and Uri-Query Options are used to specify the target resource of a request to a CoAP origin server. The options encode the different components of the request URI in a way that no percent-encoding is visible in the option values and that the full URI can be reconstructed at any involved endpoint. The syntax of CoAP URIs is defined in Section 6.

The steps for parsing URIs into options is defined in Section 6.4. These steps result in zero or more Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in a request, where each option holds the following values:

- the Uri-Host Option specifies the Internet host of the resource being requested,
- the Uri-Port Option specifies the transport layer port number of the resource,
- each Uri-Path Option specifies one segment of the absolute path to the resource, and
- each Uri-Query Option specifies one argument parameterizing the resource.

Note: Fragments ([RFC3986], Section 3.5) are not part of the request URI and thus will not be transmitted in a CoAP request.

The default value of the Uri-Host Option is the IP literal representing the destination IP address of the request message. Likewise, the default value of the Uri-Port Option is the destination UDP port. The default values for the Uri-Host and Uri-Port Options are sufficient for requests to most servers. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers.

The Uri-Path and Uri-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Uri-Path Option MUST NOT be "." or ".." (as the request URI must be resolved before parsing it into options).

The steps for constructing the request URI from the options are defined in Section 6.5. Note that an implementation does not

necessarily have to construct the URI; it can simply look up the target resource by looking at the individual options.

Examples can be found in Appendix B.

5.10.2. Proxy-Uri and Proxy-Scheme

The Proxy-Uri Option is used to make a request to a forward-proxy (see Section 5.7). The forward-proxy is requested to forward the request or service it from a valid cache, and return the response.

The option value is an absolute-URI ([RFC3986], Section 4.3).

Note that the forward-proxy MAY forward the request on to another proxy or directly to the server specified by the absolute-URI. In order to avoid request loops, a proxy MUST be able to recognize all of its server names, including any aliases, local variations, and the numeric IP addresses.

An endpoint receiving a request with a Proxy-Uri Option that is unable or unwilling to act as a forward-proxy for the request MUST cause the return of a 5.05 (Proxying Not Supported) response.

The Proxy-Uri Option MUST take precedence over any of the Uri-Host, Uri-Port, Uri-Path or Uri-Query options (which MUST NOT be included at the same time in a request containing the Proxy-Uri Option).

As a special case to simplify many proxy clients, the absolute-URI can be constructed from the Uri-* options. When a Proxy-Scheme Option is present, the absolute-URI is constructed as follows: A CoAP URI is constructed from the Uri-* options as defined in Section 6.5. In the resulting URI, the initial scheme up to, but not including the following colon is then replaced by the content of the Proxy-Scheme Option. Note that this case is only applicable if the components of the desired URI other than the scheme component actually can be expressed using Uri-* options; e.g., to represent a URI with a userinfo component in the authority, only Proxy-Uri can be used.

5.10.3. Content-Format

The Content-Format Option indicates the representation format of the message payload. The representation format is given as a numeric content format identifier that is defined in the CoAP Content Format Registry (Section 12.3). In the absence of the option, no default value is assumed, i.e. the representation format of any representation message payload is indeterminate (Section 5.5).

5.10.4. Accept

The CoAP Accept option can be used to indicate which Content-Format is acceptable to the client. The representation format is given as a numeric Content-Format identifier that is defined in the CoAP Content-Format Registry (Section 12.3). If no Accept option is given, the client does not express a preference (thus no default value is assumed). The client prefers the representation returned by the server to be in the Content-Format indicated. The server returns the preferred Content-Format if available. If the preferred Content-Format cannot be returned, then a 4.06 "Not Acceptable" MUST be sent as a response, unless another error code takes precedence for this response.

5.10.5. Max-Age

The Max-Age Option indicates the maximum time a response may be cached before it is considered not fresh (see Section 5.6.1).

The option value is an integer number of seconds between 0 and 2**32-1 inclusive (about 136.1 years). A default value of 60 seconds is assumed in the absence of the option in a response.

The value is intended to be current at the time of transmission. Servers that provide resources with strict tolerances on the value of Max-Age SHOULD update the value before each retransmission. (See also Section 5.7.1.)

5.10.6. ETag

An entity-tag is intended for use as a resource-local identifier for differentiating between representations of the same resource that vary over time. It is generated by the server providing the resource, which may generate it in any number of ways including a version, checksum, hash or time. An endpoint receiving an entity-tag MUST treat it as opaque and make no assumptions about its content or structure. (Endpoints that generate an entity-tag are encouraged to use the most compact representation possible, in particular in regards to clients and intermediaries that may want to store multiple ETag values.)

5.10.6.1. ETag as a Response Option

The ETag Option in a response provides the current value (i.e., after the request was processed) of the entity-tag for the "tagged representation". If no Location-* options are present, the tagged representation is the selected representation (Section 5.5.3) of the target resource. If one or more Location-* options are present and

thus a location URI is indicated (Section 5.10.7), the tagged representation is the representation that would be retrieved by a GET request to the location URI.

An ETag response option can be included with any response for which there is a tagged representation (e.g., it would not be meaningful in a 4.04 or 4.00 response). The ETag Option MUST NOT occur more than once in a response.

There is no default value for the ETag Option; if it is not present in a response, the server makes no statement about the entity-tag for the tagged representation.

5.10.6.2. ETag as a Request Option

In a GET request, an endpoint that has one or more representations previously obtained from the resource, and has obtained ETag response options with these, can specify an instance of the ETag Option for one or more of these stored responses.

A server can issue a 2.03 Valid response (Section 5.9.1.3) in place of a 2.05 Content response if one of the ETags given is the entity-tag for the current representation, i.e. is valid; the 2.03 Valid response then echoes this specific ETag in a response option.

In effect, a client can determine if any of the stored representations is current (see Section 5.6.2) without needing to transfer them again.

The ETag Option MAY occur zero, one or more times in a request.

5.10.7. Location-Path and Location-Query

The Location-Path and Location-Query Options together indicate a relative URI that consists either of an absolute path, a query string or both. A combination of these options is included in a 2.01 (Created) response to indicate the location of the resource created as the result of a POST request (see Section 5.8.2). The location is resolved relative to the request URI.

If a response with one or more Location-Path and/or Location-Query Options passes through a cache that interprets these options and the implied URI identifies one or more currently stored responses, those entries MUST be marked as not fresh.

Each Location-Path Option specifies one segment of the absolute path to the resource, and each Location-Query Option specifies one argument parameterizing the resource. The Location-Path and

Location-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Location-Path Option MUST NOT be "." or "..".

The steps for constructing the location URI from the options are analogous to Section 6.5, except that the first five steps are skipped and the result is a relative URI-reference, which is then interpreted relative to the request URI. Note that the relative URI-reference constructed this way always includes an absolute-path (e.g., leaving out Location-Path but supplying Location-Query means the path component in the URI is "/").

The options that are used to compute the relative URI-reference are collectively called Location-* options. Beyond Location-Path and Location-Query, more Location-* options may be defined in the future, and have been reserved option numbers 128, 132, 136, and 140. If any of these reserved option numbers occurs in addition to Location-Path and/or Location-Query and are not supported, then a 4.02 (Bad Option) error MUST be returned.

5.10.8. Conditional Request Options

Conditional request options enable a client to ask the server to perform the request only if certain conditions specified by the option are fulfilled.

For each of these options, if the condition given is not fulfilled, then the server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

If the condition is fulfilled, the server performs the request method as if the conditional request options were not present.

If the request would, without the conditional request options, result in anything other than a 2.xx or 4.12 response code, then any conditional request options MAY be ignored.

5.10.8.1. If-Match

The If-Match Option MAY be used to make a request conditional on the current existence or value of an ETag for one or more representations of the target resource. If-Match is generally useful for resource update requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource (i.e., the "lost update" problem).

The value of an If-Match option is either an ETag or the empty string. An If-Match option with an ETag matches a representation with that exact ETag. An If-Match option with an empty value matches any existing representation (i.e., it places the precondition on the existence of any current representation for the target resource).

The If-Match Option can occur multiple times. If any of the options match, then the condition is fulfilled.

If there is one or more If-Match Option, but none of the options match, then the condition is not fulfilled.

5.10.8.2. If-None-Match

The If-None-Match Option MAY be used to make a request conditional on the non-existence of the target resource. If-None-Match is useful for resource creation requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource. The If-None-Match Option carries no value.

If the target resource does exist, then the condition is not fulfilled.

(It is not very useful to combine If-Match and If-None-Match options in one request, because the condition will then never be fulfilled.)

5.10.9. Size1 Option

The Size1 option provides size information about the resource representation in a request. The option value is an integer number of bytes. Its main use is with block-wise transfers [I-D.ietf-core-block]. In the present specification, it is used in 4.13 responses (Section 5.9.2.9) to indicate the maximum size of request entity that the server is able and willing to handle.

6. CoAP URIs

CoAP uses the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. Resources are organized hierarchically and governed by a potential CoAP origin server listening for CoAP requests ("coap") or DTLS-secured CoAP requests ("coaps") on a given UDP port. The CoAP server is identified via the generic syntax's authority component, which includes a host component and optional UDP port number. The remainder of the URI is considered to be identifying a resource which can be operated on by the methods defined by the CoAP protocol. The "coap" and "coaps" URI schemes can thus be compared to the "http" and "https" URI schemes respectively.

The syntax of the "coap" and "coaps" URI schemes is specified in this section in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", "query", "segment", "IP-literal", "IPv4address" and "reg-name" are adopted from [RFC3986].

Implementation Note: Unfortunately, over time the URI format has acquired significant complexity. Implementers are encouraged to examine [RFC3986] closely. E.g., the ABNF for IPv6 addresses is more complicated than maybe expected. Also, implementers should take care to perform the processing of percent decoding/encoding exactly once on the way from a URI to its decoded components or back. Percent encoding is crucial for data transparency, but may lead to unusual results such as a slash in a path component.

6.1. coap URI Scheme

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

If the host component is provided as an IP-literal or IPv4address, then the CoAP server can be reached at that IP address. If host is a registered name, then that name is considered an indirect identifier and the endpoint might use a name resolution service, such as DNS, to find the address of that host. The host MUST NOT be empty; if a URI is received with a missing authority or an empty host, then it MUST be considered invalid. The port subcomponent indicates the UDP port at which the CoAP server is located. If it is empty or not given, then the default port 5683 is assumed.

The path identifies a resource within the scope of the host and port. It consists of a sequence of path segments separated by a slash character (U+002F SOLIDUS "/").

The query serves to further parameterize the resource. It consists of a sequence of arguments separated by an ampersand character

(U+0026 AMPERSAND "&"). An argument is often in the form of a "key=value" pair.

The "coap" URI scheme supports the path prefix "/.well-known/" defined by [RFC5785] for "well-known locations" in the name-space of a host. This enables discovery of policy or other information about a host ("site-wide metadata"), such as hosted resources (see Section 7).

Application designers are encouraged to make use of short, but descriptive URIs. As the environments that CoAP is used in are usually constrained for bandwidth and energy, the trade-off between these two qualities should lean towards the shortness, without ignoring descriptiveness.

6.2. coaps URI Scheme

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

All of the requirements listed above for the "coap" scheme are also requirements for the "coaps" scheme, except that a default UDP port of [IANA_TBD_PORT] is assumed if the port subcomponent is empty or not given, and the UDP datagrams MUST be secured through the use of DTLS as described in Section 9.1.

Considerations for caching of responses to "coaps" identified requests are discussed in Section 11.2.

Resources made available via the "coaps" scheme have no shared identity with the "coap" scheme even if their resource identifiers indicate the same authority (the same host listening to the same UDP port). They are distinct name spaces and are considered to be distinct origin servers.

6.3. Normalization and Comparison Rules

Since the "coap" and "coaps" schemes conform to the URI generic syntax, such URIs are normalized and compared according to the algorithm defined in [RFC3986], Section 6, using the defaults described above for each scheme.

If the port is equal to the default port for a scheme, the normal form is to elide the port subcomponent. Likewise, an empty path component is equivalent to an absolute path of "/", so the normal form is to provide a path of "/" instead. The scheme and host are case-insensitive and normally provided in lowercase; IP-literals are

in recommended form [RFC5952]; all other components are compared in a case-sensitive manner. Characters other than those in the "reserved" set are equivalent to their percent-encoded bytes (see [RFC3986], Section 2.1): the normal form is to not encode them.

For example, the following three URIs are equivalent, and cause the same options and option values to appear in the CoAP messages:

coap://example.com:5683/~sensors/temp.xml
coap://EXAMPLE.com/%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]. At this stage the URL is in ASCII encoding [RFC0020], even though the decoded components will be interpreted in UTF-8 [RFC3629] after step 5, 8 and 9.

   NOTE: It doesn't matter what it is resolved relative to, since we already know it is an absolute URL at this point.

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

4. If |url| has a <fragment> component, then fail this algorithm.

5. If the <host> component of |url| does not represent the request's destination IP address as an IP-literal or IPv4address, include a Uri-Host Option and let that option's value be the value of the <host> component of |url|, converted to ASCII lowercase, and then converting all percent-encodings ("%" followed by two hexadecimal digits) to the corresponding characters.

   NOTE: In the usual case where the request's destination IP address is derived from the host part, this ensures that a Uri-Host Option is only used for a <host> component of the form reg-name.

6. If |url| has a <port> component, then let |port| be that component's value interpreted as a decimal integer; otherwise, let |port| be the default port for the scheme.

7. If |port| does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be |port|.

8. If the value of the <path> component of |url| is empty or consists of a single slash character (U+002F SOLIDUS "/"), then move to the next step.

   Otherwise, for each segment in the <path> component, include a Uri-Path Option and let that option's value be the segment (not including the delimiting slash characters) after converting each percent-encoding ("%" followed by two hexadecimal digits) to the corresponding byte.

9. If |url| has a <query> component, then, for each argument in the <query> component, include a Uri-Query Option and let that option's value be the argument (not including the question mark and the delimiting ampersand characters) after converting each percent-encoding to the corresponding byte.

Note that these rules completely resolve any percent-encoding.

6.5. Composing URIs from Options

The steps to construct a URI from a request's options are as follows. These steps either result in a URI, or they fail. In these steps, percent-encoding a character means replacing each of its (UTF-8 encoded) bytes by a "%" character followed by two hexadecimal digits representing the byte, where the digits A-F are in upper case (as defined in [RFC3986] Section 2.1; to reduce variability, the hexadecimal notation for percent-encoding in CoAP URIs MUST use uppercase letters). The definitions of "unreserved" and "sub-delims" are adopted from [RFC3986].

1. If the request is secured using DTLS, let |url| be the string "coaps://". Otherwise, let |url| be the string "coap://".

2. If the request includes a Uri-Host Option, let |host| be that option's value, where any non-ASCII characters are replaced by their corresponding percent-encoding. If |host| is not a valid reg-name or IP-literal or IPv4address, fail the algorithm. If the request does not include a Uri-Host Option, let |host| be the IP-literal (making use of the conventions of [RFC5952]) or IPv4address representing the request's destination IP address.

3. Append |host| to |url|.

4. If the request includes a Uri-Port Option, let |port| be that option's value. Otherwise, let |port| be the request's destination UDP port.

5. If |port| is not the default port for the scheme, then append a single U+003A COLON character (:) followed by the decimal representation of |port| to |url|.

6. Let |resource name| be the empty string. For each Uri-Path Option in the request, append a single character U+002F SOLIDUS (/) followed by the option's value to |resource name|, after converting any character that is not either in the "unreserved" set, "sub-delims" set, a U+003A COLON (:) or U+0040 COMMERCIAL AT (@) character, to its percent-encoded form.

7. If |resource name| is the empty string, set it to a single character U+002F SOLIDUS (/).

8. For each Uri-Query Option in the request, append a single character U+003F QUESTION MARK (?) (first option) or U+0026 AMPERSAND (&) (subsequent options) followed by the option's value to |resource name|, after converting any character that is not either in the "unreserved" set, "sub-delims" set (except U+0026 AMPERSAND (&)), a U+003A COLON (:), U+0040 COMMERCIAL AT (@), U+002F SOLIDUS (/) or U+003F QUESTION MARK (?) character, to its percent-encoded form.

9. Append |resource name| to |url|.

10. Return |url|.

Note that these steps have been designed to lead to a URI in normal form (see Section 6.3).

7. Discovery

7.1. Service Discovery

As a part of discovering the services offered by a CoAP server, a client has to learn about the endpoint used by a server.

A server is discovered by a client by the client (knowing or) learning a URI that references a resource in the namespace of the server. Alternatively, clients can use Multicast CoAP (see Section 8) and the "All CoAP Nodes" multicast address to find CoAP servers.

Unless the port subcomponent in a "coap" or "coaps" URI indicates the UDP port at which the CoAP server is located, the server is assumed to be reachable at the default port.

The CoAP default port number 5683 MUST be supported by a server that offers resources for resource discovery (see Section 7.2 below) and SHOULD be supported for providing access to other resources. The default port number [IANA_TBD_PORT] for DTLS-secured CoAP MAY be supported by a server for resource discovery and for providing access to other resources. In addition other endpoints may be hosted at other ports, e.g. in the dynamic port space.

Implementation Note: When a CoAP server is hosted by a 6LoWPAN node, header compression efficiency is improved when it also supports a port number in the 61616-61631 compressed UDP port space defined in [RFC4944] (note that, as its UDP port differs from the default port, it is a different endpoint from the server at the default port).

7.2. Resource Discovery

The discovery of resources offered by a CoAP endpoint is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. To maximize interoperability in a CoRE environment, a CoAP endpoint SHOULD support the CoRE Link Format of discoverable resources as described in [RFC6690], except where fully manual configuration is desired. It is up to the server which resources are made discoverable (if any).

7.2.1. 'ct' Attribute

This section defines a new Web Linking [RFC5988] attribute for use with [RFC6690]. The Content-Format code "ct" attribute provides a hint about the Content-Formats this resource returns. Note that this is only a hint, and does not override the Content-Format Option of a CoAP response obtained by actually requesting the representation of the resource. The value is in the CoAP identifier code format as a decimal ASCII integer and MUST be in the range of 0-65535 (16-bit unsigned integer). For example application/xml would be indicated as "ct=41". If no Content-Format code attribute is present then nothing about the type can be assumed. The Content-Format code attribute MAY include a space-separated sequence of Content-Format codes, indicating that multiple content-formats are available. The syntax of the attribute value is summarized in the production ct-value in Figure 12, where cardinal, SP and DQUOTE are defined as in [RFC6690].

```
ct-value =  cardinal
         /  DQUOTE cardinal *( 1*SP cardinal ) DQUOTE
```

Figure 12

8. Multicast CoAP

CoAP supports making requests to a IP multicast group. This is defined by a series of deltas to Unicast CoAP. A more general discussion of group communication with CoAP is in [I-D.ietf-core-groupcomm].

CoAP endpoints that offer services that they want other endpoints to be able to find using multicast service discovery, join one or more of the appropriate all-CoAP-nodes multicast addresses (Section 12.8) and listen on the default CoAP port. Note that an endpoint might receive multicast requests on other multicast addresses, including the all-nodes IPv6 address (or via broadcast on IPv4); an endpoint MUST therefore be prepared to receive such messages but MAY ignore them if multicast service discovery is not desired.

8.1. Messaging Layer

A multicast request is characterized by being transported in a CoAP message that is addressed to an IP multicast address instead of a CoAP endpoint. Such multicast requests MUST be Non-confirmable.

A server SHOULD be aware that a request arrived via multicast, e.g. by making use of modern APIs such as IPV6_RECVPKTINFO [RFC3542], if available.

To avoid an implosion of error responses, when a server is aware that a request arrived via multicast, it MUST NOT return a RST in reply to NON. If it is not aware, it MAY return a RST in reply to NON as usual. Because such a Reset message will look identical to an RST for a unicast message from the sender, the sender MUST avoid using a Message ID that is also still active from this endpoint with any unicast endpoint that might receive the multicast message.

At the time of writing, multicast messages can only be carried in UDP, not in DTLS. This means that the security modes defined for CoAP in this document are not applicable to multicast.

8.2. Request/Response Layer

When a server is aware that a request arrived via multicast, the server MAY always ignore the request, in particular if it doesn't have anything useful to respond (e.g., if it only has an empty payload or an error response). The decision for this may depend on the application. (For example, in [RFC6690] query filtering, a server should not respond to a multicast request if the filter does not match. More examples are in [I-D.ietf-core-groupcomm].)

If a server does decide to respond to a multicast request, it should not respond immediately. Instead, it should pick a duration for the period of time during which it intends to respond. For purposes of this exposition, we call the length of this period the Leisure. The specific value of this Leisure may depend on the application, or MAY be derived as described below. The server SHOULD then pick a random point of time within the chosen Leisure period to send back the unicast response to the multicast request. If further responses need to be sent based on the same multicast address membership, a new leisure period starts at the earliest after the previous one finishes.

To compute a value for Leisure, the server should have a group size estimate G, a target data transfer rate R (which both should be chosen conservatively) and an estimated response size S; a rough lower bound for Leisure can then be computed as

lb_Leisure = S * G / R

E.g., for a multicast request with link-local scope on an 2.4 GHz IEEE 802.15.4 (6LoWPAN) network, G could be (relatively conservatively) set to 100, S to 100 bytes, and the target rate to 8 kbit/s = 1 kB/s. The resulting lower bound for the Leisure is 10 seconds.

If a CoAP endpoint does not have suitable data to compute a value for Leisure, it MAY resort to DEFAULT_LEISURE.

When matching a response to a multicast request, only the token MUST match; the source endpoint of the response does not need to (and will not) be the same as the destination endpoint of the original request.

For the purposes of interpreting the Location-* options and any links embedded in the representation and, the request URI (base URI) relative to which the response is interpreted, is formed by replacing the multicast address in the Host component of the original request URI by the literal IP address of the endpoint actually responding.

8.2.1. Caching

When a client makes a multicast request, it always makes a new request to the multicast group (since there may be new group members that joined meanwhile or ones that did not get the previous request). It MAY update a cache with the received responses. Then it uses both cached-still-fresh and 'new' responses as the result of the request.

A response received in reply to a GET request to a multicast group MAY be used to satisfy a subsequent request on the related unicast request URI. The unicast request URI is obtained by replacing the authority part of the request URI with the transport layer source address of the response message.

A cache MAY revalidate a response by making a GET request on the related unicast request URI.

A GET request to a multicast group MUST NOT contain an ETag option. A mechanism to suppress responses the client already has is left for further study.

8.2.2. Proxying

When a forward-proxy receives a request with a Proxy-Uri or URI constructed from Proxy-Scheme that indicates a multicast address, the proxy obtains a set of responses as described above and sends all responses (both cached-still-fresh and new) back to the original client.

This specification does not provide a way to indicate the unicast-modified request URI (base URI) in responses thus forwarded. Proxying multicast requests is discussed in more detail in [I-D.ietf-core-groupcomm]; one proposal to address the base URI issue can be found in section 3 of [I-D.bormann-coap-misc].

9. Securing CoAP

This section defines the DTLS binding for CoAP.

During the provisioning phase, a CoAP device is provided with the security information that it needs, including keying materials and access control lists. This specification defines provisioning for the RawPublicKey mode in Section 9.1.3.2.1. At the end of the provisioning phase, the device will be in one of four security modes with the following information for the given mode. The NoSec and RawPublicKey modes are mandatory to implement for this specification.

NoSec: There is no protocol level security (DTLS is disabled). Alternative techniques to provide lower layer security SHOULD be used when appropriate. The use of IPsec is discussed in [I-D.bormann-core-ipsec-for-coap]. Certain link layers in use with constrained nodes also provide link layer security, which may be appropriate with proper key management.

PreSharedKey: DTLS is enabled and there is a list of pre-shared keys [RFC4279] and each key includes a list of which nodes it can be used to communicate with as described in Section 9.1.3.1. At the extreme there may be one key for each node this CoAP node needs to communicate with (1:1 node/key ratio). Conversely, if more than two entities share a specific pre-shared key, this key only enables the entities to authenticate as a member of that group and not as a specific peer.

RawPublicKey: DTLS is enabled and the device has an asymmetric key pair without a certificate (a raw public key) that is validated using an out-of-band mechanism [I-D.ietf-tls-oob-pubkey] as described in Section 9.1.3.2. The device also has an identity calculated from the public key and a list of identities of the nodes it can communicate with.

Certificate: DTLS is enabled and the device has an asymmetric key pair with an X.509 certificate [RFC5280] that binds it to its Authority Name and is signed by some common trust root as described in Section 9.1.3.3. The device also has a list of root trust anchors that can be used for validating a certificate.

In the "NoSec" mode, the system simply sends the packets over normal UDP over IP and is indicated by the "coap" scheme and the CoAP default port. The system is secured only by keeping attackers from being able to send or receive packets from the network with the CoAP nodes; see Section 11.5 for an additional complication with this approach.

The other three security modes are achieved using DTLS and are indicated by the "coaps" scheme and DTLS-secured CoAP default port. The result is a security association that can be used to authenticate (within the limits of the security model) and, based on this authentication, authorize the communication partner. CoAP itself does not provide protocol primitives for authentication or authorization; where this is required, it can either be provided by communication security (i.e., IPsec or DTLS) or by object security (within the payload). Devices that require authorization for certain operations are expected to require one of these two forms of security. Necessarily, where an intermediary is involved, communication security only works when that intermediary is part of

the trust relationships; CoAP does not provide a way to forward different levels of authorization that clients may have with an intermediary to further intermediaries or origin servers -- it therefore may be required to perform all authorization at the first intermediary.

9.1. DTLS-secured CoAP

Just as HTTP is secured using Transport Layer Security (TLS) over TCP, CoAP is secured using Datagram TLS (DTLS) [RFC6347] over UDP (see Figure 13). This section defines the CoAP binding to DTLS, along with the minimal mandatory-to-implement configurations appropriate for constrained environments. The binding is defined by a series of deltas to Unicast CoAP. DTLS is in practice TLS with added features to deal with the unreliable nature of the UDP transport.

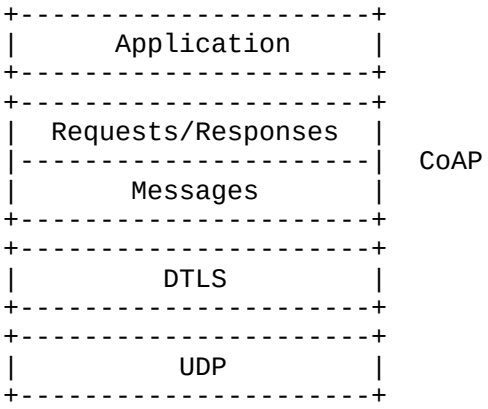


Figure 13: Abstract layering of DTLS-secured CoAP

In some constrained nodes (limited flash and/or RAM) and networks (limited bandwidth or high scalability requirements), and depending on the specific cipher suites in use, all modes of DTLS may not be applicable. Some DTLS cipher suites can add significant implementation complexity as well as some initial handshake overhead needed when setting up the security association. Once the initial handshake is completed, DTLS adds a limited per-datagram overhead of approximately 13 bytes, not including any initialization vectors/ nonces (e.g., 8 bytes with TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]), integrity check values (e.g., 8 bytes with TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]) and padding required by the cipher suite. Whether and which mode of using DTLS is applicable for a CoAP-based application should be carefully weighed considering the specific cipher suites that may be applicable, and whether the session maintenance makes it

compatible with application flows and sufficient resources are available on the constrained nodes and for the added network overhead. (For some modes of using DTLS, this specification identifies a mandatory to implement cipher suite. This is an implementation requirement to maximize interoperability in those cases where these cipher suites are indeed appropriate. The specific security policies of an application may determine the actual (set of) cipher suites that can be used.) DTLS is not applicable to group keying (multicast communication); however, it may be a component in a future group key management protocol.

9.1.1. Messaging Layer

The endpoint acting as the CoAP client should also act as the DTLS client. It should initiate a session to the server on the appropriate port. When the DTLS handshake has finished, the client may initiate the first CoAP request. All CoAP messages MUST be sent as DTLS "application data".

The following rules are added for matching an ACK or RST to a CON message or a RST to a NON message: The DTLS session MUST be the same and the epoch MUST be the same.

A message is the same when it is sent within the same DTLS session and same epoch and has the same Message ID.

Note: When a Confirmable message is retransmitted, a new DTLS sequence_number is used for each attempt, even though the CoAP Message ID stays the same. So a recipient still has to perform deduplication as described in Section 4.5. Retransmissions MUST NOT be performed across epochs.

DTLS connections in RawPublicKey and Certificate mode are set up using mutual authentication so they can remain up and be reused for future message exchanges in either direction. Devices can close a DTLS connection when they need to recover resources but in general they should keep the connection up for as long as possible. Closing the DTLS connection after every CoAP message exchange is very inefficient.

9.1.2. Request/Response Layer

The following rules are added for matching a response to a request: The DTLS session MUST be the same and the epoch MUST be the same.

This means the response to a DTLS secured request MUST always be DTLS secured using the same security session and epoch. Any attempt to supply a NoSec response to a DTLS request simply does not match the

request and (unless it does match an unrelated NoSec request) therefore MUST be rejected.

9.1.3. Endpoint Identity

Devices SHOULD support the Server Name Indication (SNI) to indicate their Authority Name in the SNI HostName field as defined in Section 3 of [RFC6066]. This is needed so that when a host that acts as a virtual server for multiple Authorities receives a new DTLS connection, it knows which keys to use for the DTLS session.

9.1.3.1. Pre-Shared Keys

When forming a connection to a new node, the system selects an appropriate key based on which nodes it is trying to reach and then forms a DTLS session using a PSK (Pre-Shared Key) mode of DTLS. Implementations in these modes MUST support the mandatory to implement cipher suite TLS_PSK_WITH_AES_128_CCM_8 as specified in [RFC6655].

Depending on the commissioning model, applications may need to define an application profile for identity hints as required and detailed in [RFC4279] (Section 5.2) to enable the use of PSK identity hints.

The security considerations of [RFC4279] (Section 7) apply. In particular, applications should carefully weigh whether they need Perfect Forward Secrecy (PFS) or not and select an appropriate cipher suite (7.1). The entropy of the PSK must be sufficient to mitigate against brute-force and (where the PSK is not chosen randomly but by a human) dictionary attacks (7.2). The cleartext communication of client identities may leak data or compromise privacy (7.3).

9.1.3.2. Raw Public Key Certificates

In this mode the device has an asymmetric key pair but without an X.509 certificate (called a raw public key); e.g., the asymmetric key pair is generated by the manufacturer and installed on the device (see also Section 11.6). A device MAY be configured with multiple raw public keys. The type and length of the raw public key depends on the cipher suite used. Implementations in RawPublicKey mode MUST support the mandatory to implement cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as specified in [I-D.mcgrew-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. The key used MUST be ECDSA-capable. The curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. Implementations MUST use the Supported Elliptic Curves Extension and Supported Point Format extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090]

can be used as an implementation method. Some guidance relevant to the implementation of this cipher suite can be found in [W3CXMLSEC]. The mechanism for using raw public keys with TLS is specified in [I-D.ietf-tls-oob-pubkey].

Implementation Note: Specifically, this means the extensions listed in Figure 14 with at least the values listed will be present in the DTLS handshake.

```
Extension: elliptic_curves
 Type: elliptic_curves (0x000a)
 Length: 4
 Elliptic Curves Length: 2
 Elliptic curves (1 curve)
   Elliptic curve: secp256r1 (0x0017)

Extension: ec_point_formats
 Type: ec_point_formats (0x000b)
 Length: 2
 EC point formats Length: 1
 Elliptic curves point formats (1)
   EC point format: uncompressed (0)

Extension: signature_algorithms
 Type: signature_algorithms (0x000d)
 Length: 4
 Data (4 bytes): 00 02 04 03
   HashAlgorithm: sha256 (4)
   SignatureAlgorithm: ecdsa (3)
```

Figure 14: DTLS extensions present for TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

9.1.3.2.1. Provisioning

The RawPublicKey mode was designed to be easily provisioned in M2M deployments. It is assumed that each device has an appropriate asymmetric public key pair installed. An identifier is calculated by the endpoint from the public key as described in Section 2 of [RFC6920]. All implementations that support checking RawPublicKey identities MUST support at least the sha-256-120 mode (SHA-256 truncated to 120 bits). Implementations SHOULD support also longer length identifiers and MAY support shorter lengths. Note that the shorter lengths provide less security against attacks and their use is NOT RECOMMENDED.

Depending on how identifiers are given to the system that verifies them, support for URI, binary, and/or human-speakable format

[RFC6920] needs to be implemented. All implementations SHOULD support the binary mode and implementations that have a user interface SHOULD also support the human-speakable format.

During provisioning, the identifier of each node is collected, for example by reading a barcode on the outside of the device or by obtaining a pre-compiled list of the identifiers. These identifiers are then installed in the corresponding endpoint, for example an M2M data collection server. The identifier is used for two purposes, to associate the endpoint with further device information and to perform access control. During (initial and ongoing) provisioning, an access control list of identifiers the device may start DTLS sessions with SHOULD also be installed and maintained.

9.1.3.3. X.509 Certificates

Implementations in Certificate Mode MUST support the mandatory to implement cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as specified in [I-D.mcgrew-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. Namely, the certificate includes a SubjectPublicKeyInfo that indicates an algorithm of id-ecPublicKey with namedCurves secp256r1 [RFC5480]; the public key format is uncompressed [RFC5480]; the hash algorithm is SHA-256; if included the key usage extension indicates digitalSignature. Certificates MUST be signed with ECDSA using secp256r1, and the signature MUST use SHA-256. The key used MUST be ECDSA-capable. The curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. Implementations MUST use the Supported Elliptic Curves Extension and Supported Point Format extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090] can be used as an implementation method.

The Authority Name in the certificate would be built out of a long term unique identifier for the device such as the EUI-64 [EUI64]. The Authority Name could also be based on the FQDN that was used as the Host part of the CoAP URI. However, the device’s IP address should not typically be used as the Authority name as it would change over time. The discovery process used in the system would build up the mapping between IP addresses of the given devices and the Authority Name for each device. Some devices could have more than one Authority and would need more than a single certificate.

When a new connection is formed, the certificate from the remote device needs to be verified. If the CoAP node has a source of absolute time, then the node SHOULD check that the validity dates of the certificate are within range. The certificate MUST be validated as appropriate for the security requirements, using functionality equivalent to the algorithm specified in [RFC5280] section 6. If the

certificate contains a SubjectAltName, then the Authority Name MUST match at least one of the authority names of any CoAP URI found in a field of URI type in the SubjectAltName set. If there is no SubjectAltName in the certificate, then the Authoritative Name MUST match the CN found in the certificate using the matching rules defined in [RFC2818] with the exception that certificates with wildcards are not allowed.

CoRE support for certificate status checking requires further study. As a mapping of OCSP [RFC2560] onto CoAP is not currently defined and OCSP may also not be easily applicable in all environments, an alternative approach may be using the TLS Certificate Status Request extension ([RFC6066] section 8, also known as "OCSP stapling") or preferably the Multiple Certificate Status Extension ([I-D.ietf-tls-multiple-cert-status-extension]), if available.

If the system has a shared key in addition to the certificate, then a cipher suite that includes the shared key such as TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA [RFC5489] SHOULD be used.

10. Cross-Protocol Proxying between CoAP and HTTP

CoAP supports a limited subset of HTTP functionality, and thus cross-protocol proxying to HTTP is straightforward. There might be several reasons for proxying between CoAP and HTTP, for example when designing a web interface for use over either protocol or when realizing a CoAP-HTTP proxy. Likewise, CoAP could equally be proxied to other protocols such as XMPP [RFC6120] or SIP [RFC3264]; the definition of these mechanisms is out of scope of this specification.

There are two possible directions to access a resource via a forward-proxy:

CoAP-HTTP Proxying: Enables CoAP clients to access resources on HTTP servers through an intermediary. This is initiated by including the Proxy-Uri or Proxy-Scheme Option with an "http" or "https" URI in a CoAP request to a CoAP-HTTP proxy.

HTTP-CoAP Proxying: Enables HTTP clients to access resources on CoAP servers through an intermediary. This is initiated by specifying a "coap" or "coaps" URI in the Request-Line of an HTTP request to an HTTP-CoAP proxy.

Either way, only the Request/Response model of CoAP is mapped to HTTP. The underlying model of Confirmable or Non-confirmable messages, etc., is invisible and MUST have no effect on a proxy function. The following sections describe the handling of requests to a forward-proxy. Reverse proxies are not specified as the proxy

function is transparent to the client with the proxy acting as if it was the origin server. However, similar considerations apply to reverse-proxies as to forward-proxies, and there generally will be an expectation that reverse-proxies operate in a similar way forward-proxies would. As an implementation note, HTTP client libraries may make it hard to operate an HTTP-CoAP forward proxy by not providing a way to put a CoAP URI on the HTTP Request-Line; reverse-proxying may therefore lead to wider applicability of a proxy. A separate specification may define a convention for URIs operating such a HTTP-CoAP reverse proxy [I-D.castellani-core-http-mapping].

10.1. CoAP-HTTP Proxying

If a request contains a Proxy-Uri or Proxy-Scheme Option with an ’http’ or ’https’ URI [RFC2616], then the receiving CoAP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated HTTP resource and return the result to the client. (See also Section 5.7 for how the request to the proxy is formulated, including security requirements.)

This section specifies for any CoAP request the CoAP response that the proxy should return to the client. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to an HTTP origin server.

Since HTTP and CoAP share the basic set of request methods, performing a CoAP request on an HTTP resource is not so different from performing it on a CoAP resource. The meanings of the individual CoAP methods when performed on HTTP resources are explained in the subsections of this section.

If the proxy is unable or unwilling to service a request with an HTTP URI, a 5.05 (Proxying Not Supported) response is returned to the client. If the proxy services the request by interacting with a third party (such as the HTTP origin server) and is unable to obtain a result within a reasonable time frame, a 5.04 (Gateway Timeout) response is returned; if a result can be obtained but is not understood, a 5.02 (Bad Gateway) response is returned.

10.1.1. GET

The GET method requests the proxy to return a representation of the HTTP resource identified by the request URI.

Upon success, a 2.05 (Content) response code SHOULD be returned. The payload of the response MUST be a representation of the target HTTP resource, and the Content-Format Option be set accordingly. The

response MUST indicate a Max-Age value that is no greater than the remaining time the representation can be considered fresh. If the HTTP entity has an entity tag, the proxy SHOULD include an ETag Option in the response and process ETag Options in requests as described below.

A client can influence the processing of a GET request by including the following option:

Accept: The request MAY include an Accept Option, identifying the preferred response content-format.

ETag: The request MAY include one or more ETag Options, identifying responses that the client has stored. This requests the proxy to send a 2.03 (Valid) response whenever it would send a 2.05 (Content) response with an entity tag in the requested set otherwise. Note that CoAP ETags are always strong ETags in the HTTP sense; CoAP does not have the equivalent of HTTP weak ETags, and there is no good way to make use of these in a cross-proxy.

10.1.2. PUT

The PUT method requests the proxy to update or create the HTTP resource identified by the request URI with the enclosed representation.

If a new resource is created at the request URI, a 2.01 (Created) response MUST be returned to the client. If an existing resource is modified, a 2.04 (Changed) response MUST be returned to indicate successful completion of the request.

10.1.3. DELETE

The DELETE method requests the proxy to delete the HTTP resource identified by the request URI at the HTTP origin server.

A 2.02 (Deleted) response MUST be returned to client upon success or if the resource does not exist at the time of the request.

10.1.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the HTTP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 2.04 (Changed) response MUST be returned to the client. If a resource has been created on the origin server, a 2.01 (Created) response MUST be returned.

10.2. HTTP-CoAP Proxying

If an HTTP request contains a Request-URI with a 'coap' or 'coaps' URI, then the receiving HTTP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated CoAP resource and return the result to the client.

This section specifies for any HTTP request the HTTP response that the proxy should return to the client. Unless otherwise specified all the statements made are RECOMMENDED behavior; some highly constrained implementations may need to resort to shortcuts. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to a CoAP origin server. The meanings of the individual HTTP methods when performed on CoAP resources are explained in the subsections of this section.

If the proxy is unable or unwilling to service a request with a CoAP URI, a 501 (Not Implemented) response is returned to the client. If the proxy services the request by interacting with a third party (such as the CoAP origin server) and is unable to obtain a result within a reasonable time frame, a 504 (Gateway Timeout) response is returned; if a result can be obtained but is not understood, a 502 (Bad Gateway) response is returned.

10.2.1. OPTIONS and TRACE

As the OPTIONS and TRACE methods are not supported in CoAP a 501 (Not Implemented) error MUST be returned to the client.

10.2.2. GET

The GET method requests the proxy to return a representation of the CoAP resource identified by the Request-URI.

Upon success, a 200 (OK) response is returned. The payload of the response MUST be a representation of the target CoAP resource, and the Content-Type and Content-Encoding header fields be set accordingly. The response MUST indicate a max-age directive that indicates a value no greater than the remaining time the representation can be considered fresh. If the CoAP response has an ETag option, the proxy should include an ETag header field in the response.

A client can influence the processing of a GET request by including the following options:

Accept: The most preferred Media-type of the HTTP Accept header field in a request is mapped to a CoAP Accept option. HTTP Accept Media-type ranges, parameters and extensions are not supported by the CoAP Accept option. If the proxy cannot send a response which is acceptable according to the combined Accept field value, then the proxy sends a 406 (not acceptable) response. The proxy MAY then retry the request with further Media-types from the HTTP Accept header field.

Conditional GETs: Conditional HTTP GET requests that include an "If-Match" or "If-None-Match" request-header field can be mapped to a corresponding CoAP request. The "If-Modified-Since" and "If-Unmodified-Since" request-header fields are not directly supported by CoAP, but are implemented locally by a caching proxy.

10.2.3. HEAD

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response.

Although there is no direct equivalent of HTTP’s HEAD method in CoAP, an HTTP-CoAP proxy responds to HEAD requests for CoAP resources, and the HTTP headers are returned without a message-body.

Implementation Note: An HTTP-CoAP proxy may want to try using a block-wise transfer [I-D.ietf-core-block] option to minimize the amount of data actually transferred, but needs to be prepared for the case that the origin server does not support block-wise transfers.

10.2.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the CoAP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 200 (OK) or 204 (No Content) response MUST be returned to the client. If a resource has been created on the origin server, a 201 (Created) response MUST be returned.

If any of the Location-* Options are present in the CoAP response, a Location header field constructed from the values of these options is returned.

10.2.5. PUT

The PUT method requests the proxy to update or create the CoAP resource identified by the Request-URI with the enclosed representation.

If a new resource is created at the Request-URI, a 201 (Created) response is returned to the client. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes is sent to indicate successful completion of the request.

10.2.6. DELETE

The DELETE method requests the proxy to delete the CoAP resource identified by the Request-URI at the CoAP origin server.

A successful response is 200 (OK) if the response includes an entity describing the status or 204 (No Content) if the action has been enacted but the response does not include an entity.

10.2.7. CONNECT

This method can not currently be satisfied by an HTTP-CoAP proxy function as TLS to DTLS tunneling has not yet been specified. For now, a 501 (Not Implemented) error is returned to the client.

11. Security Considerations

This section analyzes the possible threats to the protocol. It is meant to inform protocol and application developers about the security limitations of CoAP as described in this document. As CoAP realizes a subset of the features in HTTP/1.1, the security considerations in Section 15 of [RFC2616] are also pertinent to CoAP. This section concentrates on describing limitations specific to CoAP.

11.1. Protocol Parsing, Processing URIs

A network-facing application can exhibit vulnerabilities in its processing logic for incoming packets. Complex parsers are well-known as a likely source of such vulnerabilities, such as the ability to remotely crash a node, or even remotely execute arbitrary code on it. CoAP attempts to narrow the opportunities for introducing such vulnerabilities by reducing parser complexity, by giving the entire range of encodable values a meaning where possible, and by

aggressively reducing complexity that is often caused by unnecessary choice between multiple representations that mean the same thing. Much of the URI processing has been moved to the clients, further reducing the opportunities for introducing vulnerabilities into the servers. Even so, the URI processing code in CoAP implementations is likely to be a large source of remaining vulnerabilities and should be implemented with special care. CoAP access control implementations need to ensure they don't introduce vulnerabilities through discrepancies between the code deriving access control decisions from a URI and the code finally serving up the resource addressed by the URI. The most complex parser remaining could be the one for the CoRE Link Format, although this also has been designed with a goal of reduced implementation complexity [RFC6690]. (See also section 15.2 of [RFC2616].)

11.2. Proxying and Caching

As mentioned in 15.7 of [RFC2616], proxies are by their very nature men-in-the-middle, breaking any IPsec or DTLS protection that a direct CoAP message exchange might have. They are therefore interesting targets for breaking confidentiality or integrity of CoAP message exchanges. As noted in [RFC2616], they are also interesting targets for breaking availability.

The threat to confidentiality and integrity of request/response data is amplified where proxies also cache. Note that CoAP does not define any of the cache-suppressing Cache-Control options that HTTP/1.1 provides to better protect sensitive data.

For a caching implementation, any access control considerations that would apply to making the request that generated the cache entry also need to be applied to the value in the cache. This is relevant for clients that implement multiple security domains, as well as for proxies that may serve multiple clients. Also, a caching proxy MUST NOT make cached values available to requests that have lesser transport security properties than to which it would make available the process of forwarding the request in the first place.

Unlike the "coap" scheme, responses to "coaps" identified requests are never "public" and thus MUST NOT be reused for shared caching unless the cache is able to make equivalent access control decisions to the ones that led to the cached entry. They can, however, be reused in a private cache if the message is cacheable by default in CoAP.

Finally, a proxy that fans out Separate Responses (as opposed to Piggy-backed Responses) to multiple original requesters may provide additional amplification (see Section 11.3).

11.3. Risk of amplification

CoAP servers generally reply to a request packet with a response packet. This response packet may be significantly larger than the request packet. An attacker might use CoAP nodes to turn a small attack packet into a larger attack packet, an approach known as amplification. There is therefore a danger that CoAP nodes could become implicated in denial of service (DoS) attacks by using the amplifying properties of the protocol: An attacker that is attempting to overload a victim but is limited in the amount of traffic it can generate, can use amplification to generate a larger amount of traffic.

This is particularly a problem in nodes that enable NoSec access, that are accessible from an attacker and can access potential victims (e.g. on the general Internet), as the UDP protocol provides no way to verify the source address given in the request packet. An attacker need only place the IP address of the victim in the source address of a suitable request packet to generate a larger packet directed at the victim.

As a mitigating factor, many constrained networks will only be able to generate a small amount of traffic, which may make CoAP nodes less attractive for this attack. However, the limited capacity of the constrained network makes the network itself a likely victim of an amplification attack.

Therefore, large amplification factors SHOULD NOT be provided in the response if the request is not authenticated. A CoAP server can reduce the amount of amplification it provides to an attacker by using slicing/blocking modes of CoAP [I-D.ietf-core-block] and offering large resource representations only in relatively small slices. E.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

CoAP also supports the use of multicast IP addresses in requests, an important requirement for M2M. Multicast CoAP requests may be the source of accidental or deliberate denial of service attacks, especially over constrained networks. This specification attempts to reduce the amplification effects of multicast requests by limiting when a response is returned. To limit the possibility of malicious use, CoAP servers SHOULD NOT accept multicast requests that can not be authenticated in some way, cryptographically or by some multicast boundary limiting the potential sources. If possible a CoAP server SHOULD limit the support for multicast requests to the specific resources where the feature is required.

On some general purpose operating systems providing a Posix-style API, it is not straightforward to find out whether a packet received was addressed to a multicast address. While many implementations will know whether they have joined a multicast group, this creates a problem for packets addressed to multicast addresses of the form FF0x::1, which are received by every IPv6 node. Implementations SHOULD make use of modern APIs such as IPV6_RECVPKTINFO [RFC3542], if available, to make this determination.

11.4. IP Address Spoofing Attacks

Due to the lack of a handshake in UDP, a rogue endpoint which is free to read and write messages carried by the constrained network (i.e. NoSec or PreSharedKey deployments with nodes/key ratio > 1:1), may easily attack a single endpoint, a group of endpoints, as well as the whole network e.g. by:

1. spoofing RST in response to a CON or NON message, thus making an endpoint "deaf"; or

2. spoofing an ACK in response to a CON message, thus potentially preventing the sender of the CON message from retransmitting, and drowning out the actual response; or

3. spoofing the entire response with forged payload/options (this has different levels of impact: from single response disruption, to much bolder attacks on the supporting infrastructure, e.g. poisoning proxy caches, or tricking validation / lookup interfaces in resource directories and, more generally, any component that stores global network state and uses CoAP as the messaging facility to handle state set/update’s is a potential target.); or

4. spoofing a multicast request for a target node which may result in both network congestion/collapse and victim DoS’ing / forced wakeup from sleeping; or

5. spoofing observe messages, etc.

Response spoofing by off-path attackers can be detected and mitigated even without transport layer security by choosing a non-trivial, randomized token in the request (Section 5.3.1). [RFC4086] discusses randomness requirements for security.

In principle, other kinds of spoofing can be detected by CoAP only in case CON semantics is used, because of unexpected ACK/RSTs coming from the deceived endpoint. But this imposes keeping track of the used Message IDs which is not always possible, and moreover detection

becomes available usually after the damage is already done. This kind of attack can be prevented using security modes other than NoSec.

With or without source address spoofing, a client can attempt to overload a server by sending requests, preferably complex ones, to a server; address spoofing makes tracing back, and blocking, this attack harder. Given that the cost of a CON request is small, this attack can easily be executed. Under this attack, a constrained node with limited total energy available may exhaust that energy much more quickly than planned (battery depletion attack). Also, if the client uses a Confirmable message and the server responds with a Confirmable separate response to a (possibly spoofed) address that does not respond, the server will have to allocate buffer and retransmission logic for each response up to the exhaustion of MAX_TRANSMIT_SPAN, making it more likely that it runs out of resources for processing legitimate traffic. The latter problem can be mitigated somewhat by limiting the rate of responses as discussed in Section 4.7. An attacker could also spoof the address of a legitimate client, which, if the server uses separate responses, might block legitimate responses to that client because of NSTART=1. All these attacks can be prevented using a security mode other than NoSec, leaving only attacks on the security protocol.

11.5. Cross-Protocol Attacks

The ability to incite a CoAP endpoint to send packets to a fake source address can be used not only for amplification, but also for cross-protocol attacks against a victim listening to UDP packets at a given address (IP address and port):

- the attacker sends a message to a CoAP endpoint with the given address as the fake source address,
- the CoAP endpoint replies with a message to the given source address,
- the victim at the given address receives a UDP packet that it interprets according to the rules of a different protocol.

This may be used to circumvent firewall rules that prevent direct communication from the attacker to the victim, but happen to allow communication from the CoAP endpoint (which may also host a valid role in the other protocol) to the victim.

Also, CoAP endpoints may be the victim of a cross-protocol attack generated through an endpoint of another UDP-based protocol such as DNS. In both cases, attacks are possible if the security properties

of the endpoints rely on checking IP addresses (and firewalling off direct attacks sent from outside using fake IP addresses). In general, because of their lack of context, UDP-based protocols are relatively easy targets for cross-protocol attacks.

Finally, CoAP URIs transported by other means could be used to incite clients to send messages to endpoints of other protocols.

One mitigation against cross-protocol attacks is strict checking of the syntax of packets received, combined with sufficient difference in syntax. As an example, it might help if it were difficult to incite a DNS server to send a DNS response that would pass the checks of a CoAP endpoint. Unfortunately, the first two bytes of a DNS reply are an ID that can be chosen by the attacker, which map into the interesting part of the CoAP header, and the next two bytes are then interpreted as CoAP's Message ID (i.e., any value is acceptable). The DNS count words may be interpreted as multiple instances of a (non-existent, but elective) CoAP option 0, or possibly as a Token. The echoed query finally may be manufactured by the attacker to achieve a desired effect on the CoAP endpoint; the response added by the server (if any) might then just be interpreted as added payload.

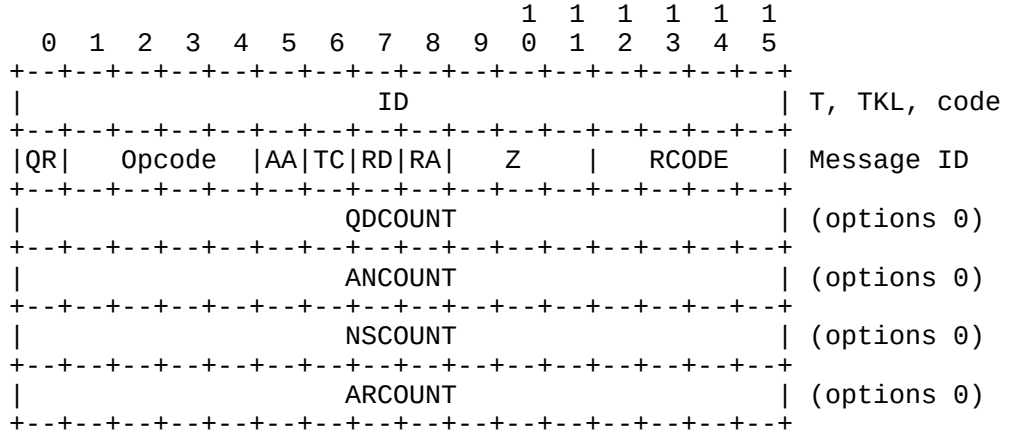


Figure 15: DNS Header vs. CoAP Message

In general, for any pair of protocols, one of the protocols can very well have been designed in a way that enables an attacker to cause the generation of replies that look like messages of the other protocol. It is often much harder to ensure or prove the absence of viable attacks than to generate examples that may not yet completely enable an attack but might be further developed by more creative minds. Cross-protocol attacks can therefore only be completely

mitigated if endpoints don't authorize actions desired by an attacker just based on trusting the source IP address of a packet. Conversely, a NoSec environment that completely relies on a firewall for CoAP security not only needs to firewall off the CoAP endpoints but also all other endpoints that might be incited to send UDP messages to CoAP endpoints using some other UDP-based protocol.

In addition to the considerations above, the security considerations for DTLS with respect to cross-protocol attacks apply. E.g., if the same DTLS security association ("connection") is used to carry data of multiple protocols, DTLS no longer provides protection against cross-protocol attacks between these protocols.

11.6. Constrained node considerations

Implementers on constrained nodes often find themselves without a good source of entropy [RFC4086]. If that is the case, the node MUST NOT be used for processes that require good entropy, such as key generation. Instead, keys should be generated externally and added to the device during manufacturing or commissioning.

Due to their low processing power, constrained nodes are particularly susceptible to timing attacks. Special care must be taken in implementation of cryptographic primitives.

Large numbers of constrained nodes will be installed in exposed environments and will have little resistance to tampering, including recovery of keying materials. This needs to be considered when defining the scope of credentials assigned to them. In particular, assigning a shared key to a group of nodes may make any single constrained node a target for subverting the entire group.

12. IANA Considerations

12.1. CoAP Code Registries

This document defines two sub-registries for the values of the Code field in the CoAP header within the Constrained RESTful Environments (CoRE) Parameters ("CoRE Parameters") registry.

Values in the two sub-registries are eight-bit values notated as three decimal digits c.dd separated by a period between the first and the second digit; the first digit c is between 0 and 7 and denotes the code class; the second and third digit dd denote a decimal number between 00 and 31 for the detail.

All Code values are assigned by sub-registries according to the following ranges:

0.00 Indicates an Empty message (see Section 4.1).

0.01-0.31 Indicates a request. Values in this range are assigned by the "CoAP Method Codes" sub-registry (see Section 12.1.1).

1.00-1.31 Reserved

2.00-5.31 Indicates a response. Values in this range are assigned by the "CoAP Response Codes" sub-registry (see Section 12.1.2).

6.00-7.31 Reserved

12.1.1. Method Codes

The name of the sub-registry is "CoAP Method Codes".

Each entry in the sub-registry must include the Method Code in the range 0.01-0.31, the name of the method, and a reference to the method's documentation.

Initial entries in this sub-registry are as follows:

Code	Name	Reference
0.01	GET	[RFCXXXX]
0.02	POST	[RFCXXXX]
0.03	PUT	[RFCXXXX]
0.04	DELETE	[RFCXXXX]

Table 5: CoAP Method Codes

All other Method Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG approval" as described in [RFC5226].

The documentation of a method code should specify the semantics of a request with that code, including the following properties:

- The response codes the method returns in the success case.
- Whether the method is idempotent, safe, or both.

12.1.2. Response Codes

The name of the sub-registry is "CoAP Response Codes".

Each entry in the sub-registry must include the Response Code in the range 2.00-5.31, a description of the Response Code, and a reference to the Response Code's documentation.

Initial entries in this sub-registry are as follows:

Code	Description	Reference
2.01	Created	[RFCXXXX]
2.02	Deleted	[RFCXXXX]
2.03	Valid	[RFCXXXX]
2.04	Changed	[RFCXXXX]
2.05	Content	[RFCXXXX]
4.00	Bad Request	[RFCXXXX]
4.01	Unauthorized	[RFCXXXX]
4.02	Bad Option	[RFCXXXX]
4.03	Forbidden	[RFCXXXX]
4.04	Not Found	[RFCXXXX]
4.05	Method Not Allowed	[RFCXXXX]
4.06	Not Acceptable	[RFCXXXX]
4.12	Precondition Failed	[RFCXXXX]
4.13	Request Entity Too Large	[RFCXXXX]
4.15	Unsupported Content-Format	[RFCXXXX]
5.00	Internal Server Error	[RFCXXXX]
5.01	Not Implemented	[RFCXXXX]
5.02	Bad Gateway	[RFCXXXX]
5.03	Service Unavailable	[RFCXXXX]
5.04	Gateway Timeout	[RFCXXXX]
5.05	Proxying Not Supported	[RFCXXXX]

Table 6: CoAP Response Codes

The Response Codes 3.00-3.31 are Reserved for future use. All other Response Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG approval" as described in [RFC5226].

The documentation of a response code should specify the semantics of a response with that code, including the following properties:

o The methods the response code applies to.

o Whether payload is required, optional or not allowed.

- The semantics of the payload. For example, the payload of a 2.05 (Content) response is a representation of the target resource; the payload in an error response is a human-readable diagnostic payload.

- The format of the payload. For example, the format in a 2.05 (Content) response is indicated by the Content-Format Option; the format of the payload in an error response is always Net-Unicode text.

- Whether the response is cacheable according to the freshness model.

- Whether the response is validatable according to the validation model.

- Whether the response causes a cache to mark responses stored for the request URI as not fresh.

12.2. Option Number Registry

This document defines a sub-registry for the Option Numbers used in CoAP options within the "CoRE Parameters" registry. The name of the sub-registry is "CoAP Option Numbers".

Each entry in the sub-registry must include the Option Number, the name of the option and a reference to the option's documentation.

Initial entries in this sub-registry are as follows:

Number	Name	Reference
0	(Reserved)	[RFCXXXX]
1	If-Match	[RFCXXXX]
3	Uri-Host	[RFCXXXX]
4	ETag	[RFCXXXX]
5	If-None-Match	[RFCXXXX]
7	Uri-Port	[RFCXXXX]
8	Location-Path	[RFCXXXX]
11	Uri-Path	[RFCXXXX]
12	Content-Format	[RFCXXXX]
14	Max-Age	[RFCXXXX]
15	Uri-Query	[RFCXXXX]
17	Accept	[RFCXXXX]
20	Location-Query	[RFCXXXX]
35	Proxy-Uri	[RFCXXXX]
39	Proxy-Scheme	[RFCXXXX]

	60	Size1	[RFCXXXX]
	128	(Reserved)	[RFCXXXX]
	132	(Reserved)	[RFCXXXX]
	136	(Reserved)	[RFCXXXX]
	140	(Reserved)	[RFCXXXX]

Table 7: CoAP Option Numbers

The IANA policy for future additions to this sub-registry is split into three tiers as follows. The range of 0..255 is reserved for options defined by the IETF (IETF Review or IESG approval). The range of 256..2047 is reserved for commonly used options with public specifications (Specification Required). The range of 2048..64999 is for all other options including private or vendor specific ones, which undergo a Designated Expert review to help ensure that the option semantics are defined correctly. The option numbers between 65000 and 65535 inclusive are reserved for experiments. They are not meant for vendor specific use of any kind and MUST NOT be used in operational deployments.

Option Number	Policy [RFC5226]
0..255	IETF Review or IESG approval
256..2047	Specification Required
2048..64999	Designated Expert
65000..65535	Reserved for experiments

Table 8: CoAP Option Number Registry Policy

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

- The meaning of the option in a request.
- The meaning of the option in a response.
- Whether the option is critical or elective, as determined by the Option Number.
- Whether the option is Safe-to-Forward, and, if yes, whether it is part of the Cache-Key, as determined by the Option Number (see Section 5.4.2).
- The format and length of the option’s value.

- Whether the option must occur at most once or whether it can occur multiple times.

- The default value, if any. For a critical option with a default value, a discussion on how the default value enables processing by implementations not implementing the critical option (Section 5.4.4).

12.3. Content-Format Registry

Internet media types are identified by a string, such as "application /xml" [RFC2046]. In order to minimize the overhead of using these media types to indicate the format of payloads, this document defines a sub-registry for a subset of Internet media types to be used in CoAP and assigns each, in combination with a content-coding, a numeric identifier. The name of the sub-registry is "CoAP Content-Formats", within the "CoRE Parameters" registry.

Each entry in the sub-registry must include the media type registered with IANA, the numeric identifier in the range 0-65535 to be used for that media type in CoAP, the content-coding associated with this identifier, and a reference to a document describing what a payload with that media type means semantically.

CoAP does not include a separate way to convey content-encoding information with a request or response, and for that reason the content-encoding is also specified for each identifier (if any). If multiple content-encodings will be used with a media type, then a separate Content-Format identifier for each is to be registered. Similarly, other parameters related to an Internet media type, such as level, can be defined for a CoAP Content-Format entry.

Initial entries in this sub-registry are as follows:

Media type	Encoding	Id.	Reference
text/plain; charset=utf-8	-	0	[RFC2046][RFC3676][RFC5147]
application/ link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/ octet-stream	-	42	[RFC2045][RFC2046]
application/exi	-	47	[EXIMIME]
application/json	-	50	[RFC4627]

Table 9: CoAP Content-Formats

The identifiers between 65000 and 65535 inclusive are reserved for experiments. They are not meant for vendor specific use of any kind and MUST NOT be used in operational deployments. The identifiers between 256 and 9999 are reserved for future use in IETF specifications (IETF review or IESG approval). All other identifiers are Unassigned.

Because the name space of single-byte identifiers is so small, the IANA policy for future additions in the range 0-255 inclusive to the sub-registry is "Expert Review" as described in [RFC5226]. The IANA policy for additions in the range 10000-64999 inclusive is "First Come First Served" as described in [RFC5226].

In machine to machine applications, it is not expected that generic Internet media types such as text/plain, application/xml or application/octet-stream are useful for real applications in the long term. It is recommended that M2M applications making use of CoAP will request new Internet media types from IANA indicating semantic information about how to create or parse a payload. For example, a Smart Energy application payload carried as XML might request a more specific type like application/se+xml or application/se-exi.

12.4. URI Scheme Registration

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap". The registration request complies with [RFC4395].

URI scheme name.
   coap

Status.

Permanent.

URI scheme syntax.
Defined in Section 6.1 of [RFCXXXX].

URI scheme semantics.
The "coap" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP). The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "http" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

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

Interoperability considerations.
None.

Security considerations.
See Section 11.1 of [RFCXXXX].

Contact.
IETF Chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
[RFCXXXX]

12.5. Secure URI Scheme Registration

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coaps". The registration request complies with [RFC4395].

URI scheme name.
coaps

Status.
Permanent.

URI scheme syntax.
Defined in Section 6.2 of [RFCXXXX].

URI scheme semantics.
The "coaps" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using Datagram Transport Layer Security (DTLS) for transport security. The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "https" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

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

Interoperability considerations.
None.

Security considerations.
See Section 11.1 of [RFCXXXX].

Contact.
IETF Chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
[RFCXXXX]

12.6. Service Name and Port Number Registration

One of the functions of CoAP is resource discovery: a CoAP client can ask a CoAP server about the resources offered by it (see Section 7). To enable resource discovery just based on the knowledge of an IP address, the CoAP port for resource discovery needs to be standardized.

IANA has assigned the port number 5683 and the service name "coap", in accordance with [RFC6335].

Besides unicast, CoAP can be used with both multicast and anycast.

Service Name.
coap

Transport Protocol.
UDP

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFCXXXX]

Port Number.
5683

12.7. Secure Service Name and Port Number Registration

CoAP resource discovery may also be provided using the DTLS-secured CoAP "coaps" scheme. Thus the CoAP port for secure resource discovery needs to be standardized.

This document requests the assignment of the port number [IANA_TBD_PORT] and the service name "coaps", in accordance with [RFC6335].

Besides unicast, DTLS-secured CoAP can be used with anycast.

Service Name.
coaps

Transport Protocol.
UDP

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.

DTLS-secured CoAP

Reference.
[RFCXXXX]

Port Number.
[IANA_TBD_PORT]

12.8. Multicast Address Registration

Section 8, "Multicast CoAP", defines the use of multicast. This document requests the assignment of the following multicast addresses for use by CoAP nodes:

IPv4 -- "All CoAP Nodes" address [TBD1], from the IPv4 Multicast Address Space Registry. As the address is used for discovery that may span beyond a single network, it should come from the Internetwork Control Block (224.0.1.x, RFC 5771).

IPv6 -- "All CoAP Nodes" address [TBD2], from the IPv6 Multicast Address Space Registry, in the Variable Scope Multicast Addresses space (RFC3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only. The address should be of the form FF0x::nn, where nn is a single byte, to ensure good compression of the local-scope address with [RFC6282].

[The explanatory text to be removed upon allocation of the addresses, except for the note about the distinct multicast addresses.]

13. Acknowledgements

Brian Frank was a contributor to and co-author of previous drafts of this specification.

Special thanks to Peter Bigot, Esko Dijk and Cullen Jennings for substantial contributions to the ideas and text in the document, along with countless detailed reviews and discussions.

Thanks to Ed Beroset, Angelo P. Castellani, Gilbert Clark, Robert Cragie, Esko Dijk, Lisa Dusseault, Mehmet Ersue, Thomas Fossati, Tom Herbst, Richard Kelsey, Ari Keranen, Matthias Kovatsch, Salvatore Loreto, Kerry Lynn, Alexey Melnikov, Guido Moritz, Petri Mutka, Colin O'Flynn, Charles Palmer, Adriano Pezzuto, Robert Quattlebaum, Akbar Rahman, Eric Rescorla, Dan Romascanu, David Ryan, Szymon Sasin, Michael Scharf, Dale Seed, Robby Simpson, Peter van der Stok, Michael Stuber, Linyi Tian, Gilman Tolle, Matthieu Vial and Alper Yegin for helpful comments and discussions that have shaped the document.

Special thanks also to the IESG reviewers, Adrian Farrel, Martin Stiemerling, Pete Resnick, Richard Barnes, Sean Turner, Spencer Dawkins, Stephen Farrell, and Ted Lemon, who contributed in-depth reviews.

Some of the text has been borrowed from the working documents of the IETF httpbis working group.

14. References

14.1. Normative References

[I-D.ietf-tls-oob-pubkey]
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-07 (work in progress), February 2013.

[I-D.mcgrew-tls-aes-ccm-ecc]
McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM ECC Cipher Suites for TLS", draft-mcgrew-tls-aes-ccm-ecc-06 (work in progress), February 2013.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC2045] Freed, N. and N.S. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

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

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

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

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3676] Gellens, R., "The Text/Plain Format and DelSp Parameters", RFC 3676, February 2004.

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

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[RFC5147] Wilde, E. and M. Duerst, "URI Fragment Identifiers for the text/plain Media Type", RFC 5147, April 2008.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

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

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, April 2013.

14.2. Informative References

[EUI64] , "GUIDELINES FOR 64-BIT GLOBAL IDENTIFIER (EUI-64) REGISTRATION AUTHORITY", April 2010, <http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>.

[EXIMIME] , "Efficient XML Interchange (EXI) Format 1.0", December 2009, <http://www.w3.org/TR/2009/CR-exi-20091208/#mediaTypeRegistration>.

[HHGTTG] Adams, D., "The Hitchhiker’s Guide to the Galaxy", October 1979.

[I-D.allman-tcpm-rto-consider] Allman, M., "Retransmission Timeout Considerations", draft-allman-tcpm-rto-consider-01 (work in progress), May 2012.

[I-D.bormann-coap-misc] Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-22 (work in progress), December 2012.

[I-D.bormann-core-ipsec-for-coap] Bormann, C., "Using CoAP with IPsec", draft-bormann-core-ipsec-for-coap-00 (work in progress), December 2012.

[I-D.castellani-core-http-mapping] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-http-mapping-07 (work in progress), February 2013.

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-10 (work in progress), October 2012.

[I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-06 (work in progress), April 2013.

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

[I-D.ietf-lwig-terminology]
Bormann, C., Ersue, M., and A. Keraenen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-04 (work in progress), April 2013.

[I-D.ietf-tls-multiple-cert-status-extension]
Pettersen, Y., "The TLS Multiple Certificate Status Request Extension", draft-ietf-tls-multiple-cert-status-extension-08 (work in progress), April 2013.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.

[RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, July 2004.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[RFC5489] Badra, M. and I. Hajjeh, "ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)", RFC 5489, March 2009.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, September 2011.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA)

Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July 2012.

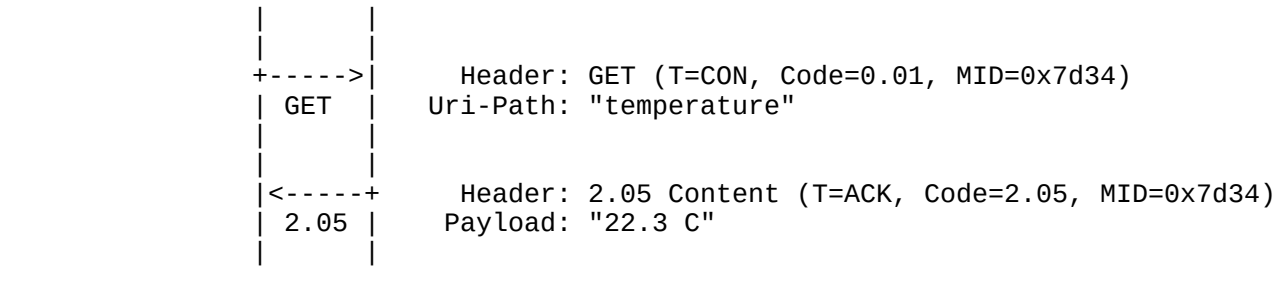
[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, April 2013.

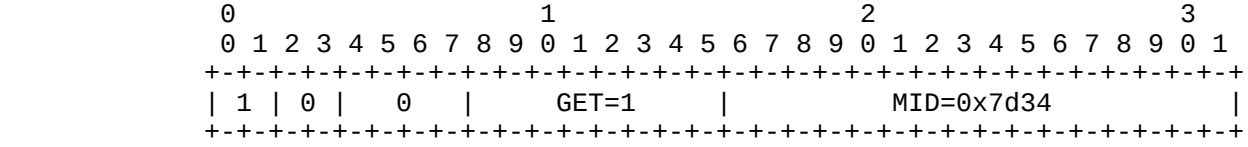
[W3CXMLSEC] Wenning, R., "Report of the XML Security PAG", October 2012, <http://www.w3.org/2011/xmlsec-pag/pagreport.html>.

Appendix A. Examples

This section gives a number of short examples with message flows for GET requests. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and multicast.

Figure 16 shows a basic GET request causing a piggy-backed response: The client sends a Confirmable GET request for the resource coap://server/temperature to the server with a Message ID of 0x7d34. The request includes one Uri-Path Option (Delta 0 + 11 = 11, Length 11, Value "temperature"); the Token is left empty. This request is a total of 16 bytes long. A 2.05 (Content) response is returned in the Acknowledgement message that acknowledges the Confirmable request, echoing both the Message ID 0x7d34 and the empty Token value. The response includes a Payload of "22.3 C" and is 11 bytes long.





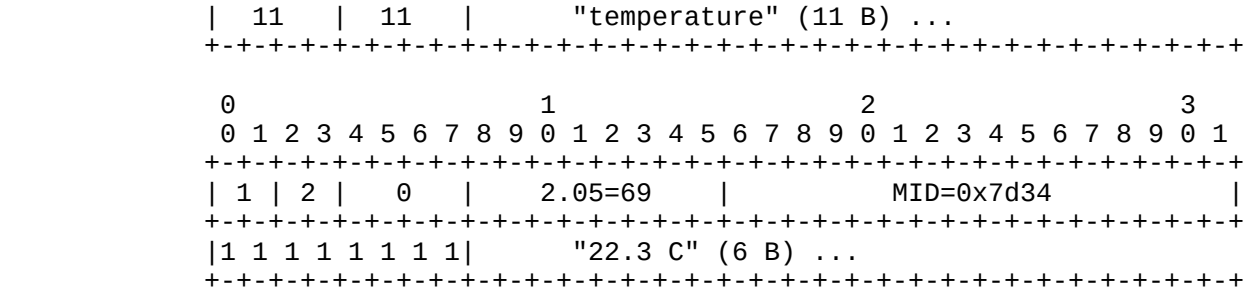
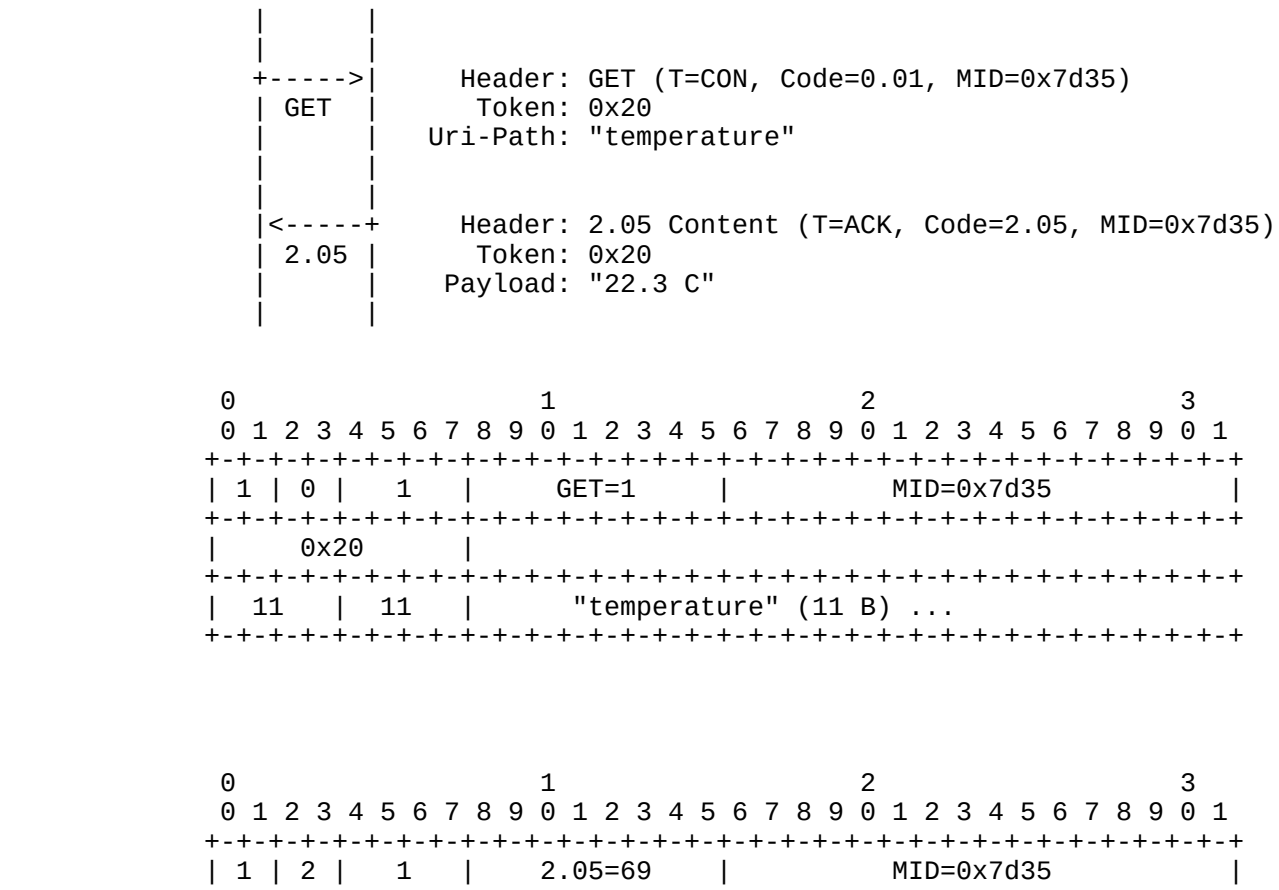


Figure 16: Confirmable request; piggy-backed response

Figure 17 shows a similar example, but with the inclusion of an non-empty Token (Value 0x20) in the request and the response, increasing the sizes to 17 and 12 bytes, respectively.



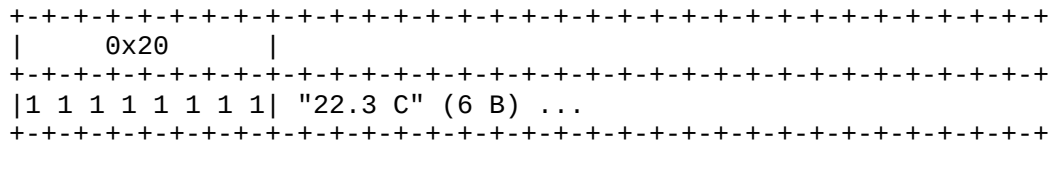


Figure 17: Confirmable request; piggy-backed response

In Figure 18, the Confirmable GET request is lost. After ACK_TIMEOUT seconds, the client retransmits the request, resulting in a piggy-backed response as in the previous example.

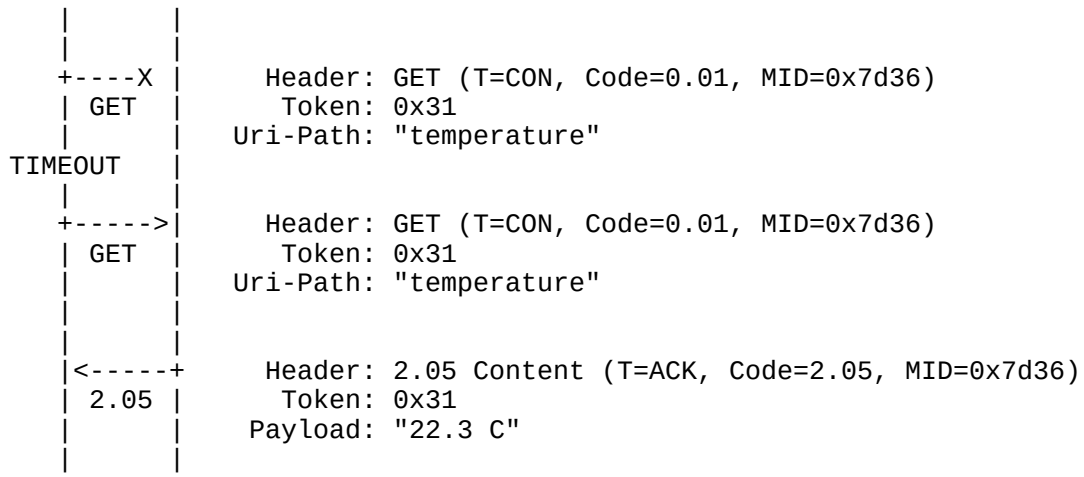
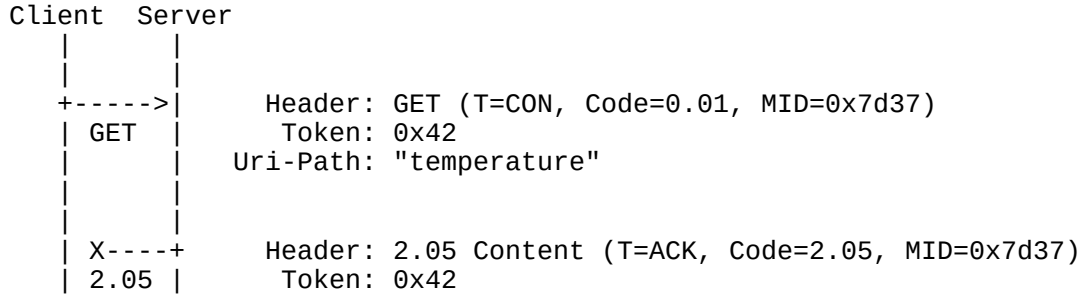


Figure 18: Confirmable request (retransmitted); piggy-backed response

In Figure 19, the first Acknowledgement message from the server to the client is lost. After ACK_TIMEOUT seconds, the client retransmits the request.



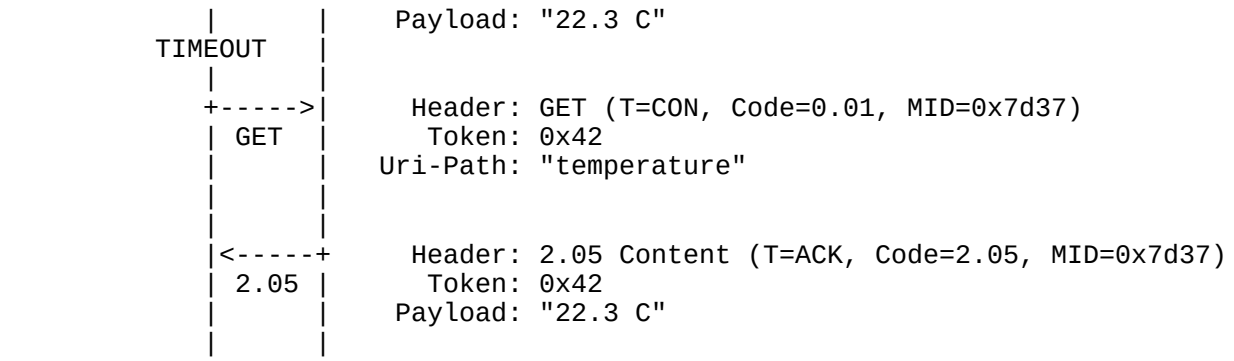


Figure 19: Confirmable request; piggy-backed response (retransmitted)

In Figure 20, the server acknowledges the Confirmable request and sends a 2.05 (Content) response separately in a Confirmable message. Note that the Acknowledgement message and the Confirmable response do not necessarily arrive in the same order as they were sent. The client acknowledges the Confirmable response.

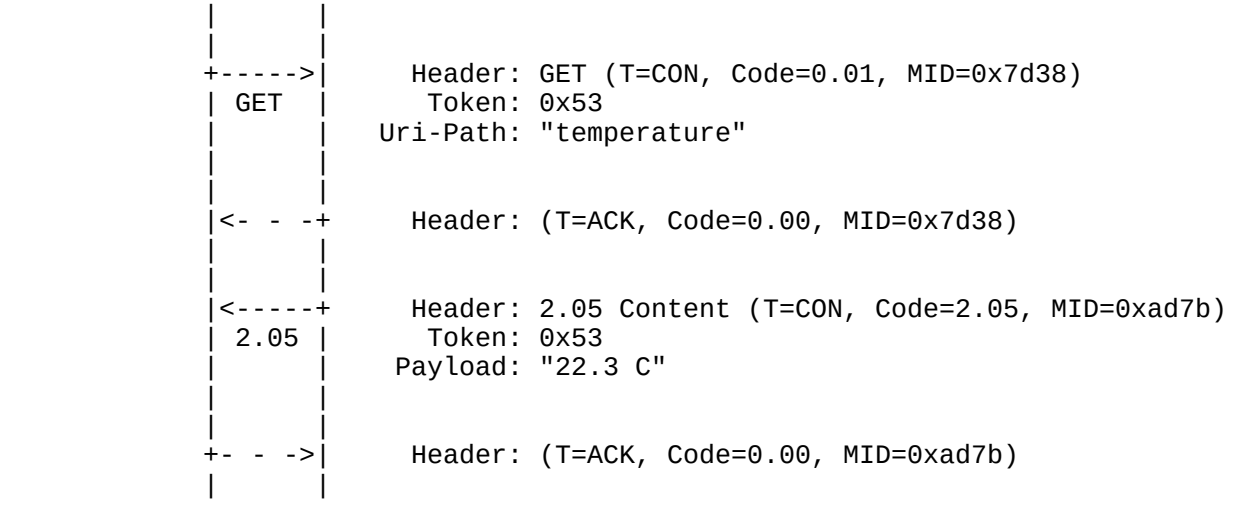


Figure 20: Confirmable request; separate response

Figure 21 shows an example where the client loses its state (e.g., crashes and is rebooted) right after sending a Confirmable request, so the separate response arriving some time later comes unexpected. In this case, the client rejects the Confirmable response with a Reset message. Note that the unexpected ACK is silently ignored.

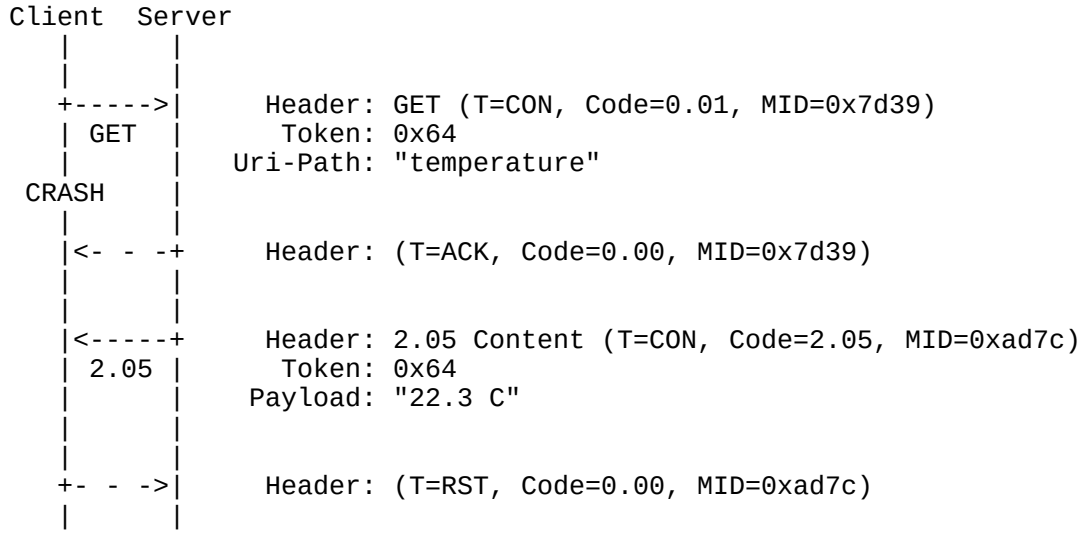


Figure 21: Confirmable request; separate response (unexpected)

Figure 22 shows a basic GET request where the request and the response are Non-confirmable, so both may be lost without notice.

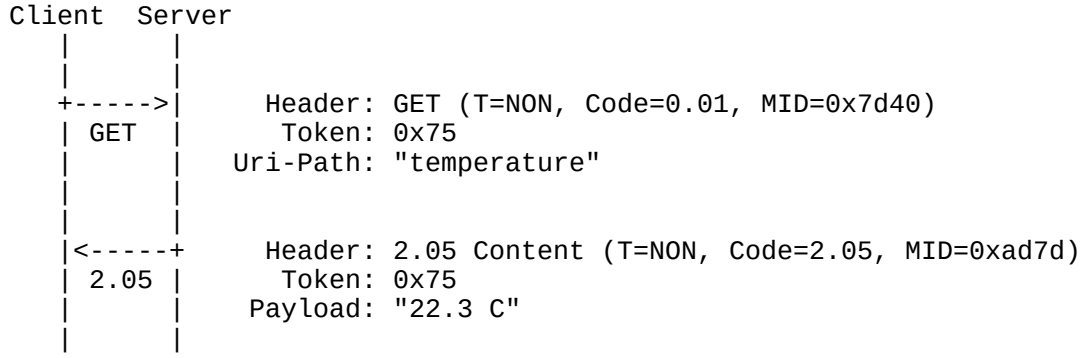


Figure 22: Non-confirmable request; Non-confirmable response

In Figure 23, the client sends a Non-confirmable GET request to a multicast address: all nodes in link-local scope. There are 3 servers on the link: A, B and C. Servers A and B have a matching resource, therefore they send back a Non-confirmable 2.05 (Content) response. The response sent by B is lost. C does not have matching response, therefore it sends a Non-confirmable 4.04 (Not Found) response.

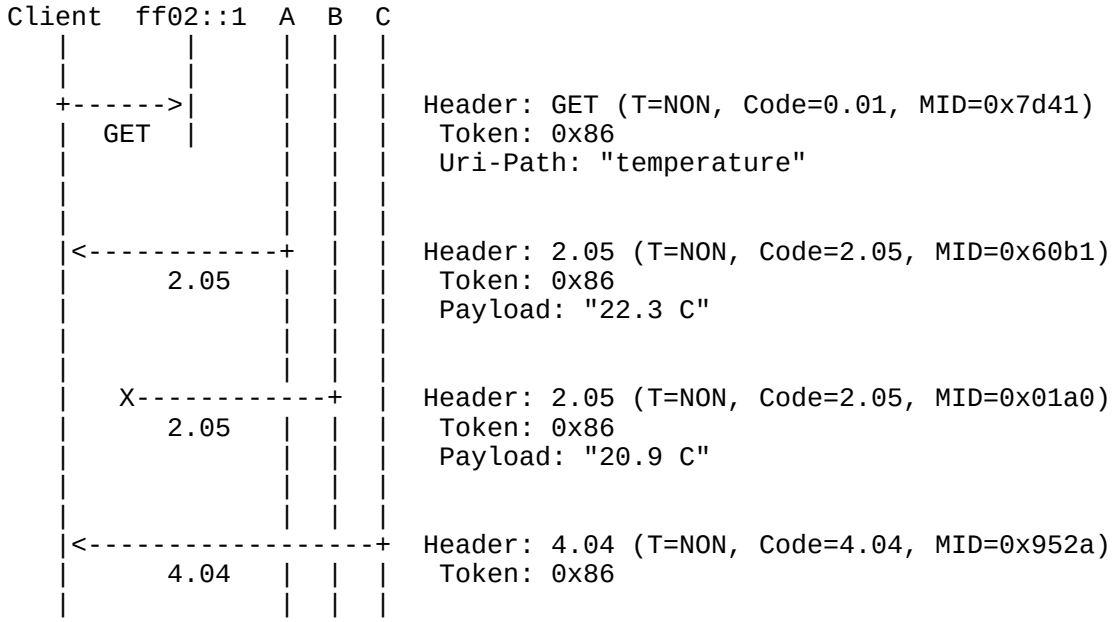


Figure 23: Non-confirmable request (multicast); Non-confirmable response

Appendix B. URI Examples

The following examples demonstrate different sets of Uri options, and the result after constructing an URI from them. In addition to the options, Section 6.5 refers to the destination IP address and port, but not all paths of the algorithm cause the destination IP address and port to be included in the URI.

o Input:

Destination IP Address = [2001:db8::2:1]
Destination UDP Port = 5683

Output:

coap://[2001:db8::2:1]/

o Input:

Destination IP Address = [2001:db8::2:1]
Destination UDP Port = 5683
Uri-Host = "example.net"

Output:

coap://example.net/

- Input:

  Destination IP Address = [2001:db8::2:1]
  Destination UDP Port = 5683
  Uri-Host = "example.net"
  Uri-Path = ".well-known"
  Uri-Path = "core"

  Output:

  coap://example.net/.well-known/core

- Input:

  Destination IP Address = [2001:db8::2:1]
  Destination UDP Port = 5683
  Uri-Host = "xn--18j4d.example"
  Uri-Path = the string composed of the Unicode characters U+3053 U+3093 U+306b U+3061 U+306f, usually represented in UTF-8 as E38193E38293E381ABE381A1E381AF hexadecimal

  Output:

  coap://xn--18j4d.example/
  %E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF

  (The line break has been inserted for readability; it is not part of the URI.)

- Input:

  Destination IP Address = 198.51.100.1
  Destination UDP Port = 61616
  Uri-Path = ""
  Uri-Path = "/"
  Uri-Path = ""
  Uri-Path = ""
  Uri-Query = "//"
  Uri-Query = "?&"

  Output:

  coap://198.51.100.1:61616//%2F//?%2F%2F&?%26

Appendix C. Changelog

(To be removed by RFC editor before publication.)

Changes from ietf-17 to ietf-18: Address comments from the IESG reviews.

- Accept is now critical.
- Add Size1 option for 4.13 responses.

Changes from ietf-15 to ietf-16: Address comments from the IESG reviews. These should not impact interoperability.

- Clarify that once there has been an empty ACK, all further ACKs to the same message also must be empty (#301).
- Define Cache-key properly (#302).
- Clarify that ACKs don’t get retransmitted, the CONs do (#303).
- Clarify: NON is like separate for CON (#304).
- Don’t use decimal response codes, keep the 3+5 structure throughout (#305).
- RFC 2119 usage in 4.5 (#306) and 8.2 (#307).
- Ensure all protocol reactions to reserved or prohibited values are defined (#308).
- URI matching rules may be scheme specific (#309).
- Don’t dally beyond MAX_TRANSMIT_SPAN during retransmission (#310).
- More about selecting a token length for anti-spoofing (#311).
- Discuss spoofing ACKs (#312).
- Qualify partial discard strategy implementation note as UDP only (#313).
- Explicitly point out that UDP and DTLS don’t mix (#314).
- Point out security consideration re URIs and access control (#315).
- Point to RFC5280 section 6 (#316).

- Add a paragraph about cert status checking (#317).
- RSA is out, ECDHE is in for cert-with-PSK, too (#318).
- Point out that requests and responses don't always come in pairs (#319).
- Clarify when there is a need for Unicode normalization (#320).
- Point out that Uri-Host doesn't handle user-part (#321).
- Clarify the use of non-FQDN Authority Names in certificates.
- Numerous editorial improvements and clarifications.

Changes from ietf-14 to ietf-15: Address comments from IETF last-call, mostly implementation notes and editorial improvements. These should not impact interoperability.

- Clarify bytes/characters and UTF-8/ASCII in "Decomposing URIs into Options" (#282).
- Make reference to ECC/CCM DTLS ciphersuite normative (#286).
- Add a quick warning that bytewise scanning for a payload marker is not a good idea (#287).
- Make reference to PROBING_RATE explicit for saturation discussion (#288).
- Mention PROCESSING_DELAY when discussion piggy-backing (#290).
- Various editorial nits: Clarify use of noun "service" (#283), Reference terminology from lwig-terminology (#284), make reference to HTTP terms more explicit (#285), add a forward reference to 5.9.2.9 (#289), 8 kbit/s is not "conservative" (#291).
- Add description of resource depletion attack (#292).
- Add description of DoS attack on congestion control (#293).
- Add discussion of using non-trivial token for protecting against hijacking (#294).
- Clarify implementation note about per-destination Message ID generation.

Changed from ietf-13 to ietf-14:

- Made Accept option non-repeatable.
- Clarified that Safe options in a 2.03 Valid response update the cache.
- Clarified that payload sniffing is acceptable only if no Content-Format was supplied.
- Clarified URI examples (Appendix B).
- Numerous editorial improvements and clarifications.

Changed from ietf-12 to ietf-13:

- Simplified message format.
  - Removed the OC (Option Count) field in the CoAP Header.
  - Changed the End-of-Options Marker into the Payload Marker.
  - Changed the format of Options: use 4 bits for option length and delta; insert one or two additional bytes after the option header if necessary.
  - Promoted the Token Option to a field following the CoAP Header.
- Clarified when a payload is a diagnostic payload (#264).
- Moved IPsec discussion to separate draft (#262).
- Added a reference to a separate draft on reverse-proxy URI embedding (#259).
- Clarified the use of ETags and of 2.03 responses (#265, #254, #256).
- Added reserved Location-* numbers and clarified Location-*.
- Added Proxy-Scheme proposal.
- Clarified terms such as content negotiation, selected representation, representation-format, message format error.
- Numerous clarifications and a few bugfixes.

Changed from ietf-11 to ietf-12:

- Extended options to support lengths of up to 1034 bytes (#202).

- Added new Jump mechanism for options and removed Fenceposting (#214).
- Added new IANA option number registration policy (#214).
- Added Proxy Unsafe/Safe and Cache-Key masking to option numbers (#241).
- Re-numbered option numbers to use Unsafe/Safe and Cache-Key compliant numbers (#241).
- Defined NSTART and restricted the value to 1 with a MUST (#215).
- Defined PROBING_RATE and set it to 1 Byte/second (#215).
- Defined DEFAULT_LEISURE (#246).
- Renamed Content-Type into Content-Format, and Media Type registry into Content-Format registry.
- A large number of small editorial changes, clarifications and improvements have been made.

Changed from ietf-10 to ietf-11:

- Expanded section 4.8 on Transmission Parameters, and used the derived values defined there (#201). Changed parameter names to be shorter and more to the point.
- Several more small editorial changes, clarifications and improvements have been made.

Changed from ietf-09 to ietf-10:

- Option deltas are restricted to 0 to 14; the option delta 15 is used exclusively for the end-of-options marker (#239).
- Option numbers that are a multiple of 14 are not reserved, but are required to have an empty default value (#212).
- Fixed misleading language that was introduced in 5.10.2 in coap-07 re Uri-Host and Uri-Port (#208).
- Segments and arguments can have a length of zero characters (#213).
- The Location-* options describe together describe one location. The location is a relative URI, not an "absolute path URI" (#218).

- The value of the Location-Path Option must not be '.' or '..' (#218).
- Added a sentence on constructing URIs from Location-* options (#231).
- Reserved option numbers for future Location-* options (#230).
- Fixed response codes with payload inconsistency (#233).
- Added advice on default values for critical options (#207).
- Clarified use of identifiers in RawPublicKey Mode Provisioning (#222).
- Moved "Securing CoAP" out of the "Security Considerations" (#229).
- Added "All CoAP Nodes" multicast addresses to "IANA Considerations" (#216).
- Over 100 small editorial changes, clarifications and improvements have been made.

Changed from ietf-08 to ietf-09:

- Improved consistency of statements about RST on NON: RST is a valid response to a NON message (#183).
- Clarified that the protocol constants can be configured for specific application environments.
- Added implementation note recommending piggy-backing whenever possible (#182).
- Added a content-encoding column to the media type registry (#181).
- Minor improvements to Appendix D.
- Added text about multicast response suppression (#177).
- Included the new End-of-options Marker (#176).
- Added a reference to draft-ietf-tls-oob-pubkey and updated the RPK text accordingly.

Changed from ietf-07 to ietf-08:

- Clarified matching rules for messages (#175)

- Fixed a bug in Section 8.2.2 on Etags (#168)
- Added an IP address spoofing threat analysis contribution (#167)
- Re-focused the security section on raw public keys (#166)
- Added an 4.06 error to Accept (#165)

Changed from ietf-06 to ietf-07:

- application/link-format added to Media types registration (#160)
- Moved content-type attribute to the document from link-format.
- Added coaps scheme and DTLS-secured CoAP default port (#154)
- Allowed 0-length Content-type options (#150)
- Added congestion control recommendations (#153)
- Improved text on PUT/POST response payloads (#149)
- Added an Accept option for content-negotiation (#163)
- Added If-Match and If-None-Match options (#155)
- Improved Token Option explanation (#147)
- Clarified mandatory to implement security (#156)
- Added first come first server policy for 2-byte Media type codes (#161)
- Clarify matching rules for messages and tokens (#151)
- Changed OPTIONS and TRACE to always return 501 in HTTP-CoAP mapping (#164)

Changed from ietf-05 to ietf-06:

- HTTP mapping section improved with the minimal protocol standard text for CoAP-HTTP and HTTP-CoAP forward proxying (#137).
- Eradicated percent-encoding by including one Uri-Query Option per &-delimited argument in a query.
- Allowed RST message in reply to a NON message with unexpected token (#135).

- Cache Invalidation only happens upon successful responses (#134).
- 50% jitter added to the initial retransmit timer (#142).
- DTLS cipher suites aligned with ZigBee IP, DTLS clarified as default CoAP security mechanism (#138, #139)
- Added a minimal reference to draft-kivinen-ipsecme-ikev2-minimal (#140).
- Clarified the comparison of UTF-8s (#136).
- Minimized the initial media type registry (#101).

Changed from ietf-04 to ietf-05:

- Renamed Immediate into Piggy-backed and Deferred into Separate -- should finally end the confusion on what this is about.
- GET requests now return a 2.05 (Content) response instead of 2.00 (OK) response (#104).
- Added text to allow 2.02 (Deleted) responses in reply to POST requests (#105).
- Improved message deduplication rules (#106).
- Section added on message size implementation considerations (#103).
- Clarification made on human readable error payloads (#109).
- Definition of CoAP methods improved (#108).
- Max-Age removed from requests (#107).
- Clarified uniqueness of tokens (#112).
- Location-Query Option added (#113).
- ETag length set to 1-8 bytes (#123).
- Clarified relation between elective/critical and option numbers (#110).
- Defined when to update Version header field (#111).
- URI scheme registration improved (#102).

- Added review guidelines for new CoAP codes and numbers.

Changes from ietf-03 to ietf-04:

- Major document reorganization (#51, #63, #71, #81).
- Max-age length set to 0-4 bytes (#30).
- Added variable unsigned integer definition (#31).
- Clarification made on human readable error payloads (#50).
- Definition of POST improved (#52).
- Token length changed to 0-8 bytes (#53).
- Section added on multiplexing CoAP, DTLS and STUN (#56).
- Added cross-protocol attack considerations (#61).
- Used new Immediate/Deferred response definitions (#73).
- Improved request/response matching rules (#74).
- Removed unnecessary media types and added recommendations for their use in M2M (#76).
- Response codes changed to base 32 coding, new Y.XX naming (#77).
- References updated as per AD review (#79).
- IANA section completed (#80).
- Proxy-Uri Option added to disambiguate between proxy and non-proxy requests (#82).
- Added text on critical options in cached states (#83).
- HTTP mapping sections improved (#88).
- Added text on reverse proxies (#72).
- Some security text on multicast added (#54).
- Trust model text added to introduction (#58, #60).
- AES-CCM vs. AES-CCB text added (#55).

- Text added about device capabilities (#59).
- DTLS section improvements (#87).
- Caching semantics aligned with RFC2616 (#78).
- Uri-Path Option split into multiple path segments.
- MAX_RETRANSMIT changed to 4 to adjust for RESPONSE_TIME = 2.

Changes from ietf-02 to ietf-03:

- Token Option and related use in asynchronous requests added (#25).
- CoAP specific error codes added (#26).
- Erroring out on unknown critical options changed to a MUST (#27).
- Uri-Query Option added.
- Terminology and definitions of URIs improved.
- Security section completed (#22).

Changes from ietf-01 to ietf-02:

- Sending an error on a critical option clarified (#18).
- Clarification on behavior of PUT and idempotent operations (#19).
- Use of Uri-Authority clarified along with server processing rules; Uri-Scheme Option removed (#20, #23).
- Resource discovery section removed to a separate CoRE Link Format draft (#21).
- Initial security section outline added.

Changes from ietf-00 to ietf-01:

- New cleaner transaction message model and header (#5).
- Removed subscription while being designed (#1).
- Section 2 re-written (#3).
- Text added about use of short URIs (#4).

- Improved header option scheme (#5, #14).
- Date option removed whiled being designed (#6).
- New text for CoAP default port (#7).
- Completed proxying section (#8).
- Completed resource discovery section (#9).
- Completed HTTP mapping section (#10).
- Several new examples added (#11).
- URI split into 3 options (#12).
- MIME type defined for link-format (#13, #16).
- New text on maximum message size (#15).
- Location Option added.

Changes from shelby-01 to ietf-00:

- Removed the TCP binding section, left open for the future.
- Fixed a bug in the example.
- Marked current Sub/Notify as (Experimental) while under WG discussion.
- Fixed maximum datagram size to 1280 for both IPv4 and IPv6 (for CoAP-CoAP proxying to work).
- Temporarily removed the Magic Byte header as TCP is no longer included as a binding.
- Removed the Uri-code Option as different URI encoding schemes are being discussed.
- Changed the rel= field to desc= for resource discovery.
- Changed the maximum message size to 1024 bytes to allow for IP/UDP headers.
- Made the URI slash optimization and method idempotence MUSTs
- Minor editing and bug fixing.

Changes from shelby-00 to shelby-01:

- Unified the message header and added a notify message type.
- Renamed methods with HTTP names and removed the NOTIFY method.
- Added a number of options field to the header.
- Combines the Option Type and Length into an 8-bit field.
- Added the magic byte header.
- Added new ETag Option.
- Added new Date Option.
- Added new Subscription Option.
- Completed the HTTP Code - CoAP Code mapping table appendix.
- Completed the Content-type Identifier appendix and tables.
- Added more simplifications for URI support.
- Initial subscription and discovery sections.
- A Flag requirements simplified.

Authors' Addresses

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

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

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

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

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

CoRE
Internet-Draft
Intended status: Standards Track
Expires: September 15, 2011

Z. Shelby
Sensinode

March 14, 2011

CoRE Link Format
draft-ietf-core-link-format-03

Abstract

This document defines Web Linking using a link format for use by constrained web servers to describe hosted resources, their attributes and other relationships between links. Based on the HTTP Link Header format defined in RFC5988, the CoRE Link Format is carried as a payload and is assigned an Internet media type. A well-known URI is defined as a default entry-point for requesting the links hosted by a server.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 15, 2011.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Web Linking in CoRE 3
1.2. Use Cases 4
1.2.1. Discovery 4
1.2.2. Resource Collections 5
1.2.3. Resource Directory 5
1.3. Terminology 5
2. Link Format 6
2.1. Target and context URIs 7
2.2. Link relations 7
2.3. Use of anchors 8
3. CoRE link extensions 8
3.1. Resource type 'rt' attribute 8
3.2. Interface description 'if' attribute 9
3.3. Content-type code 'ct' attribute 9
3.4. Maximum size estimate 'sz' attribute 9
4. Well-known Interface 10
4.1. Query Filtering 10
5. Examples 11
6. Security Considerations 13
7. IANA Considerations 13
7.1. Attribute Registry 13
7.2. Well-known 'core' URI 14
7.3. New 'hosts' relation type 14
7.4. New link-format Internet media type 15
8. Acknowledgments 15
9. Changelog 16
10. References 18
10.1. Normative References 18
10.2. Informative References 18
Author's Address 19

1. Introduction

The Constrained RESTful Environments (CoRE) working group aims at realizing the Representational State Transfer (REST) architecture [REST] in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited memoryt) and networks (e.g. 6LoWPAN [RFC4944]). CoRE is aimed at Machine-to-Machine (M2M) applications such as smart energy and building automation.

The discovery of resources hosted 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 [RFC2616] Web Server is typically called Web Discovery and the description of relations between resources is called Web Linking [RFC5988]. In this document we refer to the discovery of resources hosted by a constrained web server, their attributes and other resource relations as CoRE Resource Discovery.

The main function of such a discovery mechanism is to provide Universal Resource Indicators (URIs, called links) for the resources hosted by the server, complemented by attributes about those resources and possible further link relations. In CoRE this collection of links is carried as a resource of its own (as opposed to HTTP headers delivered with a specific resource). This document specifies a link format for use in CoRE Resource Discovery by extending the HTTP Link Header Format [RFC5988] to describe these link descriptions. The CoRE Link Format is carried as a payload and is assigned an Internet media type. A well-known URI "/.well-known/core" is defined as a default entry-point for requesting the list of links about resources hosted by a server, and thus performing CoRE Resource Discovery.

1.1. Web Linking in CoRE

What is the difference between the CoRE Link Format and [RFC5988]? Technically the CoRE Link Format is a serialization of a typed link as specified in [RFC5988], used to describe relationships between resources, so-called "Web Linking". In this specification Web Linking is extended with specific constrained M2M attributes, links are carried as a message payload rather than in an HTTP Link Header, and a default interface is defined to discover resources hosted by a server. This specification also defines a new relation type "hosts", which indicates that the resource is hosted by the server from which the link document was requested.

Why not just use the HTTP Link Header? In HTTP, the Link Header can be used to carry link information about a resource along with an HTTP

response. This works well for the typical use case for a web server and browser, where further information about a particular resource is useful after accessing it. In CoRE the main use case for Web Linking is the discovery of which resources a server hosts in the first place. Although some resources may have further links associated with them, this is expected to be an exception. For that reason the CoRE Link Format serialization is carried as a resource representation of a well-known URI. The CoRE Link Format does re-use the format of the HTTP Link Header serialization defined in [RFC5988].

1.2. Use Cases

Typical use cases for Web Linking on today's web include e.g. describing the author of a web page, or describing relations between web pages (next chapter, previous chapter etc.). Web Linking can also be applied to M2M applications, where typed links are used to assist a machine client in finding and understanding how to use resources on a server. In this section a few use cases are described for how the CoRE Link Format could be used in M2M applications. For further technical examples see Section 5. As there are a large range of M2M applications, these use cases are purposely generic. This document assumes that different deployments or application domains will define the appropraite REST interface descriptions along with Resource Types to make discovery meaningful.

1.2.1. Discovery

In M2M application, for example home or building automation, there is a need for local clients and servers to find and interact with each other without human intervention. The CoRE Link Format can be used by servers in such environments to enable Resource Discovery of the resources hosted by the server.

Resource Discovery can be performed either unicast or multicast. When a server's IP address is already known, either a priori or resolved via the Domain Name System (DNS), unicast disovery is performed in order to locate the entry point to the resource of interest. This is performed using a GET to /.well-known/core on the server, which returns a payload in the CoRE Link Format. A client would then match the appropriate Resource Type, Interface Description and possible Content-Type for its application. These attributes may also be included in the query string in order to filter the number of links returned in a response.

Multicast resource discovery is useful when a client needs to locate a resource within a limited scope, and that scope supports IP multicast. A GET request to the appropriate multicast address is

made for /.well-known/core. In order to limit the number and size or responses, a query string is recommended with the known attributes. Typically a resource would be discovered based on its Resource Type and/or Interface Description, along with possible application specific attributes.

1.2.2. Resource Collections

RESTful designs of M2M interfaces often make use of collections of resources. For example an index of temprature sensors on a data collection node or a list of alarms on a home security controller. The CoRE Link Format can be used to make it possible to find the entry point to a collection and traverse its members. The entry point of a collection would always be included in /.well-known/core to enable its discovery. The members of the collection can be defined either through the interface description of the resource along with a parameter resource for the size of the collection, or by using the link format to describe each resource in the collection. These links could be located under /.well-known/core or hosted for example in the root resource of the collection.

1.2.3. Resource Directory

In many deployment scenarios, for example constrained networks with sleeping servers, or large M2M deployments with bandwidth limited access networks, it makes sense to deploy resource directory entities which store links to resources stored on other servers. Think of this as a limited search engine for constrained M2M resources.

The CoRE Link Format can be used by a server to register resources with a resource directory, or to allow a resource directory to poll for resources. Resource polling uses the same process as unicast or multicast discovery, however usually without filtering. Resource registration can be achived by having each server POST their resources to /.well-known/core on the resource directory. This in turn adds links to the resource directory under an appropriate resource. These links can then be discovered by any client by a performing a GET on the resource directory using a query string filter.

1.3. Terminology

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

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

makes use of the following terminology:

Web Linking
: A framework for indicating the relationships between web resources.

Link
: Also called "typed links" in RFC5988. A link is a typed connection between two resources identified by URIs. Made up of a context URI, a link relation type, a tarfet URI, and optional target attributes.

Link Format
: A particular serialisation of typed links.

CoRE Link Format
: A particular serialization of typed links based the HTTP Link Header serialization defined in Section 5 of RFC5988, but carried as a resource representation with a MIME type.

Attribute
: Properly called "Target Attribute" in RFC5988. A set of key/value pairs that descibe the link or its target.

CoRE Resource Discovery
: When a client discovers the list of resources hosted by a server, their attributes and other link relations by accessing /.well-known/core.

2. Link Format

The CoRE Link Format extends the HTTP Link Header format specified in [RFC5988], which is specified in Augmented Backus-Naur Form (ABNF) notation [RFC5234]. The format does not require special XML or binary parsing, is fairly compact, and is extensible - all important characteristics for CoRE. It should be noted that this link format is just one serialization of typed links defined in [RFC5988], others include HTML link, Atom feed links [RFC4287] or HTTP Link Headers. It is expected that resources discovered in the CoRE Link Format may also be made available in alternative formats on the greater Internet. The CoRE Link Format is only expected to be supported in constrained networks and M2M systems.

Section 5 of [RFC5988] did not require an Internet media type for the defined link format, as it was defined to be carried in an HTTP header. This specification thus defines a Internet media type for the CoRE Link Format (see Section 7.4).

The CoRE link format uses the ABNF description and associated rules in Section 5 of [RFC5988]. In addition, the pchar rule is taken from [RFC3986]. The "Link:" text is omitted as that is part of the HTTP Link Header. As in [RFC5988], multiple link descriptions are separated by commas. Note that commas can also occur in quoted strings and URIs but do not end a description. The CoRE link format MUST use UTF-8 encoding, which SHOULD be in NFC (Unicode Normalization Form C). See Section 3 of [RFC5198], which explains why it useful to represent Unicode in a single unique form.

2.1. Target and context URIs

Each link conveys one target URI as a URI-reference inside angle brackets ("<>"). The context URI of a link (also called base URI in [RFC3986]) conveyed in the CoRE Link Format is by default built from the scheme and authority parts of the target URI. In the absence of this information in the target URI, the context URI is built from the scheme and authority that was used for referencing the resource returning the set of links, replacing the path with an empty path. Thus by default links can be thought of as describing a target resource hosted by the server. Other relations can be expressed by including an anchor parameter (which defines the context URI) along with an explicit relation parameter. This is an important difference to the way the HTTP Link Header format is used, as it is included in the header of an HTTP response for some URI (this URI is by default the context URI). Thus the HTTP Link Header is by default relating the target URI to the URI that was requested. In comparison, the CoRE link format includes one or more links, each describing a resource hosted by a server by default. Other relations can be expressed by using the anchor parameter. See Section 5 of [RFC3986] for a description of how URIs are constructed from URI references.

2.2. Link relations

Since links in the CoRE Link Format are typically used to describe resources hosted by a server, and thus in the absence of the relation parameter the new relation type "hosts" is assumed (see Section 7.3). The "hosts" relation type indicates that the target URI is a resource hosted by the server given by the base URI, or, if present, the anchor parameter.

To express other relations a links can make use of any registered relation parameter or target attributes by including the relation parameter. The context of a relation can be defined using the anchor parameter. In this way, relations between resources hosted on a server, or between hosted resources and external resources can be expressed.

2.3. Use of anchors

As per Section 5.2 of [RFC5988] a link description MAY include an "anchor" attribute, in which case the context is the URI included in that attribute. This is used to describe a relationship between two resources. A consuming implementation can however choose to ignore such links. It is not expected that all implementations will be able to derive useful information from explicitly anchored links.

3. CoRE link extensions

The following CoRE specific target attributes are defined. These attributes describe information useful in accessing the target link of the relation, and in some cases may be URIs. These URIs MUST be treated as indicators, and are not meant to be actually retrieved like a URL. When attributes are compared, they MUST be compared as strings. Relationships to resources that are meant to be retrieved should be expressed as separate links using the anchor attribute and the appropriate relation type.

```
link-extension    = ( "rt" "=" quoted-string )
link-extension    = ( "if" "=" quoted-string )
link-extension    = ( "ct" "=" integer )
link-extension    = ( "sz" "=" integer )
integer           = 1*DIGIT
```

3.1. Resource type ’rt’ attribute

The resource type "rt" attribute is used to assign a semantically important type to a resource. One can think of this as a noun describing the resource. In the case of a temperature resource this could be an application-specific semantic type like "OutdoorTemperature", a Universal Resource Name (URN) like "urn:temperature:outdoor" or a URI referencing a specific concept in an ontology like "http://sweet.jpl.nasa.gov/2.0/phys.owl#Temperature". Multiple resource type attributes MAY appear in a link.

The resource type attribute is not meant to used to assign a human readable name to a resource. The "title" attribute defined in [RFC5988] is meant for that purpose.

3.2. Interface description 'if' attribute

The interface description "if" attribute is used to provide a name, URI or URN indicating a specific interface definition used to interact with the target resource. One can think of this as describing verbs usable on a resource. The interface description attribute is meant to describe the generic REST interface to interact with a resource or a set of resources. It is expected that an interface description will be re-used by different resource types. For example the resource types "OutdoorTemperature", "DewPoint" and "RelHumidity" could all be accessible using the interface description "http://www.example.org/myapp.wadl#sensor".

The interface description could be for example the URI of a Web Application Description Language (WADL) definition of the target resource "http://www.example.org/myapp.wadl#sensor", a URN indicating the type of interface to the resource "urn:myapp:sensor", or an application-specific name "Sensor". Multiple interface description attributes MAY appear in a link.

3.3. Content-type code 'ct' attribute

The Content-type code "ct" attribute provides a hint about the Internet media type 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 [I-D.ietf-core-coap]. 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 MUST NOT appear more than once in a link.

Alternatively, the "type" attribute MAY be used to indicate an Internet media type as a quoted-string [RFC5988]. It is not however expected that constrained implementations are able to parse quoted-string Content-type values. A link MAY include either a ct attribute or a type attribute, but MUST NOT include both.

3.4. Maximum size estimate 'sz' attribute

The maximum size estimate attribute "sz" gives an indication of the maximum size of the resource indicated by the target URI. This attribute is not expected to be included for small resources that can comfortably by carried in a single Maxiumum Transmission Unit (MTU), but SHOULD be included for resources larger than that. The maximum size estimate attribute MUST NOT appear more than once in a link.

4. Well-known Interface

Resource discovery in CoRE is accomplished through the use of a well-known resource URI which returns a list of links about resources hosted by that server and other link relations. Well-known resources have a path component that begins with "/.well-known/" as specified in [RFC5785]. This document defines a new well-known resource for CoRE Resource Discovery "/.well-known/core".

A server implementing this specification MUST support this resource on the default port appropriate for the protocol for the purpose of resource discovery. It is however up to the application which links are included and how they are organized. The resource /.well-known/core is meant to be used to return links to the entry points of resource interfaces on a server. More sophisticated link organization can be achieved by including links to CoRE Link Format resources located elsewhere on the server, for example to achieve an index. In the absence of any links, a zero-length payload is returned. The resource representation of this resource MUST be the CoRE Link Format described in Section 2.

The CoRE resource discovery interface supports the following interactions:

- Performing a GET on /.well-known/core to the default port returns a set of links available from the server (if any) in the CoRE Link Format. These links might describe resources hosted on that server, on other servers, or express other kinds of link relations as described in Section 2.

- Filtering may be performed on any of the link format attributes using a query string as specified in Section 4.1. For example [GET /.well-known/core?n=TemperatureC] would request resources with the name TemperatureC. A server is not however required to support filtering.

- More capable servers such as proxies could support a resource directory by requesting the resource descriptions of other end-points or allowing servers to POST requests to /.well-known/core. The details of such resource directory functionality is however out of scope for this document, and is expected to be specified separately.

4.1. Query Filtering

A server implementing this document MAY recognize the query part of a resource discovery URI as a filter on the resources to be returned. The query part should conform to the following syntax. Note that

this only defines querying for a single parameter at a time.

```
filter-query = resource-param "=" query-pattern
resource-param = "uri" | parmname
query-pattern = 1*pchar [ "*" ]
```

The resource-param "uri" refers to the URI-reference between the "<" and ">" characters of a link. Other resource-param values refer to the link attribute they name. Filtering is performed by comparing the query-pattern against the value of the attribute identified by the resource-param for each link-value in the collection of resources identified by the URI path.

If the decoded query-pattern does not end with "*", a link value matches the query only if the value of the attribute or URI-reference denoted by the resource-param is bytewise identical to the query-pattern. If the decoded query-pattern ends with "*", it is sufficient that the remainder of the query-pattern be a prefix of the value denoted by the resource-param. It is not expected that very constrained nodes support filtering. Implementations not supporting filtering MUST simply ignore the query string and return the whole resource for unicast requests.

When using a transfer protocol like the Constrained Application Protocol (CoAP) that supports multicast requests, special care is taken. A multicast request with a query string MUST not be responded to if filtering is not supported (to avoid a needless response storm).

5. Examples

A few examples of typical link descriptions in this format follows. Multiple resource descriptions in a representation are separated by commas. Linefeeds never occur in the actual format, but are shown in the example for readability.

This example includes links to two different sensors sharing the same interface description.

REQ: GET /.well-known/core

RES: 200 OK
</sensors/temp>;ct=41;rt="TemperatureC";if="sensor",
</sensors/light>;ct=41;rt="LightLux";if="sensor"

This example arranges link descriptions hierarchically, with the entry point including a link to a sub-resource containing links about the sensors.

```
REQ: GET /.well-known/core

RES: 200 OK
</sensors>;rt="index";ct=40

REQ: GET /sensors

RES: 200 OK
</sensors/temp>;ct=41;rt="TemperatureC";if="sensor",
</sensors/light>;ct=41;rt="LightLux";if="sensor"
```

An example query filter may look like:

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

RES: 200 OK
</sensors/light>;ct=41;rt="LightLux";if="sensor"
```

This example shows the use of an anchor attribute to relate the temperature sensor resource to an external description and to an alternative URL.

```
REQ: GET /.well-known/core

RES: 200 OK
</sensors>;ct=40;rt="index";rt="Sensor Index",
</sensors/temp>;rt="TemperatureC";if="sensor",
</sensors/light>;ct=41;rt="LightLux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

If a client is interested to find relations about a particular resource, it can perform a query on the anchor parameter:

```
REQ: GET /.well-known/core?anchor=/sensors/temp

RES: 200 OK
<http://www.example.com/sensors/temp123>;anchor="/sensors/temp"
;rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

6. Security Considerations

This document needs the same security considerations as described in Section 7 of [RFC5988]. The /.well-known/core resource may be protected e.g. using DTLS when hosted on a CoAP server as per [I-D.ietf-core-coap] Section 10.2.

Multicast requests using CoAP for the well-known link-format resources could be used to perform denial of service on a constrained network. A multicast request SHOULD only be accepted if the request is sufficiently authenticated and secured.

CoRE link format parsers should be aware that a link description may be cyclical, i.e., contain a link to itself. These cyclical links could be direct or indirect (i.e., through referenced link resources). Care should be taken when parsing link descriptions and accessing cyclical links.

7. IANA Considerations

7.1. Attribute Registry

This document defines a registry for the link extension attributes defined for use with the CoRE Link Format. The name of the registry is "CoRE Link Format Attributes".

Each entry in the registry must include the attribute name, the attribute, the format of the attribute and a reference to the attribute's documentation.

Initial entries in this registry are as follows:

Name	Attribute	Type	Reference
Resource Type	rt	Quoted String	&SELF;
Interface Description	if	Quoted String	&SELF;
Content-Type	ct	Integer	&SELF;
Maximum Size Estimate	sz	Integer	&SELF;

Table 1: CoAP Option Numbers

New attributes defined for the CoRE Link Format MUST NOT collide with existing attributes defined in [RFC5988].

The IANA policy for future additions to this registry is "IETF

Review" as described in [RFC5226].

The documentation of an attribute should specify the semantics of the attribute, including the following properties:

- The meaning of the attribute.
- The format of the attribute's value.
- Whether the attribute can occur multiple times.
- The default value, if any.

7.2. Well-known 'core' URI

This memo registers the "core" well-known URI in the Well-Known URI Registry as defined by [RFC5785].

URI suffix: core

Change controller: IETF

Specification document(s): [[this document]]

Related information: None

7.3. New 'hosts' relation type

This memo registers the new "hosts" Web Linking relation type as per [RFC5988].

Relation Name: hosts

Description: Refers to a resource hosted by the server indicated by the link context.

Reference: [[this document]]

Notes: This relation is used in CoRE where links are retrieved as a /.well-known/core resource representation, and by default the context of the links is the server at coap://authority from which /.well-known/core was requested.

Application Data: None

7.4. New link-format Internet media type

This memo registers the a new Internet media type for the CoRE link format, application/link-format.

Type name: application

Subtype name: link-format

Required parameters: None

Optional parameters: The query string may contain uri= to match the URI, or any other attribute defined for the link format to match that attribute as defined in this document.

Encoding considerations: UTF-8 (NFC)

Security considerations: None

Interoperability considerations:

Published specification: [[this document]]

Applications that use this media type: CoAP server and client implementations for resource discovery and HTTP applications that use the link-format as a payload.

Additional information:

Magic number(s):

File extension(s):

Macintosh file type code(s):

Intended usage: COMMON

Restrictions on usage: None

Author: CoRE WG

Change controller: IETF

8. Acknowledgments

Special thanks to Peter Bigot, who has made a considerable number reviews and text contributions that greatly improved the document.

In particular, Peter is responsible for the ABNF descriptions and the idea for a new "hosts" relation type.

Thanks to Mark Nottingham and Eran Hammer-Lahav for the discussions and ideas that led to this draft, and to Carsten Bormann, Martin Thomson and Peter Saint-Andre for extensive comments and contributions that improved the text.

Thanks to Michael Stuber, Richard Kelsey, Cullen Jennings, 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 Beroset, Gilman Tolle, Robby Simpson, Colin O'Flynn and David Ryan for helpful comments and discussions that have shaped the document.

9. Changelog

Changes from ietf-02 to ietf-03:

- Removed 'obs' attribute definition, now defined in the CoAP Observation spec (#99).
- Changed Resource name (n=) to Resource type (rt=) and d= to if= (#121).
- Hierarchical organization of links under /.well-known/core removed (#95).
- Bug in Section 3.1 on byte-wise query matching fixed (#91).
- Explanatory text added about alternative Web link formats (#92).
- Fixed a bug in Section 2.2.4 (#93).
- Added use case examples (#89).
- Clarified how the CoRE link format is used and how it differs from RFC5988 (#90, #98).
- Changed the Interface definition format to quoted-string to match the resource type.
- Added an IANA registry for CoRE Link Format attributes (#100).

Changes from ietf-01 to ietf-02:

- o Added references to RFC5988 (#41).
- o Removed sh and id link-extensions (#42).
- o Defined the use of UTF-8 (#84).
- o Changed query filter definition for any parameter (#70).
- o Added more example, now as a separate section (#43).
- o Mentioned cyclical links in the security section (#57).
- o Removed the sh and id attributes, added obs and sz attributes (#42).
- o Improved the context and relation description wrt RFC5988 and requested a new "hosts" default relation type (#85).

Changes from ietf-00 to ietf-01:

- o Editorial changes to correct references.
- o Formal definition for filter query string.
- o Removed URI-reference option from "n" and "id".
- o Added security text about multicast requests.

Changes from shelby-00 to ietf-00:

- o Fixed the ABNF link-extension definitions (quotes around URIs, integer definition).
- o Clarified that filtering is optional, and the query string is to be ignored if not supported (and the URL path processed as normally).
- o Required support of wildcard * processing if filtering is supported.
- o Removed the aussumption of a default content-type assumption.

10. References

10.1. Normative References

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

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

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

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

10.2. Informative References

[I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[REST] Fielding, "Architectural Styles and the Design of Network-based Software Architectures", , 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

[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.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

Author's Address

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach@sensinode.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 16, 2011

K. Hartke
Universitaet Bremen TZI
Z. Shelby
Sensinode
March 15, 2011

Observing Resources in CoAP
draft-ietf-core-observe-02

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 specification provides a simple 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 September 16, 2011.

Table of Contents

1. Introduction 3
2. Overview 3
3. Observation Relationships 5
3.1. Establishment 5
3.2. Maintenance 6
3.3. Termination 6
4. Notifications 7
4.1. Strategies 8
4.2. Retransmission 8
4.3. Reordering 9
4.4. Caching 10
5. Observe Option 10
6. Interactions with other CoAP features 11
6.1. Request Methods 11
6.2. Block-wise Transfers 11
6.3. Resource Discovery 12
7. Security Considerations 12
8. IANA Considerations 13
9. Acknowledgements 13
10. References 13
10.1. Normative References 13
10.2. Informative References 14
Appendix A. Examples 15
A.1. Proxying 16
Appendix B. Changelog 18
Authors' Addresses 19

1. Introduction

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 state of a resource on a CoAP server can change over time. We want to give CoAP clients the ability to observe this change. However, existing approaches from the HTTP world, such as repeated polling or long-polls, generate significant complexity and/or overhead and thus are less applicable in the constrained CoAP world. Instead, a much simpler mechanism is provided to solve the basic problem of resource observation. Note that there is no intention for this mechanism to solve the full set of problems that the existing HTTP solutions solve, or to replace publish/subscribe networks that solve a much more general problem [RFC5989].

This specification describes an architecture and a protocol design that realizes the well-known subject/observer design pattern within the REST-based environment of CoAP.

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

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

2. Overview

In the subject/observer design pattern, an object, called the subject, maintains a list of interested parties, called observers, and notifies them automatically when a predefined condition, event or state change occurs. The subject provides a way for observers to register themselves with the subject. This pattern supports a clean separation between components, such as data storage and user interface.

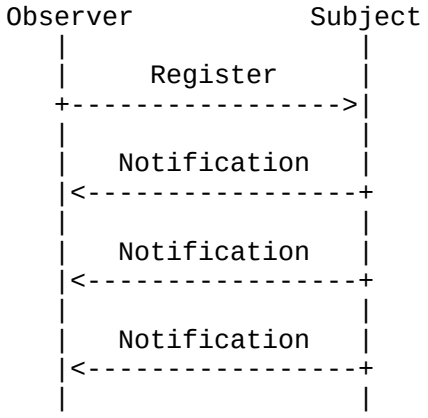


Figure 1: Subject/Observer Design Pattern

The design pattern is realized in CoAP as follows:

Subject: In the context of CoAP, the subject is a resource located at some CoAP server. The state of the resource may change over time, ranging from infrequent updates to continuous state transformations.

Observer: The observer is a CoAP client that is interested in the current state of the resource at any given time.

Observation Relationship: A client registers itself with a resource by sending a modified GET request to the server. The request causes the server to establish an observation relationship between the client and the resource. The response to the GET request supplies the client with a representation of the current resource state.

Notification: Whenever the state of a resource changes, the server notifies each client that has an observation relationship to that resource. The notification is an additional response to the GET request; it supplies the client with a representation of the new resource state. The response echoes the token specified in the request, so the client can easily correlate notifications.

Figure 2 shows an example of a CoAP client establishing an observation relationship to a resource on a CoAP server and being notified, once upon registration and then whenever the state of the resource changes. The request to establish an observation relationship and all notifications are identified by the new Observe Option defined in this document.

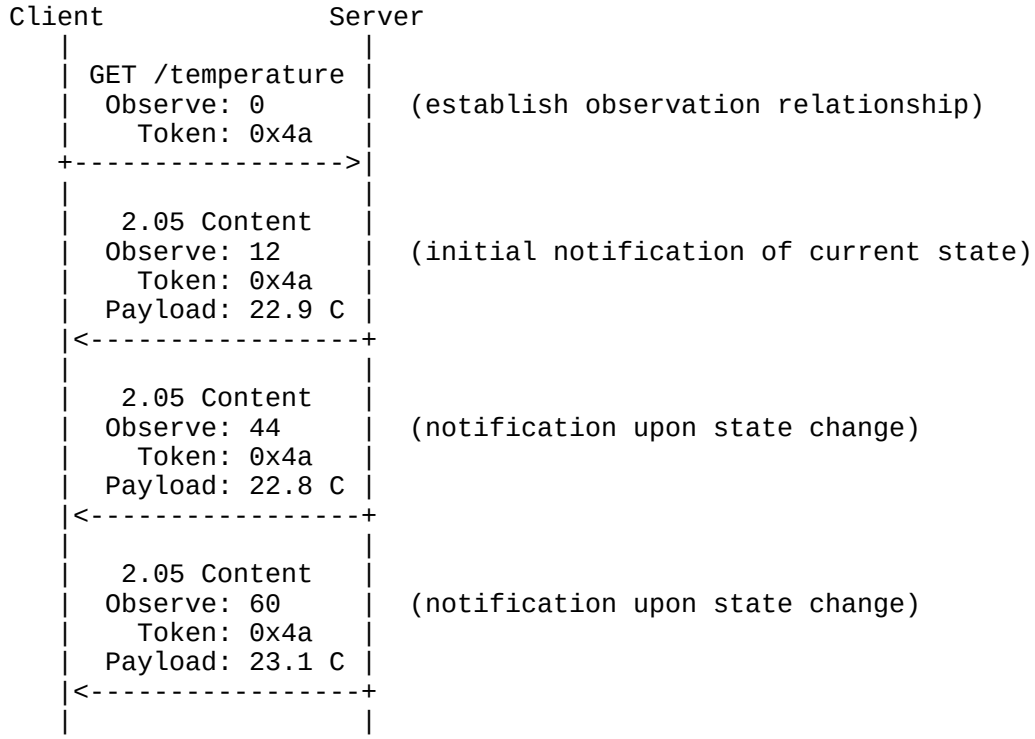


Figure 2: Observing a Resource in CoAP

3. Observation Relationships

3.1. Establishment

A client registers itself with a resource by performing a GET request that includes an Observe Option. (See Section 5 for the option definition.) When a server receives such a request, it services the request like a GET request without this option and, if the resulting response indicates success, establishes an observation relationship between the client and the target resource.

The token specified by the client in the GET request will be echoed by the server in the initial response and in all notifications sent to the client as part of the observation relationship. The server will also include an Observe Option in each response/notification to indicate that the observation relationship was successfully established. (See Section 4 for the details of notifications.)

A server that is unable or unwilling to establish an observation

relationship between a client and a resource MUST silently ignore the Observe Option and process the GET request as usual. The resulting response will not include an Observe Option, implying that no observation relationship was established.

3.2. Maintenance

A client MAY refresh an observation relationship at any time. (For example, when it didn't receive a notification for some time, it is not clear whether the resource never changed or the server rebooted and lost its state -- this is similar to the keep-alive problem of transport protocols, see e.g. the discussion in [RFC1122].) However, it is RECOMMENDED that the client does not refresh the relationship for the time specified in the Max-Age Option of the most recent notification received, including the initial response.

A client refreshes an observation relationship simply by repeating the GET request with the Observe Option. When a server receives such a repeated request (i.e. a GET request from a client for which an observation relationship already exists), it MUST NOT establish a second relationship but replace or update the existing one. If a GET request does not include an Observe Option, the server MUST end any relationship that may exist between the client and the target resource.

The exact rules for determining if two requests relate to the same observation relationship are as follows:

- The request URI of the two requests MUST match.
- The sources of the two requests MUST match. How this is determined depends on the security mode used (see Section 10 of [I-D.ietf-core-coap]): With NoSec, the IP address and port number of the request sources must match. With other security modes, in addition to the IP address and UDP port number matching, the requests must have the same security context.
- The Message IDs and any Token Options in the two requests MUST NOT be taken into account.

3.3. Termination

The observation relationship between a client and a resource ends when one of the following conditions occurs:

- The server sends a notification response with an error response code (4.xx or 5.xx).

- The client rejects a confirmable notification with a RST message.
- The last attempt of transmitting a confirmable notification to the client times out. (In this case, the server MAY also end all other observation relationships that the client has.)

A client MAY terminate an observation relationship by performing one of the following actions:

- The client rejects a confirmable notification with a RST message.
- The client performs a GET request on the resource without an Observe Option.

4. Notifications

When an observation relationship is established between a client and a resource, the client is notified of resource state changes by additional responses sent in reply to the GET request to the client. Each such notification response MUST include an Observe Option and echo the token specified by the client in the request. The order in which observers are notified about a state change 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 is changed in a way that would cause a basic GET request to return an error (for example, when the resource is deleted), the server SHOULD notify the client by sending a notification with an appropriate error code and MUST end the observation relationship.

The representation format (i.e. the media type) used in any notification resulting from an observation relationship 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 response and MUST end the observation relationship.

A notification can be sent confirmable or non-confirmable. A server can employ different strategies for notifying a client; see Section 4.1 below. The objective is that the state observed by the client eventually becomes consistent with the actual state of the resource.

If a client does not recognize the token in a confirmable notification, it MUST NOT acknowledge the message and SHOULD reject the message with a RST message (in which case the server MUST end the

observation). Otherwise, the client MUST acknowledge the message with an ACK message as usual. See Section 4.2 for details on the retransmission of confirmable messages.

Note that notifications may arrive in a different order than sent by the server due to network latency. If a notification arrives before the initial response to a request, the client can take the notification as initial response in place of the actual initial response. The client must be prepared to receive notifications after an error notification or after the client has requested the server to end the observation relationship. See Section 4.3 for further details on message reordering.

Notifications MAY be cached by CoAP end-points under the same conditions as with all responses. This is detailed in Section 4.4.

4.1. Strategies

The objective when notifying clients of state changes is that the state observed by the client eventually becomes consistent with the actual state of the resource. This allows the server some liberties in how it sends notifications, as long as it works towards this objective.

A notification can be sent confirmable or non-confirmable. The message type used is typically application-dependent and MAY be determined by the server for each notification individually. For example, for resources that change in a somewhat predictable or regular fashion, notifications can be sent in non-confirmable messages. For resources that change infrequently, notifications can be sent in confirmable messages. The server can combine these two approaches depending on the frequency of state changes and the importance of individual notifications.

A server MAY choose to omit notifying a client of a state change if it knows that it will send another notification soon (e.g., when the state is changing frequently or maybe even continuously). Similarly, it MAY choose to notify a client about the same state change more than once. For example, when state changes occur in bursts, the server can omit some notifications, send others in non-confirmable messages, and make sure that the client observes the latest state change by repeating the last notification in a confirmable message when the burst is over.

4.2. Retransmission

According to the core CoAP protocol, 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 changed in the meantime. There are many reasons why a client might not acknowledge a confirmable message, ranging from short interruptions in the network to a permanent failure of the client.

When a server is retransmitting a confirmable message with a notification, is waiting for an acknowledgement, and wants to notify the client of a state change using a new confirmable message, it MUST stop retransmitting the old notification and MUST attempt to transmit 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 observation relationship is ended.

4.3. Reordering

Messages with notifications can arrive in a different order than they were sent. Since the objective is eventual consistency, a client can safely discard a notification that arrives later than a newer notification.

For this purpose, the server keeps a single 16-bit unsigned integer variable. The variable is incremented approximately every second, wrapping around every 2^16 seconds (roughly 18.2 hours). The server MUST include the current value of the variable as the value of the Observe Option each time it sends a notification. The server MUST NOT send two notifications with the same value of the variable that pertain to the same resource to the same client.

A client MAY discard a notification as outdated (not fresh) under the following condition:

```
(V1 - V2) % (2^16) < (2^15)    and    T2 < (T1 + (2^14))
```

where T1 is a client-local timestamp of the latest valid notification received for this resource (in seconds), T2 a client-local timestamp of the current notification, V1 the value of the Observe Option of the latest valid notification received, and V2 the value of the Observe Option of the current notification. 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 (but is not needed any more, because reordering is not expected to occur on the order of 2^14 seconds). Note that the constants of 2^14 and 2^15 are non-critical, as is the even speed of

the clocks involved; e.g., the second check can be implemented by marking a response as fresh on reception and downgrading all responses periodically every, say, 2^13 seconds; once it has been downgraded twice, it no longer participates in freshness checks.

4.4. Caching

As notifications are just additional responses to a GET request, the same rules on caching apply as to all responses: CoAP end-points MAY cache the responses and thereby reduce the response time and network bandwidth consumption. Both the freshness model and the validation model are supported.

When a response is fresh in the cache, GET requests can be satisfied without contacting the origin server. This is particularly useful when the cache is located at an CoAP intermediary such as a proxy or reverse proxy. (Note that the freshness of the stored response is determined by its Max-Age Option, not the existence of an observation relationship. So a request can cause the end-point to refresh cache and observation relationship even while having an relationship.)

When an end-point has one or more responses stored, it can use the ETag Option to give the origin server an opportunity to select a stored response to be used. The end-point SHOULD add an ETag Option specifying the entity-tag of each stored response that is applicable. It MUST keep those responses in the cache until it terminates the observation relationship or sends a GET request with a new set of entity-tags. 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 sends 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 request.

5. Observe Option

No.	C/E	Name	Format	Length	Default
10	Elective	Observe	uint	0-2 B	(none)

Table 1: New Options

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 once, but also lets the server notify the client of changes to the resource state.

In a response, the Observe Option indicates that an observation relationship has been established. The option's value is a sequence number that can be used for reordering detection (see Section 4.3). The value is encoded as a variable-length unsigned integer (see Appendix A of [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 basic GET request if the server does not support observations.

6. Interactions with other CoAP features

6.1. Request Methods

If a client has an observation relationship with a resource and performs a POST, PUT or DELETE request on that resource, the request MUST NOT affect the observation relationship. However, since such a request can affect the observed resource, it can cause the server to send a notification with a resource state representation or end the observation relationship with an error notification (e.g., when a DELETE request is successful and an observed resource no longer exists).

Note that a client cannot perform a GET request on a resource to retrieve a representation of the current resource state without affecting an existing observation relationship to that resource: the client is already notified by the server with a fresh representation whenever the state changes. If the client wants to make sure that is has a fresh representation and wants to continue being notified, it should refresh the observation relationship (see Section 3.2). If the client wants to make sure it has a fresh representation and does not want to continue being notified, it should perform a GET request without an Observe Option (see Section 3.3).

6.2. Block-wise Transfers

Resources that are the subject of an observation relationship 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 Block option in the GET request. If the server

supports block-wise transfers, it SHOULD take note of the block size not just for the initial response but also for further notifications in this observation relationship.

- o Notification responses can make use of the Block option. The client SHOULD use the Observe option value from the last block. All blocks in a notification response SHOULD also carry an ETag option to ensure they are reassembled correctly.

6.3. Resource Discovery

Clients can discover resources that are interesting to observe using CoRE Resource Discovery [I-D.ietf-core-link-format]. Links with the "obs" attribute indicate resources that MUST support the mechanism in this document and are RECOMMENDED to change their state at least once in a while.

The "obs" attribute is used as a flag, and thus it has no value component. The attribute MUST NOT appear more than once in a link.

7. Security Considerations

The security considerations of the base protocol [I-D.ietf-core-coap] apply.

Note that the considerations about amplification attacks are somewhat amplified in an observation relationship. In NoSec mode, a server MUST therefore strictly limit the number of messages generated from an observation relationship that it sends between receiving packets that confirm the actual interest of the recipient 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 observation relationships. Servers MAY want to access-control this creation of state. As degraded behavior, the server can always fall back to a basic GET request (without an Observe option) if it is unwilling or unable to establish the observation relationship, including if resources for state are exhausted or nearing exhaustion.

Intermediaries MUST be careful to ensure that notifications cannot be employed to create a loop. A simple way to break any loops is to employ caches for forwarding notifications in intermediaries.

8. IANA Considerations

The following entry is added to the CoAP Option Numbers registry:

Number	Name	Reference
10	Observe	[RFCXXXX]

Table 2: New CoAP Option Numbers

The following entry is added to the CoRE Link Format Attribute registry:

Name	Reference
obs	[RFCXXXX]

Table 3: New CoRE Link Format Attributes

9. Acknowledgements

Carsten Bormann was an original author of this draft and is acknowledged for significant contribution to this document.

Thanks to Daniele Alessandrelli, Peter Bigot, Angelo Castellani, Gilbert Clark, Esko Dijk, Brian Frank and Salvatore Loreto for helpful comments and discussions that have shaped the document.

Klaus Hartke was funded by the Klaus Tschira Foundation.

10. References

10.1. Normative References

[I-D.ietf-core-block]
Shelby, Z. and C. Bormann, "Blockwise transfers in CoAP", draft-ietf-core-block-02 (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-05 (work in progress), March 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-03 (work in progress), March 2011.

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

10.2. Informative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[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.

Appendix A. Examples

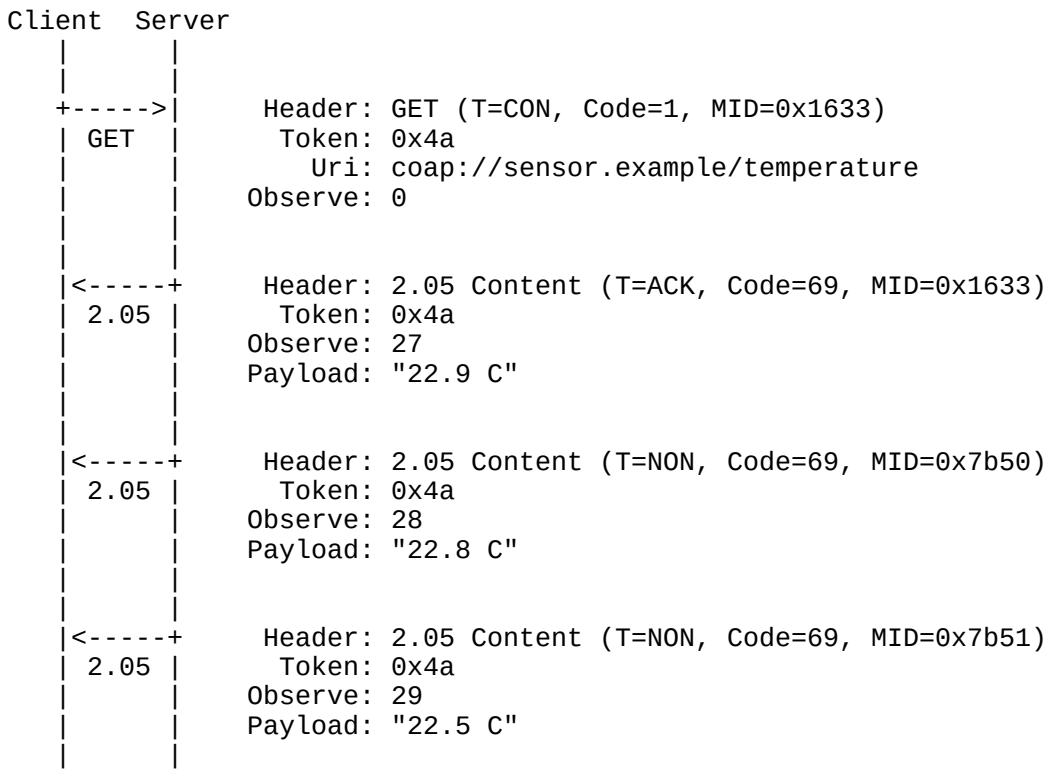


Figure 3: Simple observation with non-confirmable notifications

A.1. Proxying

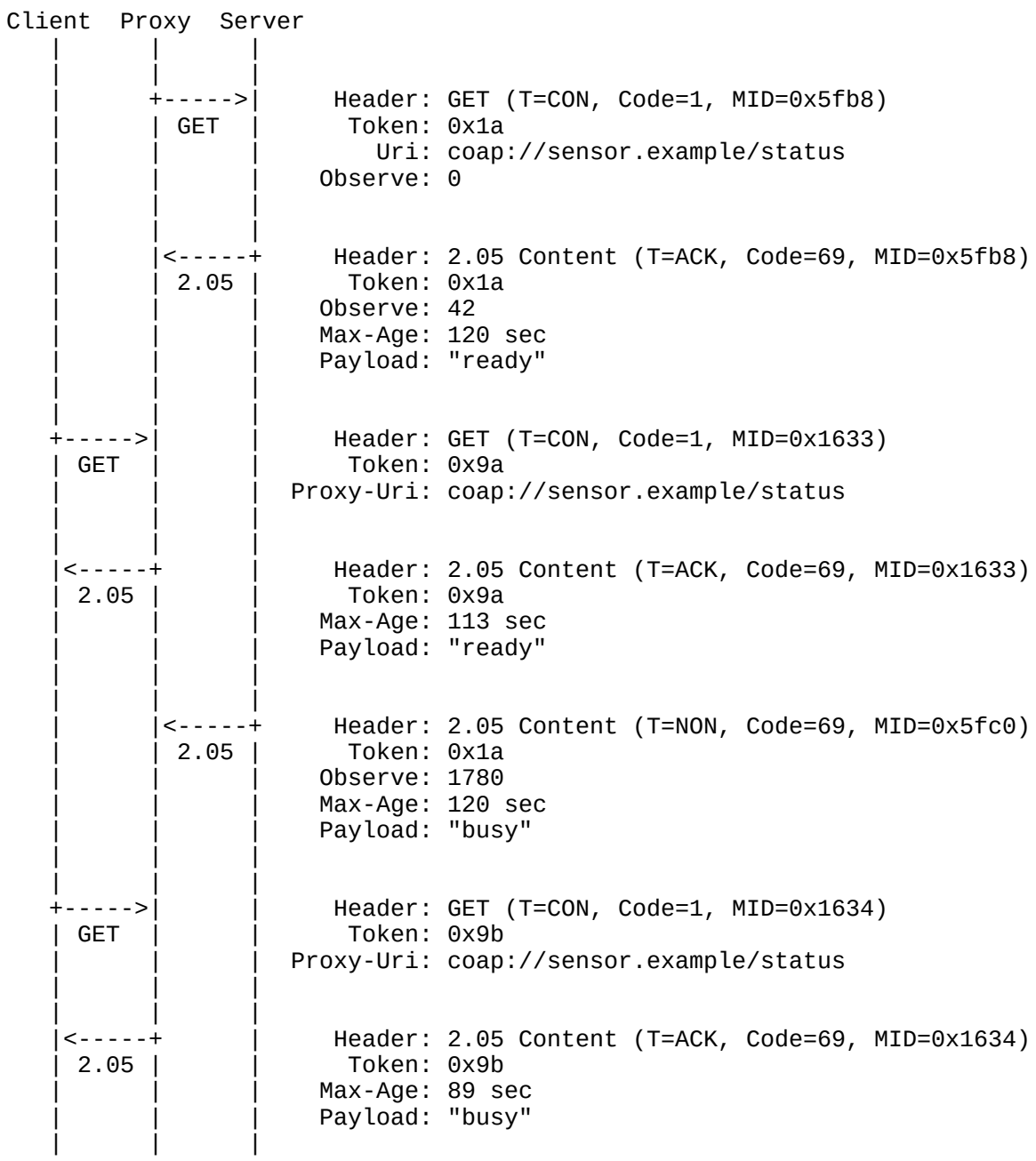
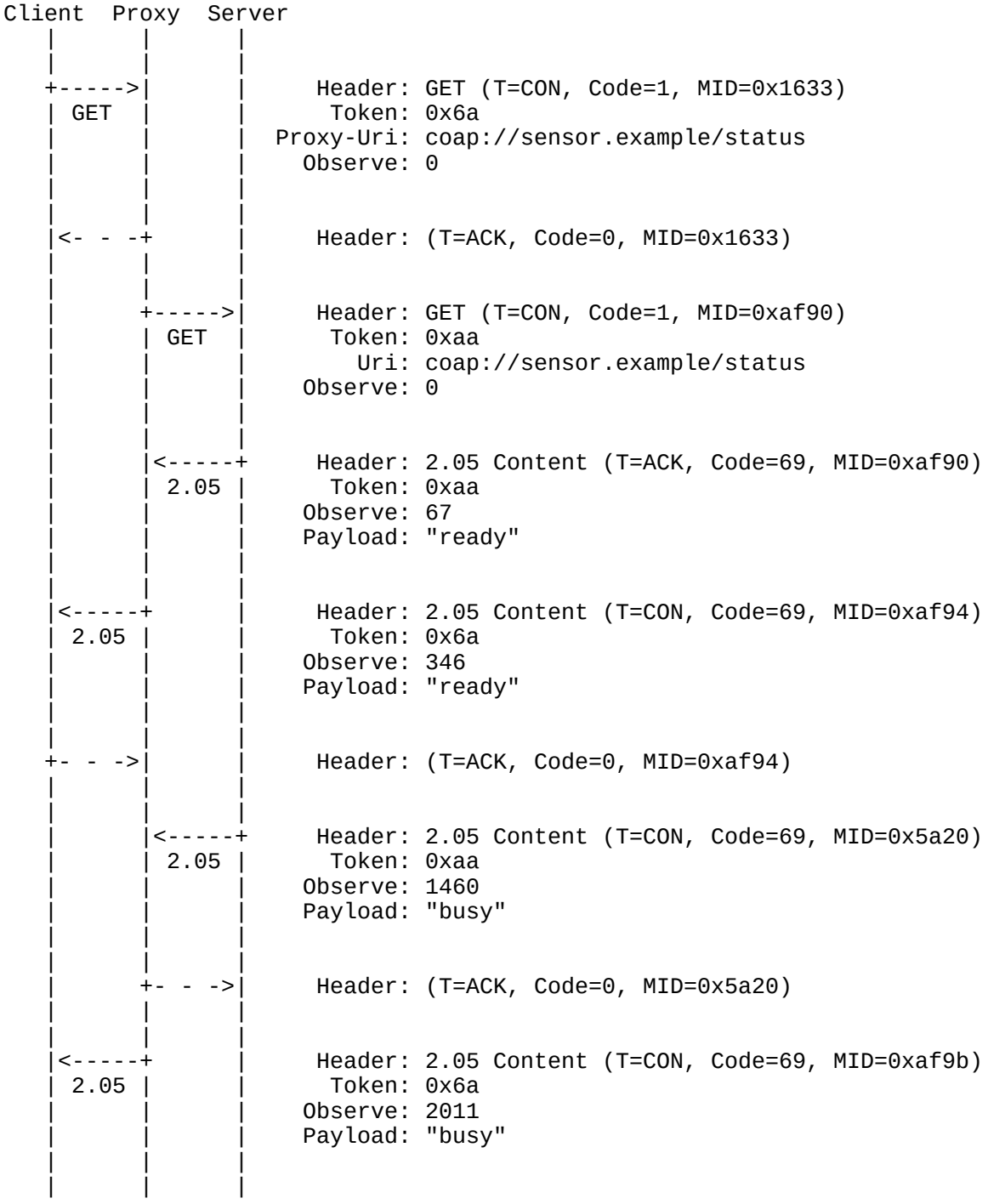


Figure 4: A proxy observes a resource to keep its cache up to date


Client Proxy Server
GET
Header: GET (T=CON, Code=1, MID=0x1633)
Token: 0x6a
Proxy-Uri: coap://sensor.example/status
Observe: 0
Header: (T=ACK, Code=0, MID=0x1633)
GET
Header: GET (T=CON, Code=1, MID=0xaf90)
Token: 0xaa
Uri: coap://sensor.example/status
Observe: 0
2.05
Header: 2.05 Content (T=ACK, Code=69, MID=0xaf90)
Token: 0xaa
Observe: 67
Payload: "ready"
2.05
Header: 2.05 Content (T=CON, Code=69, MID=0xaf94)
Token: 0x6a
Observe: 346
Payload: "ready"
Header: (T=ACK, Code=0, MID=0xaf94)
2.05
Header: 2.05 Content (T=CON, Code=69, MID=0x5a20)
Token: 0xaa
Observe: 1460
Payload: "busy"
Header: (T=ACK, Code=0, MID=0x5a20)
2.05
Header: 2.05 Content (T=CON, Code=69, MID=0xaf9b)
Token: 0x6a
Observe: 2011
Payload: "busy"

```
   +- - ->|      |     Header: (T=ACK, Code=0, MID=0xaf9b)
   |      |      |
```

Figure 5: A client observes a resource through a proxy

Appendix B. Changelog

Changes from ietf-01 to ietf-02:

- Removed the requirement of periodic refreshing (#126).
- The new "Observe" Option replaces the "Lifetime" Option.
- New mechanism to detect message reordering.
- Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:

- Changed terminology from "subscriptions" to "observation relationships" (#33).
- Changed the name of the option to "Lifetime".
- Clarified establishment of observation relationships.
- Clarified that an observation is only identified by the URI of the observed resource and the identity of the client (#66).
- Clarified rules for establishing observation relationships (#68).
- Clarified conditions under which an observation relationship is terminated.
- Added explanation on how clients can terminate an observation relationship before the lifetime ends (#34).
- Clarified that the overriding objective for notifications is eventual consistency of the actual and the observed state (#67).
- Specified how a server needs to deal with clients not acknowledging confirmable messages carrying notifications (#69).
- Added a mechanism to detect message reordering (#35).
- Added an explanation of how notifications can be cached, supporting both the freshness and the validation model (#39, #64).

- Clarified that non-GET requests do not affect observation relationships, and that GET requests without "Lifetime" Option affecting relationships is by design (#65).
- Described interaction with block-wise transfers (#36).
- Added Resource Discovery section (#99).
- Added IANA Considerations.
- Added Security Considerations (#40).
- 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
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Phone: +358407796297
Email: zach@sensinode.com

CORE
Internet-Draft
Intended status: Informational
Expires: July 30, 2011

G. Moritz
University of Rostock

January 26, 2011

SOAP-over-CoAP Binding
draft-moritz-core-soap-over-coap-00

Abstract

The scope of this document is to provide a basis for a lightweight SOAP over CoAP binding, to allow usage of SOAP Web services also in resource constrained networks.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

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

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

This Internet-Draft will expire on July 30, 2011.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Motivation 3
1.2. Requirements Language 4
1.3. Terminology and Definitions 4
1.4. Requirements 4
2. Use of CoAP 5
2.1. CoAP Content-Type 5
3. Binding Name 5
4. Transport Layer Binding 6
4.1. Source Address and Port 6
5. Addressing 6
5.1. URI Scheme 6
5.2. Destination Addressing 6
6. Message Patterns 7
6.1. Request response 7
6.2. Retransmission 8
7. CoAP Header Options 8
7.1. Unicast one-way 8
7.2. Unicast request-response 10
8. IANA Considerations 12
9. Security Considerations 12
10. References 12
10.1. Normative References 12
10.2. Informative References 12
Author’s Address 13

1. Introduction

The intention of this document is to provide a basic approach for a SOAP-over-CoAP binding. Readers of this document should be basically familiar with the CoAP draft [I-D.ietf-core-coap], SOAP [W3C.REC-soap12-part0-20070427], SOAP HTTP Binding [W3C.REC-soap12-part1-20070427] and SOAP UDP binding specifications [SOAP-over-UDP]. Parts of this document are derived from these existing specifications. This document will not provide a comprehensive specification, but may be a basis for further discussions and to identify required changes in the current CoAP [I-D.ietf-core-coap] protocol design which is on the way to become an IETF standard. These changes might not be identified directly as the binding described herein will not exploit from all features of CoAP but use CoAP as an 'application layer transport protocol' as SOAP already does with HTTP [W3C.REC-soap12-part1-20070427].

1.1. Motivation

The motivation behind this document is based on the initial I-D [I-D.moritz-6lowapp-dpws-enhancements] and the resulting discussions. The described binding herein is a major enabler for the efforts towards usage of SOAP Web services for highly resource constrained devices like wireless sensor nodes. As core protocol the OASIS Devices Profile for Web Services (DPWS) [DPWS] is applied. By combining DPWS with EXI, message size can be reduced significantly as presented in [I-D.moritz-6lowapp-dpws-enhancements]. By providing a binding of SOAP to CoAP, the message overhead can be reduced significantly. Furthermore, SOAP is not required to use inappropriate mechanisms like TCP congestion control. Reliably messaging is guaranteed by CoAP internal mechanisms and thus expensive retransmissions independent of successful or unsuccessful delivery as required by the SOAP-over-UDP binding are avoided. In summary, the major advantages are:

- more compact message format
- probably lower message parsing efforts compared to standard HTTP headers
- avoid inappropriate TCP usage implied by SOAP-over-HTTP binding
- avoid unreliable nature of unicast SOAP-over-UDP binding

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].

1.3. Terminology and Definitions

Defined below are the basic definitions for the terms used in this specification.

Receiver
: The endpoint terminating a SOAP/CoAP message

Sender
: The endpoint originating a SOAP/CoAP message

SOAP/CoAP message
: A CoAP message containing a SOAP envelope in the CoAP message body

This specification uses the constructs [action], [destination], [message id], [reply endpoint], [address] as defined by in WS-Addressing [W3C.PR-ws-addr-core-20060321].

The SOAP CoAP Binding is optional and SOAP nodes are not required to implement it. A SOAP node that correctly and completely implements the SOAP CoAP Binding may to be said to ’conform to the CoAP Binding.’

1.4. Requirements

This specification intends to meet the following requirements:

- Support a one-way message-exchange pattern (MEP) where a SOAP envelope is carried in a CoAP message from Sender to Receiver only.

- Support a request-response message-exchange pattern (MEP) where SOAP envelopes are carried in CoAP messages from Sender to Receiver and back from the origin Receiver to the origin Sender.

Even if reasonable, supporting multicast transmissions of SOAP envelopes carried in CoAP messages are out of scope of this version of this document and require further research. This binding supports SOAP 1.2 [W3C.REC-soap12-part0-20070427] Envelopes. Supporting SOAP 1.1 envelopes is out of scope but might be added in future versions of this document.

Messages conforming to either SOAP specification can use this binding. This specification relies on WS-Addressing. The SOAP envelope MUST use the mechanisms defined in WS-Addressing [W3C.PR-ws-addr-core-20060321].

2. Use of CoAP

This binding of SOAP to CoAP is intended to make appropriate use of CoAP as an application protocol. For example, successful and failure responses are indicated by the appropriated CoAP response codes (e.g. 2xx, 4xx, 5xx). This binding is not intended to fully exploit the features of CoAP, but rather to use CoAP specifically for the purpose of communicating with other SOAP nodes implementing the same binding.

2.1. CoAP Content-Type

Conforming implementations of this binding:

- MAY be capable of sending and receiving SOAP/CoAP messages serialized using media type 'application/soap+xml'.
- MAY be capable of sending and receiving SOAP/CoAP messages serialized using media type 'application/exi'.
- MUST send requests and responses using such media types provide for at least the transfer of SOAP XML Infoset, including 'application/soap+xml' and 'application/exi'.

A SOAP/CoAP message MUST contain the CoAP Content-type Identifies Option and MUST contain a value which satisfies the three points above.

Note: CoAP has no header option which corresponds to HTTP Accept. Thus the web methods known from the HTTP binding is not possible. In the current CoAP draft only view information are available how to define own header fields.

3. Binding Name

This binding is identified by the URI (see SOAP 1.2 Part 1 [W3C.REC-soap12-part1-20070427] SOAP Protocol Binding Framework): http://www.ws4d.org/2011/01/soap/bindings/CoAP/

Note: The binding name is tentative but required to indicate the corresponding binding e.g. in the WSDL of a service.

4. Transport Layer Binding

The CoAP binding MUST operate over UDP transport layer.

Note: CoAP defines a maximum message size which refers to the IP layer. The existing UDP binding instead refers only to UDP and defines a general maximum packet size independent of IP.

4.1. Source Address and Port

The source address MUST be supplied at the IP packet level and MUST be the IPv4 address (limited to unicast addresses) or IPv6 address (limited to unicast addresses) of the sender; the receiver SHOULD reject IP packets containing a SOAP/CoAP message that have inappropriate values for the source address.

Even though CoAP is intended to be used on the well-known ports defined in CoAP-04, the use of CoAP is not limited to these ports. As a result, it is possible to have a dedicated CoAP server for handling SOAP processing on a distinct port.

5. Addressing

5.1. URI Scheme

The SOAP CoAP binding defines a base URI according to the rules in CoAP-04. I.e., the base URI is the CoAP Request-URI or the value of the CoAP Location option field.

5.2. Destination Addressing

WS-Addressing defines a URI, 'http://www.w3.org/2005/08/addressing/anonymous', that can appear in the [address] property of an endpoint reference. If the [reply endpoint] property of a SOAP message transmitted over CoAP has an [address] property with this value, the UDP source address (and source port) is considered to be the address to which reply messages should be sent.

The implied value of the [reply endpoint] property for SOAP messages transmitted over CoAP is an endpoint reference with an [address] property whose value is 'http://www.w3.org/2005/08/addressing/anonymous'.

6. Message Patterns

This specification supports the following message patterns:

- Unicast one-way
- Unicast request, unicast response

as described in the remainder of this section. All SOAP/CoAP messages MUST use the POST method of CoAP, whereby POST is interpreted in accordance to RFC2616.

Note: In the current draft of CoAP-04, response Code 2.00 OK is only possible for GET methods. But GET is not the appropriate method for this binding.

Note: There is no Action field similar to SOAPAction in HTTP, so extensions are required to the basic CoAP Options. Hence the web method feature of the HTTP binding cannot be made possible.

CoAP defines Proxy mechanism for caching of responses. Because the CoAP binding defined herein is intended for service communication and not RESTful resource manipulation, caching should be avoided. CoAP defines the Max-Age option to be default a non-zero value. Hence each SOAP/CoAP message MUST contain the Max-Age option and the value MUST be zero.

6.1. Request response

To match a request with a response the CoAP Token Option can be used. The CoAP Token Option SHOULD be carried in all request-response SOAP/CoAP messages. WS-Addressing defines the [message id] property to identify messages in time and space and to match requests with response. In case of using the CoAP/SOAP binding the [message id] property is SHOULD be empty and MUST contain a value in case if the CoAP Token Option is not present.

Note: The intention of this paragraph is to reduce message size. The [message id] property has a typical size of 45 Bytes and cannot by compressed by knowledge based encodings like EXI because the value of this property is unique in each message. The CoAP Token Option may be much more compact by providing the same functionality.

CoAP defines the feature of deferred responses where a transaction ACK is sent as response to a CON message to indicate a deferred response. If the value of the [reply endpoint] of the request is 'http://www.w3.org/2005/08/addressing/anonymous', the ACK message SHOULD NOT carry any payload (e.g. a SOAP Envelope) in the CoAP

message body if not required by the application logic. If the value of the [reply endpoint] is not 'http://www.w3.org/2005/08/addressing/anonymous', the origin Receiver cannot guarantee that the response is intended to be sent to the same entity like the client and SHOULD include a SOAP Envelope providing details of the request for the entity behind the [reply endpoint].

6.2. Retransmission

To avoid repeated packet collisions, any retransmission implementation SHOULD observe good practices such as using exponential back-off algorithms and spreading. An implementation SHOULD use the Confirmable (CON) transaction message mechanism described in the CoAP specification to avoid unnecessary retransmissions. For each transmission of such a message, the value of the [message id] property and the CoAP Token Option MUST be the same.

Note: WS-Event Delivery should not use CON due to ACK flood at Event Source. Multicast messages also should use NON messages. Not sure if such requirements are in scope of this document.

7. CoAP Header Options

In this section, the CoAP header and CoAP header options usage is described in detail.

7.1. Unicast one-way

The unicast one-way message pattern consists of only one request without a response. Due to the CoAP binding the request consists of one CoAP transaction. If CoAP Confirmable messaging is used, an empty CoAP ACK message must be send back to the origin sender to indicate success or failure.

The request is formulated according to the table below, but can be extended for application specific needs.

CoAP field/header option	Value
CoAP Method	POST is the only supported method of this binding.
Request URI	The request URI confirming a CoAP URI [xxx] and identifying the WS-Addressing [address] endpoint property (network address). If the value of the WS-Addressing [address] endpoint property is 'http://www.w3.org/2005/08/addressing/anonymous' (directly set or implied by an empty [address] property), the CoAP Uri-Path Option and the CoAP Uri-Query Option are empty.
Message ID	If messaging model is CON, carrying value in accordance to messaging model of CoAP . Otherwise not included.
Content-type Option	Media type of CoAP message body.
Token Option	Token in accordance to CoAP specification identifying the transaction in time and space. Only required if message is a CoAP Confirmable message.
CoAP message body	SOAP message serialized according to the rules for carrying SOAP messages in the media type given by the Content-type Option.

Table 1: Unicast one-way Request

If the request is send as Non-Confirmable CoAP message, no response is send back to the origin requester. If the request is a Confirmable CoAP message, the response consists only of a CoAP ACK without any data in the CoAP message body. In case of a Conformable CoAP request the response is formulated according to the table below, but can be extended for application specific needs.

CoAP field/header option	Value
CoAP Status Code	Status Code in accordance to codes defined in CoAP and interpreted as described in the SOAP-over-HTTP binding. Note: CoAP describes only a subset of HTTP status codes and adds own codes. This must require further alignment.

	Message ID	If messaging model is CON, carrying value in accordance to messaging model of CoAP . Otherwise not included.
	Token Option	Token in accordance to CoAP specification identifying the transaction in time and space. Only required if message is a CoAP Confirmable message.

Table 2: Unicast one-way ACK Response

Application developers must be aware that the one-way message exchange patterns returns neither failure nor success messages if Non-Confirmable messages are used.

7.2. Unicast request-response

The unicast request-response message pattern consists of one request and one response. Due to the CoAP binding the request consists of one or more CoAP transactions.

The request is formulated according to the table below, but can be extended for application specific needs.

CoAP field/header option	Value
CoAP Method	POST is the only supported method of this binding.
Request URI	The request URI confirming a CoAP URI [xxx] and identifying the WS-Addressing [address] endpoint property (network address). If the value of the WS-Addressing [address] endpoint property is 'http://www.w3.org/2005/08/addressing/anonymous' (directly set or implied by an empty [address] property), the CoAP Uri-Path Option and the CoAP Uri-Query Option are empty.
Message ID	If messaging model is CON, carrying value in accordance to messaging model of CoAP . Otherwise not included.
Content-type Option	Media type of CoAP message body.
Token Option	Token in accordance to CoAP specification identifying the transaction in time and space. Only required if message is a CoAP Confirmable message.

CoAP message body	SOAP message serialized according to the rules for carrying SOAP messages in the media type given by the Content-type Option.

Table 3: Unicast request-response Request

If the request is send as Non-Confirmable CoAP message, no CoAP ACK is send back to the origin requester. In this case, for the response, the origin Receiver has to initiate a new CoAP transaction which can be either Non-Confirmable or Confirmable.

If the request is a Confirmable CoAP message, the response consists of a CoAP ACK which may carry the response SOAP envelope as data in the CoAP message body. For the response, the origin receiver may also initiate a new CoAP transaction after sending the CoAP ACK to the origin Sender, which can be either also Non-Confirmable or Confirmable. (see deferred responses in CoAP)

CoAP field/header option	Value
CoAP Status Code	Status Code in accordance to codes defined in CoAP and interpreted as described in the SOAP-over-HTTP binding. Note: CoAP describes only a subset of HTTP status codes and adds own codes. This must require further alignment.
Message ID	If messaging model is CON, carrying value in accordance to messaging model of CoAP . Otherwise not included.
Token Option	Token in accordance to CoAP specification identifying the transaction in time and space. Only required if message is a CoAP Confirmable message.
Content-type Option	Media type of CoAP message body.
CoAP message body	SOAP message serialized according to the rules for carrying SOAP messages in the media type given by the Content-type Option.

Table 4: Unicast request-response ACK with Response

8. IANA Considerations

No IANA issues have been identified in this draft.

9. Security Considerations

Will be added in future versions.

10. References

10.1. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

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

[W3C.PR-ws-addr-core-20060321]
Gudgin, M., Hadley, M., and T. Rogers, "Web Services Addressing 1.0 - Core", W3C PR PR-ws-addr-core-20060321, March 2006.

[W3C.REC-soap12-part0-20070427]
Mitra, N. and Y. Lafon, "SOAP Version 1.2 Part 0: Primer (Second Edition)", World Wide Web Consortium Recommendation REC-soap12-part0-20070427, April 2007, <http://www.w3.org/TR/2007/REC-soap12-part0-20070427>.

10.2. Informative References

[DPWS] Driscoll and Mensch, "OASIS Devices Profile for Web Services (DPWS) Version 1.1", 2009, <http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01>.

[I-D.moritz-6lowapp-dpws-enhancements]
Moritz, G., "DPWS for 6LoWPAN", draft-moritz-6lowapp-dpws-enhancements-01 (work in progress), June 2010.

[SOAP-over-UDP]
Jeyaraman, "OASIS SOAP-over-UDP Version 1.1", 2009, <http://docs.oasis-open.org/ws-dd/soapoverudp/1.1/os/wsdd-soapoverudp-1.1-spec-os.html>.

[W3C.REC-soap12-part1-20070427]
Gudgin, M., Karmarkar, A., Nielsen, H., Hadley, M., Mendelsohn, N., Lafon, Y., and J. Moreau, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)", World Wide Web Consortium Recommendation REC-soap12-part1-20070427, April 2007, <http://www.w3.org/TR/2007/REC-soap12-part1-20070427>.

Author's Address

Guido Moritz
University of Rostock
18051 Rostock,
Germany

Phone: +49 381 498 7269
Email: guido.moritz@uni-rostock.de

CoRE
Internet-Draft
Intended status: Informational
Expires: September 12, 2011

A. Rahman, Ed.
InterDigital Communications, LLC

March 11, 2011

Group Communication for CoAP
draft-rahman-core-groupcomm-04

Abstract

This is a working document intended to trigger discussion and develop draft language for the CoAP protocol specification in the area of group communication (including multicast functionality). 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.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 12, 2011.

Table of Contents

1. Conventions and Terminology 3
2. Introduction 3
2.1. Background 3
2.2. Problem Statement and Scope 4
3. Multicast Solutions 4
3.1. IP Multicast 4
3.1.1. Group Addressing 5
3.1.2. Group URIs 5
3.1.3. Group Discovery 5
3.1.4. Group Resource Manipulation 6
3.1.5. Multicast Transmission Methods 7
3.1.6. Congestion Control 8
3.2. Overlay Multicast 8
4. CoAP Application Group Management 9
5. Alternate Group Transmission Methods 11
5.1. Serial Unicast 11
6. CoAP Multicast and HTTP Unicast Interworking 11
7. Security Considerations 13
8. IANA Considerations 14
9. Conclusions 14
10. Acknowledgements 14
11. References 14
11.1. Normative References 14
11.2. Informative References 15
Author's Address 17

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 terminologies used in this draft.

Multicast: A 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].

Group Communication: One source node sends a CoAP message to more than one destination node. The source and destination nodes can be constrained or non-constrained nodes. This may include serial unicast, multicast, or hybrid unicast-to-multicast solutions.

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 CoRE are documented in [I-D.shelby-core-coap-req].

In this draft, we focus and expand discussions on requirements pertaining to 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 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 multicast in the current CoAP requirement to group communication, using either (reliable or unreliable) multicast or unicast.

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 can increase the efficiency 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 proposed solutions and analysis of impacts to the CoAP protocol and implementations.

3. Multicast Solutions

The classic model of a multicast application is that of a single source distributing content to many recipients. Multicast solutions have evolved from "bottom" to "top", i.e., from the network layer (IP multicast) to application layer multicast. A study published in 2005 identified new solutions in the "middle" (referred to as overlay multicast) that utilize an infrastructure based on proxies [STUDY1].

Each of these classes of multicast solutions may be compared using metrics such as link stress and level of host complexity [STUDY2]. The approach adopted here is to begin with IP multicast and present a complete picture to introduce some key concepts, then expand to cover more general scenarios such as group management and CoAP-to-HTTP proxies.

3.1. 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 Internet, leading to alternative solutions for applications like interactive gaming and publish/subscribe. However, the packet economy and minimal host complexity of IP multicast make it worth investigating for intra-domain group communication in constrained environments.

3.1.1. Group Addressing

The header compression proposal of 6LoWPAN [I-D.ietf-6lowpan-hc] includes a format to support Unicast-Prefix-based IPv6 Multicast Addresses such as [RFC3956], which is itself a scheme to simplify the deployment of PIM-SM [RFC4601]. Since the use of header compression for CoAP is highly desirable, it should be investigated if addressing mode can be used for IP multicast.

3.1.2. Group URIs

An approach to map group authorities onto 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].

//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 (or multicast) address. Each example group URI shown above might be mapped to a unique multicast IP addresses as defined in [RFC3956].

3.1.3. Group Discovery

CoAP defines a resource discovery capability but, in the absence of a multicast infrastructure, it is limited to link-local scope; 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 CoAP nodes within subdomains. A service is specified by a name of the form Instance.Type.Domain, where the type for CoAP nodes is _coap._udp and the domain is a DNS domain name that identifies a building zone as in the examples above. For each CoAP end-point in the zone, a PTR record with the name _coap._udp is defined and it points to an SRV record having the

Instance.Type.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 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.

3.1.4. Group Resource Manipulation

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 comissioning and can be illustrated by the example "return all the resources matching a .well-known URI". MGET 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 solving this problem:

- packet economy on constrained networks
- M2M resource discovery (solves the "chicken-and-egg" problem)
- apparent simultaneity of events (e.g. lighting applications)

For data links (or transports) that don’t support IP multicast, a serial unicast alternative must be provided. In either case it should be possible to enumerate the members of a group. For links that do support IP multicast, there are a few implications:

- All multicast requests MUST be sent to the well-known CoAP port

- All multicast requests SHOULD operate on /.well-known/core URIs

The justification for these constraints is that all nodes in a given group (defined over the IP address space) 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 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.

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 list of multicast, serial unicast, or some combination, having a distinct scheme for group operations might be a useful signal for the proxy receiving the request to look up the group membership and replicate serial unicasts as well as multicast packets.

3.1.5. Multicast Transmission Methods

3.1.5.1. Unreliable IP Multicast

The CoRE WG charter specified support of non-reliable multicast. In the current CoAP protocol design [I-D.ietf-core-coap], unreliable multicast is realized by the source sending non-confirmable messages.

3.1.5.2. Reliable IP Multicast

[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 TID or a new TID for repeated requests? Carsten suggests the use of bloom filters to suppress duplicate responses.

Note that non-idempotent operations (POST) cannot be supported without a *truly* reliable multicast protocol.]

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

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.

Integrity - destination receives m at most once from sender and only

if sender sent m to a group including destination.

Agreement - If a correct destination of g receives m, then all correct destinations of g receive m.

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

* Destination node sends acknowledgement: in CoAP, multicast messages are confirmable when reliability is required.
* Route redundancy
* Source node transmits multiple times

3.1.6. Congestion Control

[In the case of MGET or Confirmable MPUT, servers should enforce a random delay within TIMEOUT before sending responses. More investigation required.]

CoAP requests may be multicast, resulting multitude of replies from different nodes, potentially causing congestion. [I-D.eggert-core-congestion-control] suggests to conservatively control sending multicast request.

Various means can be implemented to prevent congestion.

Currently in the CoAP protocol, the MAX_RETRANSMIT value is set to 5 by default. It is suggested that two values are used separately for retransmissions, one for unicast transmission between a single source and sink. The current value of 5 is used for such transmission. Another constant is defined for multicast or group communication. The retransmission value for group transmission should be much lower, such as 1.

3.2. Overlay Multicast

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

Note that the CoAP (MLD) proxy may or may not be connected to an external multicast 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.

4. CoAP Application Group Management

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 one 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 be configured or dynamically formed.

The CoAP protocol needs to support CoAP group management features independently of any underlying IP multicast support. For example, a constrained node needs to be able to specify which group it intends to join using a CoAP request by providing the group address.

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

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

Figure 1: CoAP Header Options for Group Management

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

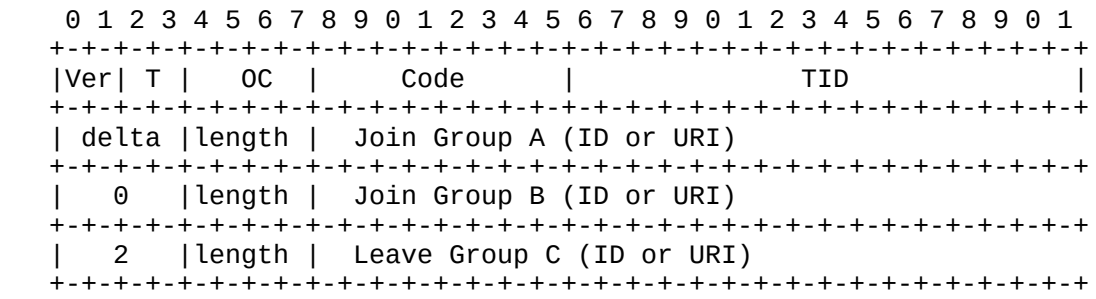


Figure 2: CoAP Message for Group Management

Header Fields for the above example:

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

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

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

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

Transaction ID: 16-bit unsigned integer assigned by the source to uniquely identify a pair of Request and Response.

CoAP defined 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 1, 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.

5. Alternate Group Transmission Methods

5.1. Serial Unicast

Unicast can also be used for the transmission of group messages. When unicast is used, no IP multicast address is provided by the application. Instead, the group URI must be expanded into unicast addresses.

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 the proxy node needs to have functionalities to support interworking of unicast and multicast as illustrated in Figure 3:

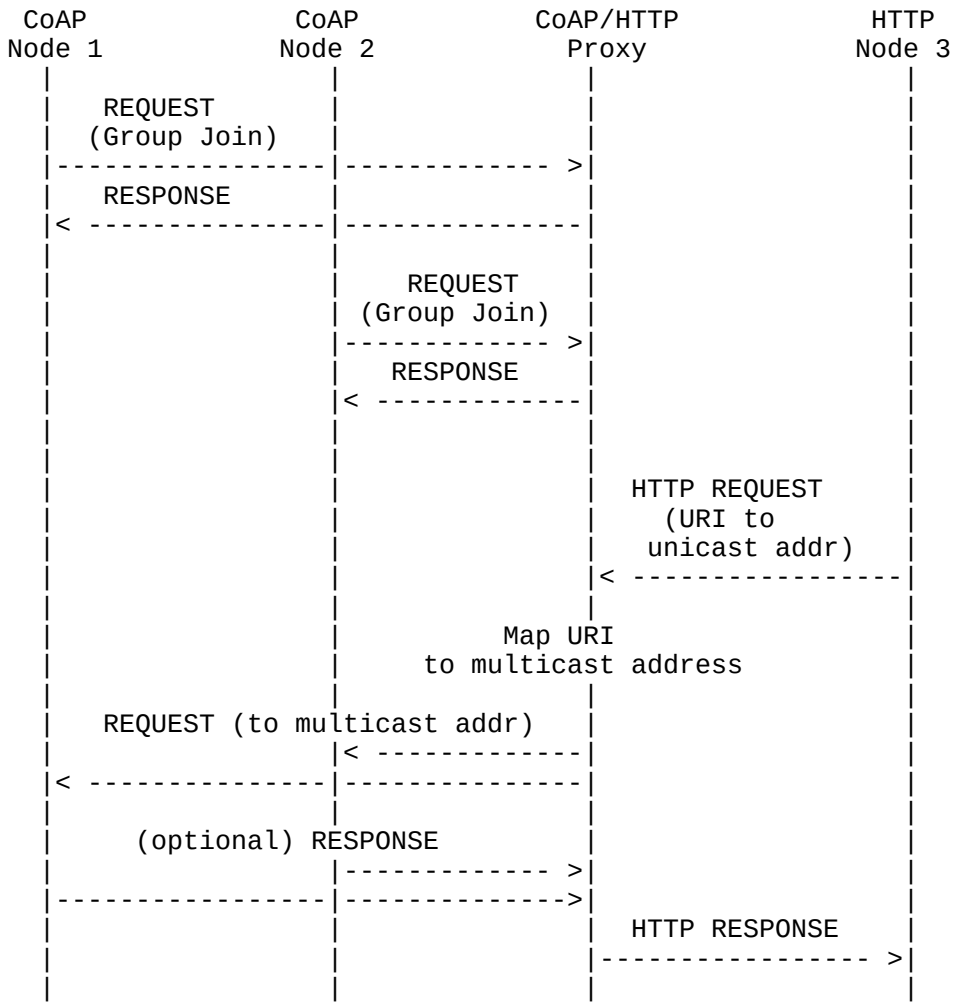


Figure 3: CoAP Multicast and HTTP Unicast Interworking

Note that Figure 3 illustrates the case of IP multicast as the underlying group communications mechanism. However the overlay multicast (Section 3.2) or CoAP application group communication (Section 4) 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 3 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 be again resolved (with the CoAP scheme) to an IP multicast destination. This may be accomplished, for example, by using DNS-SD (Section 3.1.3). The proxy node will then 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 3 illustrates that it will be generated by the proxy node and sent back to the client in the general Internet that sent the HTTP Request (i.e. node 1). 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."

7. 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. The following requirements for securing group communications in CoAP were derived from a study of these previous investigations as well as understanding of CoAP specific needs:

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 at the CoAP level, 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 ROLL [I-D.ietf-roll-rpl]. Insecure or inappropriate routing (including 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).

8. IANA Considerations

This document makes no request of IANA.

9. Conclusions

Consider the proposals for group communication described in this draft for incorporation into the overall CoAP protocol specification.

10. 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.

11. References

11.1. 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.

[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.

[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.

[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.

11.2. 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-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-02 (work in progress), December 2010.

[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-02 (work in progress), October 2010.

[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-18 (work in progress), February 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>.

Author's Address

Akbar Rahman (editor)
InterDigital Communications, LLC

Email: Akbar.Rahman@InterDigital.com

CoRE
Internet-Draft
Intended status: Informational
Expires: September 15, 2011

P. van der Stok
Philips Research
K. Lynn
Consultant
March 14, 2011

CoAP Utilization for Building Control
draft-vanderstok-core-bc-03

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. RFC 3986 defines the URI components as (1) a scheme, (2) an authority, used here to locate the building, area, or node under control, (3) a path, used here to locate the resource under control, and (4) a query part (fragments are not supported in CoAP.) Next, it is shown that DNS can be used to locate URIs on the scale necessary in large commercial BAC deployments. Finally, a method is proposed for mapping URIs onto legacy BAC resources, e.g., to facilitate application-layer gateways.

This proposal supports the view that (1) 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, and (2) service discovery is complimentary to resource discovery and facilitates control network scaling.

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 9
4. Discovery 11
4.1. DNS-Based Service Discovery 11
4.2. Service vs Host Names 12
4.3. Browsing for Services 12
4.4. Resource vs Service Discovery 12
4.5. CoRE Link Extensions for DNS-SD 13
4.5.1. Service Name "sn" attribute 13
4.5.2. Service Type "st" attribute 14
4.5.3. Service Subtype "ss" attribute 14
5. Data Representations in CoAP 14
5.1. Network architectures 15
5.2. Gateways to legacy networks 17
5.3. Commissioning CoAP devices 18
5.3.1. DNS-SD server present 19
5.3.2. DNS-SD server not present 20
6. Conclusions 20
7. Security considerations 21
8. IANA considerations 22
9. Acknowledgements 22
10. Changelog 22
11. References 22
11.1. Normative References 22
11.2. Informative References 24
Authors' Addresses 25

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.

The term "service" often means different things to different communities and even different things to the same community. In building control protocol standards, service is often used to refer to a function in the RPC sense. In this context, we generally substitute the term "function". In the IETF community, service may often refer to an abstract capability such as "datagram delivery". In this submission we use the term service, in the sense defined by "DNS-based Service Discovery" [I-D.cheshire-dnsext-dns-sd], as equivalent to a CoAP end-point (or server).

A CoAP end-point is identified by the authority part of a URI. We refer to this end-point (which is resolved to an {IP address, port} tuple) as a "node". By "device" we generally mean the physical object handled by the installer. While a device may host more than one service, for simplicity we assume here that a given device may only host a single CoAP node.

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 that is targeted to a network of nodes 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 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 nodes in a typical building control application are resource constrained, making the standardization of a lightweight

application protocol like CoAP a necessary requirement for IP to penetrate the device market. This approach is further indicated by the low energy consumption requirement of battery-less nodes. 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 the physical medium for data objects and function invocation. Many of these industry standards also specify lower-level functionality such as 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 basic syntax assumptions 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 aims at 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 is related to the chosen 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 nodes that is targeted. A group scope may be defined as link-local, or as a tree maintained by IP-multicast 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 protocol oriented assumptions are:

- Nodes may be addressed by one or more groups.

- Resources addressed by a group must be uniformly named across all targeted nodes.

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 node in several building areas by nodes inside or outside the building, and humans may intervene to change control settings.

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 6.1 of [I-D.ietf-core-coap] and is compatible with the "http" scheme specification [RFC2616]. The host part of the authority may be represented either as a literal IP address or as a fully qualified domain name. While scheme is irrelevant from the perspective of the service, it is used in service discovery to identify the protocol used to access the service.

TBD: we have yet to fully explore the utility of a separate scheme (e.g., "coapm") to support group communication models as described in [I-D.rahman-core-groupcomm].

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 node or group of nodes, 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 node 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 CoAP spec [I-D.ietf-core-coap] states "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]." 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 it is conventional to treat different ports as representing different authorities and there is no assurance that a coap server will consistently acquire the same dynamic port.

A building can be unambiguously addressed by it 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 node within the building. Arriving at the node identified by //bldg.5533BA-125a.nl, the receiving service can parse the path portion of the URI and perform the requested method 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. //node-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 seems to lead to rather verbose URI strings in the packet, contrary to the small packet assumption. However, the design of CoAP is such that the authority portion of the URI need not be transmitted in requests sent to servers. The question arises as to whether the syntax of the authority part needs to be standardized for building control. Given the examples later in the text, this appears more to be the concern of the building owner or the installer than a standardization concern.

2.3. Path part

Every network addressable resource is completely identified by a URI scheme://authority/path. The path part of the URI specifies the resource within a given node. The representation of object types and their associated attributes are typically subjects for standardization. There is no widely accepted standard for uniformly naming building control device structure in a URI. A vigorous effort is undertaken by the oBIX working group of OASIS [oBIX], but its current impact is limited.

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 node with name t-sensor1 exists, is of a given standard type (e.g. ZigBee temperature sensor), and that this standard type has the readable attribute: temperature. When commands are sent to a group of nodes it MUST be the case that the targeted resource has the same path on all targeted nodes. 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).

A better long-term solution is to build on the concepts presented in [I-D.ietf-core-link-format] and identify resources of a given object model in terms of a registered "/.well-known" prefix. 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 pathnames that may occur under this prefix. This allows the XXX standard development organization to assume responsibility for defining the name space and resources associated with the prefix. The registered /.well-known/XXX URI effectively defines a standard object model, or schema. Manufacturers may optionally define proprietary resources that can be discovered dynamically using methods described below.

3. Group Naming and Addressing

Given a network configuration and associated prefixes, 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 groups become:

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 on floor 1, west wing, ..."
//all.bu036.floor1.west.bldg6...	"all nodes 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 nodes 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]. Some techniques in this proposal, e.g. service discovery as described below, can be accomplished with a single coap-specific multicast address as long as the desired scope is building-wide.

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 an 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 Uri-Host option [I-D.ietf-core-coap] need not be sent as part of the message. However to identify the resource that is addressed, a short version of the resource path can be inserted as an option as explained in [I-D.ietf-core-link-format].

The binding of a group FQDN to 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 source or receiver node. 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 its name. In the limit, the group can degrade to a single controller that represents a non-networked subsystem replacing the original networked group of nodes. Group membership may be managed by a protocol such as Multicast Listener Discovery [RFC5790].

A group defines a set of nodes. All resources on a given node are referenced by the multicast address(es) to which the node belongs. A given node might belong to a number of groups. For example the node 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". Assuming that belonging to a group has as only consequence for the group member

that 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 server but not necessarily on the low-resource destinations or sources. Assuming that resolution of addresses only happens at node start-up, the complexity of the DNS server need not affect the responsiveness of the nodes.

In summary, the authority portion of the URI is used to identify a node (group) and the resulting DNS name is bound to a unicast (multicast) address. Naming is building or organization dependent, must be flexible, and does not require standardization efforts but SHOULD conform to some uniform convention.

4. Discovery

4.1. 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 such as CoAP nodes 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>, where the service type for CoAP nodes is "_coap._udp". The identifier "_udp" provides a transport protocol hint as required by the SRV record definition [RFC2782] and "_coap" identifies the application protocol. A PTR record with the label "_coap._udp" is defined for each CoAP end-point in the zone, and this record’s value is set to the service instance name (which in turn identifies to SRV and TXT records for the CoAP end-point).

DNS-SD also supports one level of subtype, which can be used to locate coap services based on object model (schema), for example: _bacnet._sub._coap._udp, _dali._sub._coap._udp, or _zigbee._sub._coap._udp. The maximum length of the type and subtype fields is 14 octets, which could allow for "schema-function" descriptors such as _dali-light, _dali-switch, etc.

The Location part of the service name is identical to the global (DNS subdomain) part of the authority in URIs that identify the resources on this node or group and may identify a building zone as in the examples above.

The Instance part of the service name may be changed during the commissioning process. It must be unique 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 node level, or it could represent a particular resource within a coap node. The SRV record contains the host (AAAA record) name and port of the service. In the case where a service name identifies a particular functional entry point, the path part of the URI may be placed in the TXT record.

4.2. Service vs Host Names

In general, the authority "www.example.com" does not refer to a canonical host name (the label of a AAAA record). Logically, it refers to the "world wide web service" for the example.com domain. Literally, the "www" is probably the label of a CNAME record that names a AAAA record that may in turn specify more than one address (in the case of round-robin load leveling between identical origin server instances).

The SRV record functions something like the CNAME in this case, except that it is capable of resolving to an IP address plus a listening socket (though, as we said, the use of dynamic sockets is not recommended in URIs). An optional TXT record may be configured wih same name as the SRV record and be used to store context-dependent key=value pairs. For example, a multi-function device might define a service name for each "base URI" that locates the start of an object resource (e.g. abs-path=/.well-known/zigbee/sensor/). Thus, the URI coap://host.example.com/temp might resolve through DNS-SD lookups to coap://[fdfd::1234]/.well-known/zigbee/sensor/temp.

4.3. Browsing for Services

CoAP nodes in a given subdomain may be enumerated by sending a DNS query for PTR records named _coap._udp to the authoritative server for that zone. A list of names for SRV records matching that ServiceType.Location is returned. Each SRV record contains the port and host name 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. Apart from defining the standard resources identified by schema=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.

4.4. Resource vs Service Discovery

While 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 resource URIs within a web service. [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 that identify all resources present on the node that are exposed as functions.

Assuming the ability to multicast the GET over the local link, coap resource discovery 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 by using /.well-known/core attributes as the basis for a unique "Instance" name, defaulting to "_coap._udp" for the ServiceType, and using some means to establish in which subdomain the service should be registered (TBD). The DNS TXT record can be populated by importing the other resource description attributes as they share the same key=value format specified in Section 6 of [I-D.cheshire-dnsext-dns-sd].

4.5. CoRE Link Extensions for DNS-SD

The following CoRE specific target attributes are proposed as extensions to [I-D.ietf-core-link-format] to support DNS-SD. The values are intended to be imported directly into a DNS- SD zone file are are subject to format and length constraints as specified in [I-D.cheshire-dnsext-dns-sd].

```
link-extension    = ( "sn" "=" quoted-string )
link-extension    = ( "st" "=" quoted-string )
link-extension    = ( "ss" "=" quoted-string )
```

4.5.1. Service Name "sn" attribute

The service name "sn" attribute is the <Instance> portion of a DNS-SD service instance name. It is stored directly in the DNS as a single DNS label of canonical precomposed UTF-8 [RFC3629] "Net-Unicode" (Unicode Normalization Form C) [RFC5198] text. However, to the extent that the "sn" attribute may be chosen to match the DNS host name of a proxy or gateway, it SHOULD use the syntax defined in Section 3.5 of [RFC1034] and Section 2.1 of [RFC1123].

The <Instance> portion of the name of a service being offered on the network SHOULD be configurable by the user setting up the service, so that he or she may give it an informative name. However, the device or service SHOULD NOT require the user to configure a name before it can be used. A sensible choice of default name can allow the device or service to be accessed in many cases without any manual

configuration at all. The default name should be short and descriptive, and MAY include the device's MAC address, serial number, or any similar hexadecimal string in an attempt to make the name globally unique.

DNS labels are currently limited to 63 octets in length and the entire service instance name may not exceed 255 octets.

4.5.2. Service Type "st" attribute

The service type "st" attribute is composed into the <ServiceType> portion of a DNS-SD instance name as follows. It is limited to 14 octets of Net-Unicode text. If ommitted, it defaults to "coap". An underscore '_' is prepended to the value of the "st" attribute, which is then concatenated with a period '.', and finally the "_udp" identifier. The resulting string is used to form labels for DNS-SD records and as such is stored directly in the DNS.

4.5.3. Service Subtype "ss" attribute

The service subtype "st" attribute, if present, follows the format and composition rules defined in the previous section. It is then concatenated with a period '.' and the <ServiceType> string defined above to form additonal labels for DNS-SD records as defined in Section 4.1.

5. Data Representations in CoAP

Before CoAP devices can come to market, manufacturers must agree that the type and attributes 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 believe 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. As pointed out earlier, this has consequences for the naming of resources. In some cases a CoAP device can handle more than one legacy representation. Given that a CoAP device 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).

5.1. Network architectures

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 nodes -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 IP/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 node returns data to the requesting zzz;yyy coap host via the same gateway.

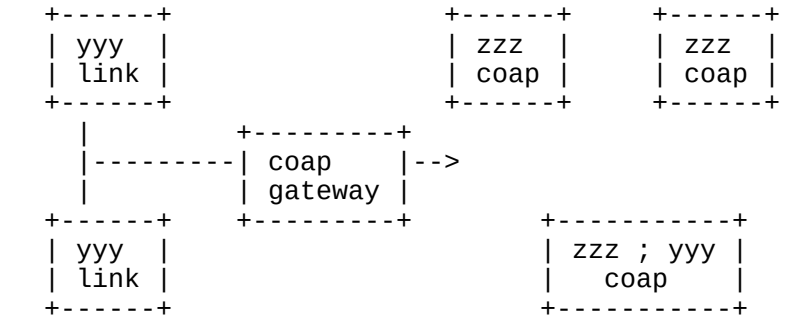


Figure 1: network with multiple representation standards

There are at least four ways that the CoAP hosts can address the legacy nodes behind the gateway.

- All nodes of legacy network YYY share the same Uri-Host also used for the coap gateway on the coap network. Every legacy node 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 node.

- All nodes 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 node. The routing protocol on the coap network makes the message arrive at the coap gateway. The gateway determines the specified legacy node from the destination IP address.

- All nodes of legacy network YYY have different authorities. This means that the possibly lenghy authority names need to be transmitted. The gateway recognizes the authorities and maps authority to legacy node.

- All nodes of legacy network YYY have different ports. This can be expresed in two ways (1) as :port in the URI, or (2) can be defined 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 node 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 device from a controller that can producee zzz commands is given. Assume the resource path /.well-known/zzz identifies the parser of the ZZZ syntax. A 12 octet message 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.

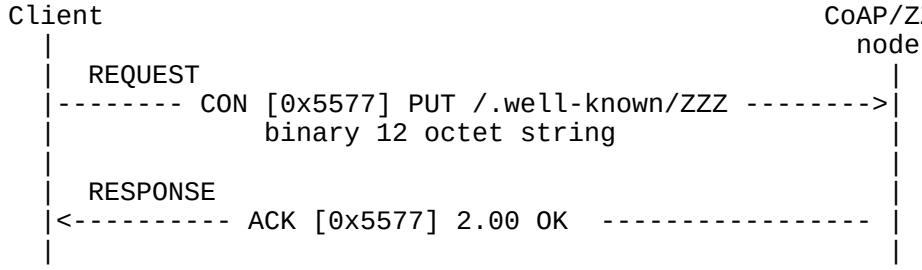


Figure 2: Sending a ZZZ command with coap to coap/ZZZ node

In the example the format of the payload is determined by the zzz standard. The path /.well-known/ZZZ is obtained from the TXT record for this service, e.g., schema=ZZZ. The 12 octet binary payload specifies the zzz command. The port number in the UDP header identifies the zzz parser.

An example of an invocation of a DALI legacy node behind a gateway is given. Assume the resource path /.well-known/DALI where the port

number identifies the DALI node. The application sets a value of 200 in the DALI node in the attribute 256 defined by the DALI spec.

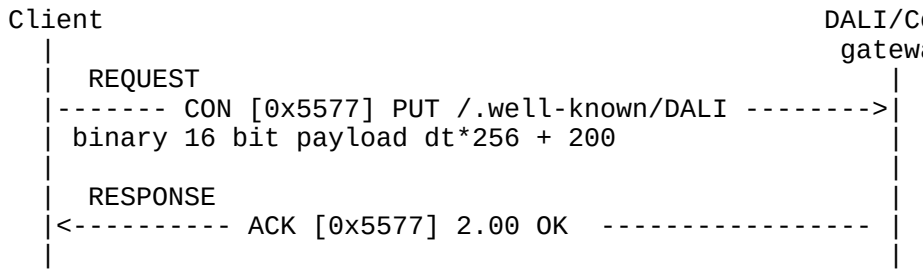


Figure 3: Sending a DALI setting with coap to coap/DALI gateway

In the example the format of the payload is determined by the legacy standard. The path /.well-known/DALI is obtained from the TXT record for this service, e.g., schema=DALI. The 16-bit binary payload specifies that the attribute dt of the dali compatible resource is set to 200. The port number in the UDP header identifies the DALI node.

5.2. Gateways to legacy networks

Two types of gateways are considered; (1) coap gateway to a single legacy link, called yyy/coap gateway, and (2) coap gateway to legacy network, called xxx/coap gateway. The source encapsulates the data with the corresponding representation inside a CoAP message and sends these messages to the gateway. In the gateway the XXX/YYY data is removed from the CoAP message and transported to the desired node. Returning an answer to the invoking host needs to be done in two different ways dependent on the addressing type of the XXX/YYY standard. The IP-node-identifier (INI) can be (1) the IP-address, (2) the IP-address, port number, (3) the URI, or (4) the path .

- The packet contains a YYY link address. In the gateway two tables are maintained. Table 1 contains a link from INI to YYY address. Table 2 contains a link of the source IP address to the destination YYY address for the active request. A yyy/coap host with IP address, IPs, sends a request to the INI, IPd, of the specified YYY device with link address Yd. The packet is routed to the coap gateway. The gateway strips the link, network and coap information from the packet and sends the message to the Yd specified in table 1 with the (Yd, IPd) pair. The gateway stores the source IP address with the destination YYY address in table 2

as pair (IPs, Yd). When an answer is returned from Yd, the gateway creates a new coap packet with the destination address, IPs, as found in table 2 and sends it on to the yyy/coap host, IPs.

- The packet contains a XXX network address. In the gateway two tables are maintained. Table 1 contains a mapping from XXX network addresses to INI for all XXX devices. Table 2 contains a mapping from IP addresses to XXX network addresses for IP devices. The xxx/coap host with IP address, IPs, sends a request to the INI, IPd, of the specified XXX device with network address Xd. The packet is routed to the coap gateway. The gateway strips the link, network and coap information from the packet and sends the message to the Xd specified in table 1 with the (Xd, IPd) pair with as source Xs as specified in table 2 with the (Xs, IPs) pair. When an answer is returned from Xd to Xs, the gateway creates a new coap packet with the destination address, IPs, as found in table 2 with the (IPs, Xs) pair and with source address IPd as found in table 1 with (Dd, IPd) pair. The gateway sends the answer on to the xxx/coap host, IPs.

It is assumed that the gateway conforms to the core standard at the internet interfaces. Consequently, all resources are visible at /.well-known/core by invoking a GET. These entries can be entered into DNS-SD with a commissioning tools as proposed in section 5.3; according to the rules specified in section 4. Filling in the address mapping tables is done in a similar way as done for other application level gateways (ALG).

5.3. Commissioning CoAP devices

Commissioning means mapping a physical device identified by its location to an URI. Given that mapping of URI to IP address is done elsewhere (e.g. DNS), the mapping of location to IP address is done during commissioning with a commissioning tool. The commissiong is done for a set of coap nodes interconnected by a wireless network. The netwok can contain zero or more 6LR routers and zero or more 6LBR. The IP infrastructure (DNS, DHCP servers) is connected to the coap network via the 6LBR. Two cases are considerd for commissioning: (1) no 6LBR and no DNS server connected, and (2) a 6LBR connects 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 provide identifiers for all these devices. The triple Instance.ServiceType.Location, as proposed in DNS-SD, is used to describe the commssioning process. The identifiers of the devices often reflect their action domain which is linked to their physical

location. The authority can be equated to the Location identifier. The Location uniquely identifies the device. The Instance is the unique identifier given to the device in the factory but which has no relation to its later function. The ServiceType together with the Location fully determine the semantics of the data returned by the device. Commissioning is the process of mapping the Location to Instance for all installed hosts. The mapping is stored in a discovery repository such that applications can communicate with the required devices.

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) exists in a host after physical installation of the devices:

- 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 Location’s to which hosts need to be assigned.
- A DHCP server (neigbor discovery) provided each host with a (link-local) IP address.

Consider the commissioning process (1) with a central DNS-SD server and (2) without a central server using mDNS.

5.3.1. DNS-SD server present

The names with the corresponding authority prefixes can be grouped by the tool and appropriate group names can be assigned. The names stored in the tool are inserted into the DNS-SD server, respecting all DNS rules with respect to domain naming, and authority structure. Assuming that the authority syntax corresponds with the structure of the building, the installer can select on a screen the subset of names belonging to the building part he wants to commission. The installer reads with a bar code reader, attached to the tool, the identifier 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 informs the DNS-SD server that the read barcode belongs to the selected authority. This is done for all devices within a given part of the building. Once this manual commissioning part is done the nodes conclude the commissioning process:

- Each node asks the DNS-SD server the authority belonging to its barcode.

- Each node updates the authority in the DNS-SD server with its IP address.

- Each node appends for each resource the path name to the authority and inserts it into the DNS-SD server with its IP address.

After the commissioning process, all resources of each node have an URI and IP address which are stored in the central DNS-SD server. When nodes are restarted, the DHCP server allocates IP addresses to the node and updates 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 link) such that each node on a given segment can communicate directly with any other node 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 link-local authorities and stored in the tool. All nodes are assumed to have selected a link-local IP address. Assume that every device has a unique barcode within the building and that the corresponding node knows the bar code number. The installer reads with a bar code reader, attached to the tool, the identifier 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 nodes. The node with the given barcode number, extends the authority with the path name of the resources. For each resource, the node multicasts the link-local IP-address and the link-local URI to the mDNS servers in the connected nodes. This concludes the commissioning of a network segment. All resources of each node have a link-local URI and a link-local IP address which are stored in the mDNS servers.

6. 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 subdomain delegation and naming can be used to express this hierarchy in the authority portion of the URI, down to the group or node 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 node(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 url to distinguish between resources on a given node, resulting in very compact identifiers.

DNS-SD [I-D.cheshire-dnsext-dns-sd] can be used to scale up service discovery beyond the local link. 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 node, locate coap nodes by subtype, or bind service names for particular coap URIs.

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. This necessitates the definition of an additional IANA mime code, and the mapping of legacy specific discovery semantics to CoAP resource discovery messages or DNS-SD lookups.

7. 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 nodes. The CoRE security analysis must be broadened to include multicast scenarios.

8. 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.

9. 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.

10. 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.

11. References

11.1. 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.

11.2. 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-04 (work in progress), January 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-02 (work in progress), December 2010.

[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., "Group Communication for CoAP", draft-rahman-core-groupcomm-04 (work in progress), March 2011.

[BACnet] Bender, J. and M. Newman, "BACnet/IP", Web http://www.bacnet.org/Tutorial/BACnetIP/index.html.

[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, 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>.

[oBIX] "oBIX working group", Web http://www.obix.org, 2003.

[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.

Authors' Addresses

Peter van der Stok
Philips Research
High Tech Campus
Eindhoven, 5656 AA
The Netherlands

Email: peter.van.der.stok@philips.com

Kerry Lynn
Consultant

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

Constrained RESTful Environments
Internet-Draft
Intended status: Informational
Expires: September 9, 2011

M. Vial
Schneider-Electric
Z. Shelby
Sensinode

March 8, 2011

Interface description with WADL in CoRE
draft-vial-core-link-format-wadl-00

Abstract

This document provides guidelines to use the Web Application Description Language (WADL) in Constrained RESTful environments. The document mainly focus on how to combine WADL with the CoRE Link Format to describe a REST interface.

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/1id-abstracts.txt.

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

This Internet-Draft will expire on September 9, 2011.

Copyright Notice

Table of Contents

1. Introduction 3
2. Requirements Language 3
3. WADL in a CoRE environment 3
3.1. CoAP adaptations 3
3.1.1. Methods 4
3.1.2. Status code 4
3.1.3. HTTP header parameters 4
3.1.4. Mutliple representations 4
3.2. CoRE resources 4
3.3. Semantic description 5
3.4. Binary XML format 5
4. Use cases 5
4.1. Resource type identifiers 5
4.2. Description of query parameters 8
4.3. Interface description and associated semantic 9
4.4. Collection of resources 9
5. Acknowledgements 11
6. IANA Considerations 11
7. Security Considerations 11
8. References 11
8.1. Normative References 11
8.2. Informative References 12
Authors' Addresses 12

1. Introduction

The Constrained RESTful Environments (CoRE) working group aims at providing a comprehensive suite of standards that will make it possible to build a REST architecture for M2M applications with highly constrained nodes and networks.

The CoRE Link Format [I-D.ietf-core-link-format] which is part of this suite defines a format to be used by CoAP servers to list hosted resources using the Web linking technique defined in RFC 5988 [RFC5988]. More specically the 'd' attribute of Link Format allows an interface designer indicate a description of the behavior, the parameters, the representation and eventually the set of sub-resources associated to a given CoRE resource. One way to describe the interface to a resource is using the Web Application Description Language (WADL).

The first part of this document will explain how to benefit from WADL to describe the REST interface of CoRE resources. Then the second part of the document will show how the previous guidelines are applied in different use cases.

The reader should keep in mind that this document does not suggest in any way constrained nodes would be able to retrieve and parse a WADL description. The interface description is rather considered as documentation with a standard and machine-processable language that will help implementors to understand an interface and eventually generate stub code.

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].

3. WADL in a CoRE environment

3.1. CoAP adaptations

The WADL language is primilarily designed to describe HTTP-based web applications. WADL is not strongly tied to the HTTP protocol [RFC2616] and any HTTP-like web protocol can be described with WADL. In a CoRE environment the CoAP protocol [I-D.ietf-core-coap] is deployed in place of the HTTP protocol as an optimized web transfer protocol. An interface designer must take into consideration the specifity of CoAP while writing the WADL definition.

3.1.1. Methods

CoAP only supports a subset of HTTP methods. So a WADL description deployed in a CoRE environment MUST only make use of methods available in CoAP namely GET, PUT, POST and DELETE.

3.1.2. Status code

CoAP decorrelates the response code representation from the actual value of the code. Hence the response code 2.00 has the value 64. When a CoAP response code is associated to the description of a response in WADL, it is RECOMMENDED to use the response code labels.

3.1.3. HTTP header parameters

CoAP does not support user-defined options in the base specification. So as a rule of thumb, header parameters are discouraged with CoAP.

3.1.4. Mutliple representations

CoAP does not support content negotiation so only one media type can be associated to a request or a response.

3.2. CoRE resources

The WADL language describes a REST resource with a resource element which associates a REST behavior to a URI. WADL offers language elements to describe the following aspects of a REST behavior: allowed methods, query string parameters, media type of the request and response content, URI of the resource and the subordinate resources. The description of the behavior can either be a reference to a resource_type element or child elements if the description is inline.

When WADL is combined with the CoRE Link Format it is RECOMMENDED to write definitions with resource_type elements. Then a Web link can reference a resource_type with the ’d’ attribute. So a Web link plays the same role as the resource element of WADL in the sense that the Web link instantiates a resource_type by linking it to an URI.

Because the resource_type element is referenced outside of the WADL description, the rules of section 2.5.1 in WADL [wadl] are not applicable. Instead the target URI of the Web link where the resource_type element is referenced MUST be used as the base URI to construct each child resource identifiers.

The ’d’ attribute of a Web link SHOULD reference a resource_type AND a WADL document to avoid potential ambiguities. resource_type

elements are identified by their XML id. The 'd' attribute MAY take the form of an URI. The path of the URI specify the WADL document while the fragment part of the URI points to a resource_type. The full URI notation may add significant overhead in a link format description thus several formats are acceptable depending on the risk of identifier conflicts. Here are few examples of 'd' attributes.

- http://www.example.org/interface.wadl#resourceType
- interface.wadl#resourceType
- resourceType

3.3. Semantic description

The main goal of the WADL description in a CoRE environment is to describe the actions that can be performed on a REST resource. The WADL document may include a grammar element with a schema to offer a detailed description of data representations hence semantically refining the description. But this mechanism is not applicable to all data representations especially if the data is not XML-based. Moreover the interface description is not meant to be directly interpreted by CoRE nodes. Thus it is RECOMMENDED to associate the semantic description of a resource with a 'n' attribute without relying only on the 'd' attribute. This separation of concerns allows an interface designer to reuse the same interface description for resources that grasp different concepts.

3.4. Binary XML format

Because CoRE deals with constrained networks, traditional XML data representations may be superseded with a more compact format for the XML information set. Efficient XML Interchange [exi] is an example of such binary XML format which heavily relies on a XML schema to achieve the best compression performances. The schema identifier can be carried inline with the binary stream or specified out-of-band. If there is no schema identifier present in the data stream but the WADL definition of the representation has a reference to grammar element, one MUST assume that the data stream is schema-informed.

4. Use cases

4.1. Resource type identifiers

Let's consider an organization which has defined two application profiles in two separate WADL documents. The first profile targets Home Automation applications while the second deals with Energy

Management. One CoRE device may implement REST services from both profiles.

The WADL descriptions are versioned to support future evolutions of the interface. The profiles have a class/type structure with a dot notation for REST resource types. They also share some concepts but with different implementations. Figure 1 and Figure 2 are short extracts of possible WADL descriptions for such profiles.

```
<application xmlns="http://wadl.dev.java.net/2009/02"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd">
  <resource_type id="sensor.temperature">
    <method name="GET" id="GetTemperature">
      <request>
        <representation mediaType="text/plain" />
      </request>
    </method>
  </resource_type>
  <resource_type id="parameter.threshold">
    <method name="PUT" id="SetThreshold>
      <request>
        <representation mediaType="text/plain" />
      </request>
    </method>
 </resource_type>
</application>
```

Home Automation WADL sample (ha1.wadl)

Figure 1

```
<application xmlns="http://wadl.dev.java.net/2009/02"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd">
  <resource_type id="meter.power">
    <method name="GET" id="GetPower">
      <request>
        <representation mediaType="application/xml" />
      </request>
    </method>
  </resource_type>
  <resource_type id="parameter.threshold">
    <method name="PUT" id="SetThreshold>
      <request>
        <representation mediaType="application/xml" />
      </request>
    </method>
  </resource_type>
</application>
```

Energy Management WADL sample (em2.wadl)

Figure 2

In a home network, the devices share the same infrastructure but usually come from different vendors and may implement many application profiles. In this context it is useful to reference a WADL interface with an absolute URI.

```
REQ: GET /.well-known/core
RES: 2.00 OK
</tmp>;n="AirTemperature";
d="http://www.example.org/ha1.wadl#sensor.temperature",
</tmp/thr>;n="TemperatureAlarm";
d="http://www.example.org/ha1.wadl#parameter.threshold"
</pwr>;n="PowerConsumption";
d="http://www.example.org/em2.wadl#meter.power",
</pwr/thr>;n="PowerAlarm";
d="http://www.example.org/em2.wadl#parameter.threshold"
```

If a deployment of devices is known to implement only REST services from one organization the resource_type identifiers may be shortened. It is however indispensable to clearly indicate a WADL document because the resource_type identifiers are only unique within a single WADL document. The example below reflects how these assumptions are actually applied.

```
REQ: GET /.well-known/core
RES: 2.00 OK
</tmp>;n="AirTemperature";d="ha1.wadl#sensor.temperature",
</tmp/thr>;n="TemperatureAlarm";d="ha1.wadl#parameter.threshold"
</pwr>;n="PowerConsumption";d="em2.wadl#meter.power",
</pwr/thr>;n="PowerAlarm";d="em2.wadl#parameter.threshold"
```

If the network is dedicated to a specific application profile it is acceptable to omit the reference to the WADL description which is supposed to be known out-of-band. Web links may have the following format:

```
REQ: GET /.well-known/core
RES: 2.00 OK
</tmp>;n="AirTemperature";d="sensor.temperature",
</thr>;n="TemperatureAlarm";d="parameter.threshold"
```

4.2. Description of query parameters

A typical usage of WADL is to provide a detailed description of how a client can build the query string component of a URI to access a parametrized resource. Below is an example that describes how a client can select the unit for a temperature sensor.

```
<application xmlns="http://wadl.dev.java.net/2009/02"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd">
  <resource_type id="sensor.temperature">
    <method name="GET" id="GetTemperature">
      <request>
        <param name="unit" style="query" default="C" required="no">
          <option value="C"><doc>Celsius</doc></option>
          <option value="K"><doc>Kelvin</doc></option>
          <option value="F"><doc>Farenheit</doc></option>
        </param>
      </request>
    </method>
  </resource_type>
</application>
```

Definition of an optional query string

Figure 3

This description give information about four valid URIs that are exposed in the following CoAP exchange.

```
REQ: GET /.well-known/core
RES: 2.00 OK
</tmp>;d="sensor.temperature"

REQ: GET /tmp
RES: 2.00 OK
20

REQ: GET /tmp?unit=C
RES: 2.00 OK
20

REQ: GET /tmp?unit=K
RES: 2.00 OK
293.15

REQ: GET /tmp?unit=F
RES: 2.00 OK
68
```

4.3. Interface description and associated semantic

The same interface description can often be reused for similar but distinct concepts. For example a temperature sensor may be able to produce the traditional air temperature but also the effective temperature which is a combination of air temperature and wind speed. Then the definition in Figure 3 is valid for both concepts and the device description could look like depicted below.

```
REQ: GET /.well-known/core
RES: 2.00 OK
</tmp>;n="DryBulbTemperature";d="sensor.temperature",
</eff>;n="EffectiveTemperature";d="sensor.temperature"
```

4.4. Collection of resources

Repeating an interface definition attribute with the same identifier for a collection of resources is especially inefficient and laborious with link format. Hopefully WADL supports templated path definitions to describe sub-resources. The template style of parameters allows an interface designer to specify the dynamic path elements of a URI thanks to a curly brace notation. It is also possible to precisely determine the data type associated to a variable path element. Below is an example of how a list of pending alarms can be described with this feature.

```
<application xmlns="http://wadl.dev.java.net/2009/02"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd">
  <resource_type id="list.alarms">
    <method name="GET" id="GetAlarmList">
      <request>
      </request>
      <response>
        <representation mediaType="application/link-format"/>
      </response>
    </method>
    <method name="POST" id="AddAlarm">
      <request>
        <representation mediaType="application/xml"
            element="Alarm"/>
      </request>
    </method>
    <resource path="{alarmId}">
      <param name="alarmId" style="template" type="xsd:int"/>
      <method name="GET" id="GetAlarm">
        <request>
        </request>
        <response>
          <representation mediaType="application/xml"
              element="Alarm"/>
        </response>
      </method>
      <method name="DELETE" id="RemoveAlarm">
        <request>
        </request>
      </method>
    </resource>
  </resource_type>
</application>
```

Definition of a collection of resources

Figure 4

Then the resource_type is referenced only once but provides an interface description for the whole collection of resources.

```
REQ: GET /.well-known/core
RES: 2.00 OK
</tmp>;n="DryBulbTemperature";d="sensor.temperature",
</alrm>;n="TemperatureAlarms";d="list.alarms",

REQ: GET /alrm
RES: 2.00 OK
</alrm/1>,
</alrm/2>

REQ: GET /alrm/1
RES: 2.00 OK
<Alarm time="" type="GreaterThan" threshold="28" />
```

5. Acknowledgements

6. IANA Considerations

This document requests no actions from IANA.

7. Security Considerations

This document has no known security implications.

8. References

8.1. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-02 (work in progress), December 2010.

[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.

[exi] W3C, "Efficient XML Interchange (EXI) Format 1.0", 2011,

<http://www.w3.org/TR/2011/PR-exi-20110120/>.

[wadl] Hadley, M., "Web Application Description Language (WADL)", 2009, <http://java.net/projects/wadl/sources/svn/content/trunk/www/wadl20090202.pdf>.

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.

Authors' Addresses

Matthieu Vial
Schneider-Electric
Grenoble,
FRANCE

Phone: +33 (0)47657 6522
Email: matthieu.vial@schneider-electric.com

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach@sensinode.com