

| | | |
|----------------------------------|------------------------|--|
| CoRE | Z. Shelby | |
| Internet-Draft | Sensinode | |
| Intended status: Standards Track | B. Frank | |
| Expires: January 9, 2011 | SkyFoundry | |
| | D. Sturek | |
| | Pacific Gas & Electric | |
| | July 8, 2010 | |

[TOC](#)

Constrained Application Protocol (CoAP)

draft-ietf-core-coap-01

Abstract

This document specifies the Constrained Application Protocol (CoAP), a specialized RESTful 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 the REST Method/Response interaction model between application end-points, supports built-in resource discovery, and includes key web concepts such as URIs and content-types. CoAP easily translates to HTTP for integration with the web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 9, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

- [1.](#) Introduction
- [2.](#) Constrained Application Protocol
 - [2.1.](#) Interaction Model
 - [2.1.1.](#) Synchronous response
 - [2.1.2.](#) Asynchronous response
 - [2.2.](#) Transaction messages
 - [2.2.1.](#) Confirmable (CON)
 - [2.2.2.](#) Non-Confirmable (NON)
 - [2.2.3.](#) Acknowledgment (ACK)
 - [2.2.4.](#) Reset (RST)
 - [2.2.5.](#) Transaction IDs
 - [2.3.](#) Methods
 - [2.3.1.](#) GET
 - [2.3.2.](#) POST
 - [2.3.3.](#) PUT
 - [2.3.4.](#) DELETE
 - [2.4.](#) Response Codes
 - [2.5.](#) Options
 - [2.5.1.](#) Option Processing
 - [2.5.2.](#) URIs
 - [2.5.3.](#) Content-type encoding
- [3.](#) Message Formats
 - [3.1.](#) CoAP header
 - [3.2.](#) Header options
 - [3.2.1.](#) Content-type Option
 - [3.2.2.](#) Uri-Scheme Option
 - [3.2.3.](#) Uri-Authority Option
 - [3.2.4.](#) Uri-Path Option
 - [3.2.5.](#) Location Option
 - [3.2.6.](#) Max-age Option
 - [3.2.7.](#) Etag Option
- [4.](#) UDP Binding

| | |
|-----------------------|---|
| 4.1. | Multicast |
| 4.2. | Retransmission |
| 4.3. | Congestion Control |
| 4.4. | Default Port |
| 5. | Caching |
| 5.1. | Cache control |
| 5.2. | Cache refresh |
| 5.3. | Proxying |
| 6. | Resource Discovery |
| 6.1. | Link Format |
| 6.2. | Naming |
| 6.3. | Query Filtering |
| 6.4. | HTTP Resource Discovery |
| 7. | HTTP Mapping |
| 8. | Protocol Constants |
| 9. | Examples |
| 10. | Security Considerations |
| 11. | IANA Considerations |
| 11.1. | Codes |
| 11.2. | Content Types |
| 12. | Acknowledgments |
| 13. | Changelog |
| 14. | References |
| 14.1. | Normative References |
| 14.2. | Informative References |
| S | Authors' Addresses |

1. Introduction

[TOC](#)

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 CoRE has been to keep message overhead small, thus limiting the use of fragmentation.

One of the main goals of CoRE is to design a generic RESTful 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, but rather to realize a subset of REST common with HTTP but optimized for M2M applications. Although CoRE 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 transactions. This document specifies the RESTful 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 [\[I-D.shelby-core-coap-req\] \(Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R. Kelsey, "CoAP Requirements and Features," April 2010.\)](#). CoAP has the following main features:

- *Constrained RESTful 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 transaction support.
- *Low header overhead and parsing complexity.
- *URI and Content-type support.
- *Built-in resource discovery.
- *Simple proxy and caching capabilities.

2. Constrained Application Protocol

[TOC](#)

This section specifies the basic functionality and processing rules of the protocol.

2.1. Interaction Model

[TOC](#)

The REST 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 REST exchange 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. A response is then sent with a Response Code and resource representation if appropriate.

Unlike HTTP, CoAP deals with these REST interchanges asynchronously over a UDP transport with support for both unicast and multicast interactions. This is achieved using transaction messages (Confirmable, Non-Confirmable, Acknowledgment, Reset) supporting optional reliability (with exponential back-off) and transaction IDs between end-points to carry REST requests and responses. These transactions are transparent to the REST interchanges. The only difference being that responses may arrive asynchronously.

One could think of CoAP as using a two-layer approach, a transactional layer used to deal with UDP and the asynchronous nature of the interactions, and the REST interactions using Method and Response codes.

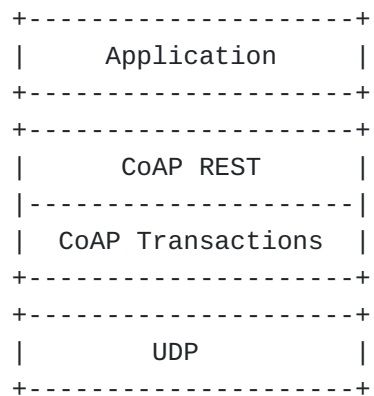


Figure 1: Abstract layering of CoAP

2.1.1. Synchronous response

[TOC](#)

The most basic interaction between the REST and Transaction layers works by sending a REST request in a confirmable CoAP message and waiting for an acknowledgment message that also carries the REST response. E.g., two possible interactions for a basic GET are shown in [Figure 2 \(Two basic GET transactions, one successful, one not found\)](#).

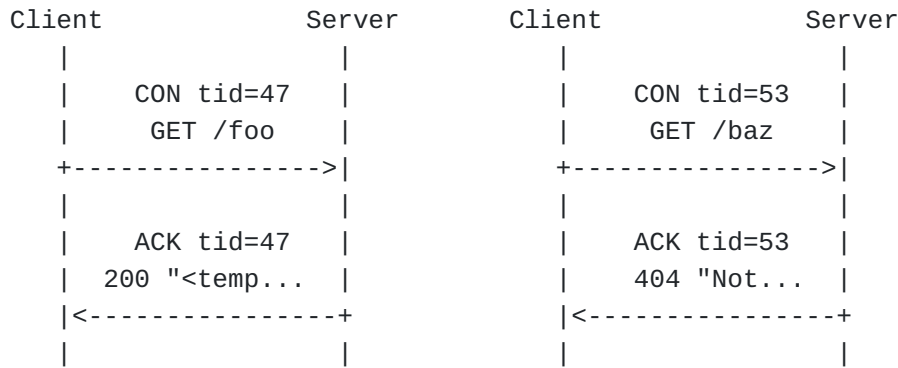


Figure 2: Two basic GET transactions, one successful, one not found

Note that at the transaction layer, the response is returned in an ACK message, independent of whether the request was successful at the REST layer. In effect, the REST response is piggy-backed on the ACK message, so no separate acknowledgment is required that the GET message was received.

The relationship between the confirmable message (CON) and the acknowledgment message (ACK) is indicated by the transaction ID, which is echoed back by the server in the ACK. Transaction IDs are short-lived, they only serve to couple CON and ACK messages.

The tight coupling between CON and ACK also relieves the ACK of the need to echo back information from the request, such as the URI or a request token supplied by the client. We say that a REST response carried in an ACK *pertains* to the REST request in the corresponding CON.

2.1.2. Asynchronous response

[TOC](#)

Not all interactions are as simple as the basic synchronous exchange shown. For example, a server might need longer to obtain the representation of the resource requested than it can wait sending back the acknowledgment, without risking the client to repeatedly retransmit the request. To handle this case, the REST response is decoupled from the transaction layer acknowledgment. Actually, the latter does not carry any REST message at all.

As the client cannot know that this will be the case, it sends exactly the same confirmable message with the same REST request. The server maybe attempts to obtain the resource (e.g., by acting as a proxy) and times out an ACK timer, or it immediately sends an acknowledgment knowing in advance that there will be no quick answer. The acknowledgment effectively is a promise that the REST request will be acted upon, see [Figure 3 \(An asynchronous GET transaction\)](#).

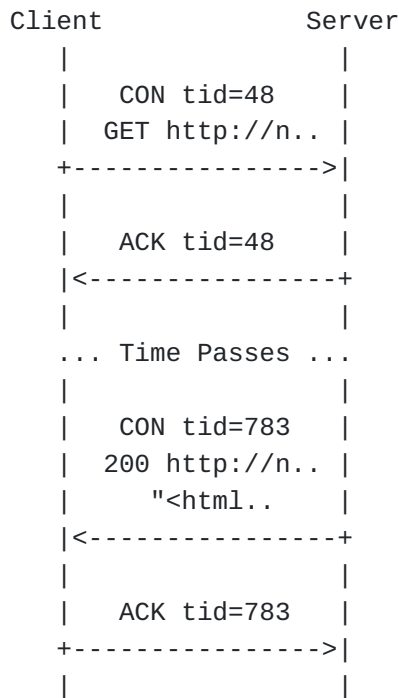


Figure 3: An asynchronous GET transaction

When the server finally has obtained the resource representation and is ready to send the response, it initiates a transaction to the client. This new transaction has its own transaction ID, so there is no automatic coupling of the response to the request. Instead, the URI (and possibly token) is echoed back to the client in order to associate the REST response to the original REST request. To ensure that this message is not lost, it is again sent as a confirmable message and answered by the client with an ACK, citing the new TID chosen by the server.

As a special failure situation, a client may no longer be aware that it sent a request, e.g., if it does not have stable storage and was rebooted in the meantime. This can be indicated by a special "Reset" message, as shown in [Figure 4 \(An orphaned transaction\)](#).

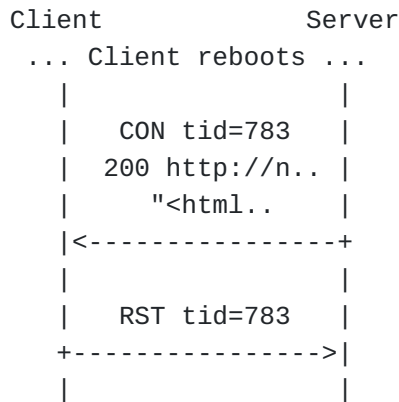


Figure 4: An orphaned transaction

2.2. Transaction messages

[TOC](#)

The CoAP transactions make use of four different message types, described in this section. These messages are transparent to the REST request/response carried over them.

2.2.1. Confirmable (CON)

[TOC](#)

Some messages require an acknowledgment, either just to know they did arrive or also to deliver the reply to a request. We call these messages "Confirmable". When no packets are lost, each Confirmable message elicits exactly one return message of type Acknowledgment or type Reset.

2.2.2. Non-Confirmable (NON)

[TOC](#)

Some other messages do not require an acknowledgment. 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.

[TOC](#)

2.2.3. Acknowledgment (ACK)

An Acknowledgment message acknowledges that a specific Confirmable message (identified by its Transaction ID) arrived. As with all of the message types itself, it may carry a payload and some options to provide more details, such as the result of a request that was carried in the Confirmable.

2.2.4. Reset (RST)

[TOC](#)

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.

2.2.5. Transaction IDs

[TOC](#)

The Transaction ID is an unsigned integer kept by a CoAP end-point for all of the CoAP Confirmable or Non-Confirmable messages it sends. Each CoAP end-point keeps a single Transaction ID variable, which is changed each time a new Confirmable or Non-Confirmable message is sent regardless of the destination address or port. The Transaction ID is used to match an Acknowledgment with an outstanding request, for retransmission and to discard duplicate messages. The initial Transaction ID should be randomized. The same Transaction ID MUST NOT be re-used within the potential retransmission window, calculated as $\text{RESPONSE_TIMEOUT} * (2 ^ \text{MAX_RETRANSMIT} - 1)$.

2.3. Methods

[TOC](#)

CoAP supports the basic RESTful methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP. In this section each method is defined along with its behavior. A unicast request with an unknown or unsupported Method Code MUST generate a message with a "405 Method Not Allowed" Response Code.

As CoAP methods manipulate resources, they have the same properties of safe (only retrieval) and idempotent (you can invoke it multiple times with the same effects) as HTTP [Section 9.1 \(Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," June 1999.\)](#) [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.

2.3.1. GET

[TOC](#)

The GET method retrieves the information of the resource identified by the request URI. Upon success a 200 (OK) response SHOULD be sent. The response to a GET is cacheable if it meets the requirements in [Section 5 \(Caching\)](#).

2.3.2. POST

[TOC](#)

The POST method is used to request the server to create a new resource under the requested URI. If a resource has been created on the server, the response SHOULD be 201 (Created) including the URI of the new resource in a Location Option with any possible status in the message body. If the POST succeeds but does not result in a new resource being created on the server, a 200 (OK) response code SHOULD be returned. Responses to this method are not cacheable.

2.3.3. PUT

[TOC](#)

The PUT method requests that the resource identified by the request URI be updated with the enclosed message body. If a resource exists at that URI the message body SHOULD be considered a modified version of that resource, and a 200 (OK) response SHOULD be returned. If no resource exists then the server MAY create a new resource with that URI, resulting in a 201 (Created) response. If the resource could not be created or modified, then an appropriate error response code SHOULD be sent. Responses to this method are not cacheable.

2.3.4. DELETE

[TOC](#)

The DELETE method requests that the resource identified by the request URI be deleted. The response 200 (OK) SHOULD be sent on success. Responses to this method are not cacheable.

2.4. Response Codes

[TOC](#)

CoAP makes use of a subset of HTTP response codes as defined in [Section 11.1 \(Codes\)](#).

2.5. Options

[TOC](#)

CoAP makes use of compact, extensible Type-Length-Value (TLV) style options. This section explains the processing of CoAP options along with a summary of the main features implemented in options such as URIs and Content-types.

2.5.1. Option Processing

[TOC](#)

If no options are to be included, the Option Count field is set to 0 below and the Payload (if any) immediately follows the Transaction ID. If options are to be included, the following rules apply. The number of options is placed in the Option Count field. Each option is then placed in order of Type, immediately following the Transaction ID with no padding. Upon reception, unknown options of class "elective" MUST be silently skipped. Unknown options of class "critical" in a Confirmable SHOULD cause the return of a response code "400 Bad Request" (TBD) including a copy of the critical option number.

2.5.2. URIs

[TOC](#)

The Universal Resource Identifier (URI) [\[RFC3986\] \(Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier \(URI\): Generic Syntax," January 2005.\)](#) is an important feature of the REST architecture, where the relative part of the URI indicates the resource being manipulated. CoAP supports URIs similarly to HTTP, e.g. `coap://[2001:DB8::101]/s/temp`. As this URI is used purely as a locator, CoAP only supports Universal Resource Locator features of [\[RFC3986\] \(Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier \(URI\): Generic Syntax," January 2005.\)](#) although throughout the document we refer to URI.

CoAP splits the URI up into its three parts with the Uri-Scheme, Uri-Authority and Uri-Path Options. The full URI can be created by concatenating those options (or their defaults if not present). Uri-Scheme and Uri-Authority options are used for proxying, whereas Uri-

Path is used for all requests. CoAP does not support "." or ".." in URIs nor does it support IRIs. A CoAP implementation SHOULD support query "?" processing (see [Section 6 \(Resource Discovery\)](#) on the use of the query string in discovery), however fragment "#" processing is not supported. All URI strings in CoAP MUST use the US-ASCII encoding defined in [\[RFC3986\] \(Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier \(URI\): Generic Syntax," January 2005.\)](#). When using the Uri-Path Option the leading slash MUST be omitted. Thus the above example "/s/temp" is included in the Uri-Path Option as "s/temp".

Application designers are encouraged to make use of short, but descriptive URIs. For example URIs 14 or less bytes in length fit in a more compact option header. In addition, very short URIs such as "/1" can be assigned as an alternative short URI for a resource by the application. CoAP resource descriptions includes an attribute to indicate if a short alternative URI of a resource is available (see [Section 6 \(Resource Discovery\)](#)).

The CoAP protocol scheme is identified in URIs with "coap://" [IANA_TBD_SCHEME].

2.5.3. Content-type encoding

[TOC](#)

In order to support heterogeneous uses, CoAP is transparent to the use of different application payloads. In order for the application process receiving a packet to properly parse a payload, its content-type should be explicitly known from the header (as e.g. with HTTP). The use of typical binary encodings for XML is discussed in

[\[I-D.shelby-6lowapp-encoding\] \(Shelby, Z., Luimula, M., and D. Peintner, "Efficient XML Encoding and 6LowApp," October 2009.\)](#).

String names of Internet media types (MIME types) [\[RFC2046\] \(Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions \(MIME\) Part Two: Media Types," November 1996.\)](#) are not optimal for use in the CoAP header. Instead, CoAP simply assigns identifiers to a subset of common media and content transfer encoding types. The content-type identifier is optionally included in the Content-type Option Header of messages to indicate the type of the message body. CoAP Content-type identifiers are defined in [Section 11.2 \(Content Types\)](#). In the absence of the Content-type Option the MIME type "text/plain" MUST BE assumed.

3. Message Formats

[TOC](#)

CoAP makes use of asynchronous transactions using a simple binary header format. This base header may be followed by options in Type-

Length-Value (TLV) format. CoAP is bound to UDP as described in [Section 4 \(UDP Binding\)](#).

Any bytes after the headers in the packet are considered the message payload, if any. The length of the message payload is implied by the datagram length. See [Section 4 \(UDP Binding\)](#) for further message length requirements.

3.1. CoAP header

[TOC](#)

This section defines the CoAP header, which is shared for all CoAP messages. CoAP makes use of an asynchronous transaction model. These transactions are used to carry RESTful exchanges, either using a Method Code (GET/PUT/POST/DELETE) to invoke interaction with a resource, or a Response Code carried in an immediate or asynchronous response.

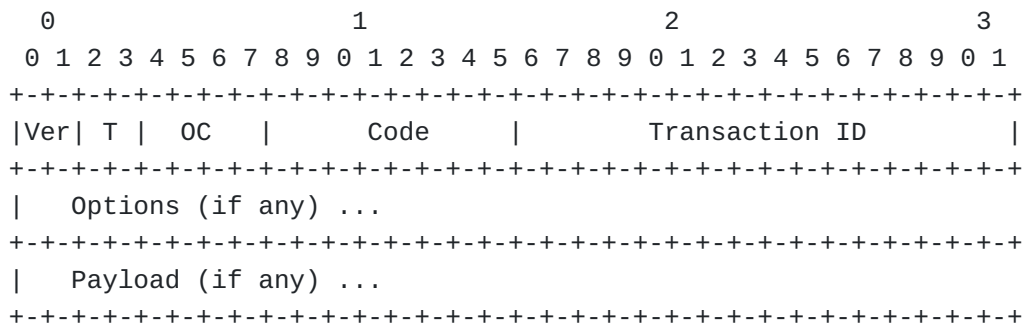


Figure 5: CoAP header format

Header Fields:

Ver: Version. 2-bit unsigned integer. Indicates the version of CoAP. Implementations of this specification MUST set

this field to 1. Other values are reserved for future versions.

T: 2-bit unsigned integer Transaction Type field. Indicates if this message is Confirmable (0), Non-Confirmable (1), Acknowledgment (2) or Reset (3).

OC: 4-bit unsigned integer Option Count field. Indicates if there are Option Headers following the base header. If set to 0 the payload (if any) immediately follows the base header. If greater than zero the field indicates the number of options to immediately follow the header.

Code: 8-bit unsigned integer. This field indicates the Method or Response Code of a message. The value 0 indicates no code. The values 1-10 are used for Method Codes as defined in [Table 1 \(Method Codes\)](#). The values 11-39 are reserved for future use. The values 40-255 are used for Response Codes as defined in [Section 11.1 \(Codes\)](#).

Transaction ID: 16-bit unsigned integer. A unique Transaction ID assigned by the source and used to match responses. The Transaction ID MUST be changed for each new request (regardless of the end-point) and MUST NOT be changed when retransmitting a request (see [Section 2.2.5 \(Transaction IDs\)](#)).

| Method | Code |
|--------|------|
| GET | 1 |
| POST | 2 |
| PUT | 3 |
| DELETE | 4 |

Table 1: Method Codes

3.2. Header options

[TOC](#)

CoAP messages may also include one or more header options in TLV format. Options MUST appear in order of option type (see [Table 2](#)

([Option headers](#)). A delta encoding is used between each option header, with the Type identifier for each Option calculated as the sum of its Option Delta field and the Type identifier of the preceding Option in the message, if any, or zero otherwise.

Each option header includes a Length field which can be extended by an octet for options with values longer than 14 octets. CoAP options include the concept of Critical (odd value) and Elective (even value) options (see [Section 2.5.1 \(Option Processing\)](#)).

Each option has the following format:

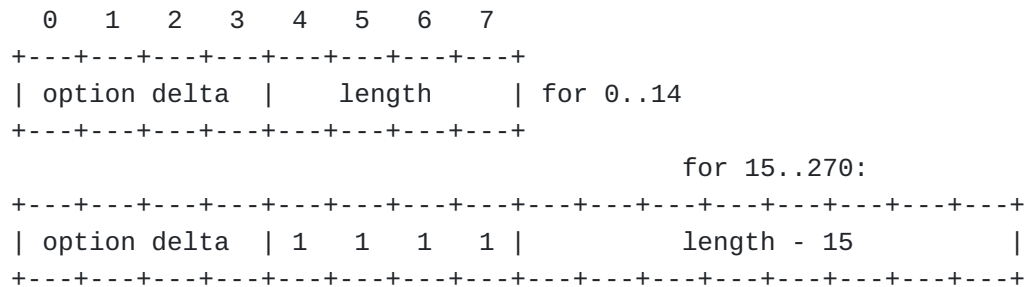


Figure 6: Header option format

Option delta: 4-bit unsigned integer. This field defines the difference between the option Type of this option and the previous option (or zero for the first option). In other words, the Type identifier is calculated by simply summing the Option delta fields of this and previous options before it. The Option Values 14, 28, ... are reserved for no-op options with no value (they are ignored) and are used for deltas larger than 14. Thus these can be used as "fenceposts" if deltas larger than 15 would otherwise be required.

Length: Length Field. Normally Length is a 4-bit unsigned integer allowing values of 0-14 octets. When the length is 15 or more, another byte is added as an 8-bit unsigned integer plus 15 allowing values of 15-270 octets.

Option Value The value in the format defined for that option in [Table 2 \(Option headers\)](#) of Length octets. Options MAY use variable length values.

The following options are defined in this document. The Default column indicates the value to be assumed in the absence of this option (if any).

| Type | C/ E | Name | Data type | Length | Default |
|------|---------|---------------|----------------------------------|---------|--------------------|
| 0 | - | Reserved | - | - | - |
| 1 | C | Content-type | 8-bit unsigned integer | 1 B | 0 (text/ plain) |
| 2 | E | Max-age | Variable length unsigned integer | 1-4 B | 60 seconds |
| 3 | C | Uri-Scheme | String | 1-270 B | "coap" |
| 4 | E | Etag | Sequence of bytes | 1-4 B | - |
| 5 | C | Uri-Authority | String | 1-270 B | "" |
| 6 | E | Location | String | 1-270 B | - |
| 7 | - | Reserved | - | - | - |
| 9 | C | Uri-Path | String | 1-270 B | "" |

Table 2: Option headers

3.2.1. Content-type Option

[TOC](#)

The Content-type Identifier Option indicates the Internet media type identifier of the message-body, see [Section 11.2 \(Content Types\)](#) for the encoding and identifier tables. A Content-type Identifier Option SHOULD be included if there is a payload included with a CoAP message. In the absence of the Content-type Option the MIME type "text/plain" (0) MUST be assumed. This option MUST be supported by all end-points.

3.2.2. Uri-Scheme Option

[TOC](#)

The Uri-Scheme Option indicates the schema part of the URI without any ":" or "://" glue. This option is most often used to access a resource via a proxy. For example, to access an HTTP resource via a proxy the option value would be "http". In the absence of this option, the URI

scheme is assumed to be "coap". [Section 2.5.2 \(URIs\)](#) specifies the rules for URIs in CoAP. This option MUST be supported by an end-point implementing proxy functionality.

3.2.3. Uri-Authority Option

[TOC](#)

The Uri-Authority Option indicates the authority (host + port) part of a URI. Examples of this option include "[2001:DB8::101]", "198.51.100.0:8000" and "sensor.example.com". This option is used when accessing a resource via a proxy. In the absence of this option, the authority is assumed to be "". [Section 2.5.2 \(URIs\)](#) specifies the rules for URIs in CoAP. This option MUST be supported by an end-point implementing proxy functionality.

3.2.4. Uri-Path Option

[TOC](#)

The Uri-Path Option indicates the absolute path part of a URI. One example of an absolute path in this option is "s/light". In the absence of this option, the path is assumed to be "/". [Section 2.5.2 \(URIs\)](#) specifies the rules for URIs in CoAP. The leading slash is assumed and MUST be omitted. This option MUST be supported by all end-points.

3.2.5. Location Option

[TOC](#)

The Location Option indicates the location of a resource as an absolute path URI and is similar to the Uri-Path Option. The Location Option MAY be included in a response to indicate the Location of a new resource created with POST or together with a 30x response code. The leading slash is assumed and MUST be omitted.

3.2.6. Max-age Option

[TOC](#)

The Max-age Option indicates the maximum age of the resource for use in cache control in seconds. The option value is represented as a variable length unsigned integer between 8 and 32 bits. A default value of 60 seconds is assumed in the absence of this option.

When included in a request, Max-age indicates the maximum age of a cached representation of that resource the client will accept. When

included in a response, Max-age indicates the maximum time the representation may be cached before it MUST be discarded.

3.2.7. Etag Option

[TOC](#)

The Etag Option is an opaque sequence of bytes which specifies the version of a resource representation. An Etag may be generated for a resource in any number of ways including a version, checksum, hash or time. An end-point receiving an Etag MUST treat it as opaque and make no assumptions about its format. The Etag MAY be included in a response to indicate to a client if a resource has changed. The Etag SHOULD be included in a request used for a cache refresh to indicate the client's current version of the resource (see [Section 5.2 \(Cache refresh\)](#)).

4. UDP Binding

[TOC](#)

The CoAP protocol operates by default over UDP. CoAP could also be used over other transports such as TCP or SCTP, the specification of which is out of this document's scope.

The goal of binding CoAP to UDP is to provide the bare minimum features for the protocol to operate over UDP, without trying to re-create the full feature set of a transport like TCP. CoAP over UDP has the following features:

- *Simple stop-and-wait retransmission reliability with exponential back-off as described in [Section 4.2 \(Retransmission\)](#) for Confirmable messages.

- *Transaction ID for response matching as described in [Section 2.2.5 \(Transaction IDs\)](#).

- *Multicast support as described in [Section 4.1 \(Multicast\)](#).

The length of the Payload in a CoAP message is calculated from the datagram length. While specific link layers make it beneficial to keep CoAP messages small enough to fit into their link layer packets (see [Section 1 \(Introduction\)](#)), this is a matter of implementation quality. The CoAP specification itself provides only an upper bound to the message size. A CoAP message SHOULD fit within a single IP packet and MUST fit within a single IP datagram. If the Path MTU is not known for a destination, an MTU of 1280 octets SHOULD be assumed.

4.1. Multicast

[TOC](#)

CoAP supports the use of multicast destination addresses. Multicast messages SHOULD be Non-Confirmable. If a Confirmable multicast message is sent then retransmission MUST NOT be performed. Furthermore, a destination end-point to a multicast Confirmable message MUST only send an Acknowledgment if the response code included indicates success (Code = 2XX) in order to eliminate error code response floods. Other mechanisms for avoiding congestion from multicast requests are being considered in [\[I-D.eggert-core-congestion-control\]](#) (Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)," June 2010.).

4.2. Retransmission

[TOC](#)

A CoAP end-point keeps track of open Confirmable messages it sent that are waiting for a response. Each entry includes at least the destination IP address and port of the original message, the message itself, a retransmission counter and a timeout. When a Confirmable is sent, an entry is made for that message with a default initial timeout of RESPONSE_TIMEOUT and the retransmission counter set to 0. When a matching Acknowledgment is received for an entry, the entry is invalidated. When a timeout is triggered for an entry and the retransmission counter is less than MAX_RETRANSMIT, the original message is retransmitted to the destination without modification, the retransmission counter is incremented, and the timeout is doubled. If the retransmission counter reaches MAX_RETRANSMIT on a timeout, then the entry is removed and the application process informed of delivery failure.

For CoAP messages sent to IP multicast addresses, retransmission MUST NOT be performed. Therefore MAX_RETRANSMIT is always set to 0 when the destination address is multicast.

4.3. Congestion Control

[TOC](#)

In addition to the exponential back-off mechanism in [Section 4.2 \(Retransmission\)](#), further congestion control optimizations are being considered and tested for CoAP. These congestion control mechanism under consideration are described in [\[I-D.eggert-core-congestion-control\]](#) (Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)," June 2010.).

4.4. Default Port

[TOC](#)

The CoAP default port number [IANA_TBD_PORT] MUST be supported by a server for resource discovery (see [Section 6 \(Resource Discovery\)](#)) 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 support a port in the 61616-61631 compressed UDP port space defined in [\[RFC4944 \(Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," September 2007.\)\]](#). The specific port number in use will be communicated in a URI and/or by some other discovery mechanism.

5. Caching

[TOC](#)

CoAP end-points are by definition constrained by bandwidth and processing power. To optimize the performance of data transfer under these constraints, we use caching features consistent with HTTP. Caching includes the following concepts:

- *Cache life of a resource is controlled via the Max-Age header option
 - *Cache refresh and versioning of a resource is controlled via the Etag header option
 - *Proxies between a client and end-point may participate in the caching process on behalf of sleeping end-points and to avoid unnecessary traffic on the constrained network
-

5.1. Cache control

[TOC](#)

When an end-point responds to a GET request by sending a representation of the resource, it SHOULD specify the Max-Age header option. The Max-Age specifies the cache life of the resource in seconds. Resources which change rapidly will have a short cache life, and resources which change infrequently should specify a long cache life. If Max-Age is unspecified in a GET response, then it is assumed to be 60 seconds. If an end-point wishes to disable caching, it must explicitly specify a Max-Age of zero seconds.

When a client reads the response from a GET request, it should cache the resource representation for the cache lifetime as specified by the

Max-Age header. During the cache lifetime, the client SHOULD use its cached version and avoid performing additional GETs for the resource. In general, the origin server end-point is responsible for determining cache age. However, in some cases a client may wish to determine its own tolerance for cache staleness. In this case, a client may specify the Max-Age header during a GET request. If the client's Max-Age is of a shorter duration than the age of a cached resource, then the proxy or end-point SHOULD perform a cache refresh. If the client specifies a Max-Age of zero seconds, then the response MUST discard the cached representation and return a fresh representation.

5.2. Cache refresh

[TOC](#)

After the expiration of the cache lifetime, clients and proxies can refresh their cached representation of a resource. Cache refresh is accomplished using a GET request which will return a representation of the resource's current state.

If the end-point has the capability to version the resource, then the end-point should include the Etag header option in the response to a GET request. The Etag is a variable length sequence of bytes which captures a version identifier of the resource. The Etag is an opaque identifier; clients MUST NOT infer any semantics from the Etag value. If an end-point specifies the Etag header option with a response, then the client SHOULD specify a matching Etag header option in their GET request during cache refresh. If the end-point's version of the resource is unmodified, then the server SHOULD return a 304 response with no payload to avoid retransmitting the resource representation.

5.3. Proxying

[TOC](#)

A proxy is defined as a CoAP end-point which services cached requests on behalf of other CoAP end-points. Any node in a CoAP network may act as a proxy, although in general the node between the constrained network and the Internet at large SHOULD always support proxy functionality.

Proxies should be used under the following scenarios:

- *Clients external to the constrained network SHOULD always make requests through a proxy to limit traffic on the constrained network
- *Clients internal to the constrained network MAY use a proxy based on network topology when performance warrants

*Clients of sleeping devices MUST use a proxy to access resources while the device is sleeping

Proxy requests are made as normal CON requests to the proxy end-point. All proxy requests MUST use the Uri-Authority header to indicate the origin server's IP address using the URI format defined by RFC 3986:

```
full uri  = "coap://" + authority + path
authority = host [ ":" port ]
host      = IP-literal / IPv4address / reg-name
           (as defined by RFC 3986)
port      = *DIGIT
```

The host part is case insensitive and may be an IPv4 literal, IPv6 literal in square brackets, or a registered name. The port number is optional, if omitted or zero-length it is assumed to be the default CoAP port (see [Section 4.4 \(Default Port\)](#)).

If a request is made using the Uri-Authority header, then the following steps are taken:

1. If the authority (host and port) is the same as the proxy end-point, then the request MUST be treated as a local request and the path part is used as Uri-Path
2. If the end-point does not support proxy functionality for the given URI or Method code, then the 400 Bad Request (TBD) error code MUST be returned
3. If the proxy does not contain a fresh cached representation of the resource, then the proxy MUST attempt to refresh its cache according to section 5.2. The origin server's IP address and port is determined by the authority part of the full URI. The Uri-Path option for the refresh request is determined by the path part of the full URI.
4. If the proxy fails to obtain a fresh cached representation, then a 502 Bad Gateway error code MUST be returned
5. The proxy returns the cached representation on behalf of the origin server

All CoAP options are considered end-to-end and MUST be stored as part of the cache entry and MUST be transmitted in the proxy's response. The Max-Age option should be adjusted by the proxy for each response using the formula: $\text{proxy-max-age} = \text{original-max-age} - \text{cache-age}$. For example if a request is made to a proxied resource that was refreshed 20sec ago and had an original Max-Age of 60sec, then that resource's proxied Max-Age is now 40sec.

6. Resource Discovery

[TOC](#)

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. The discovery of resources provided by an HTTP Web Server is typically called Web Discovery. In this document we refer to the discovery of resources offered by a CoAP end-point as resource discovery. CoAP resource discovery enables both unicast and multicast discovery of resources including filtering on attributes of resource descriptions. CoAP resource discovery can also be used to discovery HTTP resources. CoAP makes the assumption that all CoAP servers host an end-point on the default CoAP port (see [Section 4.4 \(Default Port\)](#)) hosting a well-known discovery resource, or otherwise have been configured or discovered using some general service discovery mechanism such as [\[I-D.cheshire-dnsext-dns-sd\] \(Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery," March 2010.\)](#). This section assumes that such a configuration or service discovery has already been performed a priori, if needed.

Resource discovery in CoAP is accomplished through the use of well-known resources which describe the links (resource descriptions) offered by that CoAP server. Well-known resources use the URI form `"/.well-known/"` as specified in [\[RFC5785\] \(Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers \(URIs\)," April 2010.\)](#). CoAP discovery defines a new well-known URI for discovery `"/.well-known/r"` [IANA_TBD_WELLKNOWN]. A CoAP end-point MUST support this URI for the purpose of resource discovery. It is however up to the application which resources descriptions are included and how they are organized. The resource representation of this resource is described in [Section 6.1 \(Link Format\)](#).

CoAP Resource Discovery supports the following interactions:

- *[GET `/.well-known/r`] returns a list of links available from a CoAP server (if any). This request may be unicast or multicast.
- *Filtering may be performed on any of the resource description attributes using a query string as specified in [Section 6.3 \(Query Filtering\)](#). For example [GET `/.well-known/r?n=Temperature`] would return resources with the name Temperature.
- *More capable end-points such as proxies could support a resource directory by requesting the resource descriptions of other end-points or accepting [POST `/.well-known/r`] messages from other CoAP end-points. This adds the resources of other end-points to a resource directory in which absolute URIs are included for the

links. The details of such an entity are however out of scope for this document.

*CoAP may also be used to discovery HTTP resources or an HTTP server may use the well-known resource and link-format as discussed in [Section 6.4 \(HTTP Resource Discovery\)](#).

End-points with a large number of resources SHOULD include resource descriptions only for important collections and organize their resource descriptions into a hierarchy of link resources. This is done by including links in the /.well-known/r list which point to other resource lists, e.g. "</.well-known/r/s>;n=Sensors;ct=40". Such a hierarchy SHOULD be under the /.well-known/r path.

6.1. Link Format

[TOC](#)

CoAP resource discovery makes use of the HTTP Link Header format specified in [\[I-D.nottingham-http-link-header\] \(Nottingham, M., "Web Linking," April 2010.\)](#) which is in Augmented Backus-Naur Form (ABNF) notation [\[RFC5234\] \(Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," January 2008.\)](#). This specification allows for the use of this simple link format by other protocols, thus not limiting it to the actual HTTP Link Header. The format does not require special XML or binary parsing, and is extensible. The format is also compatible with the naming and descriptions used in [\[I-D.cheshire-dnsext-dns-sd\] \(Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery," March 2010.\)](#).

[\[I-D.nottingham-http-link-header\] \(Nottingham, M., "Web Linking," April 2010.\)](#) did not request a MIME type for this link format, as it assumes that would usually be carried in an HTTP header. This specification thus requests a MIME type for this header as in [Section 11.2 \(Content Types\) \[IANA_TBD_LINK\]](#).

The CoAP link-format defines default attributes for CoAP resource discovery. All link parameters are optional and custom link-extensions MAY be defined. Multiple descriptions are separated by commas as in [\[I-D.nottingham-http-link-header\] \(Nottingham, M., "Web Linking," April 2010.\)](#). The following ABNF specification defines the format:


```

link-value      = "<" URI-Reference ">" *( ";" link-param )
link-param      = ( ( "d" "=" URI )
                    | ( "sh" "=" URI )
                    | ( "n" "=" string )
                    | ( "ct" "=" media-code *( "," media-code ) )
                    | ( "id" "=" integer )
                    | ( link-extension ) )
link-extension  = ( parmname [ "=" ( ptoken | string ) ] )
ptoken          = 1*ptokenchar
ptokenchar      = "!" | "#" | "$" | "%" | "&" | "'" | "("
                  | ")" | "*" | "+" | "-" | "." | "/" | DIGIT
                  | ":" | "<" | "=" | ">" | "?" | "@" | ALPHA
                  | "[" | "]" | "^" | "_" | "`" | "{" | "|"
                  | "}" | "~"
media-code       = Identifier for media type in decimal ASCII,
                  see Section 11.2
URI-reference    = Imported from [RFC3986]
URI              = Imported from [RFC3986]

```

link-value: The link-value is the relative URI of the resource on that end-point or an absolute full URI in the case of e.g. a directory agent or HTTP resource. The key "uri=" is used to match this field in a query string. Protocol identifier, host and port parts of the URI MAY be included if they differ from the default CoAP values.

d: Description. A URI that points to the definition of that resource interface, for example in WADL.

sh: Short URI. Alternative short URI which can also be used to access this resource.

n: Instance Name. See [Section 6.2 \(Naming\)](#).

ct: Content-type. The Internet media type this resource returns in the CoAP identifier code format as a decimal ASCII integer as per [Section 11.2 \(Content Types\)](#). For example application/xml would be indicated as "ct=41". If a resource supports multiple content-types they can be separated by a comma in this attribute. If no Content-type attribute is present then text/plain is assumed.

id: Unique Identifier. The id field is a unique identifier (e.g. UUID) for this resource for use in e.g. search directories.

An example of three typical CoAP link descriptions in this format follows. Multiple resource descriptions in a representation are

separated by commas. Linefeeds are included in the example just for readability.

```
</>;n=Info,  
</sensor/temp>;sh=/t;ct=0,41;n=Temperature,  
</sensor/light>;sh=/l;ct=41;n=Light,
```

6.2. Naming

[TOC](#)

CoAP resource naming is designed to be compatible with DNS-based Service Discovery naming. The `n=` attribute carries what can be considered the "Instance" part of a DNS-SD name. The "Service" and "Domain" parts of DNS naming are implied as Service is assumed to be `_coap._udp` and Domain is assumed to be `.local`. The Service part MAY be included as `_http._tcp` in the case of an HTTP resource. The Domain MAY be included if different than `.local`.

Whereas service discovery is used to find the IP address, port and protocol of an unknown service, resource discovery is a fine-grained discovery of resource URIs within a web service. CoAP resource descriptions can be imported into DNS-SD for exposure to service discovery by using the `n=` attribute as the "Instance", `_coap_udp` as the Service and a default of `.local` as the Domain. The DNS TXT record can be generated by importing the resource description attributes as they share the same format as Section 6 of [\[I-D.cheshire-dnsext-dns-sd\] \(Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery," March 2010.\)](#).

6.3. Query Filtering

[TOC](#)

A CoAP end-point SHOULD support the query string `/.well-known/r?` with `uri=` or any of the resource description attributes for the purpose of filtering a discovery. Wildcard `*` endings SHOULD be supported. An exact match is performed on the query string, and a 200 OK response is returned with a link-style structure that contains the matching entries (if any). If resource descriptions are organized hierarchically, a query on the root resource `/.well-known/r` SHOULD return all matching resource descriptions from the entire hierarchy if possible.

For a unicast query with no matching entries a 404 Not Found code is returned, however no response is sent in case of a multicast query as described in [Section 4.1 \(Multicast\)](#) to prevent a flood of responses.

6.4. HTTP Resource Discovery

[TOC](#)

This mechanism may also be used to discover HTTP resources as HTTP lacks a built-in discovery mechanism and multicast support. When HTTP resources are listed in the link-format HTTP they SHOULD be indicated by including `http://` in the link-value field or the `_http._tcp` Service in the name attribute.

An HTTP server may make use of the `/.well-known/r` URI scheme and link-format representation defined in this section for the purpose of discovery using HTTP. This may especially be useful for a CoAP-HTTP proxy to expose the resources available in a constrained domain to HTTP clients.

7. HTTP Mapping

[TOC](#)

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 REST 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 or SIP, the definition of which is out of scope.

The mapping of CoAP to HTTP is a 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. The mapping of HTTP to CoAP requires checking for methods, response codes, options and content-types that are not supported by CoAP. A mapping 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.

The caching and proxying of CoAP is specified in [Section 5 \(Caching\)](#). In a similar manner, caching and proxying MAY be performed between CoAP and HTTP by an intermediate node. A proxy SHOULD respond with a 502 (Bad Gateway) error to HTTP requests which can not be successfully mapped to CoAP. CoAP transaction messages are transparent to REST and MUST have no affect on a proxy function.

8. Protocol Constants

[TOC](#)

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

| | |
|-------------------------|----------|
| RESPONSE_TIMEOUT | 1 second |
|-------------------------|----------|

9. Examples

[TOC](#)

[Figure 7 \(Basic request/response\)](#) shows a basic request sequence. A client makes a Confirmable GET request for the resource /temperature to the server with a Transaction ID of 1234. The request includes one Uri-Path Option (delta 0 + 9 = 9) "temperature" of Len = 11. This request is a total of 16 octets long. The corresponding Acknowledgment is of Code 200 OK and includes a Payload of "22.3 C". The Transaction ID is 1234, thus the transaction is successfully completed. The response is 10 octets long and a Content-type of 0 (text/plain) is assumed as there is no Content-type Option.

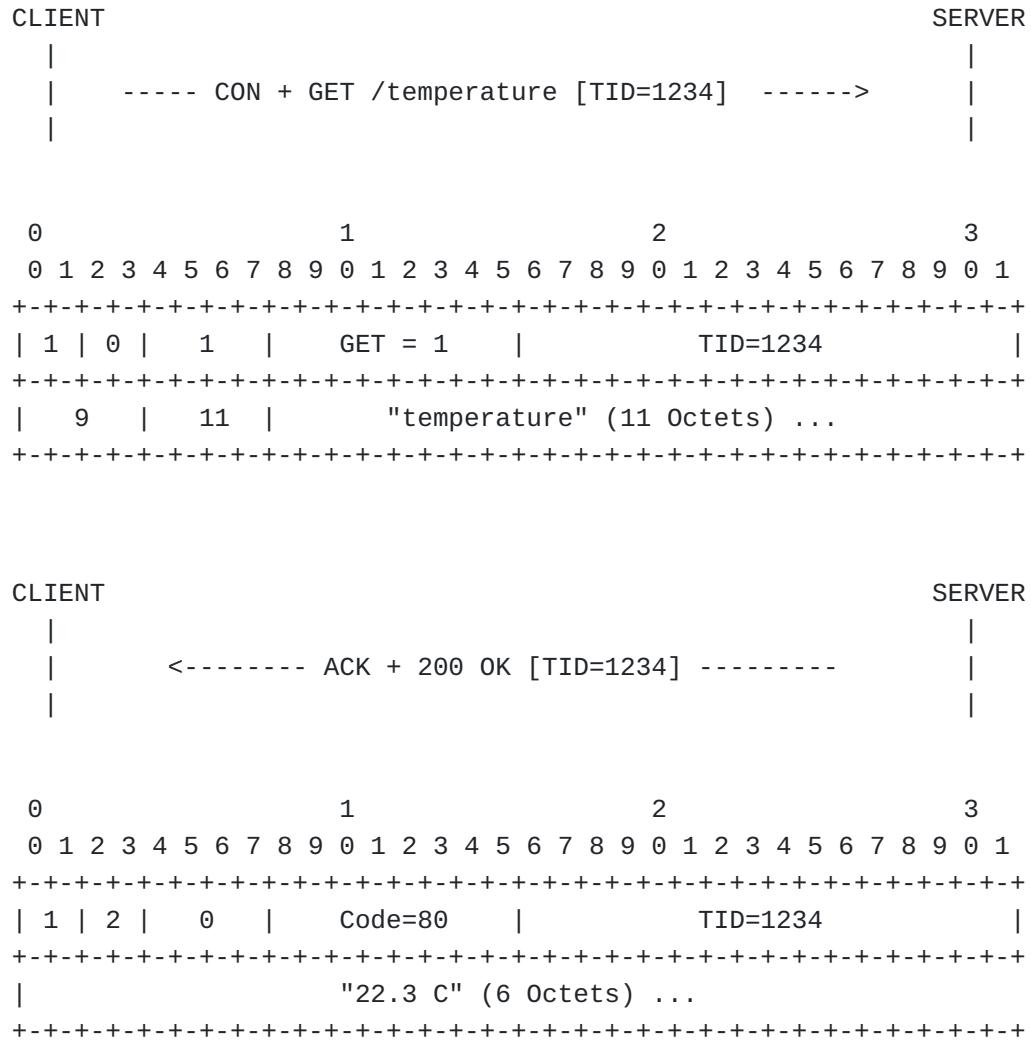


Figure 7: Basic request/response

[Figure 8 \(Basic request/response\)](#) shows an example of a retransmission using the previous request. The first ACK from the server is lost, and after RESPONSE_TIMEOUT seconds the client retransmits the request.

Figure 9: Basic request/response

[Figure 10 \(Basic request/response\)](#) shows an example of multicast resource discovery. Here a client sends a request for `/.well-known/r` with a query for `?n=Light` (Resource name = Light) to all-nodes link-scope multicast. There are 3 servers on the link: A, B and C. Servers A and B have a matching resource, therefore they send back a successful 200 OK response with the matching resource in the payload. C does not attempt to send a response.

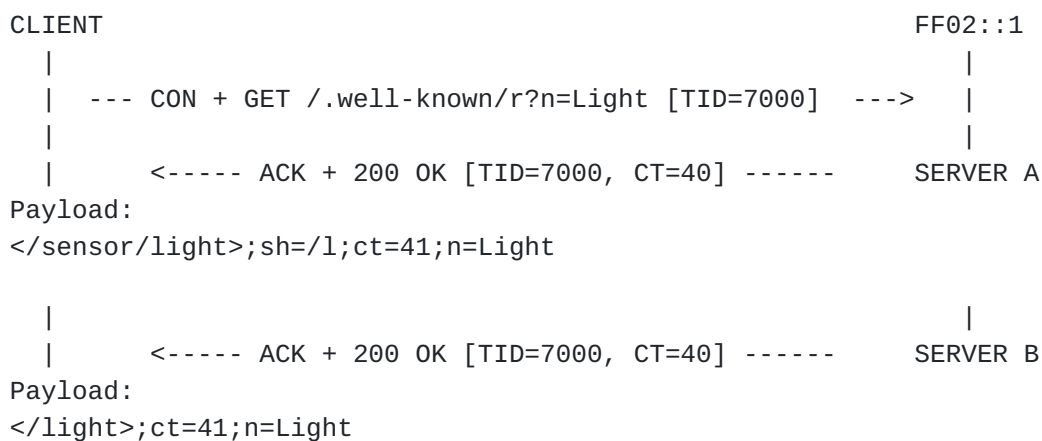


Figure 10: Basic request/response

10. Security Considerations

TOC

TODO: Expand this section to a full security analysis, including how to use CoAP with various security options.

Some of the features considered in this document will need further security considerations during a protocol design. For example the use of string URLs may have entail security risks due to complex processing on limited microcontroller implementations.

The CoAP protocol will be designed for use with e.g. (D)TLS, IPsec or object security. A protocol design should consider how integration with these security methods will be done, how to secure the CoAP header and other implications.

11. IANA Considerations

[TOC](#)

[IANA_TBD_SCHEME] This document suggests the scheme `coap://` to identify this protocol in a URI. The string "coap" should similarly be used in well-known port and service discovery registrations.

[IANA_TBD_PORT] Apply for a well-known port number in the 0-1023 space as CoAP end-points are usually executed by an operating system or root process. <http://www.iana.org/assignments/port-numbers>

[IANA_TBD_MIME] A new registry is required for the Internet MIME type identifier space for CoAP as described in [Section 11.2 \(Content Types\)](#).

[IANA_TBD_LINK] MIME type for link format, `application/link-format` to be requested.

[IANA_TBD_WELLKNOWN] Apply for a well-known URI for the purpose of resource discovery at `/.well-known/r` as per [\[RFC5785\] \(Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers \(URIs\)," April 2010.\)](#).

11.1. Codes

[TOC](#)

CoAP makes use of (a subset of) the HTTP status codes defined in [\[RFC2616\] \(Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," June 1999.\)](#). The HTTP status code is encoded into an 8-bit unsigned integer code with the mapping defined in [Table 3 \(CoAP Codes\)](#). The use of these codes is defined throughout this document using the HTTP Name.

| Code | HTTP Name |
|------|----------------------------|
| 40 | 100 Continue |
| 80 | 200 OK |
| 81 | 201 Created |
| 124 | 304 Not Modified |
| 160 | 400 Bad Request |
| 164 | 404 Not Found |
| 165 | 405 Method Not Allowed |
| 175 | 415 Unsupported Media Type |
| | |
| 200 | 500 Internal Server Error |
| 202 | 502 Bad Gateway |

| |
|----------------------------|
| Table 3: CoAP Codes |
|----------------------------|

11.2. Content Types

| |
|---------------------|
| TOC |
|---------------------|

Internet media types are identified by a string in HTTP, such as "application/xml". This string is made up of a top-level type "application" and a sub-type "xml" [\[RFC2046\] \(Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions \(MIME\) Part Two: Media Types," November 1996.\)](#). In order to minimize the overhead of using these media types to indicate the type of message payload, CoAP defines an identifier encoding scheme for a subset of Internet media types. It is expected that this table of identifiers will be extensible and maintained by IANA for values of 0-200 [IANA_TBD_MIME]. The Content-type Option is formatted as an 8-bit unsigned integer. Initial mappings from Internet media types to a suitable identifier is shown in [Table 4 \(Media type identifiers\)](#). Composite high-level types (multipart and message) are not supported. Identifier values from 201-255 are reserved for vendor specific, application specific or experimental use and are not maintained by IANA.

| Internet media type | Identifier |
|---|------------|
| text/plain (UTF-8) | 0 |
| text/xml (UTF-8) | 1 |
| text/csv (UTF-8) | 2 |
| text/html (UTF-8) | 3 |
| image/gif | 21 |
| image/jpeg | 22 |
| image/png | 23 |
| image/tiff | 24 |
| audio/raw | 25 |
| video/raw | 26 |
| application/link-format [IANA_TBD_LINK] | 40 |
| 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 |
| application/x-bxml | 48 |
| application/fastinfoset | 49 |
| application/soap+fastinfoset | 50 |
| application/json | 51 |

Table 4: Media type identifiers

12. Acknowledgments

[TOC](#)

Special thanks to Carsten Bormann and Klaus Hartke for substantial contributions to the ideas and text in the document ([Section 2.1.1 \(Synchronous response\)](#), [Section 2.1.2 \(Asynchronous response\)](#), [Section 2.2 \(Transaction messages\)](#), [Section 3.2 \(Header options\)](#)), along with countless detailed reviews and discussions.

Thanks to Michael Stuber, Richard Kelsey, Cullen Jennings, Guido Moritz, Peter Van Der Stok, Adriano Pezzuto, Lisa Dusseault, Alexey Melnikov, Gilbert Clark, Salvatore Loreto, Petri Mutka, Szymon Sasin, Robert Quattlebaum, Robert Cragie, Angelo Castellani, Tom Herbst, Ed Beroeset, Gilman Tolle, Robby Simpson, Peter Bigot, Colin O'Flynn and David Ryan for helpful comments and discussions that have shaped the document.

13. Changelog

[TOC](#)

Changes from ietf-00 to ietf-01:

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

- o Date option removed while being designed (#6)
- o New text for CoAP default port (#7)
- o Completed proxying section (#8)
- o Completed resource discovery section (#9)
- o Completed HTTP mapping section (#10)
- o Several new examples added (#11)
- o URI split into 3 options (#12)
- o MIME type defined for link-format (#13, #16)
- o New text on maximum message size (#15)
- o Location Option added

Changes from shelby-01 to ietf-00:

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

Changes from shelby-00 to shelby-01:

- o Unified the message header and added a notify message type.

- o Renamed methods with HTTP names and removed the NOTIFY method.
- o Added a number of options field to the header.
- o Combines the Option Type and Length into an 8-bit field.
- o Added the magic byte header.
- o Added new Etag option.
- o Added new Date option.
- o Added new Subscription option.
- o Completed the HTTP Code - CoAP Code mapping table appendix.
- o Completed the Content-type Identifier appendix and tables.
- o Added more simplifications for URI support.
- o Initial subscription and discovery sections.
- o A Flag requirements simplified.

14. References

[TOC](#)

14.1. Normative References

[TOC](#)

| | |
|-----------------------------------|---|
| [I-D.nottingham-http-link-header] | Nottingham, M., " Web Linking ," draft-nottingham-http-link-header-09 (work in progress), April 2010 (TXT). |
| [RFC2046] | Freed, N. and N. Borenstein , " Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types ," RFC 2046, November 1996 (TXT). |
| [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 (TXT , PS , PDF , HTML , XML). |
| [RFC3986] | Berners-Lee, T. , Fielding, R. , and L. Masinter , " Uniform Resource Identifier (URI): Generic Syntax ," STD 66, RFC 3986, January 2005 (TXT , HTML , XML). |
| [RFC5234] | |

| | |
|-----------|--|
| | Crocker, D. and P. Overell, " Augmented BNF for Syntax Specifications: ABNF ," STD 68, RFC 5234, January 2008 (TXT). |
| [RFC5785] | Nottingham, M. and E. Hammer-Lahav, " Defining Well-Known Uniform Resource Identifiers (URIs) ," RFC 5785, April 2010 (TXT). |

14.2. Informative References

[TOC](#)

| | |
|--------------------------------------|---|
| [I-D.cheshire-dnsext-dns-sd] | Cheshire, S. and M. Krochmal, " DNS-Based Service Discovery ," draft-cheshire-dnsext-dns-sd-06 (work in progress), March 2010 (TXT). |
| [I-D.eggert-core-congestion-control] | Eggert, L., " Congestion Control for the Constrained Application Protocol (CoAP) ," draft-eggert-core-congestion-control-00 (work in progress), June 2010 (TXT). |
| [I-D.shelby-6lowapp-encoding] | Shelby, Z., Luimula, M., and D. Peintner, " Efficient XML Encoding and 6LowApp ," draft-shelby-6lowapp-encoding-00 (work in progress), October 2009 (TXT). |
| [I-D.shelby-core-coap-req] | Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R. Kelsey, " CoAP Requirements and Features ," draft-shelby-core-coap-req-01 (work in progress), April 2010 (TXT). |
| [RFC4944] | Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, " Transmission of IPv6 Packets over IEEE 802.15.4 Networks ," RFC 4944, September 2007 (TXT). |

Authors' Addresses

[TOC](#)

| | |
|--------|--|
| | Zach Shelby |
| | Sensinode |
| | Kidekuja 2 |
| | Vuokatti 88600 |
| | FINLAND |
| Phone: | +358407796297 |
| Email: | zach@sensinode.com |
| | |
| | Brian Frank |
| | SkyFoundry |
| | Richmond, VA |
| | USA |

| | |
|--------|--|
| Phone: | |
| Email: | brian@skyfoundry.com |
| | |
| | Don Sturek |
| | Pacific Gas & Electric |
| | 77 Beale Street |
| | San Francisco, CA |
| | USA |
| Phone: | +1-619-504-3615 |
| Email: | d.sturek@att.net |