CoRE	Z. Shelby	
Internet-Draft	Sensinode	
Intended status: Standards Track	B. Frank	
Expires: March 31, 2011	SkyFoundry	
	D. Sturek	
	Pacific Gas & Electric	
	September 27, 2010	

TOC

Constrained Application Protocol (CoAP)
draft-ietf-core-coap-02

Abstract

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

Status of this Memo

Table of Contents

---

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 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, 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 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 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 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 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 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 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 requests and responses. These transactions are transparent to the request/response 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 request/response interactions using Method and Response codes.

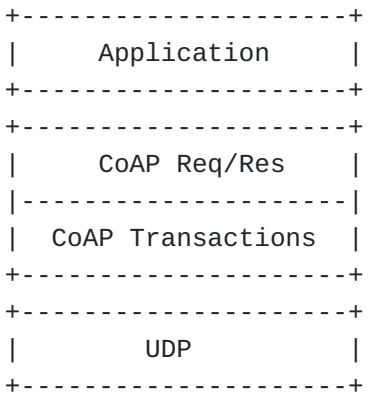

Figure 1: Abstract layering of CoAP

2.1.1. Synchronous response

TOC

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

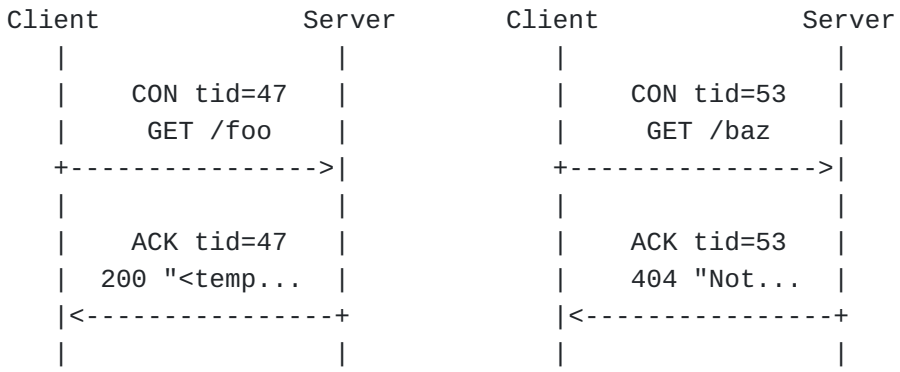

Figure 2: Two basic GET transactions, one successful, one not found

---

Note that at the transaction layer, the response is returned in an ACK message, independent of whether the request was successful at the Req/Res layer. In effect, the 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 response carried in an ACK *pertains* to the 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 response is decoupled from the transaction layer acknowledgment. Actually, the latter does not carry any message at all.

As the client cannot know that this will be the case, it sends exactly the same confirmable message with the same 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 request will be acted upon, see Figure 3 (An asynchronous GET transaction).

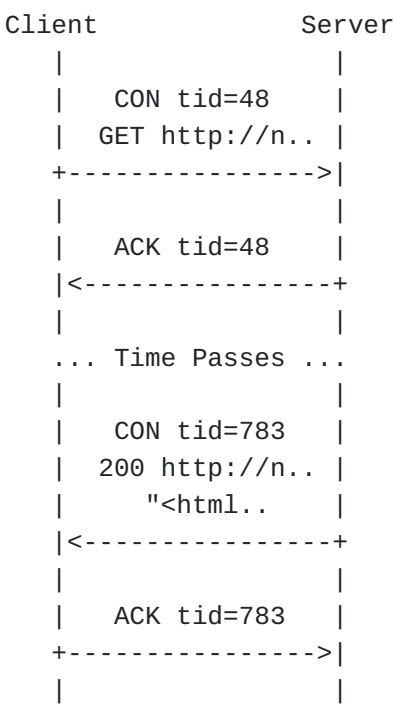

Figure 3: An asynchronous GET transaction

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 response to the original 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).

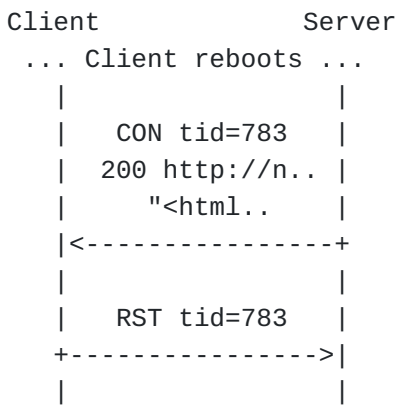

Figure 4: An orphaned transaction

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 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 RESPONSE_TIMEOUT * (2 ^ MAX_RETRANSMIT - 1).

2.3. Methods

TOC

CoAP supports the basic 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. Unlike PUT, POST is not idempotent because the URI in the request indicates the resource that will handle the enclosed body. This resource indicated by the POST may be used for data processing, a gateway to other protocols and it may create a new resource as a result of the POST.

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 or created 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.

TOC

2.3.4. DELETE

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 in the payload of the response.

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 web 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 default coap:// scheme, Uri-Authority and Uri-Path Options. The full URI can be created by concatenating those parts (or their defaults if not present). CoAP does not support "." or ".." in URIs nor does it support IRIs. A CoAP implementation SHOULD support query "?" processing, 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".
The authority part of a URI is important in determining the correct representation to return on end-points maintaining virtual servers and for intermediate components such as proxies. For this reason it is important that the full URI can be reconstructed when needed. However, at the same time, it is often advantageous for CoAP to elide the Uri-Authority when it is unknown or identical to the IPv6 destination address for efficiency. The following rules apply to processing a CoAP request:

1. If the Uri-Authority option is absent and the remainder of the URI uniquely identifies a resource the server MAY proceed to execute the request.

2. If an origin server is able to determine the IP destination address of the request, it MAY assume this as the authority of the URI.

3. If no authority can be determined and the server requires the authority to identify the resource it MUST reject the request with "400 Bad Request" (TBD: 400 is already overloaded, thus a new response code may be created for this purpose).

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. The CoRE Link Format includes an attribute to indicate if a short alternative URI of a resource is available (REF).
The CoAP protocol scheme is identified in URIs with "coap://" [IANA_TBD_SCHEME].

TOC

2.5.3. Content-type encoding

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 request/response 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.

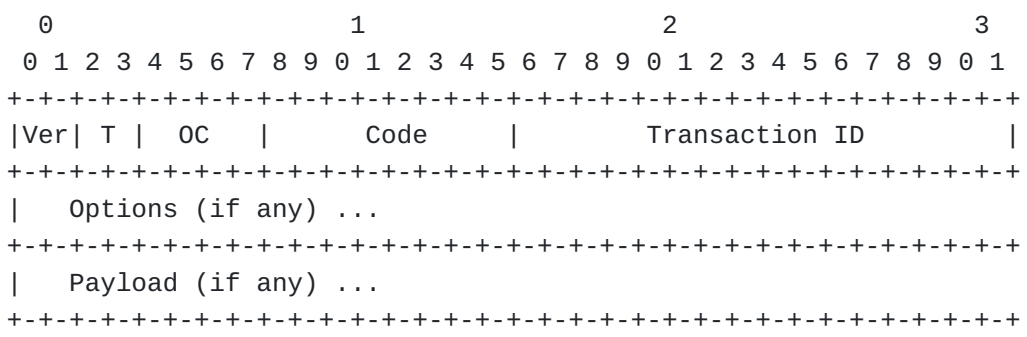

Figure 5: CoAP header format

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:

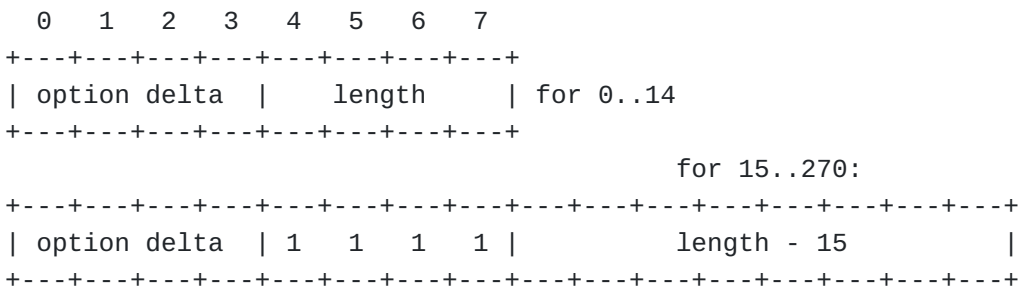

Figure 6: Header option format

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	-	Reserved	-	-
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. This option MUST NOT occur more than once in a header.

3.2.2. 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 by servers to determine which resource to return and by intermediate components, e.g. when accessing a resource via a proxy. Section 2.5.2 (URIs) specifies the rules for URIs in CoAP. This option SHOULD be included in a request when the authority of the URI is known. This option MUST be supported by an end-point implementing proxy functionality. This option MUST NOT occur more than once in a header.

3.2.3. Uri-Path Option

TOC

The Uri-Path Option indicates the absolute path part of a URI. One example of an absolute path in his 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. This option MUST NOT occur more than once in a header.

3.2.4. 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. This option MUST NOT occur more than once in a header.

3.2.5. 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. This option MUST NOT occur more than once in a header.

3.2.6. 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 may also be used with Datagram Transport Layer Security (DTLS) as described in Section 10 (Security Considerations). 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.

TOC

5.3. Proxying

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)).
When a request is made to a proxy, then the following steps are taken:

1. If the authority (host and port) is recognized as identifying 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 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.

3. If the proxy fails to obtain a fresh cached representation, then a 502 Bad Gateway error code MUST be returned

> 4. 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: proxy-max-age = original-max-age - 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. A CoAP end-point SHOULD support the CoRE Link Format of discoverable resources as described in (REF).

---

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 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 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 request/response exchanges 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
>
> **MAX_RETRANSMIT** 5

---

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.

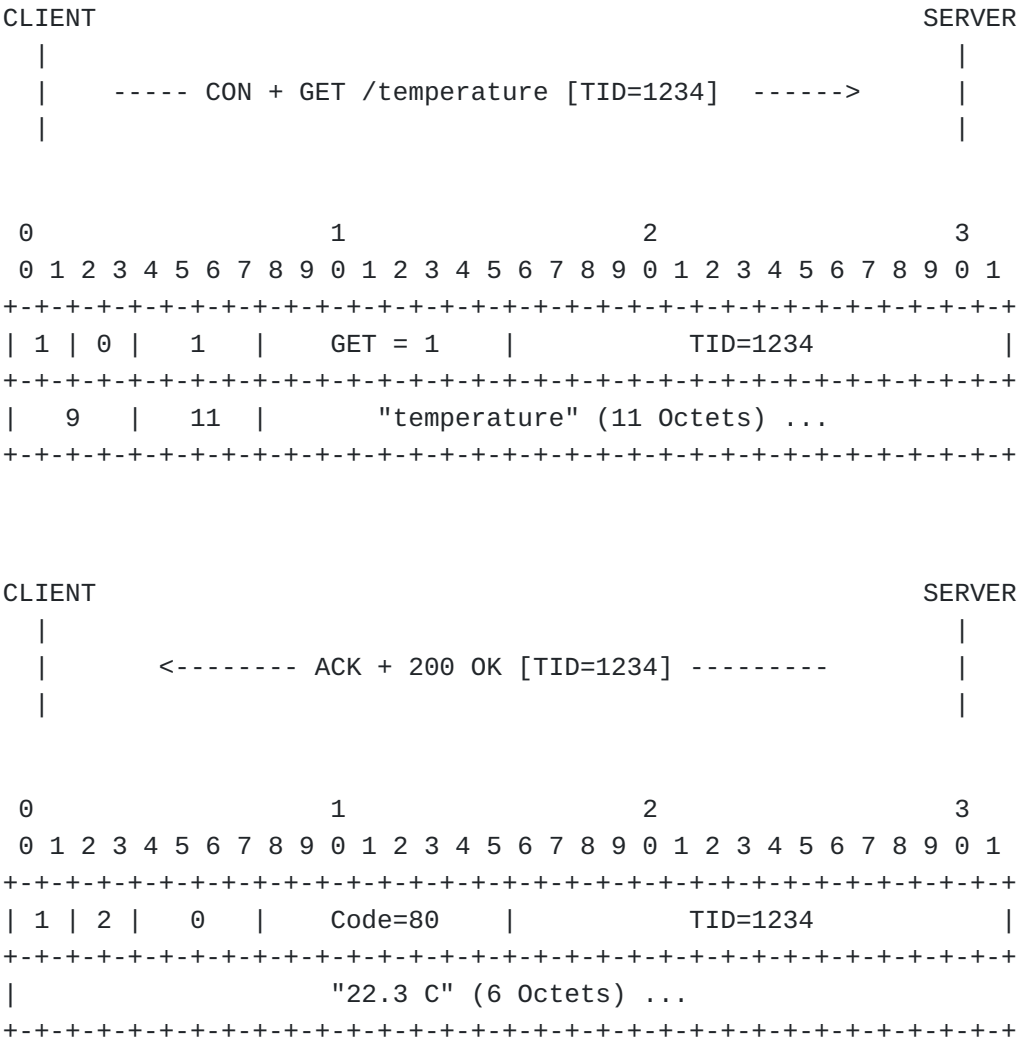

Figure 7: Basic request/response

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.

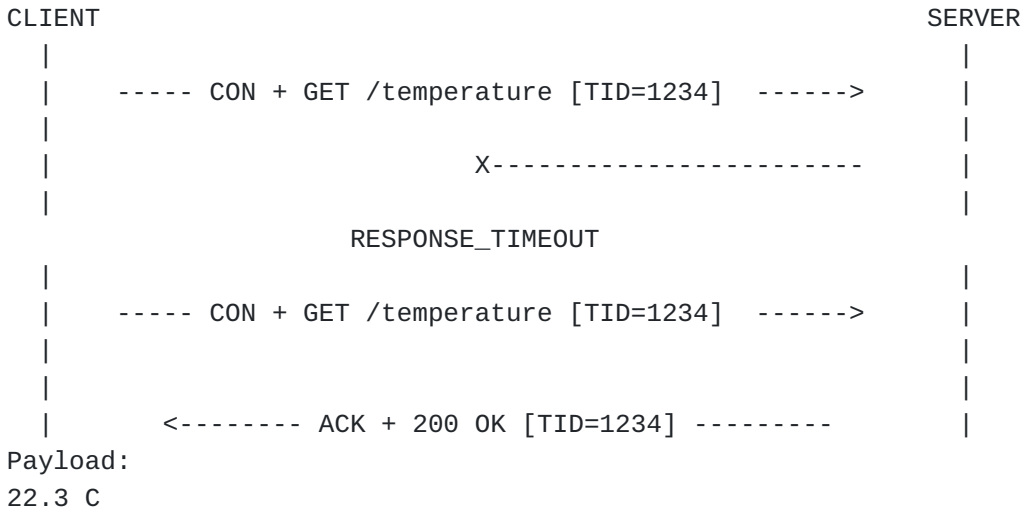

Figure 8: Basic request/response

Figure 9 (Basic request/response) shows an example of resource discovery. Here a unicast GET request is made to the server for /.well-known/core, which returns a list of two resource descriptions. The client then decides to make a request for the short URI of /sensor/light (/l). Requesting /sensors/light would result in the same representation.

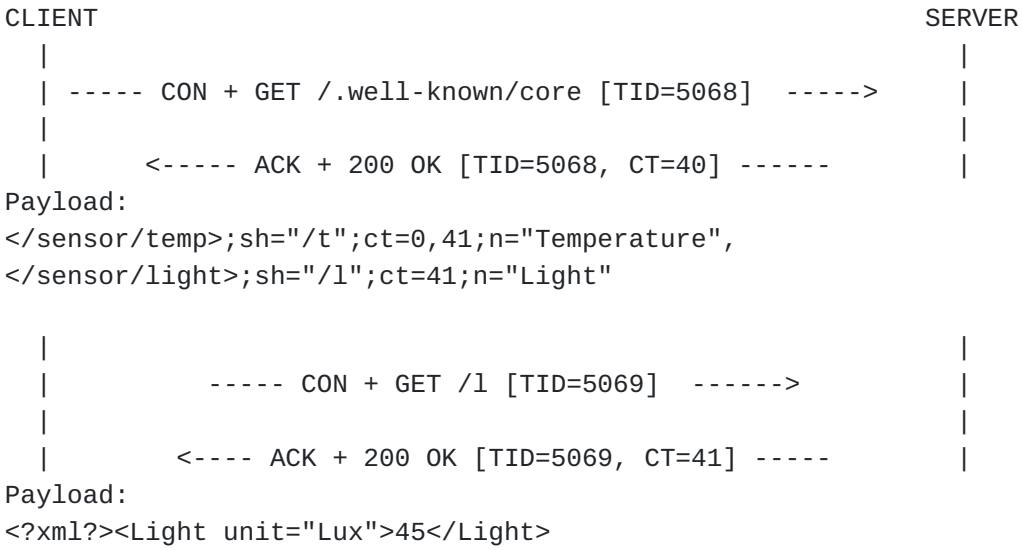

Figure 9: Basic request/response

Figure 10 (Basic request/response) shows an example of a multicast request. Here a client sends a request for /.well-known/core 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.

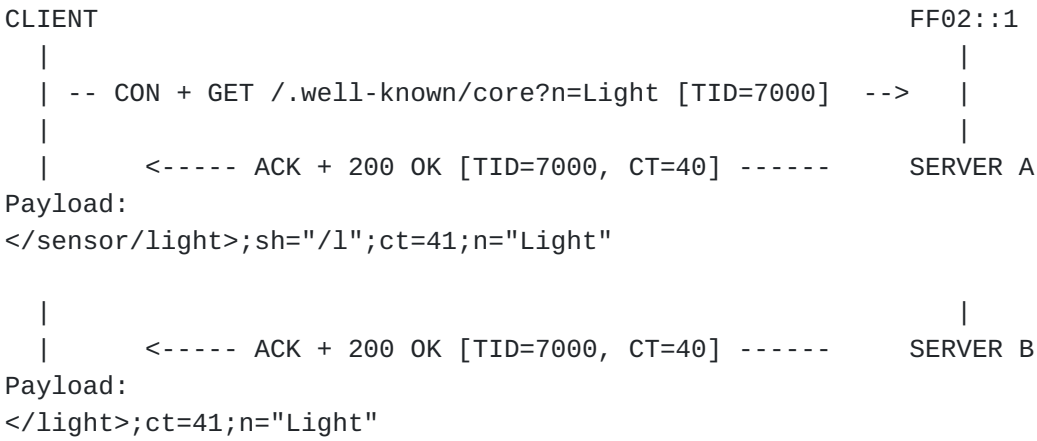

Figure 10: Basic request/response

10. Security Considerations

TOC

This section describes mechanisms that can be used to secure CoAP and analyzes the possible threats to the protocol and its limitations. Security bootstrapping (setting up keys) in constrained environments is considered in [I-D.oflynn-core-bootstrapping] (Sarikaya, B. and R. Cragie, "Initial Configuration of Resource-Constrained Devices," July 2010.).

10.1. Securing CoAP with IPSec

TOC

One mechanism to secure CoAP in constrained environments is the IPsec Encapsulating Security Payload (ESP) [RFC2406] (Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)," November 1998.). 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 most IEEE 802.15.4 radio chips are compatible with AES-CBC (with 128-bit keys) [RFC3602] (Frankel, S., Glenn, R., and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec," September 2003.) as defined for use with IPsec in [RFC4835] (Manral, V., "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)," April 2007.). When using IPsec to secure CoAP, both authentication and confidentiality SHOULD be applied as recommended in [RFC2406] (Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)," November 1998.). 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 backend 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.

TOC

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] (Rescorla, E. and N. Modadugu, “Datagram Transport Layer Security,” April 2006.) over UDP. This section describes 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) DTLS may not be applicable. The protocol is an order of magnitude more complex than CoAP and has appreciable handshake overhead needed to maintain security sessions. DTLS makes sense for applications where the session maintenance makes is compatible with application flows and sufficient resources are available on the constrained nodes and for the added network overhead.
As with IPSec, DTLS should be configured with a cypher suite compatible with any possible hardware engine on the node, for example AES-CBC in the case of IEEE 802.15.4. Implementations MUST support the mandatory to implement cipher suite TLS_RSA_WITH_AES_128_CBC_SHA as specified in [RFC5246] (Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” August 2008.).

10.3. Threat analysis and protocol limitations

TOC

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] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.) are also pertinent to CoAP. This section concentrates on describing limitations specific to CoAP and CoRE.

10.3.1. Processing URIs

TOC

TODO

TOC

10.3.2. Proxying and Caching

TODO

10.3.3. Attacks on TIDs

TOC

TODO

10.3.4. Risk of amplification using multicast

TOC

TODO

10.3.5. Asynchronous responses

TOC

TODO

---

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).

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
203	503 Service Unavailable
204	504 Gateway Timeout

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 [draft-shelby-core-link-format]	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 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, Peter Bigot, Colin O'Flynn and David Ryan for helpful comments and discussions that have shaped the document.

---

13. Changelog

TOC

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)

- 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.

- Completed the Content-type Identifier appendix and tables.
- Added more simplifications for URI support.
- Initial subscription and discovery sections.
- A Flag requirements simplified.

14. References

TOC

14.1. Normative References

TOC

[I-D.oflynn-core-bootstrapping]	Sarikaya, B. and R. Cragie, "Initial Configuration of Resource-Constrained Devices," draft-oflynn-core-bootstrapping-01 (work in progress), July 2010 (TXT).
[RFC2046]	Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types," RFC 2046, November 1996 (TXT).
[RFC2406]	Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)," RFC 2406, November 1998 (TXT, HTML, XML).
[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).
[RFC3602]	Frankel, S., Glenn, R., and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec," RFC 3602, September 2003 (TXT).
[RFC3986]	Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," STD 66, RFC 3986, January 2005 (TXT, HTML, XML).
[RFC4347]	Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security," RFC 4347, April 2006 (TXT).
[RFC4835]	Manral, V., "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)," RFC 4835, April 2007 (TXT).
[RFC5246]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, August 2008 (TXT).

14.2. Informative References

TOC

[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