| | | |
|---|---|---|
| CoRE | Z. Shelby | |
| Internet-Draft | Sensinode | |
| Intended status: Standards Track | B. Frank | |
| Expires: December 9, 2010 | SkyFoundry | |
| | D. Sturek | |
| | Pacific Gas & Electric | |
| | June 7, 2010 | |

TOC

**Constrained Application Protocol (CoAP)**
**draft-ietf-core-coap-00**

**Abstract**

This document specifies the Constrained Application Protocol (CoAP), a specialized 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 typical throughput of 10s of kbit/s. CoAP provides request/reply and subscribe/notify interaction models between application end-points, supports built-in resource discovery, and includes key web concepts such as URIs and RESTful methods. 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). 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 and building automation applications. This document specifies the RESTful Constrained Application Protocol (CoAP) which 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 [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:

*RESTful protocol design minimizing the complexity of mapping with HTTP.

*UDP binding with multicast and retransmission support.

*Low header overhead and parsing complexity.

*URI and Content-type support.

*Built-in resource discovery.

*Simple subscription for a resource with a resulting notification mechanism.

*Simple caching based on a relative time limit ("max-age").

The mapping of CoAP with HTTP is defined, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way.

---

## 2. Constrained Application Protocol

TOC

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

---

### 2.1. Interaction Model

TOC

The interaction model of CoAP is client/server with request or notify messages initiating a transaction responded to with a matching response based on a transaction ID. Machine-to-machine interactions with a RESTful design typically result in a CoAP implementation acting in both client and server roles (called an end-point). A CoAP request is similar to an HTTP request, and is sent by a client to request an action (using a method) on a resource (identified by a URI) on a server.
In addition to this typical request/response model, CoAP also supports an asynchronous subscribe/notify interaction model. A CoAP notify is the inverse of a request, where a server sends a notify message to a client about a resource on the server (identified by a URI). A notify includes the representation, Etag and/or Date of the resource. Example message exchanges can be found from Section 9 (Examples).
This document specifies the interaction of two CoAP end-points, one of which acting as a client, and the other acting as a server. A host may run any number of CoAP end-points.

---

#### 2.1.1. Request messages

TOC

A CoAP end-point acting as a client sends a request with the following rules. The Version field is set to 0. The Type Flag is set to 0 indicating a request. The A Flag SHOULD be set requesting a response and enabling retransmission in case of a timeout (see Section 4.1 (Retransmission)). The A Flag MAY be unset in cases when a response is too costly (such as a multicast message) or not useful (e.g. real-time

streaming). The Method field MUST be set with a value of 0-4. A new TRANSACTION_ID is generated, and this value is placed in the Transaction ID Field. See Section 2.1.4 (Option fields) for options rules. If a payload is to be included in the message, it immediately follows the last option or the Transaction ID if none.
For each request sent with the A flag set, a CoAP end-point keeps track of the destination IP address and Transaction ID of the request for the purpose of matching responses. The retransmission procedure is described in Section 4.1 (Retransmission).
Upon receiving a request, a CoAP end-point performs the following validation and processing:

> o The Version Field MUST be 0.
>
> o The Type Flag MUST be 0.
>
> o The Method Field MUST be 0-4.
>
> o If the Number of Options Field is > 0, then each option is validated and processed as in Section 2.1.4 (Option fields).
>
> o The length of the Payload is calculated from the datagram length.
>
> o The Method, URI, any options and Payload are passed on to the corresponding application process.
>
> o If the A bit is set, an appropriate response message MUST be sent to the source IPv6 address and port of the request with the same Transaction ID of the request. If the A bit is unset, a response message MUST NOT be sent.

---

### 2.1.2. Notify messages (Experimental)

TOC

The sending of a notify message is similar to sending a request message, with the following difference: The Type Flag is set to 2. The processing of a notify message is similar to processing a request message.

---

### 2.1.3. Response message

TOC

A response message is created with the following rules. The Version Field is set to 0. The Type Flag is set to 1. The Code is set to one of the supported response codes in Section 11.1 (Codes). The Transaction

ID MUST be set to that of the corresponding request. See Section 2.1.4 (Option fields) for options rules. An optional Payload may be included as appropriate for the request.
Upon receiving a response, a CoAP end-point performs the following validation and processing:

> o The Version Field MUST be 0.
>
> o The Type Flag MUST be 1.
>
> o The Code Field MUST contain a valid code.
>
> o If the Number of Options Field is > 0, then each option is validated and processed as in Section 2.1.4 (Option fields).
>
> o The length of the Payload is calculated from the datagram length.
>
> o The Transaction ID is used to match the response to an open request entry, and the response code, any options and Payload are passed on to the corresponding application process. If no match is found, the message is silently discarded.

---

### 2.1.4. Option fields

TOC

If no options are to be included, the Option Number Field is set to 0 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 Number of Options Field. Each option is then placed in order of Type, immediately following the Transaction ID with no padding. Upon reception, unknown options MUST be silently skipped.

---

### 2.1.5. Transaction IDs

TOC

The Transaction ID is an unsigned integer kept by a CoAP end-point for all of the CoAP request or notify messages it sends. Each CoAP end-point keeps a single Transaction ID variable, which is changed each time a new request or notify message is sent regardless of the destination address or port. The Transaction ID is used to match a response with an outstanding request or notify, for retransmission and to discard duplicate messages. The initial Transaction ID should be randomized.

---

### 2.2. Methods

TOC

CoAP supports the basic RESTful methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP methods. In this section each method is defined along with its behavior. In addition, CoAP defines a new SUBSCRIBE method for requesting soft-state subscriptions for resources. 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.

---

#### 2.2.1. 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).

---

#### 2.2.2. 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 the header and any possible status in the message body. If the POST does not result in a new resource being created on the server, a 200 (OK) response code is returned.
Responses to this method are not cacheable.

---

#### 2.2.3. 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. If no resource exists then the server MAY create a new resource with that URI.
Responses to this method are not cacheable.

### 2.2.4. 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.

### 2.2.5. SUBSCRIBE (Experimental)

TOC

CoAP supports a built-in subscribe/notify push model for an end-point to notify another end-point about a resource of interest. This push is accomplished using the CoAP notify message type, whose URI corresponds to the resource of interest on the end-point sending the notify message. A notify may include the latest representation of the resource in its payload and/or the Etag Option.
The SUBSCRIBE method allows an end-point to request notifications about a resource. A request of this method MAY include the Subscription-lifetime Option defined in Section 3.2.6 (Subscription-lifetime Option (Experimental)). In the absence of this Option, its maximum lifetime is assumed. End-points MUST NOT send notify messages without a valid subscription. Subscriptions are soft-state, and must be refreshed by sending a new SUBSCRIBE message before the end of its lifetime.
Servers keep track of subscriptions, and upon change a notify message is sent to the source address and port of the original SUBSCRIBE request with the URI of the resource in question. Notifications MAY be sent with the A bit set, enabling a server to detect if a subscriber is no longer valid. A subscription SHOULD be removed after MAX_RETRANSMIT failures when the A bit is set. A server is not required to support subscriptions for its resources (thus this feature is optional), and MAY limit the number of simultaneous subscriptions.

## 2.3. 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 which resource is being manipulated. CoAP supports variable-length string URIs with the Uri Option. As this URI is used 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 supports relative references in the Uri Option (e.g.

/sensors/temperature) for messages to a CoAP end-point, and absolute URIs for use with a proxy (coap://[2001:1ba3::450a]/sensors/temperature), and does not support "." and ".." schemes. A CoAP implementation MAY support query "?" processing if needed, however fragment "#" processing is not supported. IRIs are 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 including a relative reference URI in the Uri Option, the leading slash MUST be omitted. Thus the above example "/sensors/temperature" is included in the Uri Option as "sensors/temperature".
The CoAP protocol scheme is identified in URIs with "coap://" (TODO: IANA considerations).

---

### 2.4. CoAP Codes

TOC

When a response message is sent in response to a request or notify message it MUST always include a response code in the header. Notify messages also include a code field, which is set to "200 OK" by default. CoAP makes use of a subset of HTTP response codes as defined in Section 11.1 (Codes).

---

### 2.5. 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 [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 MIME 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.

---

TOC

## 3. Message Formats

CoAP makes use of three message types - request, notify and response, 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 or the Length Field of the magic byte header if included. When bound to UDP the entire message MUST fit within a single datagram. When used with 6LoWPAN [RFC4944] (Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," September 2007.), messages SHOULD fit into a single IEEE 802.15.4 frame to minimize fragmentation.

---

### 3.1. CoAP header

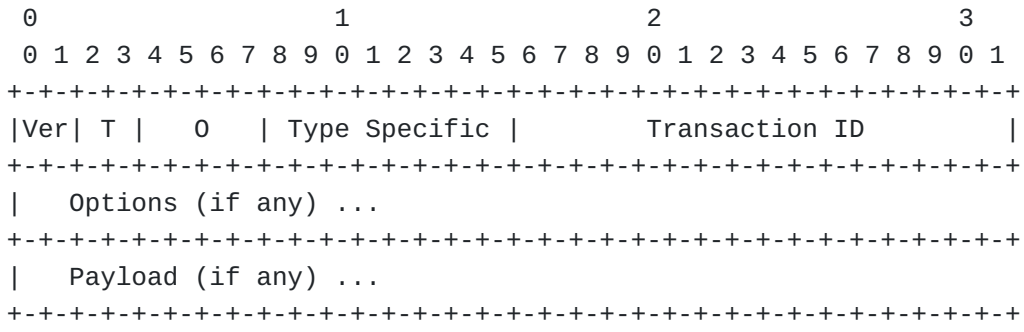
TOC

This section defines the CoAP header, which is shared for all message types.

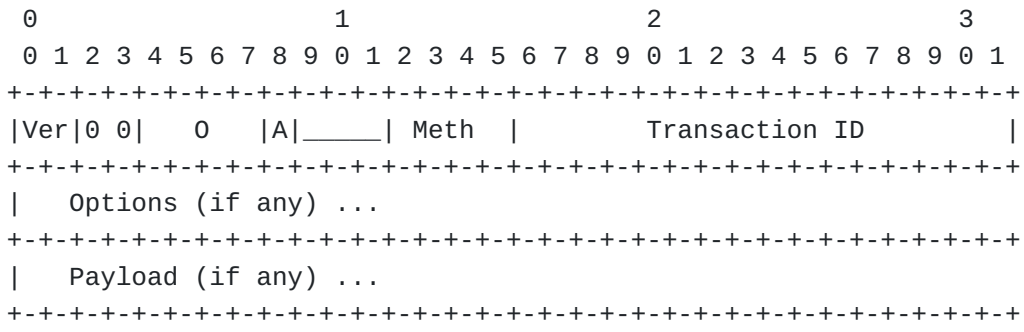
> **Request:** A CoAP request message is sent by a client to request a URI on a server using one of the methods listed in Table 1 (Method codes).
>
> **Response:** A CoAP response message is sent in response to a CoAP request or notify when appropriate. Responses include a Transaction ID corresponding to that of the request. A response is always sent when the A flag is set in a request, and is never sent when the A flag is not set. A response is always sent to the source IP address and port of the corresponding request or notify.
>
> **Notify: (Experimental)** A CoAP notify message is sent by a server to notify a client about a resource (identified by a URI) on the server as a result of a valid subscription for that resource.

---

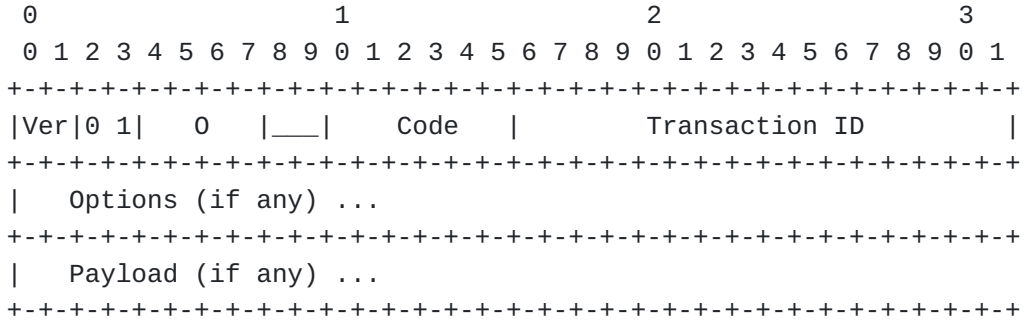
Template:




Request (T=0):




Response (T=1):




Notify (T=2):

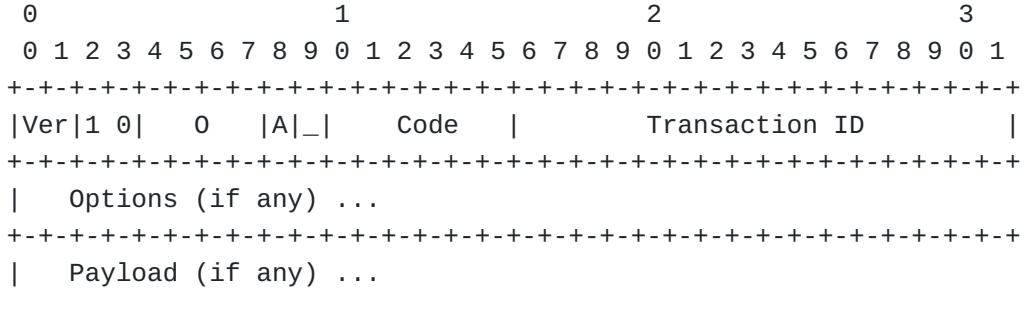


**Figure 1: CoAP header format**

---

**Header Fields:**

**Ver:** Version. 2-bit unsigned integer. Indicates the version of CoAP. Implementations of this specification MUST set this field to 0. The values 1-3 are reserved for future versions.

**T:** 2-bit Message Type flag. Indicates if this message is a CoAP request (0), response (1) or notify (2) header. The value 3 is forbidden to avoid collision with the magic byte 'r'.

**O:** 4-bit Number of Options 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.

**A:** 1-bit Acknowledgement flag. When set to 1, indicates that the destination MUST respond with a response message matching this request (see Section 4 (UDP Binding)). When set to 0, the destination MUST NOT send a response to this request.

**Meth:** 4-bit unsigned integer. This field indicates the CoAP Method of the request according to Table 1 (Method codes). Methods are described in detail in Section 2.2 (Methods).

**Code:** 6-bit unsigned integer. This field indicates the code of a response or notify message 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.

**_:** This field is unused. It MUST be initialized to zero by the sender and MUST be ignored by the receiver.

| Method | Code |
|---|---|
| GET | 0 |
| POST | 1 |
| PUT | 2 |
| DELETE | 3 |
| SUBSCRIBE | 4 |

**Table 1: Method codes**

### 3.2. Header options

TOC

CoAP messages may also include one or more header options in TLV format. Each option has the following format:

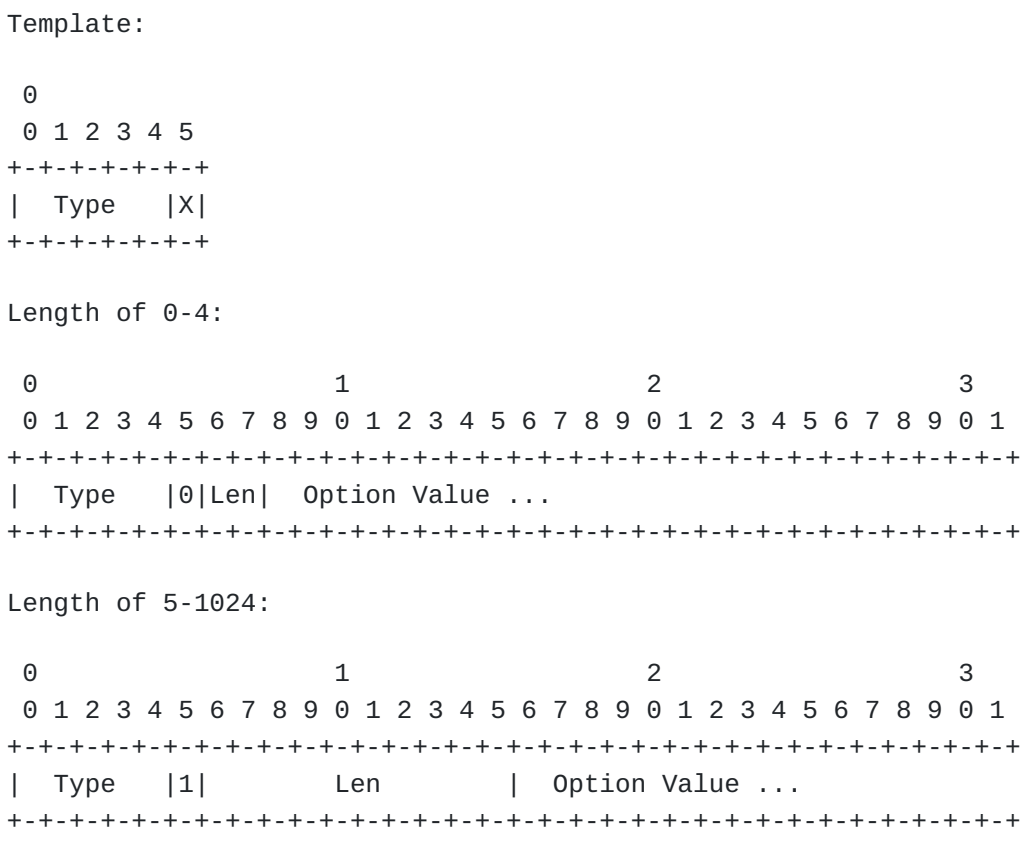



**Figure 2: Header option format**

---

**Type:** 5-bit unsigned integer. The type of the option as defined in Table 2 (Option headers), allowing for up to 32 options. Future specifications may define new CoAP option types. Option types 30-32 are reserved for experimental purposes.

**X:** 1-bit Extended Length Flag. When 0 the Length is a 2-bit unsigned integer. When 1 the option header is extended by an octet and Length is a 10-bit unsigned integer.

**Len:** Length Field. When X is 0 Length is a 2-bit unsigned integer allowing values of 0-3 octets. When X is 1 Length is a 10-bit unsigned integer allowing values of 0-1023 octets.

**Option Value** The value in the format defined for that option in Table 2 (Option headers) with a length of Option Len octets. Options may use variable length unsigned integer values of Len Field octets in network byte order as shown in Figure 3 ( Variable length unsigned integer value format).

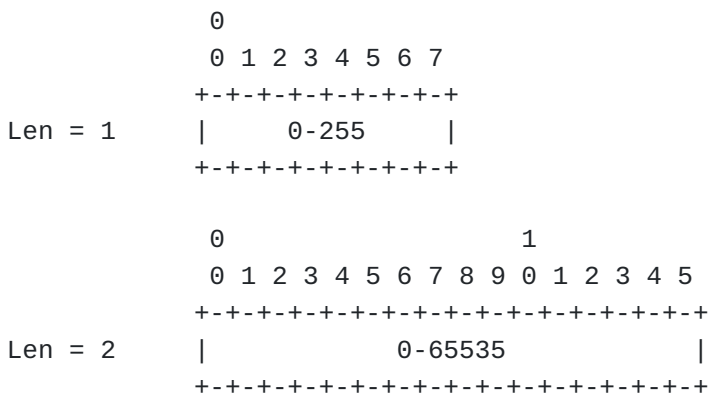



**Figure 3: Variable length unsigned integer value format**

The following options are defined in this document.

| Type | Name | Data type | Length | Rules |
|---|---|---|---|---|
| 0 | Content-type | Variable uint | 1-2 B | |
| 1 | Uri | String | 1-32768 B | Never in response |
| 2 | Not used | - | - | - |
| 3 | Max-age | Variable uint | 0-4 B | |
| 4 | Etag | Variable uint | 1-4 B | |
| 5 | Date | Variable integer | 4-6 B | Never in request |
| 6 | Subscription-lifetime | Variable uint | 1-3 B | With SUBSCRIBE or its response |

**Table 2: Option headers**

TOC

#### 3.2.1. Content-type Option

The Content-type Identifier Option indicates the Internet Media Type 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, and MUST not be included for a zero-length payload. In the absence of the Content-type Option the MIME type "text/plain" MUST be assumed.

---

#### 3.2.2. Uri Option

TOC

The Uri Option indicates the string URI of the resource that may be included in request and notify messages. In the absence of this option, the URI is assumed to be "/". Section 2.3 (URIs) specifies the rules for URIs in CoAP. When including a relative reference URI in the Uri Option, the leading slash MUST be omitted.

---

#### 3.2.3. Max-age Option

TOC

The Max-age Option indicates the maximum age of the resource for use in cache control in seconds. The option is represented as a variable length unsigned integer maximum 32-bits in length. A length of 0 is used to indicate a Max-age of 0.
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 or a notify, Max-age indicates the maximum time the representation may be cached before it MUST be discarded.

---

#### 3.2.4. Etag Option

TOC

The Etag Option is a variable length unsigned integer 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 notify message to indicate to a client if a resource has changed.

---

TOC

### 3.2.5. Date Option

The Date Option indicates the creation time and date of a given resource representation. It MAY be used in response and notify messages. The integer value is the number of seconds, after midnight UTC, January 1, 1970. This time format cannot represent time values prior to January 1, 1970. The latest UTC time value that can be represented by a 31 bit integer value is 03:14:07 on January 19, 2038. Time values beyond 03:14:07 on January 19, 2038, are represented by 39 bit integer values which is sufficient to represent dates that should be enough for anyone. For applications requiring more accuracy, a 48-bit integer MAY be included representing this value in milliseconds instead of seconds.

---

### 3.2.6. Subscription-lifetime Option (Experimental)

TOC

The Subscription-lifetime Option indicates the subscription lifetime and is optionally included with the SUBSCRIBE method (see Section 2.2.5 (SUBSCRIBE (Experimental))). The corresponding response MUST include a Subscription-lifetime Option confirming (or modifying) the subscription lifetime.
The value of this option is a variable length unsigned integer up to 24-bits indicating the lifetime of the subscription in seconds with a maximum value of 194 days. In a response the server MAY return a different value that fits its own scheduling better. A value of all 0 in a request indicates cancellation of a subscription and in a response indicates subscription failure or rejection.

---

## 4. UDP Binding

TOC

The CoAP protocol operates by default over UDP. CoAP could 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 TCP. CoAP over UDP has the following features:

- Simple stop-and-wait retransmission reliability with exponential back-off as described in Section 4.1 (Retransmission) when the A Flag is set.

- Transaction ID for response matching as described in Section 2.1.5 (Transaction IDs).

*Multicast support without retransmission. CoAP supports the use of multicast destination addresses when bound to UDP. Although the A bit may be used to force a response, retransmission MUST NOT be performed.

When a CoAP message is sent using UDP, the length of the Payload is calculated from the datagram length. When bound to UDP the entire message MUST fit within a single datagram of length 1024 octets. When used with 6LoWPAN [RFC4944] (Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," September 2007.), messages SHOULD fit into a single link-layer frame to minimize fragmentation if possible (often on the order of 60-90 octets).

---

### 4.1. Retransmission

TOC

A CoAP end-point keeps track of open request or notify messages expecting a response (A Flag set). Each entry includes at least the destination address and port of the original message, the message itself, a retransmission counter (UDP only) and a timeout. When a request or notify message is sent with the A Flag set, 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 response is received for an entry, the entry is removed. 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.

---

### 4.2. Default Port

TOC

CoAP SHOULD use a default port of 61616 which is within the 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.). As this port is in the dynamic port space, it however can not be reserved for CoAP use.

---

## 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 publishes a resource for a GET request, 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.

---

TOC

### 5.2. Cache refresh

After the expiration of the cache lifetime, clients and proxies can refresh their cached representation of a resource. Cache refresh is accomplished using 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 integer which captures a version checksum 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, 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 it SHOULD specify a 304 response with no payload to avoid retransmitting the resource representation.

---

### 5.3. Proxying

TOC

See [I-D.frank-6lowapp-chopan] (Frank, B., “Chopan - Compressed HTTP Over PANs,” September 2009.).
TODO:

- *Are interception proxies are still required to deal with a) sleeping nodes and b) protecting Internet HTTP traffic from overwhelming the CoAP network?
- *But interception proxies breaks end-to-end IP encapsulation and requires support at the routing level
- *Often the interception proxy is the same as the HTTP-to-CoAP gateway, so we need to decide how these topics dovetail
- *In Chopan, the sleeping problem was tackled by having sleeping nodes check-in with their proxies while awake, notify model might solve this problem to some extent but still have to coordinate the sleep/awake times
- *In Chopan we actually used caching to deal with POSTs, etc because otherwise how do you send a request to a sleeping node? The current caching sections are to be exclusive to GETs, but we still need to solve the problem for other types if methods.

---

TOC

## 6. Resource Discovery

The discovery of resources offered by a CoAP end-point is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is called Web Discovery. In this document we refer to the discovery of resources offered by a CoAP end-point as Resource Discovery.

CoAP makes the assumption that end-points are available on the default CoAP port, or otherwise have been configured or discovered using some general service discovery mechanism such as [I-D.cheshire-dnsext-multicastdns] (Cheshire, S. and M. Krochmal, "Multicast DNS," March 2010.). This section assumes that such a configuration or service discovery has already been performed, if needed.

Resource Discovery in CoAP is accomplished through the use of well-known resources which describe the links offered by that CoAP end-point. Well-known resources have the URI form "/.well-known/" as specified in [RFC5785] (Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)," April 2010.). Every CoAP end-point MUST support this well-known resource. The resource representation of this is described in Section 6.1 (Link Format).

CoAP requests the following well-known URL for discovery: "/.well-known/resources" (TODO: Formal description for use in request as per [RFC5785] (Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)," April 2010.)).

CoAP Resource Discovery supports the following interactions:

- [request GET /.well-known/resources] returns the list of links available from a CoAP end-point.
- A CoAP end-point may notify interested clients when this description has changed by sending [notify /.well-known/resources]. This resource MAY support subscription.
- More capable end-points such as proxies MAY support a resource directory by accepting [request POST /.well-known/resources] messages from other CoAP end-points. This adds the resources of other end-points to an agent directory in which absolute URIs are included for the links.

End-points with a large number of resources SHOULD organize their resource descriptions into a hierarchy of link resources. This is done by including links in the /.well-known/resources list which point to other resource lists, e.g. /.well-known/resources/sensors.

---

TOC

### 6.1. Link Format

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.). 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.
CoAP defines a subset of the [I-D.nottingham-http-link-header] (Nottingham, M., "Web Linking," April 2010.) features and specific parameters that have known meaning for CoAP resource discovery. A CoAP end-point MAY make use of link extension parameters as needed. The CoAP link format does not start with the "Link:" text. The following formal description is used for forming and parsing this link format:

```
link-value        = "<" URI-Reference ">" *( ";" link-param )
link-param        = ( ( "desc" "=" URI )
                  | ( "name" "=" quoted-string )
                  | ( "type" "=" ( media-type | media-code) )
                  | ( "id" "=" integer )
                  | ( link-extension ) )
link-extension    = ( parmname [ "=" ( ptoken | quoted-string ) ] )
ptoken            = 1*ptokenchar
ptokenchar        = "!" | "#" | "$" | "%" | "&" | "'" | "("
                  | ")" | "*" | "+" | "-" | "." | "/" | DIGIT
                  | ":" | "<" | "=" | ">" | "?" | "@" | ALPHA
                  | "[" | "]" | "^" | "_" | "`" | "{" | "|"
                  | "}" | "~"
media-code        = see Section 11.2
media-type        = type-name "/" subtype-name
```

The link-value is the relative URI of the resource on that end-point or an absolute URI in the case of a directory agent. The desc parameter is a URI that points to the definition of that resource interface, for example in WADL. The name parameter is a descriptive or ontology name of the resource class. This name parameter SHOULD be in an m-DNS [I-D.cheshire-dnsext-multicastdns] (Cheshire, S. and M. Krochmal, "Multicast DNS," March 2010.) compatible form. The type parameter includes Internet media type this resource returns in ascii or code format. The id field is a unique identifier (e.g. UUID) for this resource for use in e.g. search directories. All link parameters are optional and custom link-extensions may be defined. An example of a typical CoAP link description in this format would be:

```
</sensor/temp>; name="TemperatureC"; type=text/xml;<CR>
</sensor/light>; name="LightLux"; type=text/xml;
```

---

## 7. HTTP Mapping

TOC

TODO

---

## 8. Protocol Constants

TOC

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

> **RESPONSE_TIMEOUT** 1 second
>
> **MAX_RETRANSMIT** 5

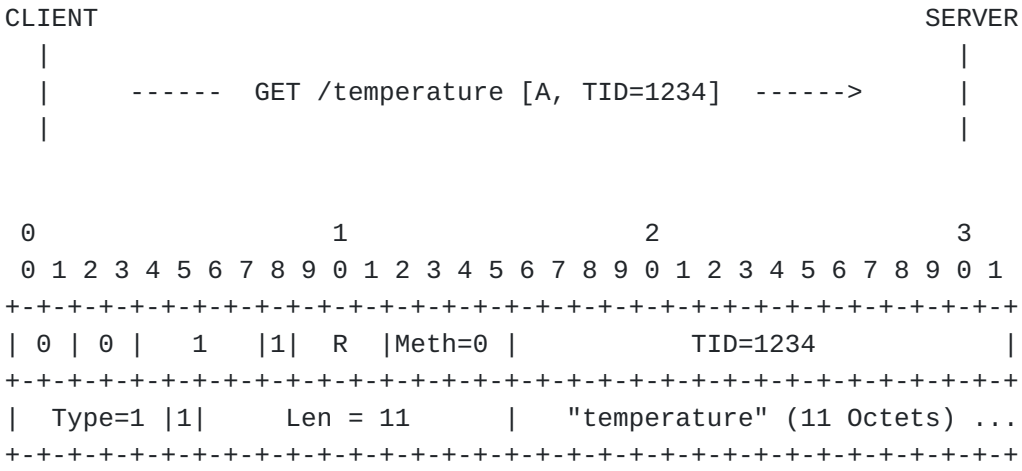
---

## 9. Examples

TOC

Figure 4 (Basic request/response) shows a basic request sequence. A client makes a GET request for the resource /temperature to the server. The A Flag is set, requesting a response and the Transaction ID is 1234. The request includes one Uri Option "temperature" of Len = 11. This request is a total of 17 octets long. The corresponding response 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.

---



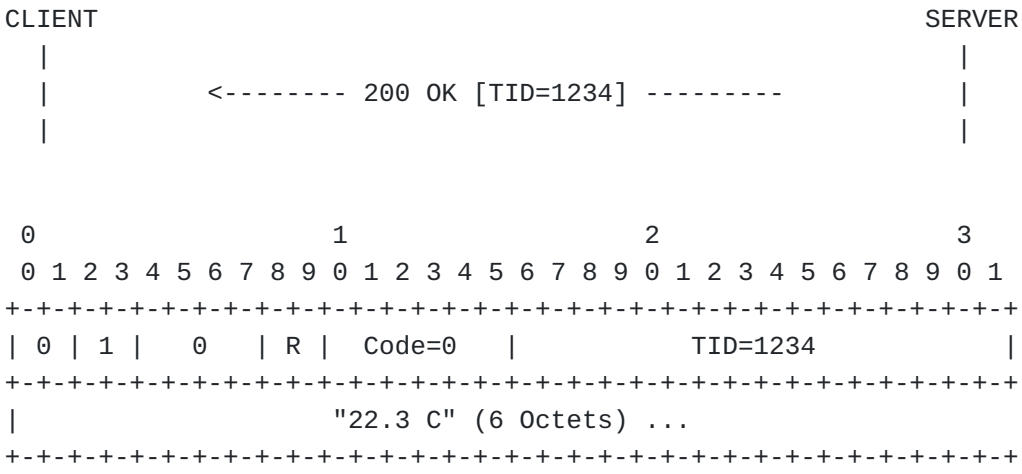




**Figure 4: Basic request/response**

---

TODO: Request with multiple packed messages (magic byte example..)
TODO: Request - Response (with retransmission)
TODO: Request - Response (discovery)
TODO: Request (SUBSCRIBE) - Response ... Resulting Notify - 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

TODO (See IANA comments in the document).

---

### 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 a 6-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 |
|---|---|
| 0 | 200 OK |
| 1 | 201 Created |
| 14 | 304 Not Modified |
| 20 | 400 Bad Request |
| 21 | 401 Unauthorized |
| 23 | 403 Forbidden |
| 24 | 404 Not Found |
| 25 | 405 Method Not Allowed |
| 29 | 409 Conflict |
| 35 | 415 Unsupported Media Type |
| | |
| 40 | 500 Internal Server Error |
| 43 | 503 Service Unavailable |

| | |
|---|---|
| 44 | 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.

The Content-type Option is formatted as a variable length unsigned integer, thus the most common media types are encoded into an 8-bit unsigned integer. This identifier is encoded as follows. Regardless of the length of the integer, the most significant 3 bits indicates the top-level media type (text, application etc.) as defined in Table 4 (Top-level type identifiers). The five initial top-level types defined in [RFC2046] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types,” November 1996.) are supported. Composite high-level types (multipart and message) are not supported. The remaining bits indicate the sub-types [RFC2046] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types,” November 1996.). This allows for up to 8 high-level types, with up to 32 sub-types for each in an 8-bit identifier and up to 8192 sub-types in a 16-bit identifier.

For example, "application/xml" would be encoded in 8-bits as:

```
5 << 5 | 00  =  10100000
```

| Top-level type | Identifier |
|---|---|
| text | 1 |
| image | 2 |
| audio | 3 |
| video | 4 |

| | |
|---|---|
| application | 5 |

**Table 4: Top-level type identifiers**

---

| Sub-type | Identifier |
|---|---|
| xml | 0 |
| plain | 1 |
| csv | 2 |
| html | 3 |

**Table 5: text sub-type identifiers**

---

| Sub-type | Identifier |
|---|---|
| gif | 0 |
| jpeg | 1 |
| png | 2 |
| tiff | 3 |

**Table 6: image sub-type identifiers**

---

| Sub-type | Identifier |
|---|---|
| raw | 0 |

**Table 7: audio sub-type identifiers**

---

| Sub-type | Identifier |
|---|---|
| raw | 0 |

**Table 8: video sub-type identifiers**

| Sub-type | Identifier |
|---|---|
| xml | 0 |
| octet-stream | 1 |
| rdf+xml | 2 |
| soap+xml | 3 |
| atom+xml | 4 |
| xmpp+xml | 5 |
| exi | 6 |
| x-bxml | 7 |
| fastinfoset | 8 |
| soap+fastinfoset | 9 |
| json | 10 |

**Table 9: application sub-type identifiers**

## 12. Acknowledgments

TOC

Thanks to Carsten Bormann, 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 and David Ryan for helpful comments and discussions.

## 13. Changelog

TOC

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.frank-6lowapp-chopan] | Frank, B., "Chopan - Compressed HTTP Over PANs," draft-frank-6lowapp-chopan-00 (work in progress), September 2009 (TXT). |
| [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). |
| [RFC4346] | Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," RFC 4346, April 2006 (TXT). |
| [RFC4347] | Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security," RFC 4347, April 2006 (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-multicastdns] | Cheshire, S. and M. Krochmal, “Multicast DNS,” draft-cheshire-dnsext-multicastdns-11 (work in progress), March 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 |