Internet Engineering Task Force (IETF)
Request for Comments: 8323
Updates: 7641, 7959
Category: Standards Track
ISSN: 2070-1721

C. Bormann
Universitaet Bremen TZI
S. Lemay
Zebra Technologies
H. Tschofenig
ARM Ltd.
K. Hartke
Universitaet Bremen TZI
B. Silverajan
Tampere University of Technology
B. Raymor, Ed.

February 2018

CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets

Abstract

The Constrained Application Protocol (CoAP), although inspired by HTTP, was designed to use UDP instead of TCP. The message layer of CoAP over UDP includes support for reliable delivery, simple congestion control, and flow control.

Some environments benefit from the availability of CoAP carried over reliable transports such as TCP or Transport Layer Security (TLS). This document outlines the changes required to use CoAP over TCP, TLS, and WebSockets transports. It also formally updates RFC 7641 for use with these transports and RFC 7959 to enable the use of larger messages over a reliable transport.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8323.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP [RFC768] can be used unimpeded as can the Datagram Transport Layer Security (DTLS) protocol [RFC6347] over UDP. The use of CoAP over UDP is focused on simplicity, has a low code footprint, and has a small over-the-wire message size.

The primary reason for introducing CoAP over TCP [RFC793] and TLS [RFC5246] is that some networks do not forward UDP packets. Complete blocking of UDP happens in between about 2% and 4% of terrestrial access networks, according to [EK2016]. UDP impairment is especially concentrated in enterprise networks and networks in geographic regions with otherwise challenged connectivity. Some networks also rate-limit UDP traffic, as reported in [BK2015], and deployment investigations related to the standardization of Quick UDP Internet Connections (QUIC) revealed numbers around 0.3% [SW2016].

The introduction of CoAP over TCP also leads to some additional effects that may be desirable in a specific deployment:

o Where NATs are present along the communication path, CoAP over TCP leads to different NAT traversal behavior than CoAP over UDP. NATs often calculate expiration timers based on the transport-layer protocol being used by application protocols. Many NATs maintain TCP-based NAT bindings for longer periods based on the assumption that a transport-layer protocol, such as TCP, offers additional information about the session lifecycle. UDP, on the other hand, does not provide such information to a NAT and timeouts tend to be much shorter [HomeGateway]. According to [HomeGateway], the mean for TCP and UDP NAT binding timeouts is 386 minutes (TCP) and 160 seconds (UDP). Shorter timeout values require keepalive messages to be sent more frequently. Hence, the use of CoAP over TCP requires less-frequent transmission of keepalive messages.

o TCP utilizes mechanisms for congestion control and flow control that are more sophisticated than the default mechanisms provided by CoAP over UDP; these TCP mechanisms are useful for the transfer of larger payloads. (However, work is ongoing to add advanced congestion control to CoAP over UDP as well; see [CoCoA].)

Note that the use of CoAP over UDP (and CoAP over DTLS over UDP) is still the recommended transport for use in constrained node networks, particularly when used in concert with block-wise transfer. CoAP over TCP is applicable for those cases where the networking infrastructure leaves no other choice. The use of CoAP over TCP leads to a larger code size, more round trips, increased RAM requirements, and larger packet sizes. Developers implementing CoAP over TCP are encouraged to consult [TCP-in-IoT] for guidance on low-footprint TCP implementations for IoT devices.

Standards based on CoAP, such as Lightweight Machine to Machine [LWM2M], currently use CoAP over UDP as a transport; adding support for CoAP over TCP enables them to address the issues above for specific deployments and to protect investments in existing CoAP implementations and deployments.

Although HTTP/2 could also potentially address the need for enterprise firewall traversal, there would be additional costs and delays introduced by such a transition from CoAP to HTTP/2. Currently, there are also fewer HTTP/2 implementations available for constrained devices in comparison to CoAP. Since CoAP also supports group communication using IP-layer multicast and unreliable communication, IoT devices would have to support HTTP/2 in addition to CoAP.

Furthermore, CoAP may be integrated into a web environment where the front end uses CoAP over UDP from IoT devices to a cloud infrastructure and then CoAP over TCP between the back-end services. A TCP-to-UDP gateway can be used at the cloud boundary to communicate with the UDP-based IoT device.

Finally, CoAP applications running inside a web browser may be without access to connectivity other than HTTP. In this case, the WebSocket Protocol [RFC6455] may be used to transport CoAP requests and responses, as opposed to cross-proxying them via HTTP to an HTTP-to-CoAP cross-proxy. This preserves the functionality of CoAP without translation -- in particular, the Observe Option [RFC7641].

To address the above-mentioned deployment requirements, this document defines how to transport CoAP over TCP, CoAP over TLS, and CoAP over WebSockets. For these cases, the reliability offered by the transport protocol subsumes the reliability functions of the message layer used for CoAP over UDP. (Note that for both a reliable transport and the message layer for CoAP over UDP, the reliability offered is per transport hop: where proxies -- see Sections 5.7 and 10 of [RFC7252] -- are involved, that layer’s reliability function does not extend end to end.) Figure 1 illustrates the layering:

```
  +--------------------------------+
  |          Application           |
  +--------------------------------+
  +--------------------------------+
  |  Requests/Responses/Signaling  |  CoAP (RFC 7252) / This Document
  |--------------------------------|
  |        Message Framing         |  This Document
  +--------------------------------+
  |      Reliable Transport        |
  +--------------------------------+
```

Figure 1: Layering of CoAP over Reliable Transports

This document specifies how to access resources using CoAP requests and responses over the TCP, TLS, and WebSocket protocols. This allows connectivity-limited applications to obtain end-to-end CoAP connectivity either (1) by communicating CoAP directly with a CoAP server accessible over a TCP, TLS, or WebSocket connection or (2) via a CoAP intermediary that proxies CoAP requests and responses between different transports, such as between WebSockets and UDP.

Section 7 updates [RFC7641] ("Observing Resources in the Constrained Application Protocol (CoAP)") for use with CoAP over reliable transports. [RFC7641] is an extension to CoAP that enables CoAP clients to "observe" a resource on a CoAP server. (The CoAP client retrieves a representation of a resource and registers to be notified by the CoAP server when the representation is updated.)

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455], [RFC7252], [RFC7641], and [RFC7959].

The term "reliable transport" is used only to refer to transport protocols, such as TCP, that provide reliable and ordered delivery of a byte stream.

Block-wise Extension for Reliable Transport (BERT):
  Extends [RFC7959] to enable the use of larger messages over a reliable transport.

BERT Option:
  A Block1 or Block2 option that includes an SZX (block size) value of 7.

BERT Block:
  The payload of a CoAP message that is affected by a BERT Option in descriptive usage (see Section 2.1 of [RFC7959]).

Transport Connection:
  Underlying reliable byte-stream connection, as directly provided by TCP or indirectly provided via TLS or WebSockets.

Connection:
  Transport Connection, unless explicitly qualified otherwise.

Connection Initiator:
The peer that opens a Transport Connection, i.e., the TCP active opener, TLS client, or WebSocket client.

Connection Acceptor:
The peer that accepts the Transport Connection opened by the other peer, i.e., the TCP passive opener, TLS server, or WebSocket server.

3. CoAP over TCP

The request/response interaction model of CoAP over TCP is the same as CoAP over UDP. The primary differences are in the message layer. The message layer of CoAP over UDP supports optional reliability by defining four types of messages: Confirmable, Non-confirmable, Acknowledgment, and Reset. In addition, messages include a Message ID to relate Acknowledgments to Confirmable messages and to detect duplicate messages.

Management of the transport connections is left to the application, i.e., the present specification does not describe how an application decides to open a connection or to reopen another one in the presence of failures (or what it would deem to be a failure; see also Section 5.4). In particular, the Connection Initiator need not be the client of the first request placed on the connection. Some implementations will want to implement dynamic connection management similar to the technique described in Section 6 of [RFC7230] for HTTP: opening a connection when the first client request is ready to be sent, reusing that connection for subsequent messages until no more messages are sent for a certain time period and no requests are outstanding (possibly with a configurable idle time), and then starting a release process (orderly shutdown) (see Section 5.5). In implementations of this kind, connection releases or aborts may not be indicated as errors to the application but may simply be handled by automatic reconnection once the need arises again. Other implementations may be based on configured connections that are kept open continuously and lead to management system notifications on release or abort. The protocol defined in the present specification is intended to work with either model (or other, application-specific connection management models).

3.1. Messaging Model

Conceptually, CoAP over TCP replaces most of the message layer of CoAP over UDP with a framing mechanism on top of the byte stream provided by TCP/TLS, conveying the length information for each message that, on datagram transports, is provided by the UDP/DTLS datagram layer.

TCP ensures reliable message transmission, so the message layer of CoAP over TCP is not required to support Acknowledgment messages or to detect duplicate messages. As a result, both the Type and Message ID fields are no longer required and are removed from the message format for CoAP over TCP.

Figure 2 illustrates the difference between CoAP over UDP and CoAP over reliable transports. The removed Type and Message ID fields are indicated by dashes.

```
      CoAP Client       CoAP Server      CoAP Client       CoAP Server
          |                    |            |                    |
          |   CON [0xbc90]     |            | (-------) [------] |
          | GET /temperature   |            | GET /temperature   |
          |   (Token 0x71)     |            |   (Token 0x71)     |
          +------------------->|            +------------------->|
          |                    |            |                    |
          |   ACK [0xbc90]     |            | (-------) [------] |
          |   2.05 Content     |            |   2.05 Content     |
          |   (Token 0x71)     |            |   (Token 0x71)     |
          |     "22.5 C"       |            |     "22.5 C"       |
          |<-------------------+            |<-------------------+
          |                    |            |                    |

              CoAP over UDP                   CoAP over reliable
                                                  transports
```

Figure 2: Comparison between CoAP over Unreliable Transports and CoAP over Reliable Transports

3.2. Message Format

The CoAP message format defined in [RFC7252], as shown in Figure 3, relies on the datagram transport (UDP, or DTLS over UDP) for keeping the individual messages separate and for providing length information.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |Ver| T |  TKL  |      Code     |          Message ID           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Token (if any, TKL bytes) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Options (if any) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |1 1 1 1 1 1 1 1|    Payload (if any) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: CoAP Message Format as Defined in RFC 7252

The message format for CoAP over TCP is very similar to the format specified for CoAP over UDP. The differences are as follows:

- o Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. The Type (T) and Message ID fields in the CoAP message header are elided.

- o The Version (Vers) field is elided as well. In contrast to the message format of CoAP over UDP, the message format for CoAP over TCP does not include a version number. CoAP is defined in [RFC7252] with a version number of 1. At this time, there is no known reason to support version numbers different from 1. If version negotiation needs to be addressed in the future, Capabilities and Settings Messages (CSMs) (see Section 5.3) have been specifically designed to enable such a potential feature.

o In a stream-oriented transport protocol such as TCP, a form of message delimitation is needed. For this purpose, CoAP over TCP introduces a length field with variable size. Figure 4 shows the adjusted CoAP message format with a modified structure for the fixed header (first 4 bytes of the header for CoAP over UDP), which includes the length information of variable size.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |  Len  |  TKL  | Extended Length (if any, as chosen by Len) ...
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |      Code     | Token (if any, TKL bytes) ...
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |  Options (if any) ...
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |1 1 1 1 1 1 1 1|    Payload (if any) ...
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: CoAP Frame for Reliable Transports

Length (Len): 4-bit unsigned integer. A value between 0 and 12 inclusive indicates the length of the message in bytes, starting with the first bit of the Options field. Three values are reserved for special constructs:

13: An 8-bit unsigned integer (Extended Length) follows the initial byte and indicates the length of options/payload minus 13.

14: A 16-bit unsigned integer (Extended Length) in network byte order follows the initial byte and indicates the length of options/payload minus 269.

15: A 32-bit unsigned integer (Extended Length) in network byte order follows the initial byte and indicates the length of options/payload minus 65805.

The encoding of the Length field is modeled after the Option Length field of the CoAP Options (see Section 3.1 of [RFC7252]).

For simplicity, a Payload Marker (0xFF) is shown in Figure 4; the Payload Marker indicates the start of the optional payload and is absent for zero-length payloads (see Section 3 of [RFC7252]). (If present, the Payload Marker is included in the message length, which counts from the start of the Options field to the end of the Payload field.)

For example, a CoAP message just containing a 2.03 code with the Token 7f and no options or payload is encoded as shown in Figure 5.

```
 0                   1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      0x01     |      0x43     |      0x7f     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 Len   =    0 ------>  0x01
 TKL   =    1 ___/
 Code  =  2.03     --> 0x43
 Token =               0x7f
```

Figure 5: CoAP Message with No Options or Payload

The semantics of the other CoAP header fields are left unchanged.

3.3. Message Transmission

Once a Transport Connection is established, each endpoint MUST send a CSM (see Section 5.3) as its first message on the connection. This message establishes the initial settings and capabilities for the endpoint, such as maximum message size or support for block-wise transfers. The absence of options in the CSM indicates that base values are assumed.

To avoid a deadlock, the Connection Initiator MUST NOT wait for the Connection Acceptor to send its initial CSM before sending its own initial CSM. Conversely, the Connection Acceptor MAY wait for the Connection Initiator to send its initial CSM before sending its own initial CSM.

To avoid unnecessary latency, a Connection Initiator MAY send additional messages after its initial CSM without waiting to receive the Connection Acceptor’s CSM; however, it is important to note that the Connection Acceptor’s CSM might indicate capabilities that impact how the Connection Initiator is expected to communicate with the Connection Acceptor. For example, the Connection Acceptor’s CSM could indicate a Max-Message-Size Option (see Section 5.3.1) that is smaller than the base value (1152) in order to limit both buffering requirements and head-of-line blocking.

Endpoints MUST treat a missing or invalid CSM as a connection error and abort the connection (see Section 5.6).

CoAP requests and responses are exchanged asynchronously over the Transport Connection. A CoAP client can send multiple requests without waiting for a response, and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Each concurrent request is differentiated by its Token, which is scoped locally to the connection.

The Transport Connection is bidirectional, so requests can be sent by both the entity that established the connection (Connection Initiator) and the remote host (Connection Acceptor). If one side does not implement a CoAP server, an error response MUST be returned for all CoAP requests from the other side. The simplest approach is to always return 5.01 (Not Implemented). A more elaborate mock server could also return 4.xx responses such as 4.04 (Not Found) or 4.02 (Bad Option) where appropriate.

Retransmission and deduplication of messages are provided by TCP.

3.4. Connection Health

Empty messages (Code 0.00) can always be sent and MUST be ignored by the recipient. This provides a basic keepalive function that can refresh NAT bindings.

If a CoAP client does not receive any response for some time after sending a CoAP request (or, similarly, when a client observes a resource and it does not receive any notification for some time), it can send a CoAP Ping Signaling message (see Section 5.4) to test the Transport Connection and verify that the CoAP server is responsive.

When the underlying Transport Connection is closed or reset, the signaling state and any observation state (see Section 7.4) associated with the connection are removed. Messages that are in flight may or may not be lost.

4. CoAP over WebSockets

CoAP over WebSockets is intentionally similar to CoAP over TCP; therefore, this section only specifies the differences between the transports.

CoAP over WebSockets can be used in a number of configurations. The most basic configuration is a CoAP client retrieving or updating a CoAP resource located on a CoAP server that exposes a WebSocket endpoint (see Figure 6). The CoAP client acts as the WebSocket client, establishes a WebSocket connection, and sends a CoAP request, to which the CoAP server returns a CoAP response. The WebSocket connection can be used for any number of requests.

```
            ___________                            ___________
           |           |                          |           |
           |          _|___      requests      ___|_          |
           |   CoAP  /  \  \  ------------->  /  /  \  CoAP   |
           |  Client \__/__/  <-------------  \__\__/ Server  |
           |           |         responses        |           |
           |___________|                          |___________|
                   WebSocket  =============>  WebSocket
                     Client     Connection     Server
```

Figure 6: CoAP Client (WebSocket Client) Accesses CoAP Server (WebSocket Server)

The challenge with this configuration is how to identify a resource in the namespace of the CoAP server. When the WebSocket Protocol is used by a dedicated client directly (i.e., not from a web page through a web browser), the client can connect to any WebSocket endpoint. Sections 8.3 and 8.4 define new URI schemes that enable the client to identify both a WebSocket endpoint and the path and query of the CoAP resource within that endpoint.

Another possible configuration is to set up a CoAP forward proxy at the WebSocket endpoint. Depending on what transports are available to the proxy, it could forward the request to a CoAP server with a CoAP UDP endpoint (Figure 7), an SMS endpoint (a.k.a. mobile phone), or even another WebSocket endpoint. The CoAP client specifies the resource to be updated or retrieved in the Proxy-Uri Option.

```
     ___________               ___________               ___________
    |           |              |           |              |           |
    |          _|___        ___|_         _|___        ___|_          |
    |   CoAP  /  \  \ ---> /  /  \ CoAP  /  \  \ ---> /  /  \  CoAP   |
    |  Client \__/__/ <--- \__\__/ Proxy \__/__/ <--- \__\__/ Server  |
    |           |              |           |              |           |
    |___________|              |___________|              |___________|
            WebSocket ===> WebSocket      UDP            UDP
              Client        Server      Client          Server
```

Figure 7: CoAP Client (WebSocket Client) Accesses CoAP Server (UDP Server) via a CoAP Proxy (WebSocket Server / UDP Client)

A third possible configuration is a CoAP server running inside a web browser (Figure 8). The web browser initially connects to a WebSocket endpoint and is then reachable through the WebSocket server. When no connection exists, the CoAP server is unreachable. Because the WebSocket server is the only way to reach the CoAP server, the CoAP proxy should be a reverse-proxy.

```
     ___________               ___________               ___________
    |           |              |           |              |           |
    |          _|___        ___|_         _|___        ___|_          |
    |   CoAP  /  \  \ ---> /  /  \ CoAP  /  /  \ ---> /  \  \  CoAP   |
    |  Client \__/__/ <--- \__\__/ Proxy \__\__/ <--- \__/__/ Server  |
    |           |              |           |              |           |
    |___________|              |___________|              |___________|
               UDP            UDP      WebSocket <=== WebSocket
             Client          Server      Server        Client
```

Figure 8: CoAP Client (UDP Client) Accesses CoAP Server (WebSocket Client) via a CoAP Proxy (UDP Server / WebSocket Server)

Further configurations are possible, including those where a WebSocket connection is established through an HTTP proxy.

4.1. Opening Handshake

Before CoAP requests and responses are exchanged, a WebSocket connection is established as defined in Section 4 of [RFC6455]. Figure 9 shows an example.

The WebSocket client MUST include the subprotocol name "coap" in the list of protocols; this indicates support for the protocol defined in this document.

The WebSocket client includes the hostname of the WebSocket server in the Host header field of its handshake as per [RFC6455]. The Host header field also indicates the default value of the Uri-Host Option in requests from the WebSocket client to the WebSocket server.

```
GET /.well-known/coap HTTP/1.1
Host: example.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: coap
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: coap
```

Figure 9: Example of an Opening Handshake

4.2. Message Format

Once a WebSocket connection is established, CoAP requests and responses can be exchanged as WebSocket messages. Since CoAP uses a binary message format, the messages are transmitted in binary data frames as specified in Sections 5 and 6 of [RFC6455].

The message format shown in Figure 10 is the same as the message format for CoAP over TCP (see Section 3.2), with one change: the Length (Len) field MUST be set to zero, because the WebSocket frame contains the length.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 | Len=0 |  TKL  |      Code     |    Token (TKL bytes) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Options (if any) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |1 1 1 1 1 1 1 1|    Payload (if any) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 10: CoAP Message Format over WebSockets

As with CoAP over TCP, the message format for CoAP over WebSockets eliminates the Version field defined in CoAP over UDP. If CoAP version negotiation is required in the future, CoAP over WebSockets can address the requirement by defining a new subprotocol identifier that is negotiated during the opening handshake.

Requests and responses can be fragmented as specified in Section 5.4 of [RFC6455], though typically they are sent unfragmented, as they tend to be small and fully buffered before transmission. The WebSocket Protocol does not provide means for multiplexing. If it is not desirable for a large message to monopolize the connection, requests and responses can be transferred in a block-wise fashion as defined in [RFC7959].

4.3. Message Transmission

As with CoAP over TCP, each endpoint MUST send a CSM (see Section 5.3) as its first message on the WebSocket connection.

CoAP requests and responses are exchanged asynchronously over the WebSocket connection. A CoAP client can send multiple requests without waiting for a response, and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Each concurrent request is differentiated by its Token, which is scoped locally to the connection.

The connection is bidirectional, so requests can be sent by both the entity that established the connection and the remote host.

As with CoAP over TCP, retransmission and deduplication of messages are provided by the WebSocket Protocol. CoAP over WebSockets therefore does not make a distinction between Confirmable messages and Non-confirmable messages and does not provide Acknowledgment or Reset messages.

4.4. Connection Health

As with CoAP over TCP, a CoAP client can test the health of the connection for CoAP over WebSockets by sending a CoAP Ping Signaling message (Section 5.4). To ensure that redundant maintenance traffic is not transmitted, WebSocket Ping and unsolicited Pong frames (Section 5.5 of [RFC6455]) SHOULD NOT be used.

5. Signaling

Signaling messages are specifically introduced only for CoAP over reliable transports to allow peers to:

- Learn related characteristics, such as maximum message size for the connection.
- Shut down the connection in an orderly fashion.
- Provide diagnostic information when terminating a connection in response to a serious error condition.

Signaling is a third basic kind of message in CoAP, after requests and responses. Signaling messages share a common structure with the existing CoAP messages. There are a code, a Token, options, and an optional payload.

(See Section 3 of [RFC7252] for the overall structure of the message format, option format, and option value formats.)

5.1. Signaling Codes

A code in the 7.00-7.31 range indicates a Signaling message. Values in this range are assigned by the "CoAP Signaling Codes" subregistry (see Section 11.1).

For each message, there are a sender and a peer receiving the message.

Payloads in Signaling messages are diagnostic payloads as defined in Section 5.5.2 of [RFC7252], unless otherwise defined by a Signaling message option.

5.2. Signaling Option Numbers

Option Numbers for Signaling messages are specific to the message code. They do not share the number space with CoAP options for request/response messages or with Signaling messages using other codes.

Option Numbers are assigned by the "CoAP Signaling Option Numbers" subregistry (see Section 11.2).

Signaling Options are elective or critical as defined in Section 5.4.1 of [RFC7252]. If a Signaling Option is critical and not understood by the receiver, it MUST abort the connection (see Section 5.6). If the option is understood but cannot be processed, the option documents the behavior.

5.3. Capabilities and Settings Messages (CSMs)

CSMs are used for two purposes:

- o Each capability option indicates one capability of the sender to the recipient.
- o Each setting option indicates a setting that will be applied by the sender.

One CSM MUST be sent by each endpoint at the start of the Transport Connection. Additional CSMs MAY be sent at any other time by either endpoint over the lifetime of the connection.

Both capability options and setting options are cumulative. A CSM does not invalidate a previously sent capability indication or setting even if it is not repeated. A capability message without any option is a no-operation (and can be used as such). An option that is sent might override a previous value for the same option. The option defines how to handle this case if needed.

Base values are listed below for CSM options. These are the values for the capability and settings before any CSMs send a modified value.

These are not default values (as defined in Section 5.4.4 in [RFC7252]) for the option. Default values apply on a per-message basis and are thus reset when the value is not present in a given CSM.

CSMs are indicated by the 7.01 (CSM) code; see Table 1 (Section 11.1).

5.3.1. Max-Message-Size Capability Option

The sender can use the elective Max-Message-Size Option to indicate the maximum size of a message in bytes that it can receive. The message size indicated includes the entire message, starting from the first byte of the message header and ending at the end of the message payload.

(Note that there is no relationship of the message size to the overall request or response body size that may be achievable in block-wise transfer. For example, the exchange depicted in Figure 13 (Section 6.1) can be performed if the CoAP client indicates a value of around 6000 bytes for the Max-Message-Size Option, even though the total body size transferred to the client is 3072 + 5120 + 4711 = 12903 bytes.)

#	C	R	Applies to	Name	Format	Length	Base Value
2			CSM	Max-Message-Size	uint	0-4	1152

C=Critical, R=Repeatable

As per Section 4.6 of [RFC7252], the base value (and the value used when this option is not implemented) is 1152.

The active value of the Max-Message-Size Option is replaced each time the option is sent with a modified value. Its starting value is its base value.

5.3.2. Block-Wise-Transfer Capability Option

#	C	R	Applies to	Name	Format	Length	Base Value
4			CSM	Block-Wise-Transfer	empty	0	(none)

C=Critical, R=Repeatable

A sender can use the elective Block-Wise-Transfer Option to indicate that it supports the block-wise transfer protocol [RFC7959].

If the option is not given, the peer has no information about whether block-wise transfers are supported by the sender or not. An implementation wishing to offer block-wise transfers to its peer therefore needs to indicate so via the Block-Wise-Transfer Option.

If a Max-Message-Size Option is indicated with a value that is greater than 1152 (in the same CSM or a different CSM), the Block-Wise-Transfer Option also indicates support for BERT (see Section 6). Subsequently, if the Max-Message-Size Option is indicated with a value equal to or less than 1152, BERT support is no longer indicated. (Note that the indication of BERT support does not oblige either peer to actually choose to make use of BERT.)

Implementation note: When indicating a value of the Max-Message-Size Option with an intention to enable BERT, the indicating implementation may want to (1) choose a particular BERT block size it wants to encourage and (2) add a delta for the header and any options that may also need to be included in the message with a BERT block of that size. Section 4.6 of [RFC7252] adds 128 bytes to a maximum block size of 1024 to arrive at a default message size of 1152. A BERT-enabled implementation may want to indicate a BERT block size of 2048 or a higher multiple of 1024 and at the same time be more generous with the size of the header and options added (say, 256 or 512). However, adding 1024 or more to the base BERT block size may encourage the peer implementation to vary the BERT block size based on the size of the options included; this type of scenario might make it harder to establish interoperability.

5.4. Ping and Pong Messages

In CoAP over reliable transports, Empty messages (Code 0.00) can always be sent and MUST be ignored by the recipient. This provides a basic keepalive function. In contrast, Ping and Pong messages are a bidirectional exchange.

Upon receipt of a Ping message, the receiver MUST return a Pong message with an identical Token in response. Unless the Ping carries an option with delaying semantics such as the Custody Option, it SHOULD respond as soon as practical. As with all Signaling messages, the recipient of a Ping or Pong message MUST ignore elective options it does not understand.

Ping and Pong messages are indicated by the 7.02 code (Ping) and the 7.03 code (Pong).

Note that, as with similar mechanisms defined in [RFC6455] and [RFC7540], the present specification does not define any specific maximum time that the sender of a Ping message has to allow when waiting for a Pong reply. Any limitations on patience for this reply are a matter of the application making use of these messages, as is any approach to recover from a failure to respond in time.

5.4.1. Custody Option

#	C	R	Applies to	Name	Format	Length	Base Value
2			Ping, Pong	Custody	empty	0	(none)

C=Critical, R=Repeatable

When responding to a Ping message, the receiver can include an elective Custody Option in the Pong message. This option indicates that the application has processed all the request/response messages received prior to the Ping message on the current connection. (Note that there is no definition of specific application semantics for "processed", but there is an expectation that the receiver of a Pong message with a Custody Option should be able to free buffers based on this indication.)

A sender can also include an elective Custody Option in a Ping message to explicitly request the inclusion of an elective Custody Option in the corresponding Pong message. In that case, the receiver SHOULD delay its Pong message until it finishes processing all the request/response messages received prior to the Ping message on the current connection.

5.5. Release Messages

A Release message indicates that the sender does not want to continue maintaining the Transport Connection and opts for an orderly shutdown, but wants to leave it to the peer to actually start closing the connection. The details are in the options. A diagnostic payload (see Section 5.5.2 of [RFC7252]) MAY be included.

A peer will normally respond to a Release message by closing the Transport Connection. (In case that does not happen, the sender of the release may want to implement a timeout mechanism if getting rid of the connection is actually important to it.)

Messages may be in flight or responses outstanding when the sender decides to send a Release message (which is one reason the sender had decided to wait before closing the connection). The peer responding to the Release message SHOULD delay the closing of the connection until it has responded to all requests received by it before the Release message. It also MAY wait for the responses to its own requests.

It is NOT RECOMMENDED for the sender of a Release message to continue sending requests on the connection it already indicated to be released: the peer might close the connection at any time and miss those requests. The peer is not obligated to check for this condition, though.

Release messages are indicated by the 7.04 code (Release).

Release messages can indicate one or more reasons using elective options. The following options are defined:

#	C	R	Applies to	Name	Format	Length	Base Value
2		x	Release	Alternative-Address	string	1-255	(none)

C=Critical, R=Repeatable

The elective Alternative-Address Option requests the peer to instead open a connection of the same scheme as the present connection to the alternative transport address given. Its value is in the form "authority" as defined in Section 3.2 of [RFC3986]. (Existing state related to the connection is not transferred from the present connection to the new connection.)

The Alternative-Address Option is a repeatable option as defined in Section 5.4.5 of [RFC7252]. When multiple occurrences of the option are included, the peer can choose any of the alternative transport addresses.

#	C	R	Applies to	Name	Format	Length	Base Value
4			Release	Hold-Off	uint	0-3	(none)

C=Critical, R=Repeatable

The elective Hold-Off Option indicates that the server is requesting that the peer not reconnect to it for the number of seconds given in the value.

5.6. Abort Messages

An Abort message indicates that the sender is unable to continue maintaining the Transport Connection and cannot even wait for an orderly release. The sender shuts down the connection immediately after the Abort message (and may or may not wait for a Release message, Abort message, or connection shutdown in the inverse direction). A diagnostic payload (see Section 5.5.2 of [RFC7252]) SHOULD be included in the Abort message. Messages may be in flight or responses outstanding when the sender decides to send an Abort message. The general expectation is that these will NOT be processed.

Abort messages are indicated by the 7.05 code (Abort).

Abort messages can indicate one or more reasons using elective options. The following option is defined:

#	C	R	Applies to	Name	Format	Length	Base Value
2			Abort	Bad-CSM-Option	uint	0-2	(none)

C=Critical, R=Repeatable

Bad-CSM-Option, which is elective, indicates that the sender is unable to process the CSM option identified by its Option Number, e.g., when it is critical and the Option Number is unknown by the sender, or when there is a parameter problem with the value of an elective option. More detailed information SHOULD be included as a diagnostic payload.

For CoAP over UDP, messages that contain syntax violations are processed as message format errors. As described in Sections 4.2 and 4.3 of [RFC7252], such messages are rejected by sending a matching Reset message and otherwise ignoring the message.

For CoAP over reliable transports, the recipient rejects such messages by sending an Abort message and otherwise ignoring (not processing) the message. No specific Option has been defined for the Abort message in this case, as the details are best left to a diagnostic payload.

5.7. Signaling Examples

An encoded example of a Ping message with a non-empty Token is shown in Figure 11.

```
      0                   1                   2
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |      0x01     |      0xe2     |      0x42     |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

      Len   =    0 -------> 0x01
      TKL   =    1 ___/
      Code  = 7.02 Ping --> 0xe2
      Token =               0x42
```

Figure 11: Ping Message Example

An encoded example of the corresponding Pong message is shown in Figure 12.

```
    0                   1                   2
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      0x01     |      0xe3     |      0x42     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

    Len   =    0 -------> 0x01
    TKL   =    1 ___/
    Code  = 7.03 Pong --> 0xe3
    Token =               0x42
```

Figure 12: Pong Message Example

6. Block-Wise Transfer and Reliable Transports

The message size restrictions defined in Section 4.6 of [RFC7252] to avoid IP fragmentation are not necessary when CoAP is used over a reliable transport. While this suggests that the block-wise transfer protocol [RFC7959] is also no longer needed, it remains applicable for a number of cases:

- Large messages, such as firmware downloads, may cause undesired head-of-line blocking when a single transport connection is used.

- A UDP-to-TCP gateway may simply not have the context to convert a message with a Block Option into the equivalent exchange without any use of a Block Option (it would need to convert the entire block-wise exchange from start to end into a single exchange).

BERT extends the block-wise transfer protocol to enable the use of larger messages over a reliable transport.

The use of this new extension is signaled by sending Block1 or Block2 Options with SZX == 7 (a "BERT Option"). SZX == 7 is a reserved value in [RFC7959].

In control usage, a BERT Option is interpreted in the same way as the equivalent Option with SZX == 6, except that it also indicates the capability to process BERT blocks. As with the basic block-wise transfer protocol, the recipient of a CoAP request with a BERT Option in control usage is allowed to respond with a different SZX value, e.g., to send a non-BERT block instead.

In descriptive usage, a BERT Option is interpreted in the same way as the equivalent Option with SZX == 6, except that the payload is also allowed to contain multiple blocks. For non-final BERT blocks, the payload is always a multiple of 1024 bytes. For final BERT blocks, the payload is a multiple (possibly 0) of 1024 bytes plus a partial block of less than 1024 bytes.

The recipient of a non-final BERT block (M=1) conceptually partitions the payload into a sequence of 1024-byte blocks and acts exactly as if it had received this sequence in conjunction with block numbers starting at, and sequentially increasing from, the block number given in the Block Option. In other words, the entire BERT block is positioned at the byte position that results from multiplying the block number by 1024. The position of further blocks to be transferred is indicated by incrementing the block number by the number of elements in this sequence (i.e., the size of the payload divided by 1024 bytes).

As with SZX == 6, the recipient of a final BERT block (M=0) simply appends the payload at the byte position that is indicated by the block number multiplied by 1024.

The following examples illustrate BERT Options. A value of SZX == 7 is labeled as "BERT" or as "BERT(nnn)" to indicate a payload of size nnn.

In all these examples, a Block Option is decomposed to indicate the kind of Block Option (1 or 2) followed by a colon, the block number (NUM), the more bit (M), and the block size (2**(SZX + 4)) separated by slashes. For example, a Block2 Option value of 33 would be shown as 2:2/0/32), or a Block1 Option value of 59 would be shown as 1:3/1/128.

6.1. Example: GET with BERT Blocks

Figure 13 shows a GET request with a response that is split into three BERT blocks. The first response contains 3072 bytes of payload; the second, 5120; and the third, 4711. Note how the block number increments to move the position inside the response body forward.

```
   CoAP Client                             CoAP Server
      |                                        |
      | GET, /status                    ------> |
      |                                        |
      | <------   2.05 Content, 2:0/1/BERT(3072)   |
      |                                        |
      | GET, /status, 2:3/0/BERT        ------> |
      |                                        |
      | <------   2.05 Content, 2:3/1/BERT(5120)   |
      |                                        |
      | GET, /status, 2:8/0/BERT       ------>  |
      |                                        |
      | <------   2.05 Content, 2:8/0/BERT(4711)   |
```

Figure 13: GET with BERT Blocks

6.2. Example: PUT with BERT Blocks

Figure 14 demonstrates a PUT exchange with BERT blocks.

```
   CoAP Client                             CoAP Server
      |                                        |
      | PUT, /options, 1:0/1/BERT(8192)     ------> |
      |                                        |
      | <------   2.31 Continue, 1:0/1/BERT         |
      |                                        |
      | PUT, /options, 1:8/1/BERT(16384)    ------> |
      |                                        |
      | <------   2.31 Continue, 1:8/1/BERT         |
      |                                        |
      | PUT, /options, 1:24/0/BERT(5683)    ------> |
      |                                        |
      | <------   2.04 Changed, 1:24/0/BERT         |
      |                                        |
```

Figure 14: PUT with BERT Blocks

7. Observing Resources over Reliable Transports

This section describes how the procedures defined in [RFC7641] for observing resources over CoAP are applied (and modified, as needed) for reliable transports. In this section, "client" and "server" refer to the CoAP client and CoAP server.

7.1. Notifications and Reordering

When using the Observe Option [RFC7641] with CoAP over UDP, notifications from the server set the option value to an increasing sequence number for reordering detection on the client, since messages can arrive in a different order than they were sent. This sequence number is not required for CoAP over reliable transports, since TCP ensures reliable and ordered delivery of messages. The value of the Observe Option in 2.xx notifications MAY be empty on transmission and MUST be ignored on reception.

Implementation note: This means that a proxy from a reordering transport to a reliable (in-order) transport (such as a UDP-to-TCP proxy) needs to process the Observe Option in notifications according to the rules in Section 3.4 of [RFC7641].

7.2. Transmission and Acknowledgments

For CoAP over UDP, server notifications to the client can be Confirmable or Non-confirmable. A Confirmable message requires the client to respond with either an Acknowledgment message or a Reset message. An Acknowledgment message indicates that the client is alive and wishes to receive further notifications. A Reset message indicates that the client does not recognize the Token; this causes the server to remove the associated entry from the list of observers.

Since TCP eliminates the need for the message layer to support reliability, CoAP over reliable transports does not support Confirmable or Non-confirmable message types. All notifications are delivered reliably to the client with positive acknowledgment of receipt occurring at the TCP level. If the client does not recognize the Token in a notification, it MAY immediately abort the connection (see Section 5.6).

7.3. Freshness

For CoAP over UDP, if a client does not receive a notification for some time, it can send a new GET request with the same Token as the original request to re-register its interest in a resource and verify that the server is still responsive. For CoAP over reliable transports, it is more efficient to check the health of the connection (and all its active observations) by sending a single CoAP Ping Signaling message (Section 5.4) rather than individual requests to confirm each active observation. (Note that such a Ping/Pong only confirms a single hop: a proxy is not obligated or expected to react to a Ping by checking all its own registered interests or all the connections, if any, underlying them. A proxy MAY maintain its own schedule for confirming the interests that it relies on being registered toward the origin server; however, it is generally inadvisable for a proxy to generate a large number of outgoing checks based on a single incoming check.)

7.4. Cancellation

For CoAP over UDP, a client that is no longer interested in receiving notifications can "forget" the observation and respond to the next notification from the server with a Reset message to cancel the observation.

For CoAP over reliable transports, a client MUST explicitly deregister by issuing a GET request that has the Token field set to the Token of the observation to be canceled and includes an Observe Option with the value set to 1 (deregister).

If the client observes one or more resources over a reliable transport, then the CoAP server (or intermediary in the role of the CoAP server) MUST remove all entries associated with the client endpoint from the lists of observers when the connection either times out or is closed.

8. CoAP over Reliable Transport URIs

CoAP over UDP [RFC7252] defines the "coap" and "coaps" URI schemes. This document introduces four additional URI schemes for identifying CoAP resources and providing a means of locating the resource:

o The "coap+tcp" URI scheme for CoAP over TCP.

o The "coaps+tcp" URI scheme for CoAP over TCP secured by TLS.

o The "coap+ws" URI scheme for CoAP over WebSockets.

o The "coaps+ws" URI scheme for CoAP over WebSockets secured by TLS.

Resources made available via these schemes have no shared identity even if their resource identifiers indicate the same authority (the same host listening to the same TCP port). They are hosted in distinct namespaces because each URI scheme implies a distinct origin server.

In this section, the syntax for the URI schemes is specified using the Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", and "query" are adopted from [RFC3986].

Section 8 ("Multicast CoAP") in [RFC7252] is not applicable to these schemes.

As with the "coap" and "coaps" schemes defined in [RFC7252], all URI schemes defined in this section also support the path prefix "/.well-known/" as defined by [RFC5785] for "well-known locations" in the namespace of a host. This enables discovery as per Section 7 of [RFC7252].

8.1. coap+tcp URI Scheme

The "coap+tcp" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over TCP.

```
  coap-tcp-URI = "coap+tcp:" "//" host [ ":" port ]
    path-abempty [ "?" query ]
```

The syntax defined in Section 6.1 of [RFC7252] applies to this URI scheme, with the following change:

- o The port subcomponent indicates the TCP port at which the CoAP Connection Acceptor is located. (If it is empty or not given, then the default port 5683 is assumed, as with UDP.)

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.2. coaps+tcp URI Scheme

The "coaps+tcp" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over TCP secured with TLS.

```
  coaps-tcp-URI = "coaps+tcp:" "//" host [ ":" port ]
    path-abempty [ "?" query ]
```

The syntax defined in Section 6.2 of [RFC7252] applies to this URI scheme, with the following changes:

- The port subcomponent indicates the TCP port at which the TLS server for the CoAP Connection Acceptor is located. If it is empty or not given, then the default port 5684 is assumed.

- If a TLS server does not support the Application-Layer Protocol Negotiation (ALPN) extension [RFC7301] or wishes to accommodate TLS clients that do not support ALPN, it MAY offer a coaps+tcp endpoint on TCP port 5684. This endpoint MAY also be ALPN enabled. A TLS server MAY offer coaps+tcp endpoints on ports other than TCP port 5684, which MUST be ALPN enabled.

- For TCP ports other than port 5684, the TLS client MUST use the ALPN extension to advertise the "coap" protocol identifier (see Section 11.7) in the list of protocols in its ClientHello. If the TCP server selects and returns the "coap" protocol identifier using the ALPN extension in its ServerHello, then the connection succeeds. If the TLS server either does not negotiate the ALPN extension or returns a no_application_protocol alert, the TLS client MUST close the connection.

- For TCP port 5684, a TLS client MAY use the ALPN extension to advertise the "coap" protocol identifier in the list of protocols in its ClientHello. If the TLS server selects and returns the "coap" protocol identifier using the ALPN extension in its ServerHello, then the connection succeeds. If the TLS server returns a no_application_protocol alert, then the TLS client MUST close the connection. If the TLS server does not negotiate the ALPN extension, then coaps+tcp is implicitly selected.

- For TCP port 5684, if the TLS client does not use the ALPN extension to negotiate the protocol, then coaps+tcp is implicitly selected.

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.3. coap+ws URI Scheme

The "coap+ws" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over WebSockets.

```
  coap-ws-URI = "coap+ws:" "//" host [ ":" port ]
    path-abempty [ "?" query ]
```

The port subcomponent is OPTIONAL. The default is port 80.

The WebSocket endpoint is identified by a "ws" URI that is composed of the authority part of the "coap+ws" URI and the well-known path "/.well-known/coap" [RFC5785] [RFC8307]. Within the endpoint specified in a "coap+ws" URI, the path and query parts of the URI identify a resource that can be operated on by the methods defined by CoAP:

```
          coap+ws://example.org/sensors/temperature?u=Cel
               \______  ______/\___________  ___________/
                      \/                   \/
                                         Uri-Path: "sensors"
    ws://example.org/.well-known/coap    Uri-Path: "temperature"
                                         Uri-Query: "u=Cel"
```

Figure 15: The "coap+ws" URI Scheme

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.4. coaps+ws URI Scheme

The "coaps+ws" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over WebSockets secured by TLS.

```
     coaps-ws-URI = "coaps+ws:" "//" host [ ":" port ]
       path-abempty [ "?" query ]
```

The port subcomponent is OPTIONAL. The default is port 443.

The WebSocket endpoint is identified by a "wss" URI that is composed of the authority part of the "coaps+ws" URI and the well-known path "/.well-known/coap" [RFC5785] [RFC8307]. Within the endpoint specified in a "coaps+ws" URI, the path and query parts of the URI identify a resource that can be operated on by the methods defined by CoAP:

```
             coaps+ws://example.org/sensors/temperature?u=Cel
                   \______  ______/\___________  ___________/
                          \/                   \/
                                             Uri-Path: "sensors"
       wss://example.org/.well-known/coap   Uri-Path: "temperature"
                                             Uri-Query: "u=Cel"
```

Figure 16: The "coaps+ws" URI Scheme

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.5. Uri-Host and Uri-Port Options

CoAP over reliable transports maintains the property from Section 5.10.1 of [RFC7252]:

> The default values for the Uri-Host and Uri-Port Options are sufficient for requests to most servers.

Unless otherwise noted, the default value of the Uri-Host Option is the IP literal representing the destination IP address of the request message. The default value of the Uri-Port Option is the destination TCP port.

For CoAP over TLS, these default values are the same, unless Server Name Indication (SNI) [RFC6066] is negotiated. In this case, the default value of the Uri-Host Option in requests from the TLS client to the TLS server is the SNI host.

For CoAP over WebSockets, the default value of the Uri-Host Option in requests from the WebSocket client to the WebSocket server is indicated by the Host header field from the WebSocket handshake.

8.6. Decomposing URIs into Options

The steps are the same as those specified in Section 6.4 of [RFC7252], with minor changes:

This step from [RFC7252]:

3. If |url| does not have a <scheme> component whose value, when converted to ASCII lowercase, is "coap" or "coaps", then fail this algorithm.

is updated to:

3. If |url| does not have a <scheme> component whose value, when converted to ASCII lowercase, is "coap+tcp", "coaps+tcp", "coap+ws", or "coaps+ws", then fail this algorithm.

This step from [RFC7252]:

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

is updated to:

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

8.7. Composing URIs from Options

The steps are the same as those specified in Section 6.5 of [RFC7252], with minor changes:

This step from [RFC7252]:

1. If the request is secured using DTLS, let |url| be the string "coaps://". Otherwise, let |url| be the string "coap://".

is updated to:

1. For CoAP over TCP, if the request is secured using TLS, let |url| be the string "coaps+tcp://". Otherwise, let |url| be the string "coap+tcp://". For CoAP over WebSockets, if the request is secured using TLS, let |url| be the string "coaps+ws://". Otherwise, let |url| be the string "coap+ws://".

This step from [RFC7252]:

4. If the request includes a Uri-Port Option, let |port| be that option’s value. Otherwise, let |port| be the request’s destination UDP port.

is updated to:

4. If the request includes a Uri-Port Option, let |port| be that option’s value. Otherwise, let |port| be the request’s destination TCP port.

9. Securing CoAP

"Security Challenges For the Internet Of Things" [SecurityChallenges] recommends the following:

> ... it is essential that IoT protocol suites specify a mandatory to implement but optional to use security solution. This will ensure security is available in all implementations, but configurable to use when not necessary (e.g., in closed environment). ... even if those features stretch the capabilities of such devices.

A security solution MUST be implemented to protect CoAP over reliable transports and MUST be enabled by default. This document defines the TLS binding, but alternative solutions at different layers in the protocol stack MAY be used to protect CoAP over reliable transports when appropriate. Note that there is ongoing work to support a data-object-based security model for CoAP that is independent of transport (see [OSCORE]).

9.1. TLS Binding for CoAP over TCP

The TLS usage guidance in [RFC7925] applies, including the guidance about cipher suites in that document that are derived from the mandatory-to-implement cipher suites defined in [RFC7252].

This guidance assumes implementation in a constrained device or for communication with a constrained device. However, CoAP over TCP/TLS has a wider applicability. It may, for example, be implemented on a gateway or on a device that is less constrained (such as a smart phone or a tablet), for communication with a peer that is likewise less constrained, or within a back-end environment that only communicates with constrained devices via proxies. As an exception to the previous paragraph, in this case, the recommendations in [RFC7525] are more appropriate.

Since the guidance offered in [RFC7925] differs from the guidance offered in [RFC7525] in terms of algorithms and credential types, it is assumed that an implementation of CoAP over TCP/TLS that needs to support both cases implements the recommendations offered by both specifications.

During the provisioning phase, a CoAP device is provided with the security information that it needs, including keying materials, access control lists, and authorization servers. At the end of the provisioning phase, the device will be in one of four security modes:

NoSec: TLS is disabled.

PreSharedKey: TLS is enabled. The guidance in Section 4.2 of [RFC7925] applies.

RawPublicKey: TLS is enabled. The guidance in Section 4.3 of [RFC7925] applies.

Certificate: TLS is enabled. The guidance in Section 4.4 of [RFC7925] applies.

The "NoSec" mode is optional to implement. The system simply sends the packets over normal TCP; this is indicated by the "coap+tcp" scheme and the TCP CoAP default port. The system is secured only by keeping attackers from being able to send or receive packets from the network with the CoAP nodes.

"PreSharedKey", "RawPublicKey", or "Certificate" is mandatory to implement for the TLS binding, depending on the credential type used with the device. These security modes are achieved using TLS and are indicated by the "coaps+tcp" scheme and TLS-secured CoAP default port.

9.2. TLS Usage for CoAP over WebSockets

A CoAP client requesting a resource identified by a "coaps+ws" URI negotiates a secure WebSocket connection to a WebSocket server endpoint with a "wss" URI. This is described in Section 8.4.

The client MUST perform a TLS handshake after opening the connection to the server. The guidance in Section 4.1 of [RFC6455] applies. When a CoAP server exposes resources identified by a "coaps+ws" URI, the guidance in Section 4.4 of [RFC7925] applies towards mandatory-to-implement TLS functionality for certificates. For the server-side requirements for accepting incoming connections over an HTTPS (HTTP over TLS) port, the guidance in Section 4.2 of [RFC6455] applies.

Note that the guidance above formally inherits the mandatory-to-implement cipher suites defined in [RFC5246]. However, modern browsers usually implement cipher suites that are more recent; these cipher suites are then automatically picked up via the JavaScript WebSocket API. WebSocket servers that provide secure CoAP over WebSockets for the browser use case will need to follow the browser preferences and MUST follow [RFC7525].

10. Security Considerations

The security considerations of [RFC7252] apply. For CoAP over WebSockets and CoAP over TLS-secured WebSockets, the security considerations of [RFC6455] also apply.

10.1. Signaling Messages

The guidance given by an Alternative-Address Option cannot be followed blindly. In particular, a peer MUST NOT assume that a successful connection to the Alternative-Address inherits all the security properties of the current connection.

11. IANA Considerations

11.1. Signaling Codes

IANA has created a third subregistry for values of the Code field in the CoAP header (Section 12.1 of [RFC7252]). The name of this subregistry is "CoAP Signaling Codes".

Each entry in the subregistry must include the Signaling Code in the range 7.00-7.31, its name, and a reference to its documentation.

Initial entries in this subregistry are as follows:

Code	Name	Reference
7.01	CSM	RFC 8323
7.02	Ping	RFC 8323
7.03	Pong	RFC 8323
7.04	Release	RFC 8323
7.05	Abort	RFC 8323

Table 1: CoAP Signaling Codes

All other Signaling Codes are Unassigned.

The IANA policy for future additions to this subregistry is "IETF Review" or "IESG Approval" as described in [RFC8126].

11.2. CoAP Signaling Option Numbers Registry

IANA has created a subregistry for Option Numbers used in CoAP Signaling Options within the "Constrained RESTful Environments (CoRE) Parameters" registry. The name of this subregistry is "CoAP Signaling Option Numbers".

Each entry in the subregistry must include one or more of the codes in the "CoAP Signaling Codes" subregistry (Section 11.1), the number for the Option, the name of the Option, and a reference to the Option’s documentation.

Initial entries in this subregistry are as follows:

Applies to	Number	Name	Reference
7.01	2	Max-Message-Size	RFC 8323
7.01	4	Block-Wise-Transfer	RFC 8323
7.02, 7.03	2	Custody	RFC 8323
7.04	2	Alternative-Address	RFC 8323
7.04	4	Hold-Off	RFC 8323
7.05	2	Bad-CSM-Option	RFC 8323

Table 2: CoAP Signaling Option Codes

The IANA policy for future additions to this subregistry is based on number ranges for the option numbers, analogous to the policy defined in Section 12.2 of [RFC7252]. (The policy is analogous rather than identical because the structure of this subregistry includes an additional column ("Applies to"); however, the value of this column has no influence on the policy.)

The documentation for a Signaling Option Number should specify the semantics of an option with that number, including the following properties:

o Whether the option is critical or elective, as determined by the Option Number.

o Whether the option is repeatable.

o The format and length of the option’s value.

o The base value for the option, if any.

11.3. Service Name and Port Number Registration

IANA has assigned the port number 5683 and the service name "coap", in accordance with [RFC6335].

Service Name:
coap

Transport Protocol:
tcp

Assignee:
IESG <iesg@ietf.org>

Contact:
IETF Chair <chair@ietf.org>

Description:
Constrained Application Protocol (CoAP)

Reference:
RFC 8323

Port Number:
5683

11.4. Secure Service Name and Port Number Registration

IANA has assigned the port number 5684 and the service name "coaps", in accordance with [RFC6335]. The port number is to address the exceptional case of TLS implementations that do not support the ALPN extension [RFC7301].

Service Name:
coaps

Transport Protocol:
tcp

Assignee:
IESG <iesg@ietf.org>

Contact:
IETF Chair <chair@ietf.org>

Description:
Constrained Application Protocol (CoAP)

Reference:
[RFC7301], RFC 8323

Port Number:
5684

11.5. URI Scheme Registration

URI schemes are registered within the "Uniform Resource Identifier (URI) Schemes" registry maintained at [IANA.uri-schemes].

Note: The following has been added as a note for each of the URI schemes defined in this document:

> CoAP registers different URI schemes for accessing CoAP resources via different protocols. This approach runs counter to the WWW principle that a URI identifies a resource and that multiple URIs for identifying the same resource should be avoided <https://www.w3.org/TR/webarch/#avoid-uri-aliases>.

This is not a problem for many of the usage scenarios envisioned for CoAP over reliable transports; additional URI schemes can be introduced to address additional usage scenarios (as being prepared, for example, in [Multi-Transport-URIs] and [CoAP-Alt-Transports]).

11.5.1. coap+tcp

IANA has registered the URI scheme "coap+tcp". This registration request complies with [RFC7595].

Scheme name:
coap+tcp

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using TCP.

Contact:
IETF Chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.1 in RFC 8323

11.5.2. coaps+tcp

IANA has registered the URI scheme "coaps+tcp". This registration request complies with [RFC7595].

Scheme name:
coaps+tcp

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using TLS.

Contact:
IETF Chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.2 in RFC 8323

11.5.3. coap+ws

IANA has registered the URI scheme "coap+ws". This registration request complies with [RFC7595].

Scheme name:
coap+ws

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket Protocol.

Contact:
IETF Chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.3 in RFC 8323

11.5.4. coaps+ws

IANA has registered the URI scheme "coaps+ws". This registration request complies with [RFC7595].

Scheme name:
  coaps+ws

Status:
  Permanent

Applications/protocols that use this scheme name:
  The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket Protocol secured with TLS.

Contact:
  IETF Chair <chair@ietf.org>

Change controller:
  IESG <iesg@ietf.org>

References:
  Section 8.4 in RFC 8323

11.6. Well-Known URI Suffix Registration

IANA has registered "coap" in the "Well-Known URIs" registry. This registration request complies with [RFC5785].

URI suffix:
  coap

Change controller:
  IETF

Specification document(s):
  RFC 8323

Related information:
  None.

11.7. ALPN Protocol Identifier

IANA has assigned the following value in the "Application-Layer Protocol Negotiation (ALPN) Protocol IDs" registry created by [RFC7301]. The "coap" string identifies CoAP when used over TLS.

Protocol:
  CoAP

Identification Sequence:
  0x63 0x6f 0x61 0x70 ("coap")

Reference:
  RFC 8323

11.8. WebSocket Subprotocol Registration

IANA has registered the WebSocket CoAP subprotocol in the "WebSocket Subprotocol Name Registry":

Subprotocol Identifier:
  coap

Subprotocol Common Name:
  Constrained Application Protocol (CoAP)

Subprotocol Definition:
  RFC 8323

11.9. CoAP Option Numbers Registry

IANA has added this document as a reference for the following entries registered by [RFC7959] in the "CoAP Option Numbers" subregistry defined by [RFC7252]:

Number	Name	Reference
23	Block2	RFC 7959, RFC 8323
27	Block1	RFC 7959, RFC 8323

Table 3: CoAP Option Numbers

12. References

12.1. Normative References

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <https://www.rfc-editor.org/info/rfc5785>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/info/rfc6455>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/info/rfc7595>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

[RFC7925] Tschofenig, H., Ed., and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8307] Bormann, C., "Well-Known URIs for the WebSocket Protocol", RFC 8307, DOI 10.17487/RFC8307, January 2018, <https://www.rfc-editor.org/info/rfc8307>.

12.2. Informative References

[BK2015] Byrne, C. and J. Kleberg, "Advisory Guidelines for UDP Deployment", Work in Progress, draft-byrne-opsec-udp-advisory-00, July 2015.

[CoAP-Alt-Transports] Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", Work in Progress, draft-silverajan-core-coap-alternative-transports-10, July 2017.

[CoCoA] Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", Work in Progress, draft-ietf-core-cocoa-02, October 2017.

[EK2016] Edeline, K., Kuehlewind, M., Trammell, B., Aben, E., and B. Donnet, "Using UDP for Internet Transport Evolution", arXiv preprint 1612.07816, December 2016, <https://arxiv.org/abs/1612.07816>.

[HomeGateway] Haetoenen, S., Nyrhinen, A., Eggert, L., Strowes, S., Sarolahti, P., and N. Kojo, "An experimental study of home gateway characteristics", Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, DOI 10.1145/1879141.1879174, November 2010.

[IANA.uri-schemes] IANA, "Uniform Resource Identifier (URI) Schemes", <https://www.iana.org/assignments/uri-schemes>.

[LWM2M] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification Version 1.0", February 2017, <http://www.openmobilealliance.org/release/LightweightM2M/V1_0-20170208-A/OMA-TS-LightweightM2M-V1_0-20170208-A.pdf>.

[Multi-Transport-URIs] Thaler, D., "Using URIs With Multiple Transport Stacks", Work in Progress, draft-thaler-appsawg-multi-transport-uris-01, July 2017.

[OSCORE] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", Work in Progress, draft-ietf-core-object-security-08, January 2018.

[RFC768]   Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC7230] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[SecurityChallenges]
Polk, T. and S. Turner, "Security Challenges For the Internet Of Things", Interconnecting Smart Objects with the Internet / IAB Workshop, February 2011, <https://www.iab.org/wp-content/IAB-uploads/2011/03/Turner.pdf>.

[SW2016]   Swett, I., "QUIC Deployment Experience @Google", IETF 96 Proceedings, Berlin, Germany, July 2016, <https://www.ietf.org/proceedings/96/slides/slides-96-quic-3.pdf>.

[TCP-in-IoT]
Gomez, C., Crowcroft, J., and M. Scharf, "TCP Usage Guidance in the Internet of Things (IoT)", Work in Progress, draft-ietf-lwig-tcp-constrained-node-networks-01, October 2017.

Appendix A. Examples of CoAP over WebSockets

This appendix gives examples for the first two configurations discussed in Section 4.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap+ws" URI might be as follows. Figure 17 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI <coap+ws://example.org/sensors/temperature?u=Cel>, for example, from a resource representation that it retrieved previously.

2. The CoAP client establishes a WebSocket connection to the endpoint URI composed of the authority "example.org" and the well-known path "/.well-known/coap", <ws://example.org/.well-known/coap>.

3. CSMs (Section 5.3) are exchanged (not shown).

4. The CoAP client sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") Options.

5. The CoAP client waits for the server to return a response.

6. The CoAP client uses the connection for further requests, or the connection is closed.

```
      CoAP        CoAP
     Client      Server
   (WebSocket  (WebSocket
     Client)     Server)

        |           |
        |           |
        +=========>|  GET /.well-known/coap HTTP/1.1
        |           |  Host: example.org
        |           |  Upgrade: websocket
        |           |  Connection: Upgrade
        |           |  Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
        |           |  Sec-WebSocket-Protocol: coap
        |           |  Sec-WebSocket-Version: 13
        |           |
        |<=========+  HTTP/1.1 101 Switching Protocols
        |           |  Upgrade: websocket
        |           |  Connection: Upgrade
        |           |  Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
        |           |  Sec-WebSocket-Protocol: coap
        :           :
        :<-------->:  Exchange of CSMs (not shown)
        |           |
        +--------->|  Binary frame (opcode=%x2, FIN=1, MASK=1)
        |           |    +-------------------------+
        |           |    | GET                     |
        |           |    | Token: 0x53             |
        |           |    | Uri-Path: "sensors"     |
        |           |    | Uri-Path: "temperature" |
        |           |    | Uri-Query: "u=Cel"      |
        |           |    +-------------------------+
        |           |
        |<---------+  Binary frame (opcode=%x2, FIN=1, MASK=0)
        |           |    +-------------------------+
        |           |    | 2.05 Content            |
        |           |    | Token: 0x53             |
        |           |    | Payload: "22.3 Cel"     |
        |           |    +-------------------------+
        :           :
        :           :
        +--------->|  Close frame (opcode=%x8, FIN=1, MASK=1)
        |           |
        |<---------+  Close frame (opcode=%x8, FIN=1, MASK=0)
        |           |
```

Figure 17: A CoAP Client Retrieves the Representation of a Resource Identified by a "coap+ws" URI

Figure 18 shows how a CoAP client uses a CoAP forward proxy with a WebSocket endpoint to retrieve the representation of the resource "coap://[2001:db8::1]/". The use of the forward proxy and the address of the WebSocket endpoint are determined by the client from local configuration rules. The request URI is specified in the Proxy-Uri Option. Since the request URI uses the "coap" URI scheme, the proxy fulfills the request by issuing a Confirmable GET request over UDP to the CoAP server and returning the response over the WebSocket connection to the client.

```
     CoAP        CoAP       CoAP
    Client      Proxy      Server
  (WebSocket  (WebSocket    (UDP
    Client)     Server)   Endpoint)

       |          |          |
       +--------->|          |  Binary frame (opcode=%x2, FIN=1, MASK=1)
       |          |          |    +------------------------------------+
       |          |          |    | GET                                |
       |          |          |    | Token: 0x7d                        |
       |          |          |    | Proxy-Uri: "coap://[2001:db8::1]/" |
       |          |          |    +------------------------------------+
       |          |          |
       |          +--------->|  CoAP message (Ver=1, T=Con, MID=0x8f54)
       |          |          |    +------------------------------------+
       |          |          |    | GET                                |
       |          |          |    | Token: 0x0a15                      |
       |          |          |    +------------------------------------+
       |          |          |
       |          |<---------+  CoAP message (Ver=1, T=Ack, MID=0x8f54)
       |          |          |    +------------------------------------+
       |          |          |    | 2.05 Content                       |
       |          |          |    | Token: 0x0a15                      |
       |          |          |    | Payload: "ready"                   |
       |          |          |    +------------------------------------+
       |          |          |
       |<---------+          |  Binary frame (opcode=%x2, FIN=1, MASK=0)
       |          |          |    +------------------------------------+
       |          |          |    | 2.05 Content                       |
       |          |          |    | Token: 0x7d                        |
       |          |          |    | Payload: "ready"                   |
       |          |          |    +------------------------------------+
       |          |          |
```

Figure 18: A CoAP Client Retrieves the Representation of a Resource Identified by a "coap" URI via a WebSocket-Enabled CoAP Proxy

Acknowledgments

We would like to thank Stephen Berard, Geoffrey Cristallo, Olivier Delaby, Esko Dijk, Christian Groves, Nadir Javed, Michael Koster, Achim Kraus, David Navarro, Szymon Sasin, Goeran Selander, Zach Shelby, Andrew Summers, Julien Vermillard, and Gengyu Wei for their feedback.

Last Call reviews from Yoshifumi Nishida, Mark Nottingham, and Meral Shirazipour as well as several IESG reviewers provided extensive comments; from the IESG, we would like to specifically call out Ben Campbell, Mirja Kuehlewind, Eric Rescorla, Adam Roach, and the responsible AD Alexey Melnikov.

Contributors

Matthias Kovatsch
Siemens AG
Otto-Hahn-Ring 6
Munich D-81739
Germany

Phone: +49-173-5288856
Email: matthias.kovatsch@siemens.com

Teemu Savolainen
Nokia Technologies
Hatanpaan valtatie 30
Tampere FI-33100
Finland

Email: teemu.savolainen@nokia.com

Valik Solorzano Barboza
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago, IL 60607
United States of America

Phone: +1-847-634-6700
Email: vsolorzanobarboza@zebra.com

Authors' Addresses

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

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

Simon Lemay
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago, IL 60607
United States of America

Phone: +1-847-634-6700
Email: slemay@zebra.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Road
Cambridge CB1 9NJ
United Kingdom

Email: Hannes.tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

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

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

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

Brian Raymor (editor)

Email: brianraymor@hotmail.com