

CORE
Internet-Draft
Updates: 7641 (if approved)
Intended status: Standards Track
Expires: August 18, 2017

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.
Microsoft
February 14, 2017

CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets
draft-ietf-core-coap-tcp-tls-06

Abstract

The Constrained Application Protocol (CoAP), although inspired by HTTP, was designed to use UDP instead of TCP. The message layer of the CoAP over UDP protocol 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 TLS. This document outlines the changes required to use CoAP over TCP, TLS, and WebSockets transports. It also formally updates [RFC7641] for use with these transports.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 18, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

- 1. Introduction 3
 - 1.1. Conventions and Terminology 5
- 2. CoAP over TCP 6
 - 2.1. Messaging Model 6
 - 2.2. Message Format 7
 - 2.3. Message Transmission 10
 - 2.4. Connection Health 11
- 3. CoAP over WebSockets 11
 - 3.1. Opening Handshake 13
 - 3.2. Message Format 14
 - 3.3. Message Transmission 15
 - 3.4. Connection Health 15
- 4. Signaling 15
 - 4.1. Signaling Codes 16
 - 4.2. Signaling Option Numbers 16
 - 4.3. Capabilities and Settings Messages (CSM) 16
 - 4.4. Ping and Pong Messages 18
 - 4.5. Release Messages 19
 - 4.6. Abort Messages 20
 - 4.7. Signaling examples 21
- 5. Block-wise Transfer and Reliable Transports 22
 - 5.1. Example: GET with BERT Blocks 23
 - 5.2. Example: PUT with BERT Blocks 24
- 6. CoAP over Reliable Transport URIs 24
 - 6.1. coap+tcp URI scheme 25
 - 6.2. coaps+tcp URI scheme 25
 - 6.3. coap+ws URI scheme 26
 - 6.4. coaps+ws URI scheme 27
 - 6.5. Uri-Host and Uri-Port Options 28
 - 6.6. Decomposing URIs into Options 28
 - 6.7. Composing URIs from Options 29

7. Securing CoAP 29

7.1. TLS binding for CoAP over TCP 29

7.2. TLS usage for CoAP over WebSockets 30

8. Security Considerations 30

8.1. Signaling Messages 31

9. IANA Considerations 31

9.1. Signaling Codes 31

9.2. CoAP Signaling Option Numbers Registry 31

9.3. Service Name and Port Number Registration 32

9.4. Secure Service Name and Port Number Registration 33

9.5. URI Scheme Registration 34

9.6. Well-Known URI Suffix Registration 36

9.7. ALPN Protocol Identifier 36

9.8. WebSocket Subprotocol Registration 36

10. References 37

10.1. Normative References 37

10.2. Informative References 38

Appendix A. Updates to RFC7641 Observing Resources in the
Constrained Application Protocol (CoAP) 40

A.1. Notifications and Reordering 40

A.2. Transmission and Acknowledgements 40

A.3. Freshness 40

A.4. Cancellation 41

Appendix B. CoAP over WebSocket Examples 41

Appendix C. Change Log 44

C.1. Since draft-core-coap-tcp-tls-02 44

C.2. Since draft-core-coap-tcp-tls-03 44

C.3. Since draft-core-coap-tcp-tls-04 44

C.4. Since draft-core-coap-tcp-tls-05 44

Acknowledgements 45

Contributors 45

Authors' Addresses 45

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP [RFC0768] or DTLS [RFC6347] over UDP can be used unimpeded. UDP is a good choice for transferring small amounts of data across networks that follow the IP architecture.

Some CoAP deployments need to integrate well with existing enterprise infrastructures, where UDP-based protocols may not be well-received or may even be blocked by firewalls. Middleboxes that are unaware of CoAP usage for IoT can make the use of UDP brittle, resulting in lost or malformed packets.

Emerging standards such as Lightweight Machine to Machine [LWM2M] currently use CoAP over UDP as a transport and require support for CoAP over TCP to address the issues above and to protect investments in existing CoAP implementations and deployments. Although HTTP/2 could also potentially address these requirements, there would be additional costs and delays introduced by such a transition. Currently, there are also fewer HTTP/2 implementations available for constrained devices in comparison to CoAP.

To address these requirements, this document defines how to transport CoAP over TCP, CoAP over TLS, and CoAP over WebSockets. Figure 1 illustrates the layering:

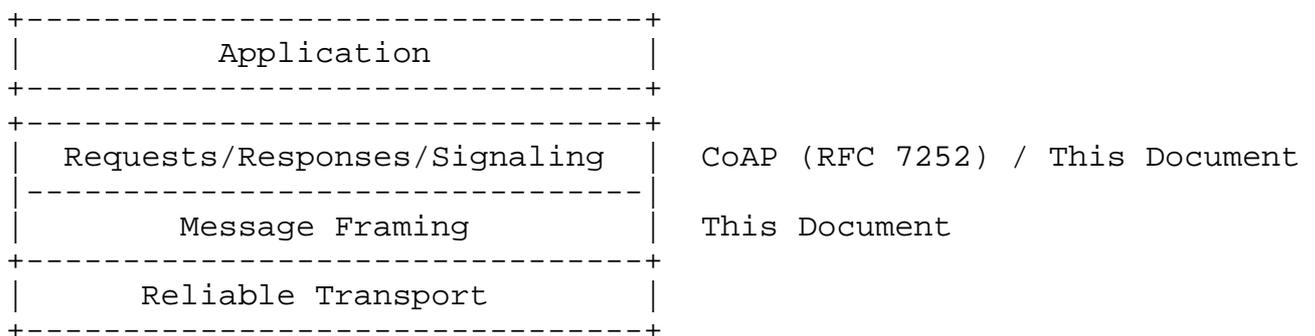


Figure 1: Layering of CoAP over Reliable Transports

Where NATs are present, CoAP over TCP can help with their traversal. 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 life cycle. UDP, on the other hand, does not provide such information to a NAT and timeouts tend to be much shorter [HomeGateway].

Some environments may also benefit from the ability of TCP to exchange larger payloads, such as firmware images, without application layer segmentation and to utilize the more sophisticated congestion control capabilities provided by many TCP implementations. Note that there is ongoing work to add more elaborate congestion control to CoAP (see [I-D.ietf-core-cocoa]).

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.

To allow IoT devices to better communicate in these demanding environments, CoAP needs to support different transport protocols, namely TCP [RFC0793], in some situations secured by TLS [RFC5246].

In addition, some corporate networks only allow Internet access via a HTTP proxy. In this case, the best transport for CoAP might be the WebSocket Protocol [RFC6455]. The WebSocket protocol provides two-way communication between a WebSocket client and a WebSocket server after upgrading an HTTP/1.1 [RFC7230] connection and may be available in an environment that blocks CoAP over UDP. Another scenario for CoAP over WebSockets is a CoAP application running inside a web browser without access to connectivity other than HTTP and WebSockets.

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 by communicating CoAP directly with a CoAP server accessible over a TCP/TLS or WebSocket connection or via a CoAP intermediary that proxies CoAP requests and responses between different transports, such as between WebSockets and UDP.

Appendix A updates Observing Resources in the Constrained Application Protocol [RFC7641] for use with CoAP over reliable transports. [RFC7641] is an extension to the CoAP core protocol 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.)

1.1. 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 [RFC2119].

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 which provide reliable and ordered delivery of a byte-stream.

BERT Option:

A Block1 or Block2 option that includes an SZX value of 7.

BERT Block:

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

Connection Initiator:

The peer that opens a reliable byte stream connection, i.e., the TCP active opener, TLS client, or WebSocket client.

Connection Acceptor:

The peer that accepts the reliable byte stream connection opened by the other peer, i.e., the TCP passive opener, TLS server, or WebSocket server.

For simplicity, a Payload Marker (0xFF) is shown in all examples for message formats:

```
...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|1 1 1 1 1 1 1 1| Payload (if any) ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

The Payload Marker indicates the start of the optional payload and is absent for zero-length payloads (see section 3 of [RFC7252]).

2. 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, Acknowledgement, and Reset. In addition, messages include a Message ID to relate Acknowledgments to Confirmable messages and to detect duplicate messages.

2.1. 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 acknowledgements or to detect duplicate messages. As a result, both the Type and Message ID fields are no longer required and are removed from the CoAP over TCP message format.

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

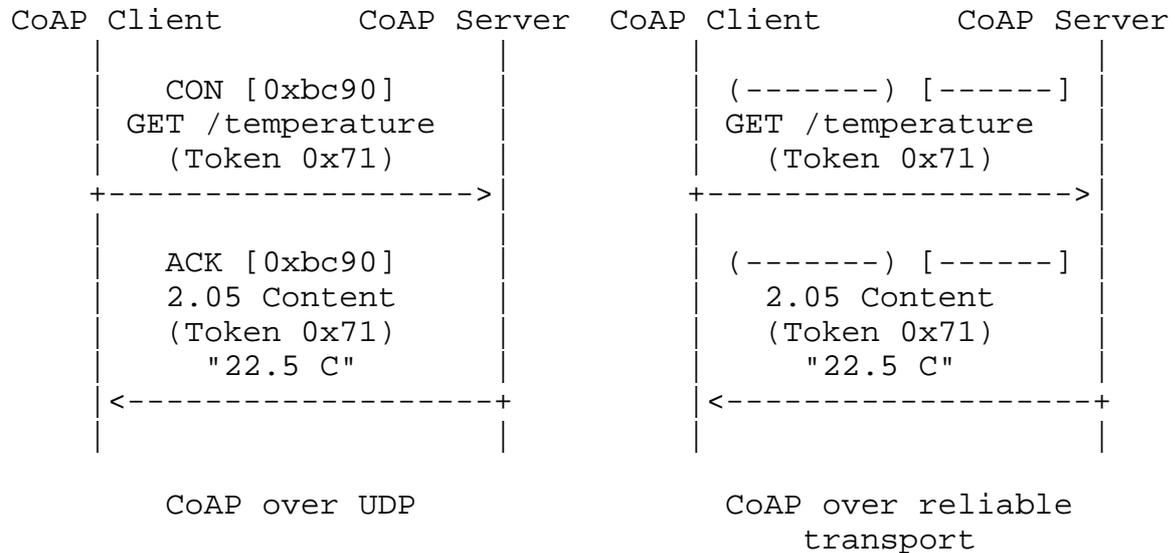


Figure 2: Comparison between CoAP over unreliable and reliable transport

2.2. 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.

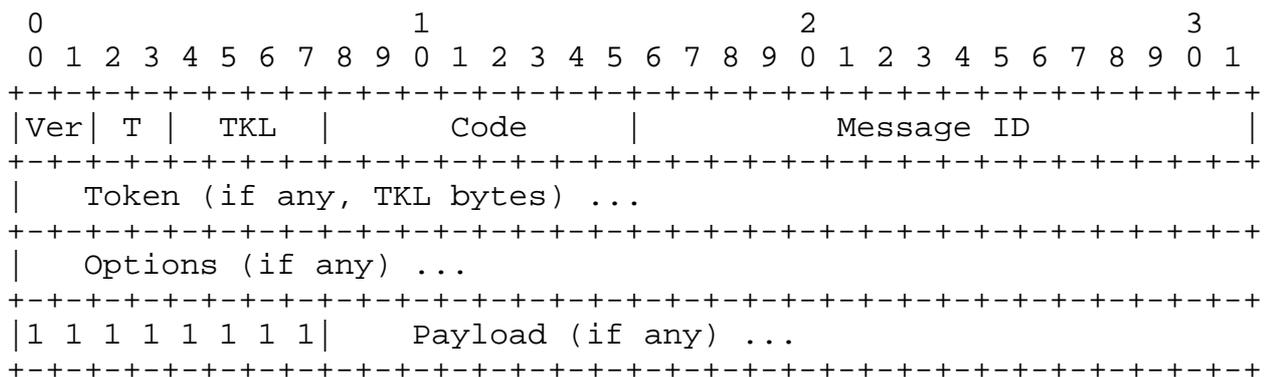


Figure 3: RFC 7252 defined CoAP Message Format

The CoAP over TCP message format 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, then Capabilities and Settings Messages (CSM see Section 4.3) 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 CoAP over UDP header), which includes the length information of variable size, shown here as an 8-bit length.

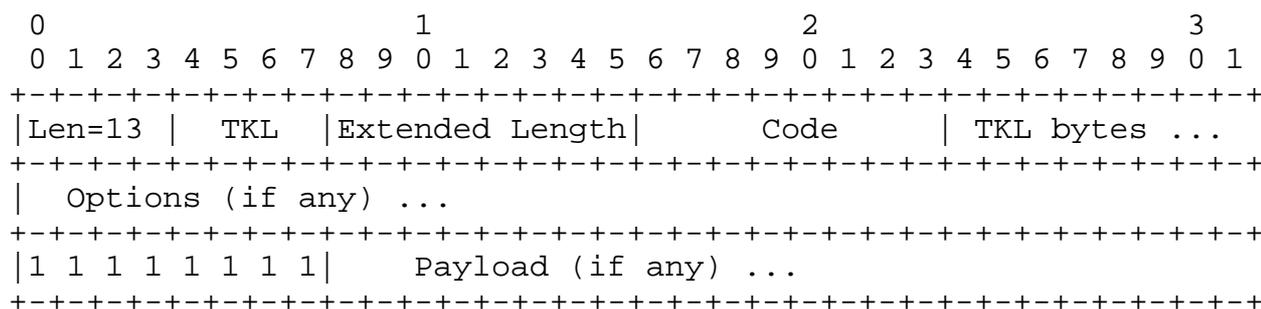


Figure 4: CoAP frame with 8-bit Extended Length field

Length (Len): 4-bit unsigned integer. A value between 0 and 12 directly 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 on CoAP Options (see section 3.1 of [RFC7252]).

The following figures show the message format for the 0-bit, 16-bit, and the 32-bit variable length cases.

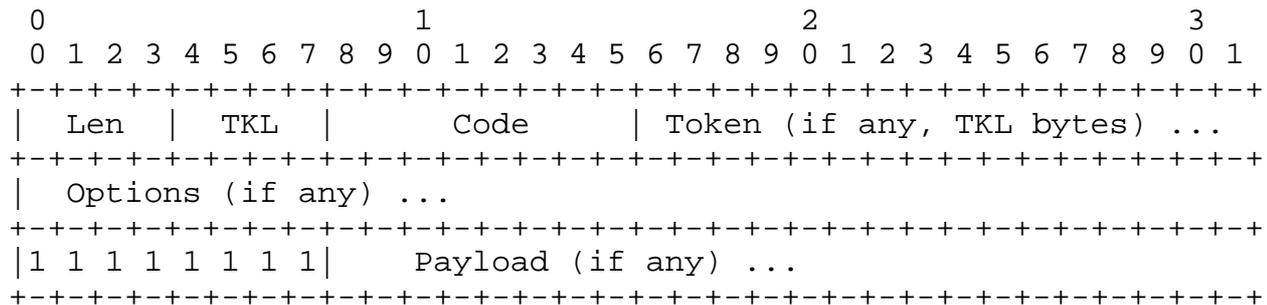


Figure 5: CoAP message format without an Extended Length field

For example: A CoAP message just containing a 2.03 code with the token 7f and no options or payload would be encoded as shown in Figure 6.

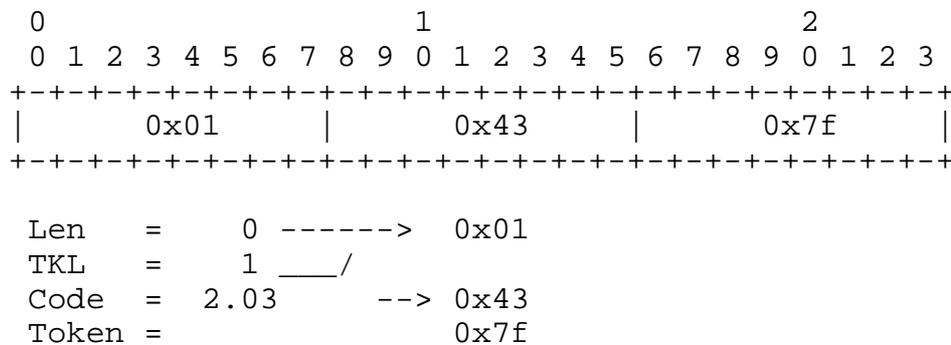


Figure 6: CoAP message with no options or payload

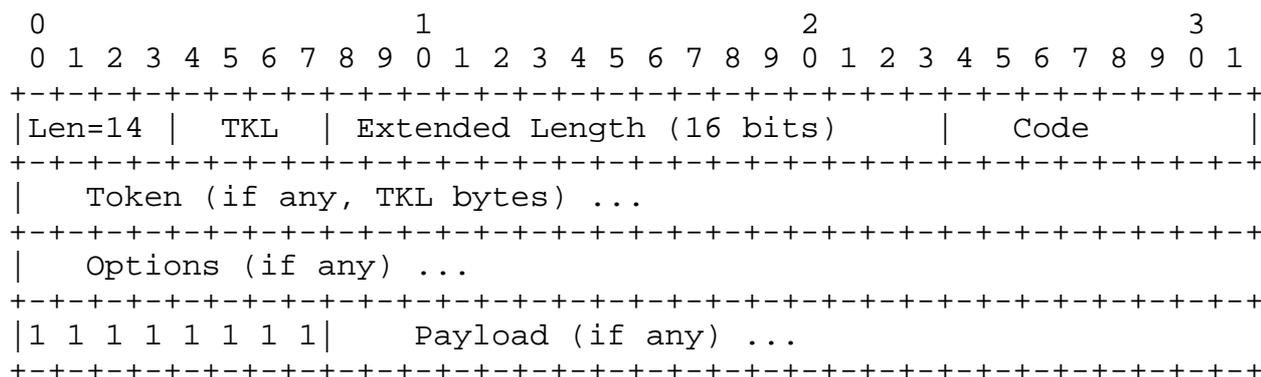


Figure 7: CoAP message format with 16-bit Extended Length field



Figure 8: CoAP message format with 32-bit Extended Length field

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

2.3. Message Transmission

Once a connection is established, both endpoints MUST send a Capabilities and Settings message (CSM see Section 4.3) as their 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 message before sending its own initial CSM message. Conversely, the Connection Acceptor MAY wait for the Connection Initiator to send its initial CSM message before sending its own initial CSM message.

To avoid unnecessary latency, a Connection Initiator MAY send additional messages without waiting to receive the Connection Acceptor's CSM; however, it is important to note that the Connection

Acceptor's CSM might advertise capabilities that impact how the initiator is expected to communicate with the acceptor. For example, the acceptor CSM could advertise a Max-Message-Size option (see Section 4.3.1) that is smaller than the base value (1152).

Endpoints MUST treat a missing or invalid CSM as a connection error and abort the connection (see Section 4.6).

CoAP requests and responses are exchanged asynchronously over the TCP/TLS 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. Concurrent requests are differentiated by their Token, which is scoped locally to the connection.

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

Retransmission and deduplication of messages is provided by the TCP/TLS protocol.

2.4. Connection Health

Empty messages (Code 0.00) can always be sent and MUST be ignored by the recipient. This provides a basic keep-alive 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 (Section 4.4) to test the connection and verify that the CoAP server is responsive.

3. 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 (Figure 9). 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.

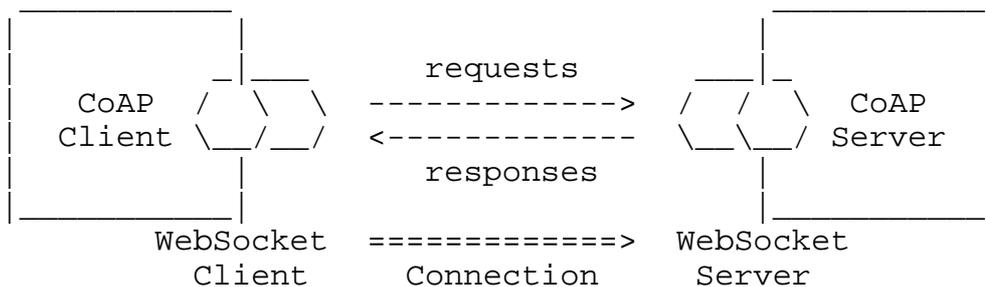


Figure 9: CoAP Client (WebSocket client) accesses CoAP Server (WebSocket server)

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. Section 6.3 and Section 6.4 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. When the WebSocket protocol is used from a web page, the choices are more limited [RFC6454], but the challenge persists.

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 10), 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.

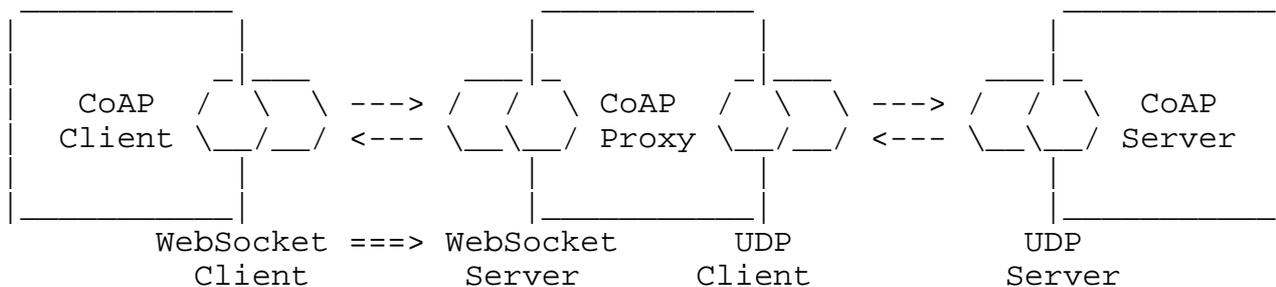


Figure 10: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

A third possible configuration is a CoAP server running inside a web browser (Figure 11). 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.

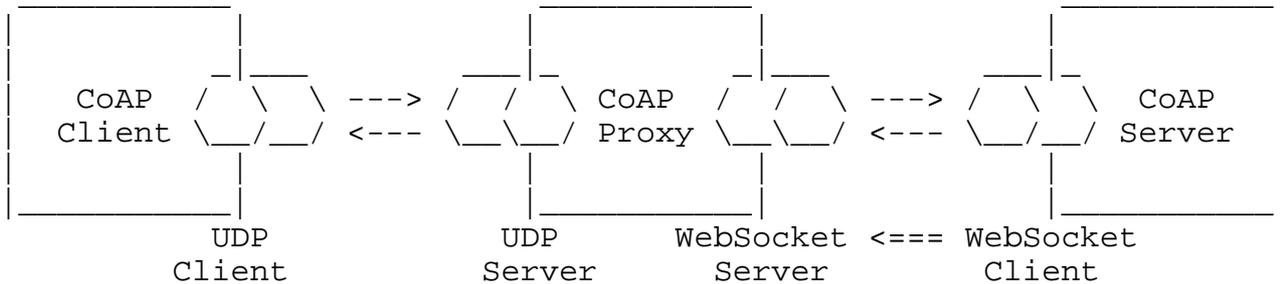


Figure 11: CoAP Client (UDP client) accesses CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

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

3.1. Opening Handshake

Before CoAP requests and responses are exchanged, a WebSocket connection is established as defined in Section 4 of [RFC6455]. Figure 12 shows an example.

The WebSocket client MUST include the subprotocol name "coap" in the list of protocols, which indicates support for the protocol defined in this document. Any later, incompatible versions of CoAP or CoAP over WebSockets will use a different subprotocol name.

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: dGh1IHNhbXBsZSBub25jZQ==
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 12: Example of an Opening Handshake

3.2. 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 13 is the same as the CoAP over TCP message format (see Section 2.2) with one change. The Length (Len) field MUST be set to zero because the WebSockets 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 13: 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 the definition of a new subprotocol identifier that is negotiated during the opening handshake.

Requests and response messages 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].

3.3. Message Transmission

As with CoAP over TCP, both endpoints MUST send a Capabilities and Settings message (CSM see Section 4.3) as their 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. Concurrent requests are differentiated by their Token, which is scoped locally to the connection.

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

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

3.4. Connection Health

As with CoAP over TCP, a CoAP client can test the health of the CoAP over WebSocket connection by sending a CoAP Ping Signaling message (Section 4.4). WebSocket Ping and unsolicited Pong frames (Section 5.5 of [RFC6455]) SHOULD NOT be used to ensure that redundant maintenance traffic is not transmitted.

4. Signaling

Signaling messages are introduced to allow peers to:

- o Related characteristics such as maximum message size for the connection
- o Shut down the connection in an orderly fashion
- o 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 is a code, a token, options, and an optional payload.

(See Section 3 of [RFC7252] for the overall structure, as adapted to the specific transport.)

4.1. 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" sub-registry (see Section 9.1).

For each message, there is 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.

4.2. 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" sub-registry (see Section 9.2).

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 4.6). If the option is understood but cannot be processed, the option documents the behavior.

4.3. Capabilities and Settings Messages (CSM)

Capabilities and Settings messages (CSM) are used for two purposes:

- o Each capability option advertises one capability of the sender to the recipient.
- o Setting options indicate a setting that will be applied by the sender.

One CSM MUST be sent by both endpoints at the start of the connection. Further CSM MAY be sent at any other time by either endpoint over the lifetime of the connection.

Both capability 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 setting before any Capabilities and Settings messages send a modified value.

These are not default values for the option as defined in Section 5.4.4 in [RFC7252]. A default value would mean that an empty Capabilities and Settings message would result in the option being set to its default value.

Capabilities and Settings messages are indicated by the 7.01 code (CSM).

4.3.1. Max-Message-Size Capability Option

The sender can use the elective Max-Message-Size Option to indicate the maximum message size in bytes that it can receive.

#	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.

4.3.2. 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 that supports block-wise transfers SHOULD indicate the Block-wise Transfer Option. If a Max-Message-Size Option is indicated with a value that is greater than 1152 (in the same or a different CSM message), the Block-wise Transfer Option also indicates support for BERT (see Section 5). Subsequently, if the Max-Message-Size Option is indicated with a value equal or less than 1152, BERT support is no longer indicated.

4.4. 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 keep-alive 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 there is 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).

4.4.1. 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. 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.

4.5. Release Messages

A Release message indicates that the sender does not want to continue maintaining the connection and opts for an orderly shutdown. 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 TCP/TLS connection. Messages may be in flight when the sender decides to send a Release message. The general expectation is that these will still be processed.

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].

The Alternative-Address Option is a repeatable option as defined in Section 5.4.5 of [RFC7252].

#	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.

4.6. Abort Messages

An Abort message indicates that the sender is unable to continue maintaining the connection and cannot even wait for an orderly release. The sender shuts down the connection immediately after the abort (and may or may not wait for a Release or 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 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

The elective Bad-CSM-Option Option 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 parameter problem with the value of an elective option. More detailed information SHOULD be included as a diagnostic payload.

One reason for an sender to generate an Abort message is a general syntax error in the byte-stream received. No specific option has been defined for this, as the details of that syntax error are best left to a diagnostic payload.

4.7. Signaling examples

An encoded example of a Ping message with a non-empty token is shown in Figure 14.

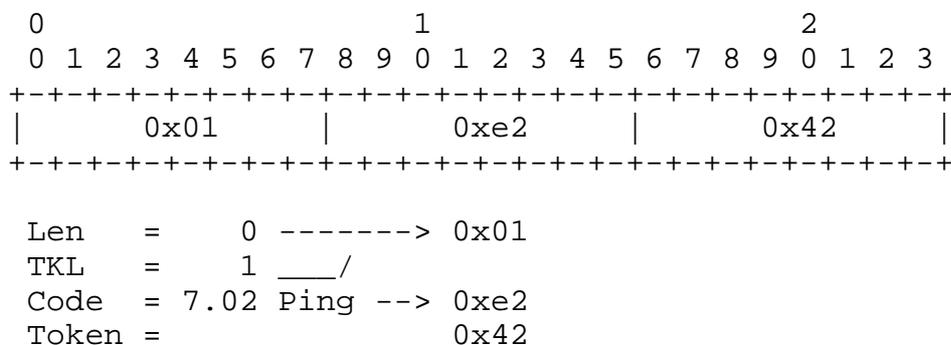


Figure 14: Ping Message Example

An encoded example of the corresponding Pong message is shown in Figure 15.

```

      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 15: Pong Message Example

5. Block-wise Transfer and Reliable Transports

The message size restrictions defined in Section 4.6 of CoAP [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:

- o large messages, such as firmware downloads, may cause undesired head-of-line blocking when a single TCP connection is used
- o 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 blockwise exchange from start to end into a single exchange)

The 'Block-wise Extension for Reliable Transport (BERT)' extends the Block 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 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 allowed to contain a multiple of 1024 bytes (non-final BERT block) or more than 1024 bytes (final BERT block).

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 with 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 with 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), more bit (M), and block size exponent ($2^{(SZX+4)}$) separated by slashes. E.g., 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.

5.1. Example: GET with BERT Blocks

Figure 16 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 16: GET with BERT blocks

5.2. Example: PUT with BERT Blocks

Figure 17 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 17: PUT with BERT blocks

6. 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 distinct namespaces and are considered to be distinct origin servers.

The syntax for the URI schemes in this section are specified using 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.

6.1. 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 changes:

- o The port subcomponent indicates the TCP port at which the CoAP server 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].

6.2. 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:

- o The port subcomponent indicates the TCP port at which the TLS server for the CoAP server is located. If it is empty or not given, then the default port 443 is assumed (this is different from the default port for "coaps", i.e., CoAP over DTLS over UDP).
- o If a TLS server does not support the Application-Layer Protocol Negotiation Extension (ALPN) [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.
- o For TCP ports other than port 5684, the TLS client MUST use the ALPN extension to advertise the "coap" protocol identifier (see Section 9.7) 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.
- o 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.
- o 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].

6.3. 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 component 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]. The path and query parts of a "coap+ws" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP:

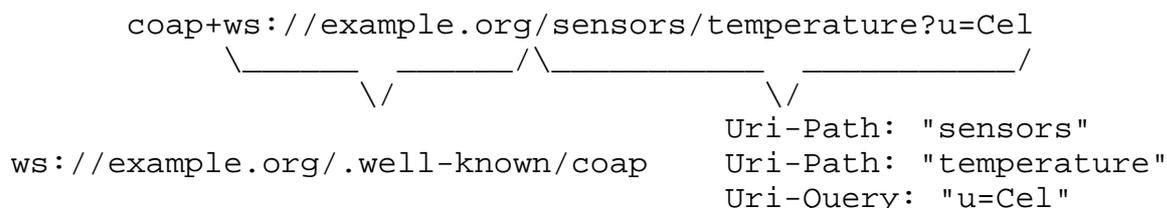


Figure 18: The "coap+ws" URI Scheme

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].

6.4. 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 component 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]. The path and query parts of a "coaps+ws" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP.

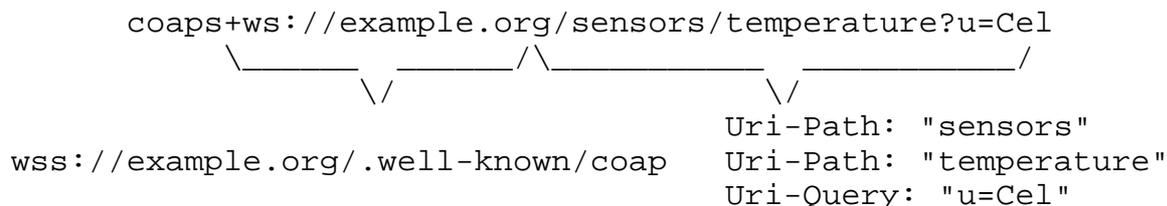


Figure 19: The "coaps+ws" URI Scheme

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].

6.5. 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.

6.6. Decomposing URIs into Options

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

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

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.

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|`.

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

6.7. Composing URIs from Options

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

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

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://".

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.

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.

7. Securing CoAP

Security Challenges for the Internet of Things [SecurityChallenges] recommends:

... 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 [I-D.ietf-core-object-security]).

7.1. TLS binding for CoAP over TCP

The TLS usage guidance in [RFC7925] applies.

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 mandatory-to-implement. The system simply sends the packets over normal TCP which 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.

7.2. 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 6.4.

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 in accepting incoming connections over a HTTPS (HTTP-over-TLS) port, the guidance in Section 4.2 of [RFC6455] applies.

8. 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.

8.1. Signaling Messages

- o 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.

9. IANA Considerations

9.1. Signaling Codes

IANA is requested to create a third sub-registry for values of the Code field in the CoAP header (Section 12.1 of [RFC7252]). The name of this sub-registry is "CoAP Signaling Codes".

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

Initial entries in this sub-registry are as follows:

Code	Name	Reference
7.01	CSM	[RFCthis]
7.02	Ping	[RFCthis]
7.03	Pong	[RFCthis]
7.04	Release	[RFCthis]
7.05	Abort	[RFCthis]

Table 1: CoAP Signal Codes

All other Signaling Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG Approval" as described in [RFC5226].

9.2. CoAP Signaling Option Numbers Registry

IANA is requested to create a sub-registry for Options Numbers used in CoAP signaling options within the "CoRE Parameters" registry. The name of this sub-registry is "CoAP Signaling Option Numbers".

Each entry in the sub-registry must include one or more of the codes in the Signaling Codes subregistry (Section 9.1), the option number, the name of the option, and a reference to the option's documentation.

Initial entries in this sub-registry are as follows:

Applies to	Number	Name	Reference
7.01	2	Max-Message-Size	[RFCthis]
7.01	4	Block-wise-Transfer	[RFCthis]
7.02, 7.03	2	Custody	[RFCthis]
7.04	2	Alternative-Address	[RFCthis]
7.04	4	Hold-Off	[RFCthis]
7.05	2	Bad-CSM-Option	[RFCthis]

Table 2: CoAP Signal Option Codes

The IANA policy for future additions to this sub-registry is based on number ranges for the option numbers, analogous to the policy defined in Section 12.2 of [RFC7252].

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.

9.3. Service Name and Port Number Registration

IANA is requested to assign the port number 5683 and the service name "coap+tcp", in accordance with [RFC6335].

Service Name.

coap+tcp

Transport Protocol.
tcp

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFCthis]

Port Number.
5683

9.4. Secure Service Name and Port Number Registration

IANA is requested to assign the port number 5684 and the service name "coaps+tcp", in accordance with [RFC6335]. The port number is requested to address the exceptional case of TLS implementations that do not support the "Application-Layer Protocol Negotiation Extension" [RFC7301].

Service Name.
coaps+tcp

Transport Protocol.
tcp

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFC7301], [RFCthis]

Port Number.
5684

9.5. URI Scheme Registration

URI schemes are registered within the "Uniform Resource Identifier (URI) Schemes" registry maintained at <http://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml> .

9.5.1. coap+tcp

IANA is requested to register the Uniform Resource Identifier (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 6.1 in [RFCthis]

9.5.2. coaps+tcp

IANA is requested to register the Uniform Resource Identifier (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 6.2 in [RFCthis]

9.5.3. coap+ws

IANA is requested to register the Uniform Resource Identifier (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 6.3 in [RFCthis]

9.5.4. coaps+ws

IANA is requested to register the Uniform Resource Identifier (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 6.4 in [RFCthis]

9.6. Well-Known URI Suffix Registration

IANA is requested to register the 'coap' well-known URI in the "Well-Known URIs" registry. This registration request complies with [RFC5785]:

URI Suffix.

coap

Change controller.

IETF

Specification document(s).

[RFCthis]

Related information.

None.

9.7. ALPN Protocol Identifier

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

Protocol.

CoAP

Identification Sequence.

0x63 0x6f 0x61 0x70 ("coap")

Reference.

[RFCthis]

9.8. WebSocket Subprotocol Registration

IANA is requested to register the WebSocket CoAP subprotocol under the "WebSocket Subprotocol Name Registry":

Subprotocol Identifier.

coap

Subprotocol Common Name.

Constrained Application Protocol (CoAP)

Subprotocol Definition.

[RFCthis]

10. References

10.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc6066>>.

- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc7301>>.
- [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, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc7925>>.

10.2. Informative References

[HomeGateway]

Eggert, L., "An experimental study of home gateway characteristics", Proceedings of the 10th annual conference on Internet measurement , 2010.

[I-D.ietf-core-cocoa]

Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-ietf-core-cocoa-00 (work in progress), October 2016.

[I-D.ietf-core-object-security]

Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-01 (work in progress), December 2016.

- [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>.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<http://www.rfc-editor.org/info/rfc768>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.
- [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, <<http://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, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<http://www.rfc-editor.org/info/rfc6454>>.
- [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, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<http://www.rfc-editor.org/info/rfc7959>>.
- [SecurityChallenges] Polk, T. and S. Turner, "Security Challenges for the Internet of Things", Interconnecting Smart Objects with the Internet / IAB Workshop , February 2011, <<http://www.iab.org/wp-content/IAB-uploads/2011/03/Turner.pdf>>.

Appendix A. Updates to RFC7641 Observing Resources in the Constrained Application Protocol (CoAP)

In this appendix, "client" and "server" refer to the CoAP client and CoAP server.

A.1. Notifications and Reordering

When using the Observe Option 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 the TCP protocol 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.

A.2. Transmission and Acknowledgements

For CoAP over UDP, server notifications to the client can be confirmable or non-confirmable. A confirmable message requires the client to either respond with an acknowledgement message or a reset message. An acknowledgement 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 which 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 acknowledgement 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 4.6).

A.3. Freshness

For CoAP over UDP, if a client does not receive a notification for some time, it MAY 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 CoAP Ping Signaling message (Section 4.4) rather than individual requests to confirm active observations.

A.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 cancelled 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 is either closed or times out.

Appendix B. CoAP over WebSocket Examples

This section gives examples for the first two configurations discussed in Section 3.

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 20 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. It 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. It 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.
4. It waits for the server to return a response.
5. The CoAP client uses the connection for further requests, or the connection is closed.

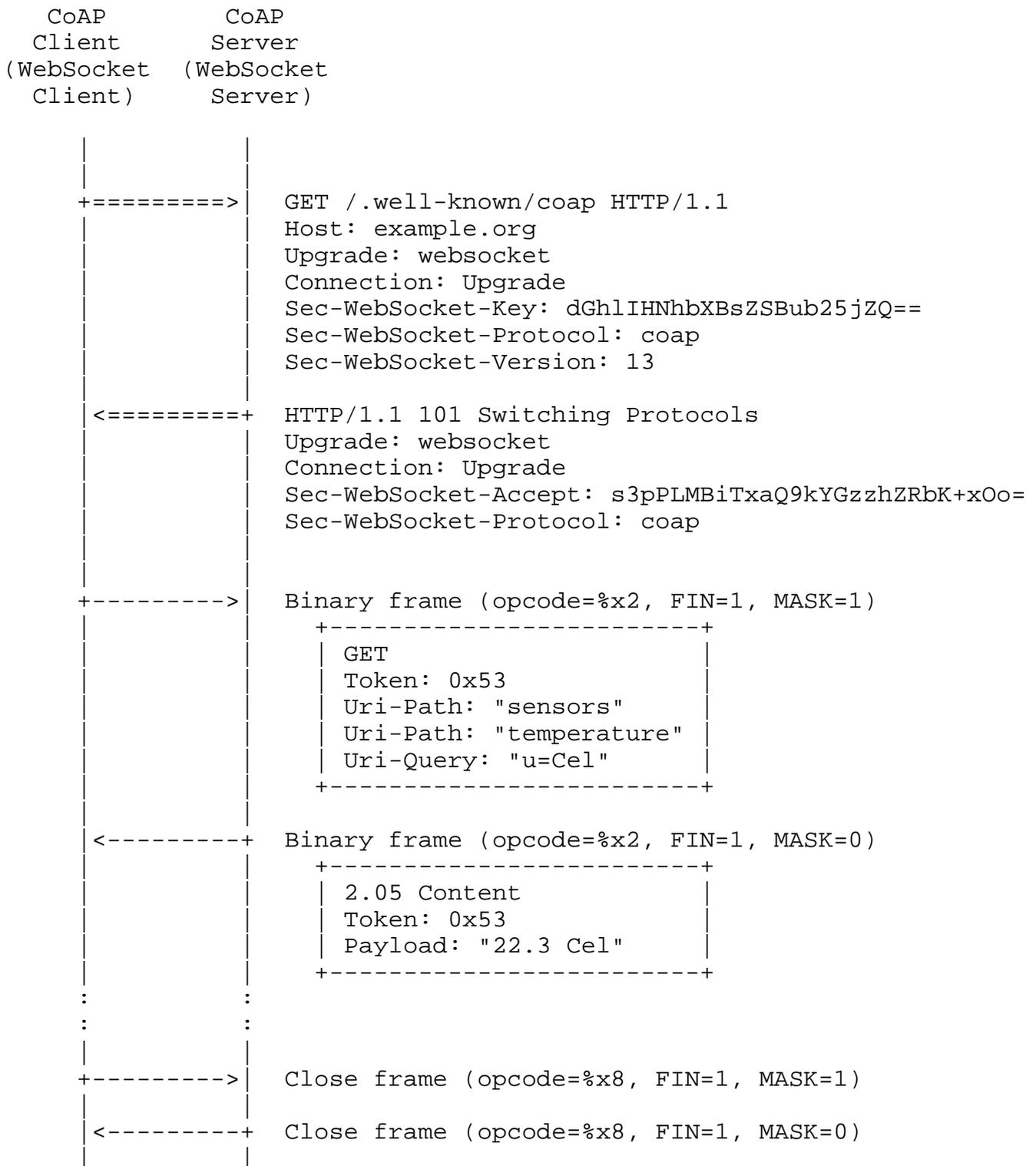


Figure 20: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 21 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.

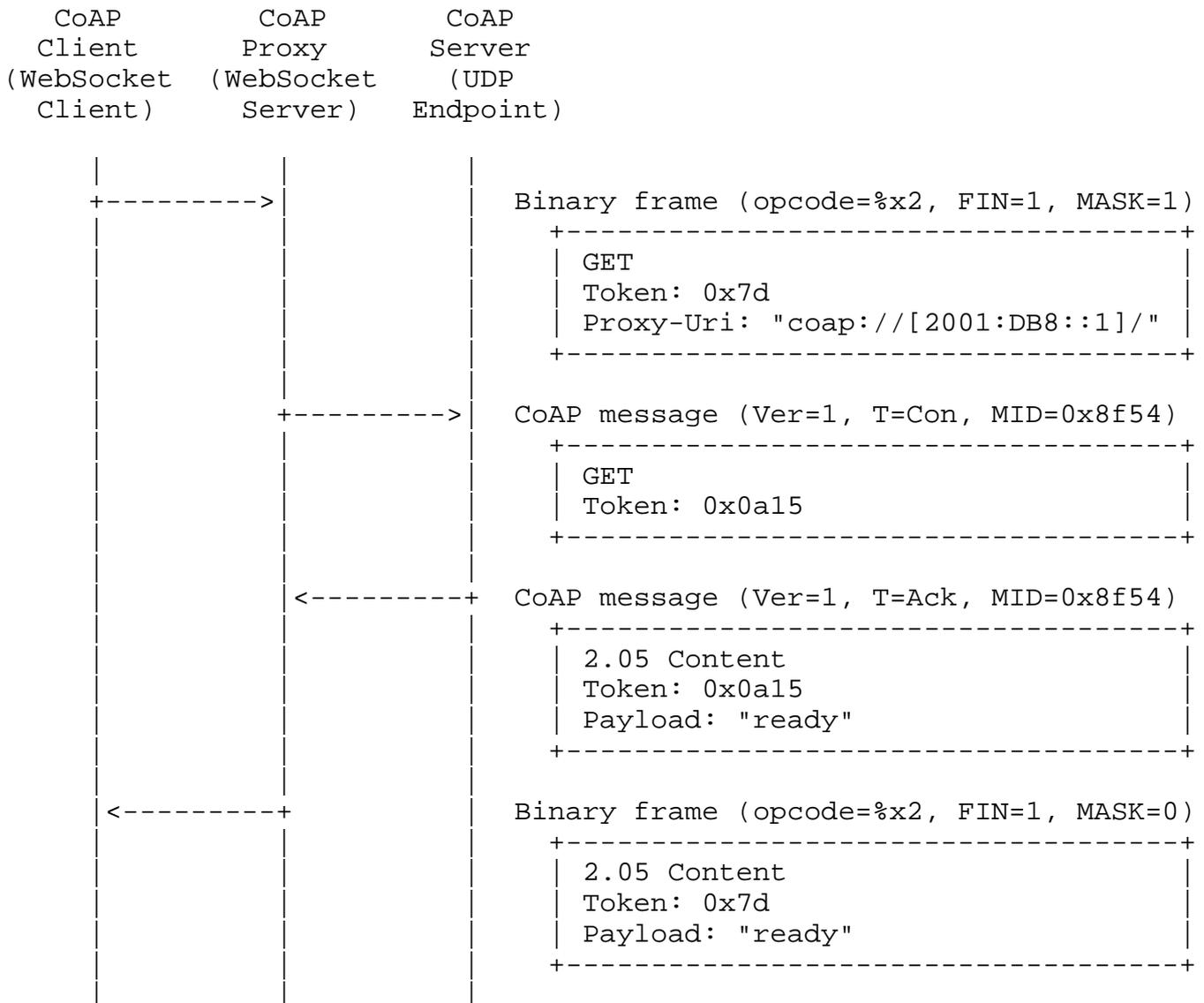


Figure 21: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSockets-enabled CoAP proxy

Appendix C. Change Log

The RFC Editor is requested to remove this section at publication.

C.1. Since draft-core-coap-tcp-tls-02

Merged draft-savolainen-core-coap-websockets-07 Merged draft-bormann-core-block-bert-01 Merged draft-bormann-core-coap-sig-02

C.2. Since draft-core-coap-tcp-tls-03

Editorial updates

Added mandatory exchange of Capabilities and Settings messages after connecting

Added support for coaps+tcp port 5684 and more details on Application-Layer Protocol Negotiation (ALPN)

Added guidance on CoAP Signaling Ping-Pong versus WebSocket Ping-Pong

Updated references and requirements for TLS security considerations

C.3. Since draft-core-coap-tcp-tls-04

Updated references

Added Appendix: Updates to RFC7641 Observing Resources in the Constrained Application Protocol (CoAP)

Updated Capability and Settings Message (CSM) exchange in the Opening Handshake to allow initiator to send messages before receiving acceptor CSM

C.4. Since draft-core-coap-tcp-tls-05

Addressed feedback from Working Group Last Call

Added Securing CoAP section and informative reference to OSCOAP

Removed the Server-Name and Bad-Server-Name Options

Clarified the Capability and Settings Message (CSM) exchange

Updated Pong response requirements

Added Connection Initiator and Connection Acceptor terminology where appropriate

Updated LWM2M 1.0 informative reference

Acknowledgements

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

Contributors

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

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 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 60607
United States of America

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

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
Great Britain

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)
Microsoft
One Microsoft Way
Redmond 98052
United States of America

Email: brian.raymor@microsoft.com