CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 13, 2014

T. Savolainen
K. Hartke
Nokia
B. Silverajan
Tampere University of Technology

July 12, 2013

CoAP over WebSockets
draft-savolainen-core-coap-websockets-00

Abstract

This document specifies how to retrieve and update CoAP resources using CoAP requests and responses over the WebSocket Protocol.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a web protocol designed for communications between resource constrained nodes. By default, CoAP operates on top of UDP or DTLS, but there is interest in using CoAP also over other types of transports, such as SMS [I-D.becker-core-coap-sms-gprs].

An interesting transport for CoAP could be the WebSocket Protocol [RFC6455]. The WebSocket protocol provides two-way communication between a client and a server after upgrading an HTTP [RFC2616] connection, and may be available in an environment that does not allow transportation of CoAP over UDP. This environment can be, for example, a corporate network with Internet access only via an HTTP proxy, or a CoAP application running in a web browser without access to connectivity means other than HTTP and WebSockets.

This document specifies how to access resources using CoAP requests and responses over the WebSocket Protocol. This allows connectivity-limited applications to obtain end-to-end CoAP connectivity either by communicating CoAP directly with a CoAP server that is accessible over a WebSocket Connection, or via an intermediary that proxies CoAP requests and responses between different transports, such as between WebSockets and UDP.

```
                 +-----------------------------------------+
                 |                                         |
                 |               Application               |
                 |                                         |
                 +-----------------------------------------+
                 |                                         |
                 |                   CoAP                  |
                 |          Requests and Responses         |
                 |                                         |
                 + - - - - - - - - - - +-------------------+
```

```
        |                   |                   |
        |       CoAP        |     CoAP over     |
        |     Messaging     |    WebSockets     |
        |                   |                   |
        +---------+---------+-------------------+
        |         |         |                   |
        |   UDP   |  DTLS   |    WebSockets     |
        |         |         |                   |
        +---------+---------+-------------------+
```

Figure 1: Abstract layering of CoAP extended by WebSockets

1.1. Overview

CoAP over WebSockets can be used in a number of configurations. The most basic configuration is a CoAP client seeking to retrieve or update a CoAP resource located at a CoAP server that exposes a WebSocket endpoint (Figure 2). The CoAP client takes the role of 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 2: CoAP client (WebSocket client)
accesses CoAP server (WebSocket server)

The challenge in this configuration is to identify 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. This

means it is necessary that the client is able to determine both the endpoint (identified by a "ws" or "wss" URI) and the path and query of the resource within that endpoint from the same URI. When the WebSocket Protocol is used from a web page, the choices are more limited [RFC6454], but the problem persists.

Section 3 proposes a new "coap+ws" URI scheme that identifies both a WebSocket endpoint and a resource within that endpoint as follows:

```
    coap+ws://example.org/path/to/endpoint?/sensors/temperature?u=degC
         \______________   ______________/ \____________  ___________/
                        \/                              \/
                                              Uri-Path: "sensors"
         ws://example.org/path/to/endpoint    Uri-Path: "temperature"
                                              Uri-Query: "u=degC"
```

Figure 3: The "coap+ws" URI Scheme

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 4), an SMS endpoint (a.k.a. mobile phone), or even another WebSocket endpoint. The 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 4: CoAP Client (WebSocket client) accesses CoAP Server
(UDP server) via a CoAP proxy (WebSocket server/UDP client)

In a completely different direction, another possible configuration is a CoAP server running inside a web browser (Figure 5). 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 not reachable; it therefore can be considered a sleepy endpoint (SEP) [I-D.dijk-core-sleepy-reqs].

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

Figure 5: CoAP Client (UDP client) accesses sleepy CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

The challenge, again, is to identify the resource. Since the CoAP server is running inside the web browser, this requires not only to identify the WebSocket client and the path and query, but also the intermediary, which is the only path to reach the server. The

problem can be avoided if the intermediary is turned into a reverse proxy or a mirror server [I-D.vial-core-mirror-server].

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

1.2. Terminology

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455] and [I-D.ietf-core-coap].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. CoAP over WebSockets

CoAP over WebSockets is intentionally very similar to CoAP as defined over UDP. Therefore, instead of presenting CoAP over WebSockets as a new protocol, this document specifies it as a series of deltas from [I-D.ietf-core-coap].

2.1. Opening Handshake

Before CoAP requests and responses can be exchanged, a WebSocket Connection needs to be established as defined in Section 4 of [RFC6455]. The WebSocket client MUST include the subprotocol name "coap.v1" in the list of protocols, which indicates support for the protocol defined in this document. Figure 6 shows an example.

```
GET /path/to/endpoint HTTP/1.1
Host: example.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: coap.v1
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: coap.v1
```

Figure 6: Example of an Opening Handshake

2.2. Message Format

Once a WebSocket Connection has been 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 is very similar to the format specified for CoAP

over UDP [I-D.ietf-core-coap]. 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. This means the "T" and "Message ID" fields in the CoAP message header can be elided.

o Furthermore, since the CoAP version is already negotiated during the opening handshake, the "Ver" field can be elided as well.

The resulting message format is shown in Figure 7. The four most-significant bits of the first byte are reserved (R). The remaining fields and structure are the same as defined in [I-D.ietf-core-coap].

```
  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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   R   |  TKL  |      Code     |    Token (TKL bytes) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Options (if any) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |1 1 1 1 1 1 1 1|    Payload (if any) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: CoAP Message Format over WebSockets

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, a multiplexing extension such as [I-D.ietf-hybi-websocket-multiplexing] can be used. Alternatively, requests and responses can be transferred in a blockwise fashion as defined in [I-D.ietf-core-block].

Messages MUST NOT be Empty (Code 0.00), i.e., they always carry either a request or a response.

2.3. Message Transmission

CoAP requests and responses are exchanged asynchronously over the WebSocket Connection, i.e., 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 the Token, which is local 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 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. Where [I-D.ietf-core-coap] makes a distinction between Confirmable and Non-Confirmable requests or responses, the normative text on Confirmable messages SHALL apply. Where [I-D.ietf-core-coap] makes a distinction between piggy-backed and separate responses, the normative text on separate responses SHALL apply.

2.4. Connection Health

If a client does not receive any response for some time after sending a CoAP request, the connection between the WebSocket client and the WebSocket server may be lost or temporarily disrupted without the client being aware of it. In this case, the client can send an unsolicited Pong frame to check the health of the WebSocket Connection, as specified in Section 5.5.3 of [RFC6455].

2.5. Closing the Connection

The WebSocket Connection is closed as specified in Section 7 of [RFC6455].

If there are requests for which the CoAP client has not received a response yet, the request is cancelled when the connection is closed.

If the CoAP client observes a resource [I-D.ietf-core-observe] over a WebSocket Connection, the CoAP server (or intermediary in the role of the CoAP server) MUST remove the client from the list of observers when the connection is closed.

3. CoAP over WebSockets URIs

For the first configuration discussed in Section 1.1, this document

defines two new URIs schemes that can be used for identifying CoAP resources and providing a means of locating these resources: "coap+ws" and "coap+wss".

Similar to the "coap" and "coaps" schemes, the "coap+ws" and "coap+wss" schemes organize resources hierarchically under a CoAP origin server. The key difference is that the server is potentially reachable on a WebSocket endpoint instead of a UDP endpoint. The endpoint is identified by an embedded "ws" or "wss" URI respectively. The remainder of the URI identifies a resource which can be operated on by the methods defined by the CoAP protocol.

The syntax of the "coap+ws" and "coap+wss" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty" and "query" are the same as in [RFC3986].

```
coap-ws-URI = "coap+" ws-URI-nq [ "?" path-abempty [ "?" query ] ]
coap-wss-URI = "coap+" wss-URI-nq [ "?" path-abempty [ "?" query ] ]

ws-URI-nq = "ws:" "//" host [ ":" port ] path-abempty
wss-URI-nq = "wss:" "//" host [ ":" port ] path-abempty
```

The port component is OPTIONAL; the default for "coap+ws" is port 80, while the default for "coap+wss" is port 443.

Fragments identifiers are not part of the request URI and thus MUST NOT be transmitted in a WebSocket handshake or a CoAP request.

4. Security Considerations

CoAP over WebSockets and CoAP over TLS-secured WebSockets do not introduce additional security issues beyond CoAP and DTLS-secured CoAP respectively [I-D.ietf-core-coap]. The security considerations of [RFC6455] apply.

5. IANA Considerations

5.1. URI Scheme Registrations

5.1.1. "coap+ws"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+ws".

URI scheme name.
   coap+ws

Status.
   Permanent.

URI scheme syntax.
   Defined in Section 3.

URI scheme semantics.
   The "coap+ws" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol.

Encoding considerations.
   The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

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

Interoperability considerations.
   None.

Security considerations.
   See Section 4.

Contact.
   IETF Chair <chair@ietf.org>

Author/Change controller.
   IESG <iesg@ietf.org>

References.
   This document.

5.1.2. "coap+wss"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+wss".

URI scheme name.
coap+wss

Status.
Permanent.

URI scheme syntax.
Defined in Section 3.

URI scheme semantics.
The "coap+wss" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol secured with Transport Layer Security (TLS).

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

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

Interoperability considerations.
None.

Security considerations.
See Section 4.

Contact.
IETF Chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
   This document.

5.2. WebSocket Subprotocol Registration

This document requests the registration of the subprotocol name "coap.v1" in the WebSocket Subprotocol Name Registry.

Subprotocol Identifier.
   coap.v1

Subprotocol Common Name.
   Constrained Application Protocol (CoAP).

Subprotocol Definition.
   This document.

6. Acknowledgements

Thanks to Nadir Javed for helpful comments and discussions that have shaped the document.

7. References

7.1. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

7.2. Informative References

[I-D.becker-core-coap-sms-gprs]
Becker, M., Li, K., Kuladinithi, K., and T. Poetsch, "Transport of CoAP over SMS, USSD and GPRS", draft-becker-core-coap-sms-gprs-03 (work in progress), February 2013.

[I-D.dijk-core-sleepy-reqs]
Dijk, E., "Sleepy Devices using CoAP - Requirements", draft-dijk-core-sleepy-reqs-00 (work in progress), June 2013.

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-12 (work in progress), June 2013.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.

[I-D.ietf-hybi-websocket-multiplexing]
Tamplin, J. and T. Yoshino, "A Multiplexing Extension for WebSockets", draft-ietf-hybi-websocket-multiplexing-11 (work in progress), July 2013.

[I-D.vial-core-mirror-server]
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-server-01 (work in progress), April 2013.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

Appendix A. Examples

This section gives examples for the first two configurations discussed in Section 1.1.

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 8 below illustrates the WebSocket and CoAP messages exchanged in detail.

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

2. It establishes a WebSocket Connection to the endpoint identified by the embedded "ws" URI, <ws://example.org/path/to/endpoint>.

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=degC") options.

4. It waits for server to return a response.

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


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

      |           |
      |           |
      +=========>|  GET /path/to/endpoint HTTP/1.1
      |           |  Host: example.org
      |           |  Upgrade: websocket
      |           |  Connection: Upgrade
```

```
        |          |  Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
        |          |  Sec-WebSocket-Protocol: coap.v1
        |          |  Sec-WebSocket-Version: 13
        |          |
        |<=========+  HTTP/1.1 101 Switching Protocols
        |          |  Upgrade: websocket
        |          |  Connection: Upgrade
        |          |  Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
        |          |  Sec-WebSocket-Protocol: coap.v1
        |          |
        |          |
        +--------->|  Binary frame (opcode=%x2, FIN=1, MASK=1)
        |          |    +-------------------------+
        |          |    | GET                     |
        |          |    | Token: 0x53             |
        |          |    | Uri-Path: "sensors"     |
        |          |    | Uri-Path: "temperature" |
        |          |    | Uri-Query: "u=degC"     |
        |          |    +-------------------------+
        |          |
        |<---------+  Binary frame (opcode=%x2, FIN=1, MASK=0)
        |          |    +-------------------------+
        |          |    | 2.05 Content            |
        |          |    | Token: 0x53             |
        |          |    | Payload: "22.3 C"       |
        |          |    +-------------------------+
        :          :
        :          :
        |          |
        +--------->|  Close frame (opcode=%x8, FIN=1, MASK=1)
        |          |
        |<---------+  Close frame (opcode=%x8, FIN=1, MASK=0)
        |          |
```

Figure 8: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 9 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 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 9: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSockets-enabled CoAP proxy

Authors' Addresses

Teemu Savolainen
Nokia
Hermiankatu 12 D
Tampere FI-33720
Finland

Email: teemu.savolainen@nokia.com

Klaus Hartke
Nokia
Hermiankatu 12 D
Tampere FI-33720
Finland

Email: klaus.hartke@nokia.com

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

Email: bilhanan.silverajan@tut.fi