Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 29, 2018

P. McManus
Mozilla

October 26, 2017

Bootstrapping WebSockets with HTTP/2
draft-mcmanus-httpbis-h2-websockets-01

Abstract

This document defines a mechanism for running the WebSocket Protocol [RFC6455] over a single stream of an HTTP/2 connection.

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 https://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 April 29, 2018.

Copyright Notice

Table of Contents

1. Introduction

The Hypertext Transfer Protocol (HTTP) provides compatible resource level semantics across different versions but it does not offer compatibility at the connection management level. Other protocols, such as WebSockets, that rely on connection management details of HTTP must be updated for new versions of HTTP.

The WebSocket Protocol [RFC6455] uses the HTTP/1.1 [RFC7230] Upgrade mechanism to transition a TCP connection from HTTP into a WebSocket connection. A different approach must be taken with HTTP/2 [RFC7540]. The multiplexing nature of HTTP/2 does not allow connection wide header and status codes such as the Upgrade and Connection request headers or the 101 response code due to its multiplexing nature. These are all required by the [RFC6455] connection establishment process.

Being able to bootstrap WebSockets from HTTP/2 allows one TCP connection to be easily shared by both protocols and extends HTTP/2's more efficient use of the network to WebSockets.

This document extends the HTTP/2 CONNECT method. The extension allows the substitution of a new protocol name to connect to rather than the external host normally used by CONNECT. The result is a tunnel on a single HTTP/2 stream that can carry data for WebSockets

(or any other protocol). The other streams on the connection may carry more extended CONNECT tunnels or traditional HTTP/2 data.

Streams that have been successfully established as protocol tunnels proceed to establish and utilize the WebSocket Protocol using the procedure defined by [RFC6455] treating the stream as if were the connection in that specification.

This tunneled stream will be multiplexed with other regular streams on the connection and enjoys the normal priority, cancellation, and flow control features of HTTP/2.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, [RFC2119].

3. The ENABLE_CONNECT_PROTOCOL SETTINGS Parameter

This document adds a new SETTINGS Parameter to those defined by [RFC7540] Section 6.5.2.

The new parameter is ENABLE_CONNECT_PROTOCOL (type = 0x8). The value of the parameter MUST be 0 or 1.

Upon receipt of ENABLE_CONNECT_PROTOCOL with a value of 1 a client MAY use the Extended CONNECT definition of this document when creating new streams. Receipt of this parameter by a server does not have any impact.

A sender MUST NOT send a ENABLE_CONNECT_PROTOCOL parameter with the value of 0 after previously sending a value of 1.

The use of a SETTINGS Parameter to opt-in to an otherwise incompatible protocol change is a use of "Extending HTTP/2" defined by section 5.5 of [RFC7540]. If a client were to use the provisions of the extended CONNECT method defined in this document without first receiving a ENABLE_CONNECT_PROTOCOL parameter with the value of 1 it would be a protocol violation.

4. The Extended CONNECT Method

The CONNECT Method of [RFC7540] Section 8.3 is modified in the following ways:

- A new pseudo-header :protocol MAY be included on request HEADERS indicating the desired protocol to be spoken on the tunnel created by CONNECT. The pseudo-header is single valued and contains a value from the HTTP Upgrade Token Registry defined by [RFC7230].

- On requests bearing the :protocol pseudo-header, the :scheme and :path pseudo-header fields SHOULD be included.

- On requests bearing the :protocol pseudo-header, the :authority pseudo-header field is interpreted according to [RFC7540] Section 8.1.2.3 instead of [RFC7540] Section 8.3. In particular the server MUST not make a new TCP connection to the host and port indicated by the :authority.

Upon receiving a CONNECT request bearing the :protocol pseudo-header the server establishes a tunnel to another service of the protocol type indicated by the pseudo-header. This service may or may not be co-located with the server.

5. Using Extended CONNECT To Bootstrap The WebSocket Protocol

The pseudo-header :protocol MUST be included in the CONNECT request and it MUST have a value of websocket to initiate a WebSocket connection on an HTTP/2 stream. Other HTTP request and response headers, such as those for manipulating cookies, may be included in the HEADERS with the CONNECT :method.

Parameters specific to WebSockets connection establishment, such as the version and sub-protocol are carried in a preamble within the HTTP/2 tunnel. This information was included in the HTTP headers in HTTP/1 but can now be conveyed in an HTTP independent fashion because the use of CONNECT with a new extension specific pseudo-header assures that the request can only be generated by a WebSocket client (See also Security Considerations).

The WebSocket peers process the opening handshake of [RFC6455] Section 4, using the information from the HTTP HEADERS and the connection preamble defined in this document. Substitutions are made for the preamble types defined here that were previously carried in HTTP/1 headers. For example, :authority pseudo-header provides the information from the Host header, and the Version preamble type provides the information from the Sec-WebSocket-Version header.

Implementations using this extended CONNECT to bootstrap WebSockets do not do the processing of the HTTP/1 Sec-WebSocket-Key and Sec-WebSocket-Accept headers as that functionality has been superceded by the :protocol pseudo-header.

After successfully processing the opening handshake the peers should proceed with The WebSocket Protocol [RFC6455] using the HTTP/2 stream from the CONNECT transaction as if it were the TCP connection referred to in [RFC6455]. The state of the WebSocket connection at this point is OPEN as defined by {{RFC6455} Section 4.1.

5.1. Connection Preamble

The connection preamble contains parameter information necessary for processing the WebSocket opening handshake. It is exchanged in both directions as the content of the first DATA frames on the HTTP/2 stream. The connection preamble parameter list is prefaced by a constant 16 byte sequence. This marker facilitates fast failure in cases where the endpoints are confused about what protocol is being spoken. The 16 bytes are: CA A4 02 0A 75 19 4D 49 86 4E A7 8F 14 86 CF 08.

Parameters such as WebSocket version, sub-protocol, and available extensions are exchanged as a series of Type, Length, Value (TLV) tuples. The preamble format is specific to WebSockets over HTTP/2. The tuples may appear in any order with the exception of the End-Preamble type which must appear last.

Each tuple consists of 1 type byte, 2 bytes specifying the network-order length of the value, and the length specified number of bytes containing the value.

Multiple TLV tuples of the same type form an ordered list of values. A single tuple contains only a single value (i.e. comma separated lists are not supported).

Several HTTP headers from [RFC6455] have corresponding preamble types defined here. Sec-WebSocket-Key and Sec-WebSocket-Accept are no longer used and do not have corresponding definitions.

5.1.1. Version Type

The Version type (0x00) MUST have a length of 1 and indicates the version of the WebSocket Protocol available. This corresponds to the Sec-WebSocket-Version header of [RFC6455]. It MUST appear at least one time in the client preamble and MUST appear exactly one time in the server preamble. The value defined by [RFC6455] is 0x0d.

5.1.2. Subprotocol Type

The Subprotocol type (0x01) corresponds to the Sec-WebSocket-Protocol header of [RFC6455] Section 11.3.4. It may appear any number of

times in the request preamble but MUST NOT appear more than once in the server preamble.

5.1.3. Extensions Type

The Extensions type (0x02) corresponds to the Sec-WebSocket-Extensions header of [RFC6455] Section 11.3.2. It may appear any number of times in the request preamble. The response preamble echoes the TLVs of the extensions selected.

5.1.4. Origin Type

The Origin type (0x03) corresponds to the Origin header defined by [RFC6454] and its use by [RFC6455]. Specifically this tuple MUST appear exactly once in a request preamble generated by a browser client and MAY appear once in requests from other clients. It MUST NOT appear in response preambles.

5.1.5. End-Preamble Type

The End-Preamble type (0xff) MUST be the last TLS in the preamble and MUST appear one time. It has a length of 0.

5.2. Example

```
   [[ From Client ]]                    [[ From Server ]]

                                        SETTINGS
                                        ENABLE_CONNECT_PROTOCOL = 1

   HEADERS + END_HEADERS
   :method = CONNECT
   :protocol = websocket
   :scheme = wss
   :path = /chat
   :authority = server.example.com:443

   DATA
   0xCA 0xA4 0x02 0x0A
   0x75 0x19 0x4D 0x49
   0x86 0x4E 0xA7 0x8F
   0x14 0x86 0xCF 0x08
   0x00 0x0001 0x0d
   0x01 0x0004 chat
   0x01 0x0009 superchat
   0x02 0x0012 permessage-deflate
   0x03 0x0016 http://www.example.com
   0xFF 0x0000

                                        HEADERS + END_HEADERS
                                        :status = 200

                                        DATA
                                        0xCA 0xA4 0x02 0x0A
                                        0x75 0x19 0x4D 0x49
                                        0x86 0x4E 0xA7 0x8F
                                        0x14 0x86 0xCF 0x08
                                        0x00 0x0001 0x0d
                                        0x01 0x0004 chat
                                        0xFF 0x0000

   DATA
   WebSocket Data

                                        DATA + END_STREAM
                                        WebSocket Data

   DATA + END_STREAM
   WebSocket Data
```

6. Design Considerations

A more native integration with HTTP/2 is certainly possible with larger additions to HTTP/2. This design was selected to minimize the solution complexity while still addressing the primary concern of running HTTP/2 and WebSockets concurrently.

7. About Intermediaries

This document does not change how WebSockets interacts with HTTP proxies. If a client wishing to speak WebSockets connects via HTTP/2 to a HTTP proxy it should continue to use a traditional (i.e. not with a :protocol pseudo-header) CONNECT to tunnel through that proxy to the WebSocket server via HTTP.

The resulting version of HTTP on that tunnel determines whether WebSockets is initiated directly or via a modified CONNECT request described in this document.

8. Security Considerations

[RFC6455] ensures that non WebSockets clients, especially XMLHttpRequest based clients, cannot make a WebSocket connection. Its primary mechanism for doing that is the use of Sec- prefixed request headers that cannot be created by XMLHttpRequest based clients. This specification addresses that concern in two ways:

- The CONNECT method is prohibited from being used by XMLHttpRequest
- The use of a pseudo-header is something that is connection specific and HTTP/2 does not ever allow to be created outside of the protocol stack.

9. IANA Considerations

This document establishes a entry for the HTTP/2 Settings Registry that was established by [RFC7540] Section 11.3

Name: ENABLE_CONNECT_PROTOCOL

Code: 0x8

Initial Value: 0

Specification: This document

10. Acknowledgments

The 2017 HTTP Workshop had a very productive discussion that helped determine the key problem and acceptable level of solution complexity.

11. Normative References

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

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/info/rfc6454>.

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

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

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

Author's Address

Patrick McManus
Mozilla

Email: mcmanus@ducksong.com