HTTPbis Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 13, 2015

Y. Hirano
Google, Inc.

August 12, 2014

WebSocket over HTTP/2
draft-hirano-httpbis-websocket-over-http2-01

Abstract

The WebSocket protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. Since it requires one TCP connection for every WebSocket connection, having multiple WebSocket connections between the same client and the same server is inefficient. On the other hand, HTTP/2 specifies a fast, secure, multiplexed framing protocol. This document provides bi-directional multiplexed communication by layering WebSocket on top of HTTP/2.

Please send feedback to the ietf-http-wg@w3.org mailing list.

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 February 13, 2015.

Table of Contents

1. Introduction

The WebSocket protocol was standardized to enable efficient bidirectional messaging mainly for browsers. However, the core spec in RFC 6455 left one problem about scalability unaddressed. That is that one WebSocket connection uses one TCP connection. Use of multiple WebSocket connections provides flexibility for web apps, while using more TCP connections leads to more load to the end hosts and also to network intermediaries.

For the HTTP/1.1, there has been effort to multiplex HTTP traffic into one TCP connection called HTTP/2. The HTTP/2 defines a general multiplexed transport on which not only HTTP but other messaging application protocol may be layered onto. We can address the scalability issue of WebSocket by using HTTP/2 framing's multiplexing functionality.

In this document, we describe how to layer WebSocket semantics onto HTTP/2 semantics by defining detailed mapping, replacement of operations and events defined in RFC 6455.

1.1. Document Organization

WebSocket over HTTP/2 is a protocol that layers the WebSocket protocol over an HTTP/2 stream rather than a TCP connection. This document introduces some abstractions and overrides some definitions in [RFC6455]. Definitions in [RFC6455] not overridden by this document such as Error Handling or Extensions are still valid.

Section 3 describes how to choose the protocol to use between native WebSocket and WebSocket over HTTP/2 for each server. Each of Section 4, Section 5 and Section 6 overrides definitions and rules in its counterpart in [RFC6455].

2. Conformance Requirements and Terminology

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].

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("MUST", "SHOULD", "MAY", etc.) used in introducing the algorithm.

Conformance requirements phrased as algorithms or specific steps can be implemented in any manner, so long as the end result is equivalent. In particular, the algorithms defined in this

specification are intended to be easy to understand and are not intended to be performant.

Native WebSocket means the WebSocket specified in [RFC6455].

"Frame" has two meanings, WebSocket frame and HTTP/2 frame. When it is obvious "WebSocket" and "HTTP/2" can be omitted. For example, "DATA frame" means "HTTP/2 DATA frame" and "Close frame" means "WebSocket Close frame".

3. Cross Protocol Negotiation

3.1. Overview

This section is non-normative..

To establish a WebSocket connection, a client needs to decide the protocol to use. Roughly speaking, if a client knows the server's preference the client will connect to the server with the protocol. Otherwise, the client tries to connect to the server with the native WebSocket.

3.2. server preference

The server can tell its preference between the WebSocket over HTTP/2 and the native WebSocket by the following means.

- Sending [ALT-SVC] information to the client

- Selecting an ALPN protocol

3.3. WebSocket over HTTP/2 capability

When two endpoints and all intermediaries between them understand WebSocket over HTTP/2, we say the communication path consisting of these nodes is capable of WebSocket over HTTP/2.

The client MUST send a SETTINGS frame containing SETTINGS_WEBSOCKET_CAPABLE before it starts the first WebSocket opening handshake on a HTTP/2 connection.

The client knows if the communication path towards the server is capable of WebSocket over HTTP/2 when the handshake response is received.

- If status code of the response is other than 501 (Not Implemented), the communication path is capable of WebSocket over HTTP/2.

- If status code of the response is 501 (Not Implemented), the communication path is not capable of WebSocket over HTTP/2.

When the server receives a handshake from a client, the server MUST send the server's opening handshake.

If the server has never received a SETTINGS frame that contains SETTINGS_WEBSOCKET_CAPABLE on the HTTP/2 connection, the server MUST send a 501 (Not Implemented) status code.

Otherwise, the server MUST NOT send a 501 (Not Implemented) status code.

The client MAY start an opening handshake with WebSocket over HTTP/2 without knowing if the communication path is capable of WebSocket over HTTP/2. When the status code of the opening handshake handshake from the server is 501 (Not Implemented), the client MAY start another opening handshake with the native WebSocket. If it comes to that, the connection failure MUST not be reported to the upper layer.

NOTE: The server may reset the stream. In such a case, the client doesn't know if the communication path is capable of WebSocket over

HTTP/2.

The client MUST not start an opening handshake with WebSocket over HTTP/2 when it knows that the communication path is not capable of WebSocket over HTTP/2.

3.4. secure connection

If the client knows that the server prefers WebSocket over HTTP/2 more than the native WebSocket and there is an existing HTTP/2 connection, the client create an HTTP/2 stream on the HTTP/2 connection.

Otherwise, the client sets up a TLS connection. The client SHOULD send one or two of the following application protocols as ProtocolNameList as specified in [ALPN] in any order.

- "http/1.1" for the native WebSocket over TLS
- "h2ws" for secure WebSocket over HTTP/2.

If the server selects the "h2ws" protocol, the client SHOULD connect to the server with WebSocket over HTTP/2 on the TLS connection. If the server selects the "http/1.1" protocol or the server does not support ALPN, the client SHOULD connect to the server with the native WebSocket on the TLS connection. If the server returns

"no_application_protocol" alert, the client MUST _Fail the WebSocket connection_.

3.5. the server's preference

The client SHOULD keep track of the [ALT-SVC] information provided by the server and use it as the server's preference.

Note that though a client uses the ALPN protocol when it sets up a TLS connection, it SHOULD not use the information after that.

4. Opening Handshake

4.1. Handshake Request

The client initiates an opening handshake by sending a HEADERS frame. The frame MUST NOT set the END_STREAM flag because WebSocket intends to establish a bi-directional communication port and to send arbitrary data after success in opening handshake. The HEADERS Name/Value section will contain all of the following headers which are associated with the WebSocket protocol [RFC6455] opening handshake. Upgrade, Connection, Sec-WebSocket-Key, and Sec-WebSocket-Version headers MUST NOT be included because we do not have to take care of protocol upgrading or verification over HTTP. The following name/value pairs MUST be present in every request:

":path": /resource name/ as used in the "Client Requirements" section of the WebSocket protocol specification. (See [RFC6455])

":authority": /host:port/ (e.g. "www.example.com:1234") as used in the "Client Requirements" section of the WebSocket protocol specification. (See [RFC6455])

"websocket-version": the WebSocket protocol version of this request. MUST be "WebSocket/13".

":scheme": the scheme portion of the URI. MUST be "ws" or "wss". (See also /secure/ flag in [RFC6455])

"websocket-origin": /origin/ as used in the "Client Requirements" section of the WebSocket protocol specification. (See [RFC6455])

In addition, the following OPTIONAL name/value pairs MAY be present:

"sec-websocket-protocol" - the Sec-WebSocket-Protocol header (See [RFC6455])

"sec-websocket-extensions" - the Sec-WebSocket-Extensions header (See [RFC6455])

Also, other HTTP compatible header name/value pairs MAY be present.

4.2. Handshake Response

The server responds to a client request with a HEADERS frame. If the

server intends to allow the client connection, the HEADERS frame MUST NOT set the END_STREAM flag and MUST have ":status" containing "101". Any status code other than 101 indicates that the WebSocket handshake has not completed and that the semantics of HTTP still apply. The client MAY send some data to the server before receiving the successful response. The server MUST ignore this data when opening handshake fails. After sending successful response, the server can send arbitrary data frames at any time. The response status line is unfolded into name/value pairs like other WebSocket handshake headers and MUST be present: ":status" - The WebSocket or fallback HTTP response status code (e.g. "101" or "101 Switching Protocols". See [RFC6455]). In addition, the following OPTIONAL name/value pairs MAY be present:

"sec-websocket-protocol" - the Sec-WebSocket-Protocol header (See [RFC6455])

"sec-websocket-extensions" - the Sec-WebSocket-Extensions header (See [RFC6455])

Also, other HTTP compatible header name/value pairs MAY be present. All header names MUST be lowercase. The successful server response MUST have ":status" containing "101".

4.2.1. The Alt-Svc header

When the Alt-Svc header field is contained in the handshake response, the client SHOULD use the advertised service if possible. Note that the Alt-Svc header field takes effect even for the handshake response whose status code is not 101.

If the client receives an opening handshake response having the Alt-Svc header field and the client is able to work with the advertised service, the client SHOULD send a Close frame with code 1006 and reason like "Alternate Service: h2ws" and then close the WebSocket connection as soon as possible. These transactions MUST be hidden and MUST NOT be notified to upper layers like the JavaScript event queue. Then, the client SHOULD connect to the advertised server with the advertised protocol.

5. Data Framing

TO BE WRITTEN

6. Closing the Connection

Some definitions in [RFC6455] are overridden in this section.

6.1. Definitions

6.1.1. Close the WebSocket Connection

To _Close the WebSocket Connection_, an endpoint closes the underlying HTTP/2 stream. If the stream is already closed, the endpoint MUST do nothing. Otherwise, the endpoint MUST send an RST_STREAM frame with an appropriate error code.

6.1.2. Start the WebSocket Closing Handshake

To _Start the WebSocket Closing Handshake_ with a status code (Section 6.4) /code/ and an optional close reason (Section 6.1.6) /reason/, an endpoint MUST send a Close control frame, as described in [RFC6455] whose status code is set to /code/ and whose close reason is set to /reason/. The last HTTP/2 frame of the WebSocket Close control frame MUST turn END_STREAM flag on.

6.1.3. The WebSocket Closing Handshake is Started

Same as Section 7.1.3 in [RFC6455].

6.1.4. The WebSocket Connection is Closed

When the underlying HTTP stream is closed, it is said that _The WebSocket Connection is Closed_ and that the WebSocket connection is in the CLOSED state. If the stream was closed after the WebSocket closing handshake was completed, the WebSocket connection is said to have been closed _cleanly_.

If the WebSocket connection could not be established, it is also said that _The WebSocket Connection is Closed_, but not cleanly.

6.1.5. The WebSocket Connection Close Code

Same as Section 7.1.5 in [RFC6455].

6.1.6. The WebSocket Connection Close Reason

Same as Section 7.1.6 in [RFC6455].

6.1.7. Fail the WebSocket Connection

Same as Section 7.1.7 in [RFC6455].

6.2. Abnormal Closures

6.2.1. Client-Initiated Closure

If at any point the underlying HTTP/2 stream is unexpectedly terminated, the client MUST _Fail the WebSocket Connection_.

Except as indicated above or as specified by the application layer (e.g. a script using the WebSocket API), clients SHOULD NOT close the connection.

6.2.2. Server-initiated closure

Same as Section 7.2.2 in [RFC6455].

6.2.3. Recovering from Abnormal Closure

Same as Section 7.2.3 in [RFC6455].

6.3. Normal Closure of Connections

Same as Section 7.3 in [RFC6455].

6.4. Status Codes

Same as Section 7.4 in [RFC6455].

7. Security Considerations

[RFC6455] frame has the masking mechanism for two purposes.

- To prevent a misbehavior of transparent proxies.
- To prevent TLS side-channel attacks such as [BEAST].

These should be addressed at the HTTP/2 framing layer and WebSocket over HTTP/2 has no masking mechanism.

8. IANA Considerations

8.1. Registration of New SETTINGS parameter

This section describes a new SETTINGS parameter.

SETTINGS_WEBSOCKET_CAPABLE(0xxx): The Client uses this parameter to declare that it wants to use WebSocket over HTTP/2. This parameter must be sent before creating any WebSocket over HTTP/2 stream in an HTTP/2 connection.

9. References

9.1. Normative References

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

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

[HTTP-2] Belshe, M., Peon, R., Thomson, M., and A. Melnikov, "Hypertext Transfer Protocol version 2", August 2014.

[ALPN] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application Layer Protocol Negotiation Extension", March 2014.

[ALT-SVC] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", July 2014.

9.2. Informative References

[BEAST] Duong, T. and J. Rizzo, "The BEAST attack", 2011.

Author's Address

Yutaka Hirano
Google, Inc.

Email: yhirano@google.com