Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 30, 2017

T. Yoshino
W. Zhu
Google, Inc.

March 29, 2017

WiSH: A General Purpose Message Framing over Byte-Stream Oriented Wire Protocols (HTTP)

draft-yoshino-wish-02

Abstract

This document defines a general purpose message framing named WiSH which supports bi-directional message-based communication over byte-stream oriented protocols such as HTTP (in its standard semantics). The WiSH framing is designed to be compatible with WebSocket. You may want to think about WiSH as a binary and bi-directional alternative to the framing defined for the server-sent events [SSE].

Table of Contents

1. Introduction

The WebSocket protocol was proposed to provide native client-server bi-directional messaging for the Web. It has been implemented and deployed widely, but there are still missing semantics and functionalities. See [BidiwebSurvey].

WiSH is a message framing format for use over the standard HTTP semantics to provide bi-directional messaging semantics. WiSH stands for Web in Strict HTTP. The communication protocol providing the HTTP semantics can be HTTP/1.1 [RFC7231], HTTP/2 [RFC7540], HTTP/2 + QUIC [QUIC], or any future protocols. Wire-protocol functionalities such as compression, multiplexing, session priority, etc. are provided by the underlying protocol [TransportAbstraction]. Unlike HTTP/2, HTTP/1.1 doesn't specify if earlier 2xx responses are allowed [RFC7540]. Therefore when HTTP/1.1 is used as the underlying protocol, full-duplex communication may be broken if the client, server or any proxy chooses to buffer or reject earlier 2xx responses. Since proxies may buffer response bodies, communication over WiSH may experience extra latency compared to WebSocket. When HTTPS is used, response buffering by proxies is less likely to happen.

Wire-protocol features of WebSocket, such as handshake or control messages, are all dropped. The WiSH framing respects the semantics of the underlying protocol (as opposed to turning it to a transport protocol). The concept of fragmentation is retained for enabling

starting message transmission before determining the final length of the message.

Application-level protocols may use WiSH as the framing protocol to support bi-directional communication over HTTP and for Web and Internet clients.

2. Background

There has been several attempts to improve bi-directional message-based communication on the Web.

The server-sent events [SSE] realized message-based communication in the server-to-client direction, by introducing a new Web API and a special message framing format while using HTTP as the wire protocol. Except for the issue of possible buffering by intermediaries, the server-sent events work well with existing intermediaries and frameworks that support HTTP.

WebSocket introduced both a new Web API and a new wire protocol to realize bi-directional message-based communication. Because the wire protocol is incompatible with HTTP, intermediaries and frameworks have to be upgraded to understand the protocol to support WebSocket.

In parallel to the development of WebSocket, HTTP has been greatly improved with HTTP/2. There are more improvements upcoming e.g. QUIC to the HTTP. At the same time, the Web APIs for HTTP have also been improved. The XMLHttpRequest is being replaced with the Fetch API [Fetch] which allows for streamed uploading and downloading of the body part of HTTP messages by using the Streams API [Streams]. The Streams API also enables implementing data transfer and various data processing (e.g. compression/decompression, message framing) in the form of the transform stream. The transform stream mechanism is designed to allow for optimizing transfer and processing by offloading some part of them from the JavaScript world.

It's desirable that further evolution of bi-directional message-based communication utilize HTTP/2 to reduce cost of development and standardization. Bidi communication should be multiplexed with normal HTTP traffic and should benefit from future transport-level improvements such as QUIC.

The WiSH idea is based on the above analysis. Combination of the Fetch API and transform streams enables efficient processing of the WiSH framing. Use of the HTTP semantics as-is reduces cost and makes the Web simpler. Once the WiSH idea is successfully adopted, binding to the WebSocket API could be introduced as further optimization for existing WebSocket users.

3. 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.

4. WiSH Protocol

WiSH frames messages over an HTTP request or response body using the framing defined in Section 5.

The "Content-Type" header value of the underlying HTTP request/response message MUST be "application/web-stream".

5. Framing

```
 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
+-+-+-+-+-------+-+-------------+-------------------------------+
|F|C|0|0|opcode |0|Payload      |Extended payload length        |
|I|M| | |4 bit  | |length       |16 bit if payload length is 126|
|N|P| | |       | |7 bit        |64 bit if payload length is 127|
+-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - - +
|                                                               |
+ - - - - - - - - - - - - - - - +-------------------------------+
|                               |Payload Data                   |
+-------------------------------+ - - - - - - - - - - - - - - - +
|                                                               |
+---------------------------------------------------------------+
```

WiSH framing is compatible with the framing defined in [RFC6455] for the WebSocket protocol.

The opcode field indicates how to interpret the payload data field. WiSH uses the following opcodes.

o %x0 denotes a continuation frame

- o %x1 denotes a text frame

- o %x2 denotes a binary frame

Any values not listed here are reserved.

The FIN bit together with the continuation frame opcode, payload length and extended payload length work in the same way as WebSocket to represent frames and messages. The fragmentation mechanism allows for flushing part of a large message payload without waiting for the total size of the message to be determined.

The CMP bit indicates whether the message is compressed. The CMP bit of the first frame MUST be set to 1 when compression is enabled for the message. Otherwise, it MUST be set to 0. The CMP bit of non-first frames MUST be always set to 0.

The message type distinction by the opcode field (text and binary) is kept to allow better Web support. One of the possible use cases is to use the text type for exchanging metadata encoded in JSON, etc., and the binary type for exchanging non-metadata messages.

The status code and status reason defined in the WebSocket protocol are dropped.

The ping and pong control message of the WebSocket protocol are dropped. If such a feature is needed, it should be provided by underlying protocols.

The permessage-deflate extension [RFC7692] is defined for the WebSocket protocol, to add a compression mechanism to it. No extension mechanism is defined for WiSH. Compression can be implemented by underlying protocols or in the application layer if needed. What contents are exchanged and in what encoding they are exchanged over WiSH are to be defined by the application layer.

6. Using WiSH over HTTP

Standard HTTP (REST) semantics should be followed, especially the choice of the HTTP method. Some HTTP semantics may not be applicable, e.g. the "Cache-Control" header, when the body is streamed. However, such limitation is not specific to WiSH.

7. Content Negotiation and WebSocket Compatibility Consideration

In this section, we discuss how to bridge WiSH and WebSocket.

The JavaScript binding and wire protocol handshake with WebSocket port is future work. Current thoughts are documented at [BidiwebBinding].

7.1. Subprotocol Negotiation

When layered over HTTP, a client and server MAY choose to negotiate a subprotocol (in the WebSocket term) to use by using the standard HTTP "Accept" and "Content-Type" headers. In order to be compatible with RFC6455, a client MAY list offered subprotocols as follows:

```
Accept: application/web-stream; protocol=foo; q=1,
    application/web-stream; protocol=bar; q=0.5
```

Following the "application/web-stream" media type, a parameter named "protocol" is specified with the subprotocol name as its value. A client offers multiple subprotocols by listing multiple "application/web-stream" media type with the "protocol" parameter with different values.

The client MAY indicate that the media type of the request body is WiSH by using the "Content-Type" header as follows:

```
Content-Type: application/web-stream
```

Where compatibility with WebSocket matters, symmetric subprotocols MUST be used. When multiple subprotocols are offered, a client MUST NOT specify the "protocol" parameter because it's not determined which subprotocol will be chosen by the server until the negotiation is done. When a single subprotocol is offered, a client MAY specify the "protocol" parameter which is the same as the one specified in the "Accept" header.

The server chooses one subprotocol from the offered ones and notifies the chosen subprotocol with the "Content-Type" header as follows:

```
Content-Type: application/web-stream; protocol=foo
```

The client SHOULD NOT start streaming any data (with the request body) before the client receives all the response headers from the server, which concludes the negotiation process.

Where compatibility with WebSocket doesn't matter, the "Content-Type" header value MAY differ between the HTTP request and HTTP response (see Section 6). This includes a combination of WiSH and non-WiSH media type.

7.2. Compression Negotiation

When layered over HTTP, a client and server MAY choose to negotiate a compression to use by using the standard HTTP "Accept-Encoding" and "Content-Encoding" headers. A client MAY list offered compression methods as follows:

```
Accept-Encoding:
    web-stream-deflate;
    client_max_window_bits; server_max_window_bits=10, q=1,
    web-stream-deflate;
    client_max_window_bits; server_max_window_bits; q=0.5
```

Each element in the header value consists of the identifier of the compression method followed by parameters configuring the method. The "web-stream-deflate" compression method in the example shows how the compression algorithm used for the "permessage-deflate" can be configured for the example. This example includes the "client_max_window_bits" for WebSocket compatibility which indicates whether or not the client supports the "client_max_window_bits" parameter.

The client MAY indicate use of the compression method for the HTTP request body by using the "Content-Encoding" header as follows:

```
Content-Encoding: web-stream-deflate
```

The server chooses one compression method from the offered ones and notifies the chosen compression method with the "Content-Encoding" header as follows:

```
Content-Encoding: web-stream-deflate
```

The server MAY also choose not to include the "Content-Encoding" header to indicate that it rejects use of any compression method.

The client SHOULD NOT start streaming any data (with the request body) before the client receives all the response headers from the server, which concludes the negotiation process.

7.3. Valid UTF-8 Requirement

In RFC6455, endpoints are required to _Fail the WebSocket Connection_ when they find that the byte stream in a text message is not a valid UTF-8 stream. To conform to the requirement, RFC6455 server frameworks check UTF-8 validness. The contents of text messages of WiSH also MUST be a valid UTF-8 stream. However, WiSH endpoints are not required to check UTF-8 validness. This provides more

flexibility to server development. For example, a server may choose to check UTF-8 validness inside a JSON parser.

8. Acknowledgements

Thank you to the following people for giving feedback to the document: Ben Christensen, Costin Manolache, Kari Hurtta, Loic Hoguin, Roberto Peon, Van Catha.

9. References

9.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

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

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10 .17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

[RFC7692] Yoshino, T., "Compression Extensions for WebSocket", RFC 7692, DOI 10.17487/RFC7692, December 2015, <http://www.rfc-editor.org/info/rfc7692>.

9.2. Non-normative References

[SSE] WHATWG, "HTML Living Standard", October 2016, <https://html.spec.whatwg.org/multipage/comms.html>.

[Fetch] WHATWG, "Fetch Standard", October 2016, <https://fetch.spec.whatwg.org/>.

[Streams] WHATWG, "Standard", October 2016, <https://streams.spec.whatwg.org/>.

[BidiwebSurvey]
Yoshino, T. and W. Zhu, "Non Request-Response Communication over the Web, and What's Missing", January 2014, <https://github.com/bidiweb/bidiweb-semantics/blob/master/SurveyOfProtocolGaps.md>.

[BidiwebBinding]
Bidiweb, , "Issue about WiSH JavaScript binding", March 2017, <https://github.com/bidiweb/wish/issues/12>.

[TransportAbstraction]
Zhu, W., "http-transport-abstraction", July 2016, <https://github.com/bidiweb/http-transport-abstraction>.

[QUIC] Hamilton, R., Iyengar, J., Swett, I., and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2", July 2016.

Authors' Addresses

Takeshi Yoshino
Google, Inc.

Email: tyoshino@google.com

Wenbo Zhu
Google, Inc.

Email: wenboz@google.com