WiSH: A General Purpose Message Framing over Byte-Stream Oriented Wire Protocols (HTTP)
draft-yoshino-wish-01

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].

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 May 4, 2017.

Copyright Notice

Copyright (c) 2016 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

...
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

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.

- %x0 denotes a continuation frame
%x1 denotes a text frame
%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 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. WebSocket Compatibility

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

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. 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


9.2. Non-normative References


Authors’ Addresses

Takeshi Yoshino
Google, Inc.

Email: tyoshino@google.com

Wenbo Zhu
Google, Inc.

Email: wenboz@google.com