HyBi Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 29, 2013

L. Stout, Ed.
&yet
J. Moffitt
E. Cestari
ProcessOne

February 25, 2013

An XMPP Sub-protocol for WebSocket
draft-moffitt-xmpp-over-websocket-02

Abstract

This document defines a binding for the XMPP protocol over a WebSocket transport layer. A WebSocket binding for XMPP provides higher performance than the current HTTP binding for XMPP.

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 August 29, 2013.

Copyright Notice

Table of Contents

1. Introduction

Applications using XMPP (see [RFC6120] and [RFC6121]) on the Web currently make use of BOSH (see [XEP-0124] and [XEP-0206]), an XMPP binding to HTTP. BOSH is based on the HTTP long polling technique, and it suffers from high transport overhead compared to XMPP's native binding to TCP. In addition, there are a number of other known issues with long polling [RFC6202], which have an impact on BOSH-based systems.

It would be much better in most circumstances to avoid tunneling XMPP over HTTP long polled connections and instead use the XMPP protocol directly. However, the APIs and sandbox that browsers have provided do not allow this. The WebSocket protocol [RFC6455] now exists to solve these kinds of problems. The WebSocket protocol is a bi-directional protocol that provides a simple message-based framing layer over raw sockets and allows for more robust and efficient communication in web applications.

The WebSocket protocol enables two-way communication between a client and a server, effectively emulating TCP at the application layer and therefore overcoming many of the problems with existing long-polling techniques for bidirectional HTTP. This document defines a WebSocket sub-protocol for the Extensible Messaging and Presence Protocol (XMPP).

2. Terminology

The basic unit of framing in the WebSocket protocol is called a message. In XMPP, the basic unit is the stanza, which is a subset of the first-level children of each document in an XMPP stream (see Section 9 of [RFC6120]). XMPP also has a concept of messages, which are stanzas whose top-level element name is message. In this document, the word "message" will mean a WebSocket message, not an XMPP message stanza (see Section 3.2).

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].

3. XMPP Sub-Protocol

3.1. Handshake

The XMPP sub-protocol is used to transport XMPP over a WebSocket connection. The client and server agree to this protocol during the WebSocket handshake (see Section 1.3 of [RFC6455]).

During the WebSocket handshake, the client MUST include the |Sec-WebSocket-Protocol| header in its handshake, and the value |xmpp| MUST be included in the list of protocols. The reply from the server MUST also contain |xmpp| in its own |Sec-WebSocket-Protocol| header in order for an XMPP sub-protocol connection to be established.

Once the handshake is complete, WebSocket messages sent or received will conform to the protocol defined in the rest of this document.

3.2. Messages

Data frame messages in the XMPP sub-protocol MUST be of the text type and contain UTF-8 encoded data. The close control frame’s contents are specified in Section 3.5. Control frames other than close are not restricted.

Unless noted in text, the word "message" will mean a WebSocket message containing a text data frame.

3.3. XMPP Stream Setup

The first message sent after the handshake is complete MUST be an XMPP opening stream tag as defined in XMPP [RFC6120] or an XML text declaration (see Section 4.3.1 of [W3C.REC-xml-20081126]) followed by an XMPP opening stream tag. The stream tag MUST NOT be closed (i.e. the closing </stream:stream> tag should not appear in the message) as

it is the start of the client's outgoing XML. The '<' character of the tag or text declaration MUST be the first character of the text payload.

The server MUST respond with a message containing an error (see Section 3.4), its own opening stream tag, or an XML text declaration followed by an opening stream tag.

Except in the case of certain stream errors (see Section 3.4), the opening stream tag, <stream:stream>, MUST appear in a message by itself.

3.4. Stream Errors

Stream level errors in XMPP are terminal. Should such an error occur, the server MUST send the stream error as a complete element in a message to the client.

If the error occurs during the opening of a stream, the stream error message MUST start with an opening stream tag (see Section 4.7.1 of [RFC6120]) and end with a closing stream tag.

After the stream error and closing stream tag have been sent, the server MUST close the connection as in Section 3.5.

3.5. Closing the Connection

Either the server or the client may close the connection at any time. Before closing the connection, the closing party MUST close the XMPP stream if it has been established. To initiate the close, the closing party MUST send a normal WebSocket close message with an empty body. The connection is considered closed when a matching close message is received (see Section 1.4 of [RFC6455]).

Except in the case of certain stream errors (see Section 3.4), the closing stream tag, </stream:stream>, MUST appear in a message by itself.

3.6. Stanzas

Each XMPP stanza MUST be sent in its own message. A stanza MUST NOT be split over multiple messages. All first level children of the <stream:stream> element MUST be treated the same as stanzas (e.g. <stream:features> and <stream:error>).

3.7. Stream Restarts

After successful SASL authentication, an XMPP stream needs to be restarted. In these cases, as soon as the message is sent (or received) containing the success indication, both the server and client streams are implicitly closed, and new streams needs to be opened. The client MUST open a new stream as in Section 3.3 and MUST NOT send a closing stream tag.

3.8. Pings and Keepalives

XMPP servers send whitespace pings as keepalives between stanzas, and XMPP clients can do the same thing. These extra whitespace characters are not significant in the protocol. Servers and clients SHOULD use WebSocket ping messages instead for this purpose.

The XMPP Ping extension [XEP-0199] allows entities to send and respond to ping requests. A client sending a WebSocket ping is equivalent to pinging the WebSocket server, which may also be the XMPP server. When the XMPP server is not also the WebSocket server, a WebSocket ping may be useful to check the health of the intermediary server.

3.9. Use of TLS

TLS cannot be used at the XMPP sub-protocol layer because the sub-protocol does not allow for raw binary data to be sent. Instead, enabling TLS SHOULD be done at the WebSocket layer using secure WebSocket connections via the |wss| URI scheme. (See Section 10.6 of [RFC6455]).

Because TLS is to be provided outside of the XMPP sub-protocol layer, a server MUST NOT advertise TLS as a stream feature (see Section 4.6 of [RFC6120]), and a client MUST ignore any advertised TLS stream feature, when using the XMPP sub-protocol.

3.10. Stream Management

Implications of, and recommendation to use, the XMPP Stream Management extension [XEP-0198] to be added.

4. Examples

Examples will be added as soon as the WebSocket protocol specification is more stable.

5. Security Considerations

Since application level TLS cannot be used (see Section 3.9), applications which need to protect the privacy of the XMPP traffic need to do so at the WebSocket or other appropriate layer.

The Security Considerations for both WebSocket (See Section 10 of [RFC6455] and XMPP (See Section 13 of [RFC6120]) apply to the WebSocket XMPP sub-protocol.

6. IANA Considerations

This specification requests IANA to register the WebSocket XMPP sub-protocol under the "WebSocket Subprotocol Name" Registry with the following data:

Subprotocol Identifier: xmpp

Subprotocol Common Name: WebSocket Transport for the Extensible Messaging and Presence Protocol (XMPP)

Subprotocol Definition: RFC XXXX

[[NOTE TO RFC EDITOR: Please change XXXX to the number assigned to this document upon publication.]]

7. Informative References

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

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6121] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence", RFC 6121, March 2011.

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.

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

[W3C.REC-xml-20081126] Sperberg-McQueen, C., Yergeau, F., Paoli, J., Bray, T., and E. Maler, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-

xml-20081126, November 2008,
<http://www.w3.org/TR/2008/REC-xml-20081126>.

[XEP-0124]
Paterson, I., Smith, D., Saint-Andre, P., and J. Moffitt, "Bidirectional-streams Over Synchronous HTTP (BOSH)", XSF XEP 0124, July 2010.

[XEP-0198]
Karneges, J., Saint-Andre, P., Hildebrand, J., Forno, F., Cridland, D., and M. Wild, "Stream Management", XSF XEP 0198, June 2011.

[XEP-0199]
Saint-Andre, P., "XMPP Ping", XSF XEP 0199, June 2009.

[XEP-0206]
Paterson, I. and P. Saint-Andre, "XMPP Over BOSH", XSF XEP 0206, July 2010.

Authors' Addresses

Lance Stout (editor)
&yet

Email: lance@andyet.net

Jack Moffitt

Email: jack@metajack.im

Eric Cestari
ProcessOne

Email: ecestari@process-one.com