Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 14, 2013

M. Hapner, Ed.
Huawei
C. Suconic
redhat

August 13, 2012

The MessageBroker WebSocket Subprotocol
draft-hapner-hybi-messagebroker-subprotocol-03

Abstract

The WebSocket protocol [I-D.ietf-hybi-thewebsocketprotocol] provides a subprotocol extension facility. The MessageBroker WebSocket Subprotocol (MBWS) is a WebSocket Subprotocol used by messaging clients to send messages to, and receive messages from an internet message broker (herein called a message broker). A message broker is a messaging intermediary that queues messages sent by its clients for asynchronous delivery to its clients.

Messages are addressed to message-broker-specific address names. Clients send messages to addresses and consume messages from addresses. Clients do not send messages directly to other clients.

Message brokers provide a range of functionality that is outside the scope of MBWS. Typically an internet message broker provides a REST API for working with this functionality; such as configuring client credentials; setting client access controls; configuring address routing; etc.

MBWS limits its scope to the definition of a WebSocket subprotocol that provides a full duplex, reliable message transport protocol between message brokers and their clients; and, between message brokers.

Since reliable message transport is often independent of a broker’s particular features, MBWS can be used as the message transport protocol for a wide range of message brokers.

The MBWS subprotocol defines a binary message frame and a text message frame. Both types of frame carry the same protocol; however, the protocol bindings differ slightly. The binary frame is a WebSocket binary message that contains an MBWS binary header followed by a binary message body. The text frame is a WebSocket UTF-8 text message that contains an MBWS text header followed by a text message body.

Status of this Memo

This Internet-Draft will expire on February 14, 2013.

Table of Contents

1. Introduction 4
2. MBWS Functionality 4
2.1. Connection Recovery 5
2.1.1. MBWS Connections 5
2.1.2. MBWS Connect 5
2.1.3. MBWS Message Sequencing 6
2.1.4. MBWS Reconnect 6
2.1.5. MBWS Prepare-to-close 7
2.1.6. MBLWS Connections 8
2.1.7. Message Metadata 8
2.1.7.1. Address List 8
2.1.7.1.1. Undeliverable Messages 8
2.1.7.2. Content-Type 9
2.1.7.3. Property List 9
3. Additional Issues 9
3.1. Sec-WebSocket-Protocol Field 9
3.2. Client Identity 9
3.3. Message Security 9
3.4. Empty Protocol Values 10
4. MBWS/MBLWS Protocol ABNF 10
5. Scenarios 11
5.1. MBWS Connection Recovery Scenario 11
5.2. MBLWS Session Scenario 12
6. Issues Outside the Scope of this Document 12
6.1. Messaging Scope 12
6.2. Message Acknowledgement Interval 13
6.3. Synchronous Messaging 13
6.4. End-to-End Reliability 13
7. References 13
Authors’ Addresses 14

1. Introduction

The WebSocket protocol [I-D.ietf-hybi-thewebsocketprotocol] provides a subprotocol extension facility. The MessageBroker WebSocket Subprotocol (MBWS) is a WebSocket Subprotocol used by messaging clients to send messages to, and receive messages from an internet message broker (herein called a message broker). A message broker is a messaging intermediary that queues messages sent by its clients for asynchronous delivery to its clients.

Messages are addressed to message-broker-specific address names. Clients send messages to addresses and consume messages from addresses. Clients do not send messages directly to other clients.

Message brokers provide a range of functionality that is outside the scope of MBWS. Typically an internet message broker provides a REST API for working with this functionality; such as configuring client credentials; setting client access controls; configuring address routing; etc.

MBWS limits its scope to the definition of a WebSocket subprotocol that provides a full duplex, reliable message transport protocol between message brokers and their clients; and, between message brokers.

Since reliable message transport is often independent of a broker’s particular features, MBWS can be used as the message transport protocol for a wide range of message brokers.

The MBWS subprotocol defines a binary message frame and a text message frame. Both types of frame carry the same protocol; however, the protocol bindings differ slightly. The binary frame is a WebSocket binary message that contains an MBWS binary header followed by a binary message body. The text frame is a WebSocket UTF-8 text message that contains an MBWS text header followed by a text message body.

2. MBWS Functionality

MBWS subprotocol defines two capabilities:

- Connection Recovery - the ability to support a logical, reliable connection that spans a sequence of WebSocket sessions
- Message Metadata - the ability to annotate a WebSocket message with metadata to support the functionality of a message broker

This document defines two subprotocols - MessageBroker WebSocket Subprotocol (MBWS) and MessageBrokerLight WebSocket Subprotocol

(MBLWS). MBWS supports both Connection Recovery and Message Metadata. MBLWS supports only Message Metadata.

The protocol description defines the logical MBWS and MBLWS subprotocols. The protocol ABNF [RFC5234] defines the binding of these protocols to MBWS binary frames and text frames. MBLWS uses the same frames as MBWS.

2.1. Connection Recovery

If a WebSocket session fails, the protocol does not define how the parties resolve what messages have been received and what messages have been lost. In many cases, this is not an issue; however, message brokers typically provide once-and-only-once QoS and WebSocket alone is not sufficient to support this.

MBWS defines a Connection Recovery subprotocol that allows a message broker client whose connection's session has failed to create a new WebSocket session that extends the connection and reliably resynchronizes its full duplex message transport such that no messages are lost or duplicated.

2.1.1. MBWS Connections

MBWS defines a connection that spans a sequence of one or more WebSocket sessions. During the time period between the failure of one of its sessions and the creation of its next session, its parties must maintain the state required to recover the connection. Since messages may be lost when a session fails, this state must contain a window of recently sent messages. MBWS provides support for identifying connections; maintaining recently sent message windows; recovering a connection on a new session; and, resynchronizing a recovered connection's message transport.

2.1.2. MBWS Connect

When a client requests a new MBWS connection it sends a Connect frame with an empty connection name. The server must respond with a Connect frame containing the name of a new connection. The MBWS client must retain this connection name so that it can be used later to recover this connection if this connection's current WebSocket session were to fail. It is recommended but not required that connection name be a URN.

Connection's are identified by a combination of client origin and connection name. Only the client origin that opened the connection can recover the connection.

An MBWS connection is closed by an MBWS Prepare-to-close followed by a WebSocket close.

2.1.3. MBWS Message Sequencing

MBWS requires clients and message brokers to use an implicit sequence numbering protocol for the messages transported by a connection. Each direction of transport defines a separate sequence. The first message sent by each endpoint is sequence number 1, the next is 2, etc. Since both parties are guaranteed to see the messages in the order sent, no explicit exchange of sequence numbers is required.

Both parties must acknowledge receipt of messages they receive. This is done by sending an Acknowledge control frame with the sequence number of the last message reliably received. When a sending endpoint receives an Acknowledge control frame from its receiving endpoint, the sending endpoint can delete from its message recovery window all messages with sequence numbers less than or equal to the Acknowledge sequence number.

If an MBWS session abnormally terminates, both the client and server should retain the state of the MBWS connection so that it can be resynchronized and continued on a new session. The client of a failed MBWS connection session has the option of reconnecting and continuing the existing connection; or, creating a new connection. Upon receipt of a new connection request, a server will clear the state of an existing MBWS connection if such exists. Upon receipt of reconnect request, a server will, if possible, resynchronize and continue the existing connection.

2.1.4. MBWS Reconnect

A client requests a connection reconnect by sending a Connect frame containing the name of the connection to be reconnected followed by a list of three message sequence numbers. The first sequence number (CSLR) is that of the last message the client has received. The second (CSLW) and third (CSUW) sequence numbers define the respective lower and upper bounds of the sequence numbers of the messages in the client’s retained message window. Upon receipt of this Connect frame, the server determines if it can reconnect based on the following criteria:

1. The connection name must match the client’s current MBWS connection name.
2. CSLR+1 must be the sequence number of a message the server can reinitiate sending with (i.e. either the message with this sequence number is in the connection’s retained messsage window or it is the sequence number of the next unsent message).

3. The message sequence number of the message the server last received (SSLR) is in the range of CSLW-1 to CSUW.

If all three criteria are met the reconnect succeeds and the server responds with a Connect frame containing the reconnected connection name and one sequence number (SSLR) which is that of the last message received by the server. Message transport then resumes with the client sending the SSLR+1 message and the server sending the CSLR+1 message.

If the criteria are not met, the reconnect request fails and the server treats it as though it were a connect request and responds with a connect response. The client recognizes that its reconnect request has been converted into a connect request because the response contains a connection name that does not match that in the client’s reconnect request

2.1.5. MBWS Prepare-to-close

MBWS adds a Prepare-to-close phase that immediately precedes the WebSocket close phase. This is done to allow both endpoints to acknowledge the receipt of the last message sent to them prior to initiating WebSocket close. The endpoints must retain the MBWS connection state until the WebSocket close has completed. Once the connection has entered the WebSocket close phase, the ability to Reconnect is unreliable; however, by this point both endpoints have acknowledged all messages sent and the failure of a Reconnect request will not result in messages being lost or duplicated. The steps of the MBWS Prepare-to-close phase are as follows:

1. Endpoint-1 sends a Prepare-to-close control frame which signals that it has sent its last message and will initiate a WebSocket close when the prepare-to-close phase is complete.
2. Endpoint-2 receives the Prepare-to-close control frame. It then sends an Acknowledge control frame with the sequence number of the last message it has received. This is done whether or not this message sequence number has been acknowledged previously.
3. Endpoint-2 sends its last message, if any, followed by a Prepare-to-close control frame.
4. Endpoint-1 receives the last messages in transit, if any, followed by the Prepare-to-close control frame. It then sends an Acknowledge control frame with the sequence number of the last message it has received. This is done whether or not this message sequence number has been acknowledged previously.
5. Endpoint-1 initiates a WebSocket close by sending a WebSocket close control frame.

It is possible and acceptable that both endpoints initiate Prepare-to-close at nearly the same time. If so, this may result in both

endpoints initiating a WebSocket close at nearly the same time.

2.1.6. MBLWS Connections

An MBLWS client does not use Connect, Acknowledge or Prepare-to-close control frames. Message transport begins immediately after the WebSocket upgrade request has been accepted by the server. MBLWS does not support connection recovery. MBLWS connections do not span WebSocket sessions. If an MBLWS connection's WebSocket session fails or is closed, the connection is closed.

2.1.7. Message Metadata

MBWS and MBLWS define a message header containing three metadata elements. In order, these are Address List, Content-Type and Property List.

2.1.7.1. Address List

For messages sent by a client to a broker, the Address List contains the list of destination Addresses to which to send the message. Empty Addresses are ignored. For messages delivered by a message broker to a client, Address List contains the single address from which the message originated, .i.e if a client receives the same message from multiple addresses it will receive each as a separate message.

It is recommended but not required that address value be a URN.

The format and semantics of Address is message broker dependent and is outside the scope of MBWS. For instance, some brokers may treat Address as a strictly local name; other brokers may support a more global form of addressing. Broker-specific message routing semantics determine how a destination Address's messages are to be routed and how message's origination Address is determined. This includes defining the meaning of an empty destination Address List and an empty origination Address.

2.1.7.1.1. Undeliverable Messages

A messages's Address may not be known to a broker. MBWS does not define how such dead-letters are handled once they are received by a message broker. MBWS requires a message broker to acknowledge every message sent to it, whether or not it can deliver it.

2.1.7.2. Content-Type

Immediately following Address List, a message header contains a Content-Type. Its value is a UTF-8 string containing the MIME discrete type [RFC2045] that describes the message's content. Content-Type may be empty.

2.1.7.3. Property List

Immediately following Content-Type, a message header contains a Property List. This list contains zero or more Properties. Each Property is a Name/Value pair with each being a UTF-8 string. MBWS does not define the semantics of Properties.

3. Additional Issues

3.1. Sec-WebSocket-Protocol Field

Sec-WebSocket-Protocol Field Values

Value
MBWS.huawei.com
MBLWS.huawei.com

WebSocket defines the subprotocol negotiation process. This starts with a client including the Sec-WebSocket-Protocol Field with one or more subprotocol names in its WebSocket upgrade request. The table above specifies the values for the two subprotocols defined in this document.

3.2. Client Identity

WebSocket uses the HTTP origin model to identify clients. MBWS uses the same client identity model.

3.3. Message Security

WebSocket supports TLS and MBWS/MBLWS recommends, but does not require, its use. In addition to providing better security, the use of TLS and port 443 insures that MBWS connections avoid the overhead and latency of having to traverse web proxies.

3.4. Empty Protocol Values

In several places, this document refers to an 'empty' UTF-8 string element. In MBWS, UTF-8 string protocol elements are length-delimited. An 'empty' element is one with a zero valued length delimiter.

4. MBWS/MBLWS Protocol ABNF

```
mbws-frame = binary-frame / text-frame
;the frame used with a WS binary message
binary-frame =
    binary-connect-frame / binary-acknowledge-frame / binary-prepare-to-clos
e-frame / binary-message-frame
binary-connect-frame =
    binary-connect-frame-id binary-connection-name binary-message-sequence-n
umber-list
binary-connect-frame-id = %x01
binary-connection-name = binary-string
binary-message-sequence-number-list =
    binary-list-length *binary-message-sequence-number
binary-acknowledge-frame =
    binary-acknowledge-frame-id binary-message-sequence-number
binary-acknowledge-frame-id = %x02
binary-message-sequence-number = varint
binary-prepare-to-close-frame = binary-prepare-to-close-frame-id
binary-prepare-to-close-frame-id = %x03
binary-message-frame =
    binary-message-frame-id binary-message-header binary-message-body
binary-message-frame-id = %x03
binary-message-header =
    binary-address-list binary-content-type binary-property-list
binary-address-list = binary-list-length *binary-address
binary-address = binary-string
binary-content-type = binary-string
binary-property-list = binary-list-length *binary-property
binary-property = binary-property-name binary-property-value
binary-property-name = binary-string
binary-property-value = binary-string
binary-message-body = *OCTET
;the frame used with a WS text message
text-frame =
    text-connect-frame / text-acknowledge-frame / text-prepare-to-close-fram
e / text-message-frame
text-connect-frame =
    text-connect-frame-id text-connection-name text-message-sequence-number-
list
text-connect-frame-id = %x31 SP
text-connection-name = text-string
text-message-sequence-number-list =
```

```
    text-list-length *text-message-sequence-number
text-acknowledge-frame =
    text-acknowledge-frame-id text-message-sequence-number
text-acknowledge-frame-id = %x32 SP
text-message-sequence-number = text-int
text-prepare-to-close-frame = text-prepare-to-close-frame-id
text-prepare-to-close-frame-id = %x33 SP
text-message-frame =
    text-message-frame-id text-message-header text-message-body
text-message-frame-id = %x33 SP
text-message-header =
    text-address-list text-content-type text-property-list
text-address-list = text-list-length *text-address
text-address = text-string
text-content-type = text-string
text-property-list = text-list-length *text-property
text-property = text-property-name text-property-value
text-property-name = text-string
text-property-value = text-string
text-message-body = UTF8-string
;UTF8 encoded character string
UTF8-string = *(OCTET)
;Google Protocol Buffers base 128 varint
varint = 1*8(OCTET)
;the number of characters in a UTF8 string
binary-string-length = varint
binary-string = binary-string-length UTF8-string
;the number of entries in a list
binary-list-length = varint
text-int = DIGIT *DIGIT SP
;the number of characters in a UTF8 string
text-string-length = text-int
text-string = text-string-length UTF8-string
;the number of entries in a list
text-list-length = text-int ;the number of entries in a list
```

Figure 1

5. Scenarios

5.1. MBWS Connection Recovery Scenario

1. Broker provides ’ws:’ and/or ’wss:’ URIs for accepting MBWS connections.
2. Client establishes an HTTP session with Broker; identifies itself using HTTP client origin; and, authenticates itself using HTTP authentication.

3. If successful, Client requests HTTP upgrade to MBWS Subprotocol.
4. If upgrade successful, Client sends Connect frame with empty connection name.
5. Broker responds with Connect frame containing a new connection name.
6. Broker starts streaming messages to client; and, Client starts streaming messages to Broker.
7. Client and Broker periodically acknowledge receipt of each other’s messages using Acknowledge control frames.
8. Client or Broker may initiate session close as defined by WebSocket.
9. If session abnormally terminates, client recovers connection by executing (1) through (3) and then continues with (10)
10. Client sends Connect frame containing the connection name it wishes to recover and the CSLR, CSLW and CSUW message sequence numbers.
11. Broker responds with Connect frame. If Connect frame contains a new connection name, broker has rejected recovery and opened a new connection, processing continues with (6). If Connect frame contains the recovery connection name and SSLR sequence number, Broker has accepted recovery.
12. Processing continues at (6) with the Client initiating sending with the SSLR+1 message; and, the Broker initiating sending with the CSLR+1 message.

5.2. MBLWS Session Scenario

1. Broker provides ’ws:’ and/or ’wss:’ URIs for accepting MBLWS sessions.
2. Client establishes an HTTP session with Broker; identifies itself using HTTP client origin; and, authenticates itself using HTTP authentication.
3. If successful, Client requests HTTP upgrade to MBWS Subprotocol.
4. If upgrade successful, Broker starts streaming available messages to client; and, Client starts streaming messages to Broker.
5. Client or Broker may initiate session close as defined by WebSocket.

6. Issues Outside the Scope of this Document

This section is non-normative.

6.1. Messaging Scope

Message brokers provide message-broker-specific functionality for routing, queueing, forwarding, filtering, transporting, etc. messages. This results in the broker delivering specific messages to

specific clients. This document defines how a message broker uses the subprotocols defined here to transport messages to/from a client. All other message broker functionality is outside the scope of this document.

6.2. Message Acknowledgement Interval

The parties of an MBWS connection decide when to send Acknowledge control frames. Typically these are sent after some number of messages have been received or some time interval has elapsed within which at least one message has been received. The choice of acknowledgement interval is outside the scope of this document.

6.3. Synchronous Messaging

Message brokers have a history of supporting synchronous messaging where clients make blocking calls to send and to receive messages. WebSocket and MBWS are natively asynchronous messaging protocols. MBWS is optimized for asynchronous, full duplex message transport. It has not been designed for synchronous messaging.

6.4. End-to-End Reliability

The responsibility for reliable message delivery over a MBWS connection is not the responsibility of the message broker alone - it is only achieved when both clients and brokers implement recovery of MBWS connections. The degree to which clients and message brokers are able to recover from failure is outside the scope of this document.

7. References

[I-D.ietf-hybi-thewebsocketprotocol]
Fette, I. and A. Melnikov, "The WebSocket protocol", draft-ietf-hybi-thewebsocketprotocol-17 (work in progress), September 2011.

[GPBE] "Google Protocol Buffers Encoding <http://code.google.com/apis/protocolbuffers/docs/encoding.html>".

[RFC2045] Freed, N. and Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", November 1966.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", January 2008.

Authors' Addresses

Mark Hapner (editor)
Huawei

Email: mhapner@huawei.com

Clebert Suconic
redhat

Email: csuconic@redhat.com