Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 28, 2012

M. Hapner, Ed.
Huawei
C. Suconic
redhat

March 27, 2012

The MessageBroker WebSocket Subprotocol
draft-hapner-hybi-messagebroker-subprotocol-01

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

Copyright Notice

Table of Contents

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 (herein, such a connection is called a ’connection’)
- 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

When a WebSocket is normally closed, both client and server can assume the messages they sent/delivered have been received by the other party.

The reliability of the WebSocket's underlying TCP connection, combined with the WebSocket close protocol, insures that both parties have implicitly acknowledged the receipt of the all messages they have been sent.

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 Connection Name and Connection Recovery

When a client requests a new 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 recovery this connection if this connection's current WebSocket session were to fail. If a connection's WebSocket session is closed, as defined by the WebSocket close protocol, it also closes the connection. It is recommended but not required that connection name be a URN.

When a client requests the recovery of a connection, it sends a Connect frame containing the name of the connection to be recovered. The message broker must then respond with a Connect frame containing a connection name. If this connection name matches the value sent by the client, the server has accepted the recovery request. If the name does not match, the server has rejected the recovery request and has opened a new connection.

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.

2.1.3. Message Synchronization of a Recovered MBWS Connection

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 party 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 frame with the sequence number of the last message reliably received. When a sending party receives an Acknowledge frame from its receiving party, the sending party can delete from its message recovery window all messages with sequence numbers less than or equal to the Acknowledge sequence number.

If a session abnormally terminates and a message broker accepts a client's request to recover the connection, both client and message broker must verify that they can resume sending messages with the message sequence number required by each. Connection recovery message resynchronization is a serial two phase process. First, the client provides the message broker with the information required for the broker to restart message delivery. This phase is named broker-message-delivery-resync. Second, the message broker provides the client with information required for the client to restart message delivery. This phase is named client-message-delivery-resync.

If both broker-message-delivery-resync and client-message-delivery-resync succeed, the connection has been recovered. If either fails,

a new connection is opened.

2.1.3.1. Broker-message-delivery-resync

The client sends an Acknowledge frame containing the sequence number of the last message it has received. The message broker validates it can resume sending with the next message in sequence.

If so, the message broker must reply with a Connect frame containing the connection name being recovered. Connection recovery then proceeds with the client-message-deliveryresync phase.

If the message broker cannot restart with this message, it must reply with a Connect frame with a new connection name. Message transport then begins on this new connection.

2.1.3.2. Client-message-delivery-resync

The message broker sends the client an Acknowledge frame containing the sequence number of the last message it has received. The client validates it can resume sending with the next message in sequence.

If so, the client must reply with a Connect frame containing the connection name being recovered. This completes a successful connection recovery and normal full duplex message transport resumes.

If the client cannot restart with this message, it must reply with a Connect frame containing an empty connection name. The message broker must then respond with a Connect frame containing a new connection name. Message transport then begins on this new connection.

2.1.4. MBLWS Connections

An MBLWS client opens a new connection with the same Connect frame protocol as used by an MBWS client. MBLWS does not support connection recovery. MBLWS connections do not span WebSocket sessions. If an MBLWS client sends a Connect frame containing a connection name, the connection name must be ignored and a new connection must be opened. MBLWS connections do not use Acknowledgement frames. If an MBLWS client sends an Acknowledgement frame, it must be ignored. If connection’s WebSocket session fails or is closed, the connection is closed.

2.1.5. Message Metadata

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

Property List.

2.1.5.1. 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 addresses from which the message originated.

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 Addresses are determined. This includes defining the meaning of an empty destination Address List and an empty origination Address List.

2.1.5.1.1. Undeliverable Messages

An 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.

2.1.5.2. 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.

2.1.5.3. 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, the protocol 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-message-frame
binary-connect-frame = binary-connect-frame-id binary-connection-name
binary-connect-frame-id = %x01
binary-connection-name = binary-string
binary-acknowledge-frame =
```

```
    binary-acknowledge-frame-id binary-message-sequence-number
binary-acknowledge-frame-id = %x02
binary-message-sequence-number = varint
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-message-frame
text-connect-frame = text-connect-frame-id text-connection-name
text-connect-frame-id = %x31 SP
text-connection-name = text-string
text-acknowledge-frame =
    text-acknowledge-frame-id text-message-sequence-number
text-acknowledge-frame-id = %x32 SP
text-message-sequence-number = text-int
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 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 connection name it wishes to recover
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 recovery connection name, broker has accepted recovery.
12. Client sends Acknowledge frame containing the sequence number of the last message it has received.
13. 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 recovery connection name, broker has accepted recovery.
14. Broker sends Acknowledge frame containing the sequence number of the last message it has received.

15. Client responds with Connect frame. If Connect frame contains an empty connection name, client has rejected recovery and processing continues with (5). If the connection name is the recovery connection name, processing continues at (6)

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. Broker starts streaming available messages to client; and, Client starts streaming messages to Broker.
4. 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 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 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