HyBi Working Group T. Yoshino
Internet-Draft Google, Inc.
Intended status: Standards Track June 19, 2013
Expires: December 21, 2013

Compression Extensions for WebSocket
draft-ietf-hybi-permessage-compression-10

Abstract

This document specifies a framework for creating WebSocket extensions that add compression functionality to the WebSocket Protocol. An extension based on this framework compresses the payload data portion of non-control WebSocket messages on a per-message basis using parameters negotiated during the opening handshake. This framework provides a general method to apply a compression algorithm to the contents of WebSocket messages. For each compression algorithm, an extension is defined by specifying parameter negotiation and compression algorithm in detail. This document also specifies one specific compression extension using the DEFLATE algorithm.

Please send feedback to the hybi@ietf.org mailing list.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document specifies a framework to add compression functionality to the WebSocket Protocol [RFC6455]. This framework specifies how to define WebSocket Per-message Compression Extensions (PMCEs) individually for various compression algorithms based on the extension concept of the WebSocket Protocol specified in Section 9 of [RFC6455]. A WebSocket client and a peer WebSocket server negotiate use of a PMCE and determine parameters to configure the compression algorithm during the WebSocket opening handshake. The client and server can then exchange non-control messages using frames with compressed data in the payload data portion. This framework specifies a general method to apply a compression algorithm to the contents of WebSocket messages. A document specifying an individual PMCE describes how to negotiate configuration parameters for the compression algorithm and how to transform (compress and decompress) data in the payload data portion in detail. A WebSocket client may offer multiple PMCEs during the WebSocket opening handshake. A peer WebSocket server received those offers may choose and accept preferred one or decline all of them. PMCEs use the RSV1 bit of the WebSocket frame header to indicate whether a message is compressed or not, so that an endpoint can choose not to compress messages with incompressible contents.

This document also specifies one specific PMCE based on the DEFLATE [RFC1951] algorithm. The extension name of the PMCE is "permessage-deflate". We chose DEFLATE since it's widely available as a library on various platforms and the overhead is small. To align the end of compressed data to an octet boundary, this extension uses the algorithm described in Section 2.1 of the PPP Deflate Protocol [RFC1979]. Endpoints can take over the LZ77 sliding window [LZ77] used to build frames for previous messages to get better compression ratio. For resource-limited devices, this extension provides parameters to limit memory usage for compression context.

2. 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.

This document references the procedure to _Fail the WebSocket Connection_. This procedure is defined in Section 7.1.7 of [RFC6455].

This document references the event that _The WebSocket Connection is Established_ and the event that _A WebSocket Message Has Been Received_. These events are defined in Section 4.1 and Section 6.2, respectively, of [RFC6455].

This document uses the Argumented Backus-Naur Form (ABNF) notation of [RFC5234]. The DIGIT (decimal 0-9) rule is included by reference, as defined in the Appendix B.1 of [RFC5234].

3. Complementary Terminology

This document defines some terms about WebSocket and WebSocket Extension mechanism that are underspecified or not defined at all in RFC6455. This terminology is effective only in this document and any other documents that refer to this section.

Non-control message means a message consists of non-control frames.

Message payload (or payload of a message) means concatenation of the payload data portion of all frames consisting the message.

Extension in use next to extension X means the extension listed next to X in the "Sec-WebSocket-Extensions" header in the server's opening handshake. Such an extension is applied to outgoing data from the application right after X on sender side but applied right before X to incoming data from the underlying transport.

Accept an extension offer means including a corresponding response in the "Sec-WebSocket-Extensions" header in the server's opening handshake.

Decline an extension offer means not including a corresponding response in the "Sec-WebSocket-Extensions" header in the server's opening handshake.

4. WebSocket Per-message Compression Extension

WebSocket Per-message Compression Extensions (PMCEs) are extensions to the WebSocket Protocol enabling compression functionality. PMCEs are built based on Section 9 of [RFC6455]. PMCEs are individually defined for each compression algorithm to be implemented, and are registered in the WebSocket Extension Name Registry created in Section 11.4 of [RFC6455]. Each PMCE refers to this framework and defines the following:

- The content to put in the "Sec-WebSocket-Extensions" header. The content includes the extension name of the PMCE and any applicable extension parameters.

- How to interpret extension parameters exchanged during the opening handshake

- How to transform the payload of a message.

One such extension is defined in Section 8 of this document and is registered in Section 10. Other PMCEs may be defined in other documents.

Section 5 describes the basic extension negotiation process. Section 6 describes how to apply the compression algorithm with negotiated parameters to the contents of WebSocket messages.

5. Extension Negotiation

To offer use of a PMCE, a client includes a "Sec-WebSocket-Extensions" header element with the extension name of the PMCE in the "Sec-WebSocket-Extensions" header in the client's opening handshake of the WebSocket connection. Extension parameters in the element represent the PMCE offer in detail. For example, a client lists preferred configuration parameter values for the compression algorithm of the PMCE. A client offers multiple PMCE choices to the server by including multiple elements in the "Sec-WebSocket-Extensions" header, one for each PMCE offered. The set of elements MAY include multiple PMCEs with the same extension name to offer use of the same algorithm with different configuration parameters. The order of elements means the client's preference. An element precedes another element has higher preference. It is RECOMMENDED that a server accepts PMCEs with higher preference if the server supports it.

To accept use of an offered PMCE, a server includes a "Sec-WebSocket-Extensions" header element with the extension name of the PMCE in the "Sec-WebSocket-Extensions" header in the server's opening handshake of the WebSocket connection. Extension parameters in the element represent the configuration parameters of the PMCE to use in detail. We call these extension parameters and their values "agreed parameters". The element MUST represent a PMCE that is fully supported by the server. The contents of the element doesn't need to be exactly the same as one of the received offers. For example, an offer with an extension parameter "X" indicating availability of the feature X may be accepted with an element without the extension parameter meaning that the server declined use of the feature.

A server MUST NOT accept a PMCE offer together with another extension if the PMCE will conflict with the extension on use of the RSV1 bit. A client that receives a response accepting a PMCE offer together with such an extension MUST _Fail the WebSocket Connection_.

A server MUST NOT accept a PMCE offer together with another extension if the PMCE will be applied to output of the extension and any of the following conditions applies to the extension:

- The extension requires boundary of fragments to be preserved between output from the extension at the sender and input to the extension at the receiver.

- The extension uses the "Extension data" field or any of the reserved bits on the WebSocket header as per-frame attribute.

A client receiving a response accepting a PMCE offer together with

such an extension MUST _Fail the WebSocket Connection_.

A server declines all offered PMCEs by not including any element with PMCE names. If a server responds with no PMCE element in the "Sec-WebSocket-Extensions" header, both endpoints proceed without Per-message Compression once _the WebSocket Connection is established_.

If a server gives an invalid response, such as accepting a PMCE that the client did not offer, the client MUST _Fail the WebSocket Connection_.

If a server responds with a valid PMCE element in the "Sec-WebSocket-Extensions" header and _the WebSocket Connection is established_, both endpoints MUST use the algorithm described in Section 6 to exchange messages, using the message payload transformation (compressing and decompressing) procedure of the PMCE returned by the server.

5.1. Negotiation Examples

The followings are example values for the "Sec-WebSocket-Extensions" header offering PMCEs. permessage-foo and permessage-bar in the examples are hypothetical extension names of PMCEs for compression algorithm foo and bar.

o Offer the permessage-foo.

```
permessage-foo
```

o Offer the permessage-foo with a parameter x with a value of 10.

```
permessage-foo; x=10
```

   The value MAY be quoted.

```
permessage-foo; x="10"
```

o Offer the permessage-foo as first choice and the permessage-bar as a fallback plan.

```
permessage-foo, permessage-bar
```

o Offer the permessage-foo with a parameter use_y which enables a feature y as first choice, and the permessage-foo without the use_y parameter as a fallback plan.

```
permessage-foo; use_y, permessage-foo
```

6. Framing

PMCEs operate only on non-control messages.

This document allocates the RSV1 bit of the WebSocket header for PMCEs, and calls the bit the "Per-message Compressed" bit. On a WebSocket connection where a PMCE is in use, this bit indicates whether a message is compressed or not.

A message with the "Per-message Compressed" bit set on the first fragment of the message is called a "compressed message". Frames of a compressed message have compressed data in the payload data portion. An endpoint receiving a compressed message decompresses the concatenation of the compressed data of the frames of the message by following the decompression procedure specified by the PMCE in use. The endpoint uses the bytes corresponding to the application data portion in this decompressed data for the _A WebSocket Message Has Been Received_ event instead of the received data as-is.

A message with the "Per-message Compressed" bit unset on the first fragment of the message is called an "uncompressed message". Frames of an uncompressed message have uncompressed original data as-is in the payload data portion. An endpoint received an uncompressed message uses the concatenation of the application data portion of the frames of the message as-is for the _A WebSocket Message Has Been Received_ event.

6.1. Compression

An endpoint MUST use the following algorithm to send a message in the form of a compressed message.

1. Compress the message payload of the original message by following the compression procedure of the PMCE. The original message may be input from the application layer or output of another WebSocket extension depending on what extensions were negotiated.

2. If this PMCE is the last extension to process outgoing messages, build frame(s) by using the compressed data instead of the original data for the message payload, and setting the "Per-message Compressed" bit of the first frame, then send the frame(s) as described in Section 6.1 of RFC6455. Otherwise, pass the transformed message payload and modified header values including "Per-message Compressed" bit value set to 1 to the extension next to the PMCE. If the extension expects frames as input, build a frame for the message and pass it.

An endpoint MUST use the following algorithm to send a message in the

form of an uncompressed message. If this PMCE is the last extension to process outgoing messages, build frame(s) by using the original data for the payload data portion as-is and unsetting the "Per-message Compressed" bit of the first frame, then send the frame(s) as described in Section 6.1 of RFC6455. Otherwise, pass the message payload and header values to the extension next to the PMCE as-is. If the extension expects frames as input, build a frame for the message and pass it.

An endpoint MUST NOT set the "Per-message Compressed" bit of control frames and non-first fragments of a data message. An endpoint received such a frame MUST _Fail the WebSocket Connection_.

PMCEs don't change the opcode field. The opcode of the first frame of a compress message indicates the opcode of the original message.

The payload data portion in frames generated by a PMCE is not subject to the constraints for the original data type. For example, the concatenation of the data corresponding to the application data portion of frames of a compressed text message is not required to be valid UTF-8. At the receiver, the payload data portion after decompression is subject to the constraints for the original data type again.

6.2. Decompression

An endpoint MUST use the following algorithm to receive a message in the form of a compressed message.

1. Concatenate the payload data portion of the received frames of the compressed message. The received frames may direct input from underlying transport or output of another WebSocket extension depending on what extensions were negotiated.

2. Decompress the concatenation by following the decompression procedure of the PMCE.

3. If this is the last extension to process incoming messages, deliver the _A WebSocket Message Has Been Received_ event to the application layer with the decompressed message payload and header values including the "Per-message Compressed" bit unset to 0. Otherwise, pass the decompressed message payload and header values including the "Per-message Compressed" bit unset to 0 to the extension next to the PMCE. If the extension expects frames as input, build a frame for the message and pass it.

An endpoint MUST use the following algorithm to receive a message in the form of an uncompressed message. If this PMCE is the last

extension to process incoming messages, deliver the _A WebSocket Message Has Been Received_ event to the application layer with the received message payload and header values as-is. Otherwise, pass the message payload and header values to the extension next to the PMCE as-is. If the extension expects frames as input, build a frame for the message and pass it.

7. Intermediaries

When an intermediary proxies a WebSocket connection, the intermediary MAY add, change or remove Per-message Compression on the messages. Such a change must not be made if the new combination of extensions after the change doesn't conform to the constraints of the extensions. The elements in the "Sec-WebSocket-Extensions" for the PMCE in the opening handshakes with the connected client and server must be altered by the intermediary accordingly to match the new combination of extensions.

8. permessage-deflate extension

This section specifies a specific PMCE called "permessage-deflate". It compresses the payload of a message using the DEFLATE algorithm [RFC1951] and the byte boundary aligning method introduced in [RFC1979].

This section uses the term "byte" with the same meaning as RFC1951, i.e. 8 bits stored or transmitted as a unit (same as an octet).

The registered extension name for this extension is "permessage-deflate".

For an offer for this extension, the following 4 extension parameters are defined. If needed to distinguish from ones for a response, these parameters are called with a prefix "client-to-server".

o "s2c_no_context_takeover"

o "c2s_no_context_takeover"

o "s2c_max_window_bits"

o "c2s_max_window_bits"

For a response for this extension, the following 4 extension parameters are defined. If needed to distinguish from ones for an offer, these parameters are called with a prefix "server-to-client".

o "s2c_no_context_takeover"

o "c2s_no_context_takeover"

o "s2c_max_window_bits"

o "c2s_max_window_bits"

A server MUST decline an offer for this extension if any of the following conditions is met:

o The offer has any extension parameter not defined for use in an offer.

o The offer has any extension parameter with an invalid value.

o The offer has multiple extension parameters with the same name.

o The server doesn't support the offered configuration.

A client MUST _Fail the WebSocket Connection_ if the peer server accepted an offer for this extension with a response meeting any of the following condition:

o The response has any extension parameter not defined for use in a response.

o The response has any extension parameter with an invalid value.

o The response has multiple extension parameters with the same name.

o The client doesn't support the configuration the response represents.

The term "LZ77 sliding window" used in this section means the buffer storing recently processed input. The LZ77 algorithm searches the buffer for match with the next input.

8.1. Method Parameters

8.1.1. Context Takeover Control

8.1.1.1. s2c_no_context_takeover

A client MAY include the "s2c_no_context_takeover" extension parameter to an offer. This parameter has no value. Using this parameter, a client prevents the peer server from using the same LZ77 sliding window it used to build frames of the last sent message to build frames of the next message. If the peer server doesn't use the same LZ77 sliding window to compress multiple messages, the client doesn't need to reserve memory to retain the LZ77 sliding window in between messages.

A server accepts an offer with the "s2c_no_context_takeover" extension parameter by including the "s2c_no_context_takeover" extension parameter in the response. This server-to-client parameter has no value.

It is RECOMMENDED that a server supports the client-to-server "s2c_no_context_takeover" extension parameter.

A server MAY include the "s2c_no_context_takeover" extension parameter to a response even if the offer to accept by the response doesn't have "s2c_no_context_takeover" extension parameter.

8.1.1.2. c2s_no_context_takeover

A client MAY include the "c2s_no_context_takeover" extension parameter to an offer. This parameter has no value. Using this parameter, a client tells the peer server a hint that the client is not likely to use the same LZ77 sliding window it used to build frames of the last sent message to build frames of the next message.

A server MAY include the "c2s_no_context_takeover" extension parameter to a response. This parameter has no value. Using this parameter, a server prevents the peer client from using the same LZ77 sliding window it used to build frames of the last sent message to build frames for the next message. This reduces the amount of memory that the server has to reserve for the connection, in the same way the "s2c_no_context_takeover" extension parameter does for the client.

A client MUST support server-to-client "c2s_no_context_takeover" extension parameter.

8.1.2. Limiting the LZ77 sliding window size

8.1.2.1. s2c_max_window_bits

A client MAY include the "s2c_max_window_bits" extension parameter to an offer. This parameter has a decimal integer value without leading zeroes between 8 to 15 inclusive indicating the base-2 logarithm of the LZ77 sliding window size.

```
s2c_max_window_bits = 1*DIGIT
```

Using this parameter, a client limits the LZ77 sliding window size that the server uses to compress messages. If the peer server doesn't use small LZ77 sliding window to compress messages, the client can reduce the memory for the LZ77 sliding window.

A server declines an offer with this parameter if the server doesn't support it.

A server accepts an offer with this extension parameter by including the "s2c_max_window_bits" extension parameter with the same or smaller value as the offer in the response. This server-to-client parameter has a decimal integer value without leading zeroes between 8 to 15 inclusive indicating the base-2 logarithm of the LZ77 sliding window size.

```
s2c_max_window_bits = 1*DIGIT
```

A server MAY include the "s2c_max_window_bits" extension parameter to a response even if the offer to accept by the response doesn't have "s2c_max_window_bits" extension parameter.

8.1.2.2. c2s_max_window_bits

A client MAY include the "c2s_max_window_bits" extension parameter to an offer. This parameter has no value or a decimal integer value without leading zeroes between 8 to 15 inclusive indicating the base-2 logarithm of the LZ77 sliding window size. Using this parameter, a client tells the peer server that the client supports the server-to-client "c2s_max_window_bits" extension parameter. If the parameter has a value, the parameter also tells the peer server a hint that the client is not likely to use LZ77 sliding window size greater than the size specified by the value to compress messages.

If a received offer has the "c2s_max_window_bits" extension parameter, the server MAY include the "c2s_max_window_bits" parameter in the response to the offer. This server-to-client parameter has a decimal integer value without leading zeroes between 8 to 15 inclusive indicating the base-2 logarithm of the LZ77 sliding window size.

```
c2s_max_window_bits = 1*DIGIT
```

Using this server-to-client parameter, a server limits the LZ77 sliding window size that the client uses to compress messages. This reduces the amount of memory that the server has to reserve for the connection, in the same way the "s2c_max_window_bits" extension parameter does for the client.

If a received offer doesn't have the "c2s_max_window_bits" extension parameter, the server MUST NOT include the "c2s_max_window_bits" extension parameter in the response to the offer.

8.1.3. Example

The simplest "Sec-WebSocket-Extensions" header in a client's opening handshake to offer use of the permessage-deflate is the following:

```
Sec-WebSocket-Extensions: permessage-deflate
```

Since the "c2s_max_window_bits" extension parameter is not specified, the server may not accept the offer with the "c2s_max_window_bits" extension parameter. The simplest "Sec-WebSocket-Extensions" header in a server's opening handshake to accept use of the permessage-deflate is the same.

The following offer sent by a client is asking the server to use the LZ77 sliding window size of 1,024 bytes or less and declaring that the client can accept the "c2s_max_window_bits" extension parameter.

```
Sec-WebSocket-Extensions:
    permessage-deflate;
    c2s_max_window_bits; s2c_max_window_bits=10
```

This offer might be rejected by the server because the server doesn't support the "s2c_max_window_bits" extension parameter. This is fine if the client cannot support a larger sliding window size, but if the client wants to fallback to the "permessage-deflate" without the "s2c_max_window_bits" option, the client should offer the fallback option explicitly like this:

```
Sec-WebSocket-Extensions:
    permessage-deflate;
    c2s_max_window_bits; s2c_max_window_bits=10,
    permessage-deflate;
    c2s_max_window_bits
```

This example offers two configurations so that the server can accept permessage-deflate by picking a supported one. To accept the first option, the server might send back, for example:

```
Sec-WebSocket-Extensions:
    permessage-deflate; s2c_max_window_bits=10
```

And to accept the second option, the server might send back, for example:

```
Sec-WebSocket-Extensions: permessage-deflate
```

8.2. Message Payload Transformation

8.2.1. Compression

An endpoint uses the following algorithm to compress a message.

1. Compress all the octets of the payload of the message using DEFLATE.

2. If the resulting data does not end with an empty DEFLATE block with no compression (the "BTYPE" bits is set to 00), append an empty DEFLATE block with no compression to the tail end.

3. Remove 4 octets (that are 0x00 0x00 0xff 0xff) from the tail end. After this step, the last octet of the compressed data contains

(possibly part of) the DEFLATE header bits with the "BTYPE" bits set to 00.

In using DEFLATE in the first step above:

- An endpoint MAY use multiple DEFLATE blocks to compress one message.

- An endpoint MAY use DEFLATE blocks of any type.

- An endpoint MAY use both DEFLATE blocks with the "BFINAL" bit set to 0 and DEFLATE blocks with the "BFINAL" bit set to 1.

- When any DEFLATE block with the "BFINAL" bit set to 1 doesn't end at byte boundary, an endpoint adds minimal padding bits of 0 to make it end at byte boundary. The next DEFLATE block follows the padded data if any.

An endpoint MUST NOT use an LZ77 sliding window longer than 32,768 bytes to compress messages to send.

If the "agreed parameters" contain the "c2s_no_context_takeover" extension parameter, the client MUST start compressing each new message with an empty LZ77 sliding window. Otherwise, the client MAY take over the LZ77 sliding window used to build the last compressed message.

If the "agreed parameters" contain the "s2c_no_context_takeover" extension parameter, the server MUST start compressing each new message with an empty LZ77 sliding window. Otherwise, the server MAY take over the LZ77 sliding window used to build the last compressed message.

If the "agreed parameters" contain the "c2s_max_window_bits" extension parameter with a value of w, the client MUST NOT use an LZ77 sliding window longer than the w-th power of 2 bytes to compress messages to send.

If the "agreed parameters" contain the "s2c_max_window_bits" extension parameter with a value of w, the server MUST NOT use an LZ77 sliding window longer than the w-th power of 2 bytes to compress messages to send.

8.2.2. Decompression

An endpoint uses the following algorithm to decompress a message.

1. Append 4 octets of 0x00 0x00 0xff 0xff to the tail end of the payload of the message.

2. Decompress the resulting data using DEFLATE.

If the "agreed parameters" contain the "s2c_no_context_takeover" extension parameter, the client MAY start decompressing each new message with an empty LZ77 sliding window. Otherwise, the client MUST take over the LZ77 sliding window used to process the last compressed message.

If the "agreed parameters" contain the "c2s_no_context_takeover" extension parameter, the server MAY start decompressing each new message with an empty LZ77 sliding window. Otherwise, the server MUST take over the LZ77 sliding window used to process the last compressed message.

If the "agreed parameters" contain the "s2c_max_window_bits" extension parameter with a value of w, the client MAY reduce the size of its LZ77 sliding window to decompress received messages down to the w-th power of 2 bytes. Otherwise, the client MUST use a 32,768 byte LZ77 sliding window to decompress received messages.

If the "agreed parameters" contain the "c2s_max_window_bits" extension parameter with a value of w, the server MAY reduce the size of its LZ77 sliding window to decompress received messages down to the w-th power of 2 bytes. Otherwise, the server MUST use a 32,768 byte LZ77 sliding window to decompress received messages.

8.2.3. Examples

This section introduces examples of how the permessage-deflate transforms messages.

8.2.3.1. A message compressed using 1 compressed DEFLATE block

Suppose that an endpoint sends a text message "Hello". If the endpoint uses 1 compressed DEFLATE block (compressed with fixed Huffman code and the "BFINAL" bit is not set) to compress the message, the endpoint obtains the compressed data to use for the message payload as follows.

The endpoint compresses "Hello" into 1 compressed DEFLATE block and flushes the resulting data into a byte array using an empty DEFLATE block with no compression:

```
0xf2 0x48 0xcd 0xc9 0xc9 0x07 0x00 0x00 0x00 0xff 0xff
```

By stripping 0x00 0x00 0xff 0xff from the tail end, the endpoint gets the data to use for the message payload:

```
0xf2 0x48 0xcd 0xc9 0xc9 0x07 0x00
```

Suppose that the endpoint sends this compressed message without fragmentation. The endpoint builds one frame by putting the whole compressed data in the payload data portion of the frame:

```
0xc1 0x07 0xf2 0x48 0xcd 0xc9 0xc9 0x07 0x00
```

The first 2 octets (0xc1 0x07) are the WebSocket frame header (FIN=1, RSV1=1, RSV2=0, RSV3=0, opcode=text, MASK=0, Payload length=7). The following figure shows what value is set in each field of the WebSocket frame header.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-------+-+-------------+
|F|R|R|R| opcode|M| Payload len |
|I|S|S|S|       |A|             |
|N|V|V|V|       |S|             |
| |1|2|3|       |K|             |
+-+-+-+-+-------+-+-------------+
|1|1|0|0|   1   |0|      7      |
+-+-+-+-+-------+-+-------------+
```

Suppose that the endpoint sends the compressed message with fragmentation. The endpoint splits the compressed data into fragments and builds frames for each fragment. For example, if the fragments are 3 and 4 octet, the first frame is:

```
0x41 0x03 0xf2 0x48 0xcd
```

and the second frame is:

```
0x80 0x04 0xc9 0xc9 0x07 0x00
```

Note that the RSV1 bit is set only on the first frame.

8.2.3.2. Sharing LZ77 Sliding Window

Suppose that a client has sent a message "Hello" as a compressed message and will send the same message "Hello" again as a compressed message.

```
0xf2 0x48 0xcd 0xc9 0xc9 0x07 0x00
```

This is the payload of the first message the client has sent. If the "agreed parameters" contain the "c2s_no_context_takeover" extension parameter, the client compresses the payload of the next message into the same bytes (if the client uses the same "BTYPE" value and "BFINAL" value). So, the payload of the second message will be:

```
0xf2 0x48 0xcd 0xc9 0xc9 0x07 0x00
```

If the "agreed parameters" did not contain the "c2s_no_context_takeover" extension parameter, the client can compress the payload of the next message into shorter bytes by referencing the history in the LZ77 sliding window. So, the payload of the second message will be:

```
0xf2 0x00 0x11 0x00 0x00
```

Note that even if some uncompressed messages (with the RSV1 bit unset) are inserted between the two "Hello" messages, they will make no difference to the LZ77 sliding window.

8.2.3.3. Using a DEFLATE Block with No Compression

```
0xc1 0x0b 0x00 0x05 0x00 0xfa 0xff 0x48 0x65 0x6c 0x6c 0x6f 0x00
```

This is a frame constituting a text message "Hello" compressed using a DEFLATE block with no compression. The first 2 octets (0xc1 0x0b) are the WebSocket frame header (FIN=1, RSV1=1, RSV2=0, RSV3=0, opcode=text, MASK=0, Payload length=7). Note that the RSV1 bit is set for this message (only on the first fragment if the message is fragmented) because the RSV1 bit is set when DEFLATE is applied to the message, including the case when only DEFLATE blocks with no compression are used. The third to 13th octet consists a payload data containing "Hello" compressed using a DEFLATE block with no compression.

8.2.3.4. Using a DEFLATE Block with BFINAL Set to 1

On platform where the flush method using an empty DEFLATE block with no compression is not avaiable, implementors can choose to flush data using DEFLATE blocks with "BFINAL" set to 1.

```
0xf3 0x48 0xcd 0xc9 0xc9 0x07 0x00 0x00
```

This is a payload of a message containing "Hello" compressed using a DEFLATE block with "BFINAL" set to 1. The first 7 octets constitute a DEFLATE block with "BFINAL" set to 1 and "BTYPE" set to 01 containing "Hello". The last 1 octet (0x00) contains the header bits

with "BFINAL" set to 0 and "BTYPE" set to 00, and 5 padding bits of 0. This octet is necessary to allow the payload to be decompressed in the same manner as messages flushed using DEFLATE blocks with BFINAL unset.

8.2.3.5. Two DEFLATE Blocks in 1 Message

Two or more DEFLATE blocks may be used in 1 message.

```
0xf2 0x48 0x05 0x00 0x00 0x00 0xff 0xff 0xca 0xc9 0xc9 0x07 0x00
```

The first 3 octets (0xf2 0x48 0x05) and the least significant two bits of the 4th octet (0x00) constitute one DEFLATE block with "BFINAL" set to 0 and "BTYPE" set to 01 containing "He". The rest of the 4th octet contains the header bits with "BFINAL" set to 0 and "BTYPE" set to 00, and the 3 padding bits of 0. Together with the following 4 octets (0x00 0x00 0xff 0xff), the header bits constitute an empty DEFLATE block with no compression. A DEFLATE block containing "llo" follows the empty DEFLATE block.

8.3. Implementation Notes

On most common software development platforms, their DEFLATE compression library provides a method to align compressed data to byte boundaries using an empty DEFLATE block with no compression. For example, Zlib [Zlib] does this when "Z_SYNC_FLUSH" is passed to the deflate function.

To obtain a useful compression ratio, an LZ77 sliding window size of 1,024 or more is RECOMMENDED.

On the direction where context takeover is disallowed, an endpoint can easily figure out whether a certain message will be shorter if compressed or not.. Otherwise, it's not easy to know whether future messages will benefit from having a certain message compressed. Implementor may employ some heuristics to determine this.

8.4. Intermediaries

When an intermediary forwards a message, the intermediary MAY change compression on the messages as far as the resulting sequence of messages conform to the constraints based on the "agreed parameters". For example, an intermediary may decompress a received message, unset the "Per-message Compressed" bit and forward it to the other peer. Since such a compression change may affect the LZ77 sliding window, the intermediary may need to parse and transform the following messages, too.

9. Security Considerations

There is a known exploit for combination of a secure transport protocol and history-based compression [CRIME]. Implementors should give attention to this point when integrating this extension with other extensions or protocols.

10. IANA Considerations

10.1. Registration of the "permessage-deflate" WebSocket Extension Name

This section describes a WebSocket extension name registration in the WebSocket Extension Name Registry [RFC6455].

Extension Identifier
   permessage-deflate

Extension Common Name
   WebSocket Per-message Deflate

Extension Definition
   This document.

Known Incompatible Extensions
   None

The "permessage-deflate" extension name is used in the "Sec-WebSocket-Extensions" header in the WebSocket opening handshake to negotiate use of the permessage-deflate extension.

10.2. Registration of the "Per-message Compressed" WebSocket Framing Header Bit

This section describes a WebSocket framing header bit registration in the WebSocket Framing Header Bits Registry [RFC6455].

Header Bit
   RSV1

Common Name
   Per-message Compressed

Meaning
   The message is compressed or not.

Reference
   Section 6 of this document.

The "Per-message Compressed" framing header bit is used on the first fragment of non-control messages to indicate whether the payload of the message is compressed by the PMCE or not.

11. Acknowledgements

Special thanks to Patrick McManus who wrote up the initial specification of a DEFLATE-based compression extension for the WebSocket Protocol to which I referred to write this specification.

Thank you to the following people who participated in discussions on the HyBi WG and contributed ideas and/or provided detailed reviews (the list is likely to be incomplete): Adam Rice, Alexey Melnikov, Arman Djusupov, Bjoern Hoehrmann, Brian McKelvey, Dario Crivelli, Greg Wilkins, Inaki Baz Castillo, Jamie Lokier, Joakim Erdfelt, John A. Tamplin, Julian Reschke, Kenichi Ishibashi, Mark Nottingham, Peter Thorson, Roberto Peon, Simone Bordet and Tobias Oberstein. Note that people listed above didn't necessarily endorse the end result of this work.

12. References

12.1. Normative References

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

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

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

[LZ77] Ziv, J. and A. Lempel, "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343.

12.2. Informative References

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[RFC1979] Woods, J., "PPP Deflate Protocol", RFC 1979, August 1996.

[Zlib] Gailly, J. and M. Adler, "Zlib", <http://zlib.net/>.

[CRIME] Rizzo, J. and T. Duong, "The CRIME attack", Ekoparty 2012, September 2012.

Author's Address

Takeshi Yoshino
Google, Inc.

Email: tyoshino@google.com