HyBi Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2013

T. Yoshino
Google, Inc.

March 13, 2013

Compression Extensions for WebSocket
draft-ietf-hybi-permessage-compression-06

Abstract

This document specifies a framework for creating WebSocket extensions that add compression functionality to the WebSocket Protocol. Extensions based on this framework compress the payload data portion of non-control WebSocket messages on per-message basis using a specified compression algorithm. One reserved bit RSV1 in the WebSocket frame header is allocated to control application of compression for each message. 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

This Internet-Draft is submitted to IETF 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 September 14, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
2. Conformance Requirements and Terminology 4
3. WebSocket Per-message Compression Extension 5
4. Extension Negotiation 6
4.1. Negotiation Examples 6
5. Framing 8
5.1. Sending 8
5.2. Receiving 8
6. permessage-deflate extension 9
6.1. Method Parameters 10
6.1.1. Context Takeover Control 10
6.1.2. Limiting the LZ77 sliding window size 10
6.1.3. Example 11
6.2. Payload Data Transformation 12
6.2.1. Compression 12
6.2.2. Decompression 13
6.2.3. Examples 14
6.3. Intermediaries 17
6.4. Implementation Notes 17
7. Security Considerations 18
8. IANA Considerations 19
8.1. Registration of the "permessage-deflate" WebSocket Extension Name 19
8.2. Registration of the "Per-message Compressed" WebSocket Framing Header Bit 19
9. Acknowledgements 20
10. References 21
10.1. Normative References 21
10.2. Informative References 21
Author's Address 22

1. Introduction

This document specifies a framework to apply a compression algorithm to octets exchanged over the WebSocket Protocol [RFC6455]. This framework uses the extension concept for the WebSocket Protocol is introduced in the Section 9 of [RFC6455]. By specifying basic extension negotiation process excluding algorithm specific extension parameters in detail and a general method of transforming contents of WebSocket messages using a compression algorithm, this framework allows us to define WebSocket Per-message Compression Extensions (PMCEs) to the WebSocket Protocol individually for various compression algorithms. A WebSocket client and a WebSocket server negotiate use of a PMCE and determines parameters to configure the compression algorithm during the WebSocket opening handshake. The client and server then exchange non-control messages using frames with compressed data in the payload data portion. Documents specifying individual PMCEs describe how to negotiate parameters and how to transform octets in the payload data portion. A WebSocket client may offer multiple PMCEs during the WebSocket opening handshake. The WebSocket server received those offers may choose and accept preferred one from them. PMCEs use the RSV1 bit of the WebSocket frame header to indicate whether the message is compressed or not, so that we 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 the DEFLATE since it's widely available as a library on various platforms and the overhead of the DEFLATE is small. To align the end of compressed data to octet boundary, this extension uses the algorithm described in the 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 the Section 7.1.7 of [RFC6455].

This document references the event that _the WebSocket Connection is established_. This event is defined in the Section 4.1 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. WebSocket Per-message Compression Extension

WebSocket Per-message Compression Extensions (PMCEs) are individually defined for various compression algorithms, and are registered in the WebSocket Extension Name Registry. Each PMCE refers to this framework and defines:

- The content to put in the "Sec-WebSocket-Extensions" header, including the extension name of the PMCE and any applicable extension parameters
- How to interpret extension parameters exchanged during the opening handshake
- How to transform payload data portion of messages.

One such extension is defined in Section 6 of this document and is registered in Section 8. Other PMCEs may be defined in other documents.

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. This bit indicates whether the compression method is applied to the contents of the message or not. An endpoint MUST NOT offer or accept use of any other extension using the RSV1 bit together with a PMCE. The "Per-message Compressed" bit MUST NOT be set on control frames and non-first fragments of a data message. Messages with the "Per-message Compressed" bit set (only on the first fragment if the message is fragmented) are called "compressed messages" and have compressed data in their payload data portion. Messages with the "Per-message Compressed" bit unset are called "uncompressed messages" and have uncompressed data in their payload data portion.

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

- Frame boundary of frames output by the non-PMCE extension needs to be preserved.

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

Section 4 describes basic extension negotiation process. Section 5 describes how to apply the compression algorithm with negotiated parameters to the contents of WebSocket messages.

4. Extension Negotiation

To offer use of a PMCE, a client includes a "Sec-WebSocket-Extensions" header element with the extension name of the offered 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 by listing capability of the client and preferred values for the algorithm's configuration parameters to use. A client offers multiple PMCE choices to the server by including multiple elements, 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 configurations.

To accept use of an offered PMCE, a server includes a

"Sec-WebSocket-Extensions" header element with the extension name of the offered extension 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. The element MUST represent a PMCE that is fully supported by the server. The server rejects all offered PMCEs by not including any element with PMCE names, in which case the connection proceeds without Per-message Compression.

If the server responds with no PMCE element in the "Sec-WebSocket-Extensions" header and _the WebSocket Connection is established_, both endpoints MUST proceed without Per-message Compression. If the server gives an invalid response, such as accepting a PMCE that the client did not offer, the client MUST _Fail the WebSocket Connection_.

If the 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 5 to exchange messages, using the payload data transformation procedure of the PMCE returned by the server.

4.1. 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.

- Offer the permessage-foo.

```
permessage-foo
```


- 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"
```

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

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

- 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
```


5. Framing

5.1. Sending

An endpoint uses the following algorithm to compressed a message to send.

1. Compress the payload data portion of the message using the compression algorithm.

2. Build frame(s) for the message by putting the resulting octets instead of the original octets.

3. Set the "Per-message Compressed" bit of the first fragment to 1.

PMCEs don't change the opcode field. The payload data portion in outgoing frames output by a PMCE is not subject to the constraints for the original data type. At the receiver, the payload data portion after decompressing is subject to the constraints for the original data type again.

To send an uncompressed message, an endpoint sets the "Per-message Compressed" bit of the first fragment of the message to 0. The payload data portion of the message is sent as-is without applying the compression.

5.2. Receiving

To receive a compressed message, an endpoint decompress the payload data portion in the frames of the message.

An endpoint receives an uncompressed message as-is without decompression.

6. permessage-deflate extension

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

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

The following 4 extension parameters are defined for this extension.

- "s2c_no_context_takeover"
- "c2s_no_context_takeover"
- "s2c_max_window_bits"
- "c2s_max_window_bits"

A server MUST decline a "permessage-deflate" offer if any of the following conditions is met:

- The offer has any extension parameter unknown to the server.
- The offer has any extension parameter with an invalid value.
- The offer has multiple extension parameters with the same name.
- The server doesn't support the offered configuration.

A client MUST _Fail the WebSocket Connection_ if the server accepted a "permessage-deflate" offer with a response meeting any of the following condition:

- The response has any extension parameter unknown to the client.
- The response has any extension parameter with an invalid value.
- The response has multiple extension parameters with the same name.
- The client doesn't support the configuration the response represents.

6.1. Method Parameters

6.1.1. Context Takeover Control

A client MAY attach the "s2c_no_context_takeover" extension parameter. The "s2c_no_context_takeover" extension parameter has no value. If a server received the "s2c_no_context_takeover" extension parameter, the server MUST NOT use the same LZ77 sliding window to compress two or more messages. Servers SHOULD be able to accept the "s2c_no_context_takeover" parameter. A server accepts an offer with this extension parameter by including the "s2c_no_context_takeover" extension parameter in the response. If a server accepted an offer with this extension parameter, the server MUST empty its LZ77 sliding window to compress messages to send each time the server builds a new message.

A server MAY attach the "c2s_no_context_takeover" extension parameter to disallow the client to use the LZ77 sliding window used to build frames for the last message the client sent to build frames for the next message to send. The "c2s_no_context_takeover" extension parameter has no value. Clients SHOULD be able to accept the "c2s_no_context_takeover" parameter. A client that received this parameter MUST reset its LZ77 sliding window for sending to empty for each message.

6.1.2. Limiting the LZ77 sliding window size

A client MAY attach the "s2c_max_window_bits" extension parameter to limit the LZ77 sliding window size that the server uses to build messages. This extension parameter MUST have a decimal integer value in the range between 8 to 15 indicating the base-2 logarithm of the LZ77 sliding window size.

```
s2c_max_window_bits = 1*DIGIT
```

A server declines an offer with this extension parameter if the server doesn't support the extension parameter. A server accepts an offer with this extension parameter by including the extension parameter with the same value as the offer in the response. If a

server accepts an offer with this extension parameter, the server MUST NOT use LZ77 sliding window size greater than the size specified by the extension parameter to compress messages

A client MAY attach the "c2s_max_window_bits" extension parameter if the client can adjust LZ77 sliding window size based on the "c2s_max_window_bits" sent by the server. This parameter has no value.

If a server received and accepts an offer with the "c2s_max_window_bits" extension parameter, the server MAY include the "c2s_max_window_bits" parameter in the response to the offer to limit the LZ77 sliding window size that the client uses to build messages. If a server received and accepts an offer without 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. The "c2s_max_window_bits" extension parameter in the server's opening handshake MUST have a decimal integer value in the range between 8 to 15 indicating the base-2 logarithm of the LZ77 sliding window size.

```
c2s_max_window_bits = 1*DIGIT
```

If a client received the "c2s_max_window_bits" extension parameter, the client MUST NOT use LZ77 sliding window size greater than the size specified by the extension parameter to build messages.

6.1.3. 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 "s2c_max_window_bits" is mandatory for the client, but if the client want to fallback to the "permessage-deflate" without the "s2c_max_window_bits", the client should offer the fallback option in addition 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 supported one from them. To accept the first option, the server sends back this for example:

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

And to accept the second option, the server sends back this for example:

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

6.2. Payload Data Transformation

6.2.1. Compression

An endpoint uses the following algorithm to compress a message.

1. Compress all the octets of the payload data portion of the

message using the DEFLATE.

2. If the resulting data does not end with an empty DEFLATE block with no compression (the "BTYPE" bit is set to 0), 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" bit set to 0.

In using the DEFLATE in the first step above:

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

- An endpoints MAY use DEFLATE blocks of any type.

- An endpoints 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 a server accepts an offer with the "c2s_no_context_takeover" extension parameter, the client MUST empty its LZ77 sliding window to compress messages to send each time the client compresses a new message to send. Otherwise, the client MAY take over the LZ77 sliding window used to build the last compressed message.

If a server accepts an offer with the "s2c_no_context_takeover" extension parameter, the server MUST empty its LZ77 sliding window to compress messages to send each time the server compresses a new message to send. Otherwise, the server MAY take over the LZ77 sliding window used to build the last compressed message.

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

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

6.2.2. 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 data portion of the message.

2. Decompress the resulting data using the DEFLATE.

If a server accepts an offer with the "s2c_no_context_takeover" extension parameter, the client MAY empty its LZ77 sliding window to decompress received messages each time the client decompresses a new received message. Otherwise, the client MUST take over the LZ77 sliding window used to process the last compressed message.

If a server accepts an offer with the "c2s_no_context_takeover" extension parameter, the server MAY empty its LZ77 sliding window to decompress received messages each time the server decompresses a new received message. Otherwise, the server MUST take over the LZ77 sliding window used to process the last compressed message.

If a server accepts an offer with 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 a server accepts an offer with 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.

6.2.3. Examples

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

6.2.3.1. 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 put in the payload data portion 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
```

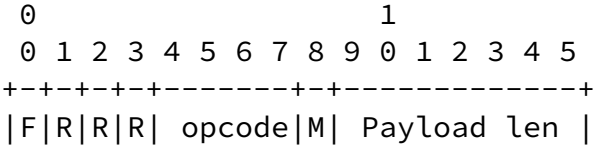
By stripping 0x00 0x00 0xff 0xff from the tail end, the endpoint gets the data to put in the payload data portion:

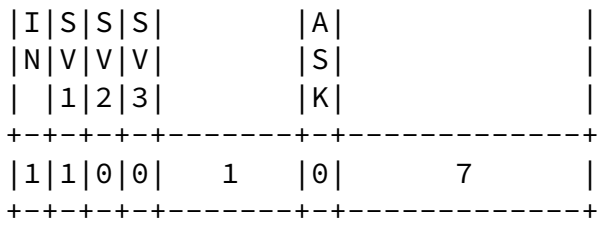
```
0xf2 0x48 0xcd 0xc9 0xc9 0x07 0x00
```

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.





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.

6.2.3.2. 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. If the server has accepted the offer with the "c2s_no_context_takeover" extension parameter, the server compresses the payload data portion of the next message into the same bytes (if the server uses the same "BTYPE" value and "BFINAL" value):

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

If the server hasn't accepted the offer with the "c2s_no_context_takeover" extension parameter, the server can compress the payload data portion of the next message into shorter bytes utilizing the history in the LZ77 sliding window:

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

Note that even if any uncompressed message (any message with the RSV1 bit unset) is inserted between the two "Hello" messages, such a message doesn't make any change on the LZ77 sliding window.

6.2.3.3. Using a DEFLATE Block with No Compression

Suppose that an endpoint compresses a text message "Hello" using a DEFLATE block with no compression. A DEFLATE block with no compression containing "Hello" flushed into a byte array using another but empty DEFLATE block with no compression is:

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

The endpoint strips the 4 octets at the tail end:

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

The endpoint builds a frame by putting the resulting data in the payload data portion of the frame:

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

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 the DEFLATE is applied to the message and it includes the case only DEFLATE blocks with no compression are used.

6.2.3.4. 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. Using a DEFLATE block with "BFINAL" set to 1 and "BTYPE" set to 1, "Hello" is compressed into:

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

So, payload of a message containing "Hello" compressed using this method is:

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

The last 1 octet (0x00) contains the header bits with "BFINAL" set to 0 and "BTYPE" set to 0, and 7 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.

6.2.3.5. 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) consist one DEFLATE block with "BFINAL" set to 0 and "BTYPE" set to 1 containing "He";. The rest of the 4th octet contains the header bits with "BFINAL" set to 0 and "BTYPE" set to 0, and the 3 padding bits of 0. Together with the following 4 octets (0x00 0x00 0xff 0xff), the header bits consist an empty DEFLATE block with no compression. A DEFLATE block containing "llo" follows the empty DEFLATE block.

6.3. Intermediaries

When an intermediary forwards messages, the intermediary MAY add, change or remove Per-message Compression on the messages. 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 framing.

6.4. Implementation Notes

On most common software development platforms, their DEFLATE compression library provide 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 attain sufficient compression ratio, the LZ77 sliding window size of 1,024 or more is RECOMMENDED.

7. Security Considerations

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

8. IANA Considerations

8.1. 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.

8.2. 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 5 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 data portion of the message is compressed by the PMCE or not.

9. 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): Alexey Melnikov, Arman Djusupov, Bjoern Hoehrmann, Brian McKelvey, Greg Wilkins, Inaki Baz Castillo, Jamie Lokier, Joakim Erdfelt, John A. Tamplin, Julian Reschke, Kenichi Ishibashi, Mark Nottingham, Peter Thorson, Roberto Peon and Simone Bordet. Note that people listed above didn't necessarily endorse the end result of this work.

10. References

10.1. 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.

10.2. 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