HyBi Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 28, 2013

T. Yoshino
Google, Inc.

January 24, 2013

WebSocket Per-message Compression
draft-ietf-hybi-permessage-compression-05

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 of non-control WebSocket messages 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 DEFLATE.

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 July 28, 2013.

Copyright Notice

Table of Contents

1. Introduction

This section is non-normative.

As well as other communication protocols, the WebSocket Protocol [RFC6455] can benefit from compression technology. This document specifies a framework for creating WebSocket extensions that apply a compression algorithm to octets exchanged over the WebSocket Protocol using its extension framework. Extensions based on this framework negotiate compression parameters during the opening handshake, and then compress the octets in non-control messages. Extensions for various compression algorithms can be specified by describing how to negotiate parameters and transform data in payloads. A client may offer multiple compression algorithms during the opening handshake by listing multiple compression extensions. The server may choose preferred one from them. Extensions based on this framework share the RSV1 bit of the WebSocket frame header to indicate whether the message is compressed or not, so that we can choose to skip messages with incompressible contents avoiding extra compression.

This document also specifies one specific extension "permessage-deflate" which is based on DEFLATE [RFC1951] algorithm. We chose DEFLATE since it's widely available as library on various platforms and the overhead it adds for each chunk 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 previous messages to get better compression ratio. For resource-limited devices, this extension provides parameters to limit memory usage for compression context.

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

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

The simplest header from the server to accept this extension is the same.

2. Conformance Requirements

Everything in this specification except for sections explicitly marked non-normative is normative.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Extension Negotiation

Extension names and negotiation methods are specified individually for each compression algorithm. There is no additional rule for extension naming. Extensions build based on this framework are collectively called "Per-message Compression Extensions".

To request use of a Per-message Compression Extension, a client MUST include an element with its extension token in the "Sec-WebSocket-Extensions" header in its opening handshake. The element contains extension parameters as specified by the specification of the extension. A client MAY list multiple Per-message Compression Extensions with the same name to offer use of the same algorithm with different configurations.

To accept use of a Per-message Compression Extension, a server MUST include an element with its extension token in the "Sec-WebSocket-Extensions" header in its opening handshake. The element contains extension parameters as specified by the specification of the extension. The parameters MUST be derived from the parameters sent by the client and the server's capability. To reject use of a Per-message Compression Extension, a server MUST

simply ignore the element in the "Sec-WebSocket-Extensions" header in the client's opening handshake.

If a client doesn't support the extension and its parameters replied from the server, the client MUST _Fail the WebSocket Connection_. Otherwise, once _the WebSocket Connection is established_, both endpoints MUST use the algorithm described in Section 4 to exchange messages.

3.1. Negotiation Example

This section is non-normative.

These are "Sec-WebSocket-Extensions" header value examples that negotiate the Per-message Compression Extension. permessage-foo and permessage-bar in the examples are extension names of Per-message Compression Extensions for hypothetical compression algorithm foo and bar.

- Request foo.

```
permessage-foo
```

- Request foo with a parameter x with 10 as its value.

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


- Request foo with a parameter z with "Hello World" (quotation for clarification) as its value. Since "Hello World" contains a space, it needs to be quoted.

```
permessage-foo; z="Hello World"
```

- Request foo and bar.

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

- Request foo with a parameter use_y which enables a feature y as first choice, and also list one without the parameter as a fallback plan.

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

4. Framing

This section describes how to apply the negotiated compression method to the contents of WebSocket messages.

This document allocates the RSV1 bit of the WebSocket header for extensions based on this framework, and names it the "Per-message Compressed" bit. Any other extension requiring the use of the RSV1

bit is incompatible with these extensions. This bit MAY be set only on the first fragment of a message. This bit indicates whether the compression method is applied to the message or not. Messages with the "Per-message Compressed" bit set (on its first fragment) are called "compressed messages". They have compressed data in their payload. Messages with the bit unset are called "uncompressed messages". They have uncompressed data in their payload.

Per-message Compression Extensions MUST NOT be used after any extension for which frame boundary needs to be preserved. Per-message Compression Extensions MUST NOT be used after any extension that uses "Extension data" field or any of the reserved bits on the WebSocket header as per-frame attribute.

Per-message Compression Extensions operates only on data frames.

4.1. Sending

To send a compressed message, an endpoint MUST use the following algorithm.

1. Compress the payload of the message using the compression method.

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.

To send an uncompressed message, an endpoint MUST set the "Per-message Compressed" bit of the first fragment of the message to 0. The payload of the message MUST be sent as-is without applying the compression method.

4.2. Receiving

To receive a compressed message, an endpoint MUST decompress its payload.

An endpoint MUST receive an uncompressed message as-is without decompression.

5. permessage-deflate extension

This section specifies a specific extension called "permessage-deflate" that compresses the payload of messages using DEFLATE [RFC1951] and byte boundary alignment method introduced in [RFC1979].

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

5.1. Method Parameters

The following 4 parameters are defined in the following subsections for this extension.

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

A server MUST ignore a "permessage-deflate" extension entry if any of the following is true:

- It has any parameter unknown to the server
- It has any parameter with an invalid value
- It is not supported by the server

A client MUST _Fail the WebSocket Connection_ if any of the following is true about the received "permessage-deflate" extension entry:

- It has any parameter unknown to the client
- It has any parameter with an invalid value
- It is not supported by the client

5.1.1. Disallow compression context takeover

A client MAY attach the "s2c_no_context_takeover" parameter to disallow the server to take over the LZ77 sliding window used to build previous messages. Servers SHOULD be able to accept the "s2c_no_context_takeover" parameter. To accept a request with this parameter, a server:

- MUST attach this parameter to its response
- MUST reset its LZ77 sliding window for sending to empty for each message

A server MAY attach the "c2s_no_context_takeover" parameter to disallow the client to take over the LZ77 sliding window used to build previous messages. 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.

These parameters have no value.

5.1.2. Limit maximum LZ77 sliding window size

A client MAY attach the "s2c_max_window_bits" parameter to limit the LZ77 sliding window size that the server uses to build messages. This 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. The ABNF [RFC5234] for the value of this parameter is 1*DIGIT. Servers MAY be able to accept the "s2c_max_window_bits" parameter. To accept a request with this parameter, the server:

- MUST attach this parameter with the same value as one of the "accepted request" to its response
- MUST NOT use LZ77 sliding window size greater than the size specified by this parameter to build messages

A client MAY attach the "c2s_max_window_bits" 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 the received request has the "c2s_max_window_bits" parameter, the server MAY respond to the request with the "c2s_max_window_bits" parameter to limit the LZ77 sliding window size that the client uses to build messages. Otherwise, the server MUST NOT accept the request with a response with the parameter. This parameter sent by the server MUST have a decimal integer value in the range between 8 to 15 indicating the base-2 logarithm of the LZ77 sliding window size. The ABNF for the value of this parameter is 1*DIGIT. A client that received this parameter MUST NOT use LZ77 sliding window size greater than the size specified by this parameter to build messages.

5.1.3. Example

This section is non-normative.

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

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

This request might be rejected by the server because it doesn't support the "s2c_max_window_bits" parameter. Since there's only one compression extension listed in the header, the server need to give up use of the Per-message Compression Extension entirely. If reduction of LZ77 sliding window size by the server is mandatory for the client, this is fine.

The next example lists two configurations so that the server can accept permessage-deflate by picking supported one from them.

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

The server can choose to accept the second extension entry by sending back this for example:

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

Since the "c2s_max_window_bits" parameter was not specified for both of the extensions, the server cannot use the "c2s_max_window_bits" parameter.

5.2. Application Data Transformation

5.2.1. Compression

An endpoint MUST use 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 block with no compression ("BTYPE" set to 0), append an empty block with no compression to the tail.

3. Remove 4 octets (that are 0x00 0x00 0xff 0xff) from the tail. After this step, the last octet of the compressed data contains the (part of) header bits with "BTYPE" set to 0.

In the first step:

- Multiple blocks MAY be used.
- Any type of block MAY be used.
- Both block with "BFINAL" set to 0 and 1 MAY be used.
- When any block with "BFINAL" set to 1 doesn't end at byte boundary, minimal padding bits of 0 MUST be added to make it end at byte boundary, and then the next block MUST start at the byte boundary if any.

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

If the server specified the "s2c_no_context_takeover" parameter, the server MUST reset its LZ77 sliding window for sending to empty for each message. Otherwise, the server MAY take over the LZ77 sliding window used to build the last compressed message.

If the server specified the "c2s_no_context_takeover" parameter, the client MUST reset its LZ77 sliding window for sending to empty for each message. Otherwise, the client MAY take over the LZ77 sliding window used to build the last compressed message.

If the server specified the "s2c_max_window_bits" parameter and its value is w, the server MUST NOT use an LZ77 sliding window greater

than w-th power of 2 bytes to build messages to send.

If the server specified the "c2s_max_window_bits" parameter and its value is w, the client MUST NOT use an LZ77 sliding window greater than w-th power of 2 bytes to build messages to send.

5.2.2. Decompression

An endpoint MUST use the following algorithm to decompress a message.

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

2. Decompress the resulting octets using DEFLATE.

If the server specified the "s2c_no_context_takeover" parameter, the

client MAY reset its LZ77 sliding window for receiving to empty for each message. Otherwise, the client MUST take over the LZ77 sliding window used to parse the last compressed message.

If the server specified the "c2s_no_context_takeover" parameter, the server MAY reset its LZ77 sliding window for receiving to empty for each message. Otherwise, the server MUST take over the LZ77 sliding window used to parse the last compressed message.

If the server specified the "s2c_max_window_bits" parameter and its value is w, the client MAY reduce the size of the 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 server specified the "c2s_max_window_bits" parameter and its value is w, the server MAY reduce the size of the 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.

5.2.3. Examples

This section is non-normative.

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

5.2.3.1. A message compressed using 1 compressed block

Suppose that a text message "Hello" is sent. When 1 compressed block (compressed with fixed Huffman code, "BFINAL" is not set) is used, compressed data to be sent in payload is obtained as follows.

Compress "Hello" into 1 compressed block and flush it into a byte array using an empty block with no compression:

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

Strip 0x00 0x00 0xff 0xff from the tail:

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

To send it without fragmentation, just build a frame putting the whole data in payload data:

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


The first 2 octets are the WebSocket protocol's overhead (FIN=1, RSV1=1, RSV2=0, RSV3=0, opcode=text, MASK=0, Payload length=7).

To send it after fragmentation, split the compressed payload and build frames for each of split data as well as fragmentation process done when the compression extension is not used. For example, the first fragment may contain 3 octets of the payload:

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

and the second (last) fragment contain 4 octets of the payload:

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

Note that RSV1 is set only on the first fragment.

5.2.3.2. Sharing LZ77 Sliding Window

Suppose that the next message to send is also "Hello". If it's disallowed by the other peer (using some extension parameter) to take over the LZ77 sliding window used for the last message, the next message is compressed into the same byte array (if the same "BTYPE" and "BFINAL" value are used). If it's allowed, the next message can be compressed into shorter payload:

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

instead of:

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

Note that even if any uncompressed message is inserted between the two "Hello" messages, it doesn't affect context sharing between the two "Hello" messages.

5.2.3.3. Using a Block with No Compression

Blocks with no compression can be also used. A block with no compression containing "Hello" flushed into a byte array using an empty block with no compression is:

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

So, payload of a message containing "Hello" converted into a DEFLATE block with no compression is:

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


If it's not fragmented, the frame for this message is:

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

The first 2 octets are the WebSocket protocol's overhead (FIN=1, RSV1=1, RSV2=0, RSV3=0, opcode=text, MASK=0, Payload length=7). Note that RSV1 must be set for this message (only on the first fragment of it) because RSV1 indicates whether DEFLATE is applied to the message including use of blocks with no compression or not.

5.2.3.4. Using a Block with BFINAL Set to 1

On platform where the flush method based on an empty block with no compression is not avaiable, implementors can choose to flush data using blocks with "BFINAL" set to 1. Using a 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 parameter setting is:

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

The last 1 octet contains the header bits with "BFINAL" set to 0 and "BTYPE" set to 0, and 7 padding bits of 0. It's necessary to make the payload able to be processed by the same manner as messages flushed using blocks with BFINAL unset.

5.2.3.5. Two Blocks in 1 Message

Two or more blocks may be used in 1 message.

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

The first 3 octets and the least significant two bits of the 4th octet consist one 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 block with no compression. Then, a block containing "llo" follows.

5.3. Intermediaries

When intermediaries forward messages, they MAY decompress and/or compress the messages according to the constraints negotiated during the opening handshake of the connection(s).

5.4. Implementation Notes

This section is non-normative.

On most common software development platforms, the operation of aligning compressed data to byte boundaries using an empty block with no compression is available as a library. For example, Zlib [Zlib] does this when "Z_SYNC_FLUSH" is passed to deflate function.

To get sufficient compression ratio, LZ77 sliding window size of 1,024 or more is recommended.

6. Security Considerations

There are no security concerns for now.

7. IANA Considerations

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

7.2. 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 4 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 Per-message Compression Extension or not.

8. Acknowledgements

Special thanks to Patrick McManus who wrote up the initial specification of 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.

9. References

9.1. 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.

9.2. 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/>.

Author's Address

Takeshi Yoshino
Google, Inc.

Email: tyoshino@google.com