HyBi Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 18, 2013

T. Yoshino
Google, Inc.

October 15, 2012

WebSocket Per-message Compression
draft-ietf-hybi-permessage-compression-02

Abstract

This specification defines a WebSocket extension that adds compression functionality to the WebSocket Protocol. It compresses the payload of non-control WebSocket messages using specified compression algorithm. One reserved bit RSV1 in the WebSocket frame header is allocated to control application of compression for each message. This specification provides one compression method available for the 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 April 18, 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 specification defines a WebSocket extension that applies a compression algorithm to octets exchanged over the WebSocket Protocol using its extension framework. This extension negotiates what compression method to use on opening handshake, and then compresses the octets in non-control messages using the method. We can apply this extension to various compression algorithms by specifying how to negotiate parameters and transform payload. A client may offer multiple compression methods on opening handshake, and then the server chooses one from them. This extension uses 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 without applying extra compression.

This specification provides one specific compression method "deflate" which is based on DEFLATE [RFC1951] for this extension. 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 method 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, method parameters to limit the usage of memory for compression context are provided.

The simplest "Sec-WebSocket-Extensions" header in the client's opening handshake to request DEFLATE based per-message compression is the following:

```
Sec-WebSocket-Extensions: permessage-compress; method=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

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

To request use of the Per-message Compression Extension, a client MUST include an element with the "permessage-compress" extension token as its extension identifier in the "Sec-WebSocket-Extensions" header in its opening handshake. The element MUST contain exactly one extension parameter named "method". The value of the "method" extension parameter is a list of compression method descriptions, ordered by preference. Each compression method description has a method name and optional method parameters. The grammar of the list is "requested-method-list" defined in the following ABNFs.

```
requested-method-list = 1#method-desc
method-desc = method-name *(";" method-param)
method-name = token
method-param = token ["=" (token | quoted-string)]
```

The list MAY contain multiple method descriptions with the same method name.

To accept use of the Per-message Compression Extension, a server MUST choose one compression method description to accept from ones listed by the client, and include an element with the "permessage-compress" extension token in the "Sec-WebSocket-Extensions" header in its opening handshake. The chosen description is called "accepted request". The element MUST contain exactly one extension parameter named "method". The value of the "method" extension parameter MUST be a compression method description. This description is called "method agreement". The method name in the "method agreement" MUST be one of the accepted request. The "method agreement" MUST conform the "accepted request". Its grammar is "method-agreement" defined in the following ABNF.

```
method-agreement = method-desc
```

The value of the "method" parameter MUST be quoted by using "quoted-string" syntax if it doesn't conform to token syntax.

If a client doesn't support the method and its configuration specified by the "method agreement", the client MUST _Fail the WebSocket Connection_. Otherwise, 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.

- Request foo method. Since foo matches token syntax, it doesn't need to be quoted.

```
permessage-compress; method=foo
```

- Request foo method with a parameter x with 10 as its value. Since the method parameter value contains a semicolon, it doesn't match token syntax. Quotation is needed.

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

- Request foo method and bar method. Since the method parameter value contains a comma, it doesn't match token syntax. Quotation is needed.

```
permessage-compress; method="foo, bar"
```

- Request foo method with parameter x with "Hello World" (quotation for clarification) as its value and bar method. Since "Hello World" contains a space, it needs to be quoted. Since quoted "Hello World" contains double quotations and a space, it needs to be quoted again.

```
permessage-compress; method="foo; x=\"Hello World\", bar"
```


4. Framing

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

This extension allocates the RSV1 bit of the WebSocket header and names it the "Per-message Compressed" bit. Any extension requiring the use of the RSV1 bit is incompatible with this extension. 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.

This extension MUST NOT be used after any extension for which frame boundary needs to be preserved. This extension 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.

This extension 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. DEFLATE method

This section defines a method named "deflate" for this extension that compresses the payload of messages using DEFLATE [RFC1951] and byte boundary alignment method introduced in [RFC1979].

5.1. Method Parameters

An endpoint MAY include one or more method parameters in the method description as defined below.

Maximum LZ77 sliding window size

A client MAY attach the "s2c_max_window_bits" method parameter to limit the LZ77 sliding window size that the server uses to build messages. If the "accepted request" has this method parameter, the server MUST NOT use LZ77 sliding window size greater than the size specified by this parameter to build messages. If the "accepted request" has this method parameter, the server MUST attach this method parameter with the same value as one of the "accepted request".

A server MAY attach the "c2s_max_window_bits" method parameter to limit the LZ77 sliding window size that the client uses to build messages. A client that received this parameter MUST NOT use LZ77 sliding window size greater than the size specified by this parameter to build messages.

These parameters MUST have an integer value in the range between 8 to 15 indicating the base-2 logarithm of the LZ77 sliding window size.

Disallow compression context takeover

A client MAY attach the "s2c_no_context_takeover" method parameter to disallow the server to take over the LZ77 sliding window used to build previous messages. If the "accepted request" has this method parameter, the server MUST reset its LZ77 sliding window for sending to empty for each message. If the "accepted request" has this method parameter, the server MUST attach this method parameter.

A server MAY attach the "c2s_no_context_takeover" method parameter to disallow the client to take over the LZ77 sliding window used to build previous messages. A client that received this parameter MUST reset its LZ77 sliding window for sending to empty for each message.

These parameters have no value.

A server MUST ignore any method parameter other than "s2c_max_window_bits" and "s2c_no_context_takeover" in the received "deflate" method description.

A client MUST _Fail the WebSocket Connection_ if there is any method parameter other than the "s2c_max_window_bits", "c2s_max_window_bits", "s2c_no_context_takeover" and "c2s_no_context_takeover" in the received "deflate" method description. A client MUST _Fail the WebSocket Connection_ if it doesn't support the method and its configuration specified by the received "deflate" method description.

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 "method agreement" has the "s2c_max_window_bits" method

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 "method agreement" has the "c2s_max_window_bits" method 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.

If the "method agreement" has the "s2c_no_context_takeover" method 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 "method agreement" has the "c2s_no_context_takeover" method 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.

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 "method agreement" has the "s2c_max_window_bits" method 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 "method agreement" has the "c2s_max_window_bits" method 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.

If the "method agreement" has the "s2c_no_context_takeover" method 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 "method agreement" has the "c2s_no_context_takeover" method 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.

5.2.3. Examples

This section is non-normative.

This section introduces examples of how the DEFLATE method transforms messages.

5.2.3.1. A message compressed using 1 compressed block

Suppose that a text message "Hello" is sent using the DEFLATE method. 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-compress" WebSocket Extension Name

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

Extension Identifier
   permessage-compress

Extension Common Name
   WebSocket Per-message Compression

Extension Definition
   This document.

Known Incompatible Extensions
   None

The "permessage-compress" token is used in the "Sec-WebSocket-Extensions" header in the WebSocket opening handshake to negotiate use of the Per-message Compression 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.

7.3. WebSocket Per-message Compression Method Name Registry

This specification creates a new IANA registry for names of compression methods to be used with the WebSocket Per-message Compression Extension in accordance with the principles set out in [RFC5226].

As part of this registry, IANA maintains the following information:

Method Identifier
: The identifier of the method, as will be used in the method description as defined Section 3 of this specification. The value must conform to the method-name ABNF as defined in Section 3 of this specification.

Method Common Name
: The name of the method, as the method is generally referred to.

Method Definition
: A reference to the document in which the method being used with this extension is defined.

WebSocket Per-message Compression method names are to be subject to the "First Come First Served" IANA registration policy [RFC5226].

IANA has added initial values to the registry as follows.

Identifier	Common Name	Definition
deflate	DEFLATE	This document

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.

9. References

9.1. Normative References

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 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