Internet Engineering Task Force (IETF)
Request for Comments: 7692
Category: Standards Track
ISSN: 2070-1721

T. Yoshino
Google, Inc.

December 2015

Compression Extensions for WebSocket

Abstract

This document defines 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 WebSocket data messages on a per-message basis using parameters negotiated during the opening handshake. This framework provides a general method for applying a compression algorithm to the contents of WebSocket messages. Each compression algorithm has to be defined in a document defining the extension by specifying the parameter negotiation and the payload transformation algorithm in detail. This document also specifies one specific compression extension using the DEFLATE algorithm.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7692.

Copyright Notice

Table of Contents

1. Introduction

This document specifies a framework for adding compression functionality to the WebSocket Protocol [RFC6455]. The framework specifies how to define WebSocket Per-Message Compression Extensions (PMCEs) for a compression algorithm based on the extension concept of the WebSocket Protocol specified in Section 9 of [RFC6455]. A WebSocket client and a peer WebSocket server negotiate the use of a PMCE and determine the parameters required to configure the compression algorithm during the WebSocket opening handshake. The client and server can then exchange data messages whose frames contain compressed data in the payload data portion.

This framework only specifies a general method for applying a compression algorithm to the contents of WebSocket messages. Each individual PMCE has to be specified in a document describing in detail how to negotiate the configuration parameters for the specific compression algorithm used by that PMCE and how to transform (compress and decompress) data in the payload data portion.

A WebSocket client may offer multiple PMCEs during the WebSocket opening handshake. A peer WebSocket server receiving the offer may choose to accept the preferred PMCE 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 DEFLATE algorithm is widely available on various platforms, and its overhead is small. The extension name of this PMCE is "permessage-deflate". To align the end of compressed data to an octet boundary, this extension uses the algorithm described in Section 2.1 of [RFC1979]. Endpoints can take over the LZ77 sliding window [LZ77] used to build frames for previous messages to achieve a 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 Sections 4.1 and 6.2, respectively, of [RFC6455].

This document uses the Augmented 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 mechanisms that are underspecified or not defined at all in [RFC6455].

data message - a message consisting of data frames as defined in Section 5.6 of [RFC6455].

message payload (or payload of a message) - the concatenation of the payload data portion of all data frames (see Section 6.2 of [RFC6455]) representing a single message.

next extension in use after extension X - the next extension listed after X in the "Sec-WebSocket-Extensions" header in the server's opening handshake as defined in Section 9.1 of [RFC6455]. Such an extension is applied to outgoing data from the application right after X on the sender side but is applied right before X to incoming data from the underlying transport.

extension in use preceding extension X - the extension listed right before 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 before X on the sender side but is applied right after X to incoming data from the underlying transport.

extension negotiation offer - each element in the "Sec-WebSocket-Extensions" header in the client's opening handshake.

extension negotiation response - each element in the "Sec-WebSocket-Extensions" header in the server's opening handshake.

corresponding extension negotiation response for an extension negotiation offer - an extension negotiation response that a server sends back to the peer client containing the same extension name as the offer and meeting the requirements represented by the offer.

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

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

4. WebSocket Per-Message Compression Extension

WebSocket PMCEs are extensions to the WebSocket Protocol enabling compression functionality. PMCEs are built based on the extension concept of the WebSocket Protocol specified in 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 referring to this framework MUST define the following:

- The extension name of the PMCE and any applicable extension parameters that MUST be included in the "Sec-WebSocket-Extensions" header during the extension negotiation offer/response.
- How to interpret the extension parameters exchanged during the opening handshake.
- How to transform the payload of a message.

One PMCE is defined in Section 7 of this document and is registered in Section 9. Other PMCEs may be defined in the future 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 MUST include the extension name of the PMCE in the "Sec-WebSocket-Extensions" header field of its opening handshake of the WebSocket connection. Extension parameters

are used to specify the PMCE offer in detail. For example, a client lists its preferred configuration parameter values for the compression algorithm of the PMCE. A client may also offer multiple PMCE choices to the server by including multiple elements in the "Sec-WebSocket-Extensions" header, one for each PMCE offered. This set of elements MAY include multiple PMCEs with the same extension name to offer the possibility to use the same algorithm with different configuration parameters. The order of elements is important as it specifies the client's preference. An element preceding another element has higher preference. It is recommended that a server accepts PMCEs with higher preference if the server supports them.

A PMCE negotiation offer provides requests and/or hints to the server.

A request in a PMCE negotiation offer indicates constraints on the server's behavior that must be satisfied if the server accepts the offer. For example, suppose that a server sends data compressed with the DEFLATE algorithm to a client. The server must keep the original bytes of data that it recently compressed and sent to the client. The client must keep the result of decompressing the bytes of data that it recently received from the server. The amount of bytes of data kept is called the LZ77 window size. The LZ77 window size of the client must not be less than the LZ77 window size of the server. In a PMCE negotiation offer, the client MUST inform the server of its LZ77 window size so that the server uses an LZ77 window size that is not greater than the LZ77 window size of the client. This restriction on the LZ77 window size is an example of a request in a PMCE negotiation offer.

A hint in a PMCE negotiation offer provides information about the client's behavior that the server may either safely ignore or refer to when the server decides its behavior. For example, suppose that a client sends data compressed with the DEFLATE algorithm to a server. The client must keep the original bytes of data that it recently compressed and sent to the server. The server must keep the result of decompressing the bytes of data that it recently received from the client. The LZ77 window size of the server must not be less than the LZ77 window size of the client. In a PMCE negotiation offer, the client MAY inform the server of the maximum LZ77 window size the client can afford so that the server can choose to use an LZ77 window size that is not greater than the maximum size of the client. This information is an example of a hint in a PMCE negotiation offer. It's waste of memory to use an LZ77 window size greater than the LZ77 window size the client actually uses. Using the hint, the server can avoid the waste of memory. Since the hint itself doesn't specify the

constraints on the endpoints, the server must use the "agreed parameters" (defined below) to explicitly ask the client not to use an LZ77 window size greater than the LZ77 window size of the server.

To accept the use of an offered PMCE, a server MUST include the extension name of the PMCE in the "Sec-WebSocket-Extensions" header field of its opening handshake of the WebSocket connection. Extension parameters represent the detailed configuration parameters for the PMCE to use. These extension parameters and their values are called "agreed parameters". The element MUST represent a PMCE that is fully supported by the server. The contents of the element don't need to be exactly the same as those of the received extension negotiation offers. For example, suppose that a server received a PMCE extension negotiation offer with an extension parameter "X" indicating that the client can enable an optional feature named X. The server may accept the PMCE offer with an element without the extension parameter "X", meaning that the server chose not to enable the feature X. In this case, the offer contains the extension parameter "X", but the "agreed parameters" don't contain the extension parameter "X".

"Agreed parameters" must represent how the requests and hints in the client's extension negotiation offer have been handled in addition to the server's requests and hints on the client's behavior, so that the client can configure its behavior without identifying exactly which PMCE extension negotiation offer has been accepted.

For example, if a client sends an extension negotiation offer that includes a parameter "enable_compression" and another without this parameter, the server accepts the former and informs the client by sending back an element that includes parameter(s) acknowledging "enable_compression". The name of the acknowledging parameter doesn't need to be exactly the same as the offer. For example, two parameters, "enable_strong_compression" and "enable_weak_compression", may be defined as acknowledging parameters for "enable_compression".

Compression features can be applied differently for each direction. For such features, the acknowledging parameter and the parameter in the reverse direction must be chosen to distinguish them. For example, in order to make parameters distinguishable, a "server_" prefix can be added to parameters affecting data sent from a server, and a "client_" prefix can be added to parameters affecting data sent from a client.

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

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

- The extension requires the boundary of frames to be preserved between the output from the extension at the sender and the 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 a per-frame attribute.

A client that receives a response accepting a PMCE extension negotiation offer together with such an extension MUST _Fail the WebSocket Connection_.

A server declining all offered PMCEs MUST NOT include 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 and the message payload transformation (compressing and decompressing) procedure of the PMCE configured with the "agreed parameters" returned by the server to exchange messages.

5.1. General Negotiation Flow

This section describes a general negotiation flow. How to handle parameters in detail must be specified in the document specifying the PMCE.

A client makes an offer including parameters identifying the following:

- Hints about how the client is planning to compress data
- Requests about how the server compresses data
- Limitations concerning the client's compression functionality

The peer server makes a determination of its behavior based on these parameters. If the server can and wants to proceed with this PMCE enabled, the server responds to the client with parameters identifying the following:

- Requests about how the client compresses data
- How the server will compress data

Based on these parameters received from the server, the client determines its behavior and if it can and wants to proceed with this PMCE enabled. Otherwise, the client starts the closing handshake with close code 1010.

5.2. Negotiation Examples

The following 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 the 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

6. Framing

PMCEs operate only on data 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 receiving 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 which extensions were negotiated.

2. Process the compressed data as follows:

   * If this PMCE is the last extension to process outgoing messages, build frame(s) using the compressed data instead of the original data for the message payload, set the "Per-Message Compressed" bit of the first frame, and then send the frame(s) as described in Section 6.1 of [RFC6455].

   * Otherwise, pass the transformed message payload and modified header values, including the "Per-Message Compressed" bit value set to 1, to the next extension after the PMCE. If the extension expects frames for 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.

1. Process the original data as follows:

   * If this PMCE is the last extension to process outgoing messages, build frame(s) using the original data for the payload data portion as is, unset the "Per-Message Compressed" bit of the first frame, and then send the frame(s) as described in Section 6.1 of [RFC6455].

   * Otherwise, pass the message payload and header values to the next extension after the PMCE as is. If the extension expects frames for 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 receiving such a frame MUST _Fail the WebSocket Connection_.

PMCEs do not change the opcode field. The opcode of the first frame of a compressed 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 output 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 be direct input from the underlying transport or output of another WebSocket extension, depending on which extensions were negotiated.

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

3. Process the decompressed message as follows:

   * 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 preceding the PMCE. If the extension expects frames for 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.

1. Process the received message as follows:

   * 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 preceding the PMCE as is. If the extension expects frames for input, build a frame for the message and pass it.

7. The "permessage-deflate" Extension

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

This section uses the term "byte" with the same meaning as used in [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".

Four extension parameters are defined for "permessage-deflate" to help endpoints manage per-connection resource usage.

- "server_no_context_takeover"
- "client_no_context_takeover"
- "server_max_window_bits"
- "client_max_window_bits"

These parameters enable two methods (no_context_takeover and max_window_bits) of constraining memory usage that may be applied independently to either direction of WebSocket traffic. The extension parameters with the "client_" prefix are used by the client to configure its compressor and by the server to configure its decompressor. The extension parameters with the "server_" prefix are used by the server to configure its compressor and by the client to configure its decompressor. All four parameters are defined for both a client's extension negotiation offer and a server's extension negotiation response.

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

- The negotiation offer contains an extension parameter not defined for use in an offer.
- The negotiation offer contains an extension parameter with an invalid value.
- The negotiation offer contains multiple extension parameters with the same name.
- The server doesn't support the offered configuration.

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

- The negotiation response contains an extension parameter not defined for use in a response.
- The negotiation response contains an extension parameter with an invalid value.
- The negotiation response contains multiple extension parameters with the same name.
- The client does not support the configuration that the response represents.

The term "LZ77 sliding window" [LZ77] used in this section means the buffer used by the DEFLATE algorithm to store recently processed input. The DEFLATE compression algorithm searches the buffer for a match with the following input.

The term "use context takeover" used in this section means that the same LZ77 sliding window used by the endpoint to build frames of the previous sent message is reused to build frames of the next message to be sent.

7.1. Extension Parameters

7.1.1. Context Takeover Control

7.1.1.1. The "server_no_context_takeover" Extension Parameter

A client MAY include the "server_no_context_takeover" extension parameter in an extension negotiation offer. This extension parameter has no value. By including this extension parameter in an extension negotiation offer, a client prevents the peer server from using context takeover. If the peer server doesn't use context takeover, the client doesn't need to reserve memory to retain the LZ77 sliding window between messages.

Absence of this extension parameter in an extension negotiation offer indicates that the client can decompress a message that the server built using context takeover.

A server accepts an extension negotiation offer that includes the "server_no_context_takeover" extension parameter by including the "server_no_context_takeover" extension parameter in the corresponding extension negotiation response to send back to the client. The "server_no_context_takeover" extension parameter in an extension negotiation response has no value.

It is RECOMMENDED that a server supports the "server_no_context_takeover" extension parameter in an extension negotiation offer.

A server MAY include the "server_no_context_takeover" extension parameter in an extension negotiation response even if the extension negotiation offer being accepted by the extension negotiation response didn't include the "server_no_context_takeover" extension parameter.

7.1.1.2. The "client_no_context_takeover" Extension Parameter

A client MAY include the "client_no_context_takeover" extension parameter in an extension negotiation offer. This extension parameter has no value. By including this extension parameter in an extension negotiation offer, a client informs the peer server of a hint that even if the server doesn't include the "client_no_context_takeover" extension parameter in the corresponding extension negotiation response to the offer, the client is not going to use context takeover.

A server MAY include the "client_no_context_takeover" extension parameter in an extension negotiation response. If the received extension negotiation offer includes the "client_no_context_takeover" extension parameter, the server may either ignore the parameter or use the parameter to avoid taking over the LZ77 sliding window unnecessarily by including the "client_no_context_takeover" extension parameter in the corresponding extension negotiation response to the offer. The "client_no_context_takeover" extension parameter in an extension negotiation response has no value. By including the "client_no_context_takeover" extension parameter in an extension negotiation response, a server prevents the peer client from using context takeover. This reduces the amount of memory that the server has to reserve for the connection.

Absence of this extension parameter in an extension negotiation response indicates that the server can decompress messages built by the client using context takeover.

A client MUST support the "client_no_context_takeover" extension parameter in an extension negotiation response.

7.1.2. Limiting the LZ77 Sliding Window Size

7.1.2.1. The "server_max_window_bits" Extension Parameter

A client MAY include the "server_max_window_bits" extension parameter in an extension negotiation 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, and MUST conform to the ABNF below.

```
server-max-window-bits = 1*DIGIT
```

By including this parameter in an extension negotiation offer, a client limits the LZ77 sliding window size that the server will use to compress messages. If the peer server uses a small LZ77 sliding window to compress messages, the client can reduce the memory needed for the LZ77 sliding window.

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

Absence of this parameter in an extension negotiation offer indicates that the client can receive messages compressed using an LZ77 sliding window of up to 32,768 bytes.

A server accepts an extension negotiation offer with this parameter by including the "server_max_window_bits" extension parameter in the extension negotiation response to send back to the client with the same or smaller value as the offer. The "server_max_window_bits" extension parameter in an extension negotiation response has a decimal integer value without leading zeroes between 8 to 15, inclusive, indicating the base-2 logarithm of the LZ77 sliding window size, and MUST conform to the ABNF below.

```
server-max-window-bits = 1*DIGIT
```

A server MAY include the "server_max_window_bits" extension parameter in an extension negotiation response even if the extension negotiation offer being accepted by the response didn't include the "server_max_window_bits" extension parameter.

7.1.2.2. The "client_max_window_bits" Extension Parameter

A client MAY include the "client_max_window_bits" extension parameter in an extension negotiation 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. If a value is specified for this parameter, the value MUST conform to the ABNF below.

```
client-max-window-bits = 1*DIGIT
```

By including this parameter in an offer, a client informs the peer server that the client supports the "client_max_window_bits" extension parameter in an extension negotiation response and, optionally, a hint by attaching a value to the parameter. If the "client_max_window_bits" extension parameter in an extension negotiation offer has a value, the parameter also informs the peer server of a hint that even if the server doesn't include the "client_max_window_bits" extension parameter in the corresponding extension negotiation response with a value greater than the one in the extension negotiation offer or if the server doesn't include the extension parameter at all, the client is not going to use an LZ77 sliding window size greater than the size specified by the value in the extension negotiation offer to compress messages.

If a received extension negotiation offer has the "client_max_window_bits" extension parameter, the server MAY include the "client_max_window_bits" extension parameter in the corresponding extension negotiation response to the offer. If the "client_max_window_bits" extension parameter in a received extension negotiation offer has a value, the server may either ignore this value or use this value to avoid allocating an unnecessarily big LZ77 sliding window by including the "client_max_window_bits" extension parameter in the corresponding extension negotiation response to the offer with a value equal to or smaller than the received value. The "client_max_window_bits" extension parameter in an extension negotiation response has a decimal integer value without leading zeroes between 8 to 15 inclusive indicating the base-2 logarithm of the LZ77 sliding window size and MUST conform to the ABNF below.

```
client-max-window-bits = 1*DIGIT
```

By including this extension parameter in an extension negotiation response, a server limits the LZ77 sliding window size that the client uses to compress messages. This reduces the amount of memory for the decompression context that the server has to reserve for the connection.

If a received extension negotiation offer doesn't have the "client_max_window_bits" extension parameter, the corresponding extension negotiation response to the offer MUST NOT include the "client_max_window_bits" extension parameter.

Absence of this extension parameter in an extension negotiation response indicates that the server can receive messages compressed using an LZ77 sliding window of up to 32,768 bytes.

7.1.3. Examples

The simplest "Sec-WebSocket-Extensions" header in a client's opening handshake to offer use of the "permessage-deflate" extension looks like this:

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

Since the "client_max_window_bits" extension parameter is not included in this extension negotiation offer, the server must not accept the offer with an extension negotiation response that includes the "client_max_window_bits" extension parameter. The simplest "Sec-WebSocket-Extensions" header in a server's opening handshake to accept use of the "permessage-deflate" extension is the same:

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

The following extension negotiation offer sent by a client is asking the server to use an LZ77 sliding window with a size of 1,024 bytes or less and declaring that the client supports the "client_max_window_bits" extension parameter in an extension negotiation response.

```
Sec-WebSocket-Extensions:
    permessage-deflate;
    client_max_window_bits; server_max_window_bits=10
```

This extension negotiation offer might be rejected by the server because the server doesn't support the "server_max_window_bits" extension parameter in an extension negotiation offer. This is fine if the client cannot receive messages compressed using a larger sliding window size, but if the client just prefers using a small window but wants to fall back to the "permessage-deflate" without the "server_max_window_bits" extension parameter, the client can make an offer with the fallback option like this:

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

The server can accept "permessage-deflate" by picking any supported one from the listed offers. To accept the first option, for example, the server may send back a response as follows:

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

To accept the second option, for example, the server may send back a response as follows:

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

7.2. Message Payload Transformation

7.2.1. 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 are 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.

When 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 a byte boundary, an endpoint MUST add minimal padding bits of 0 to make it end at a byte boundary. The next DEFLATE block follows the padded data if any.

An endpoint fragments a compressed message by splitting the result of running this algorithm. Even when only part of the payload is available, a fragment can be built by compressing the available data

and choosing the block type appropriately so that the end of the resulting compressed data is aligned at a byte boundary. Note that for non-final fragments, the removal of 0x00 0x00 0xff 0xff MUST NOT be done.

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 "client_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. Note that even if the client has included the "client_no_context_takeover" extension parameter in its offer, the client MAY take over the LZ77 sliding window used to build the last compressed message if the "agreed parameters" don't contain the "client_no_context_takeover" extension parameter. The client-to-server "client_no_context_takeover" extension parameter is just a hint for the server to build an extension negotiation response.

If the "agreed parameters" contain the "server_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 "client_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. Note that even if the client has included in its offer the "client_max_window_bits" extension parameter with a value smaller than one in the "agreed parameters", the client MAY use an LZ77 sliding window with any size to compress messages to send as long as the size conforms to the "agreed parameters". The client-to-server "client_max_window_bits" extension parameter is just a hint for the server to build an extension negotiation response.

If the "agreed parameters" contain the "server_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.

7.2.2. 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 "server_no_context_takeover" extension parameter, the client MAY decompress each new message with an empty LZ77 sliding window. Otherwise, the client MUST decompress each new message using the LZ77 sliding window used to process the last compressed message.

If the "agreed parameters" contain the "client_no_context_takeover" extension parameter, the server MAY decompress each new message with an empty LZ77 sliding window. Otherwise, the server MUST decompress each new message using the LZ77 sliding window used to process the last compressed message. Note that even if the client has included the "client_no_context_takeover" extension parameter in its offer, the server MUST decompress each new message using the LZ77 sliding window used to process the last compressed message if the "agreed parameters" don't contain the "client_no_context_takeover" extension parameter. The client-to-server "client_no_context_takeover" extension parameter is just a hint for the server to build an extension negotiation response.

If the "agreed parameters" contain the "server_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 "client_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. Note that even if the client has included in its offer the "client_max_window_bits" extension parameter with a value smaller than one in the "agreed parameters", the client MUST use an LZ77 sliding window of a size that conforms the "agreed parameters" to compress messages to send. The client-to-server "client_max_window_bits" extension parameter is just a hint for the server to build an extension negotiation response.

7.2.3. Examples

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

7.2.3.1. A Message Compressed Using One Compressed DEFLATE Block

Suppose that an endpoint sends a text message "Hello". If the endpoint uses one compressed DEFLATE block (compressed with fixed Huffman code and the "BFINAL" bit 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 one 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
```

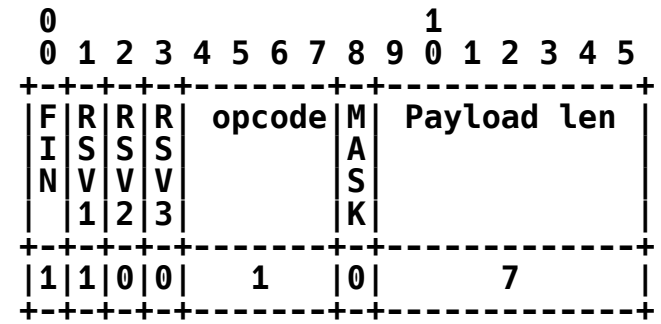
By stripping 0x00 0x00 0xff 0xff from the tail end, the endpoint gets the data to use for the message payload:

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

Suppose that the endpoint sends this compressed message without fragmentation. The endpoint builds one frame by putting all of the 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 octets, 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.

7.2.3.2. 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
```

The above is the payload of the first message that the client has sent. If the "agreed parameters" contain the "client_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 "client_no_context_takeover" extension parameter, the client can compress the payload of the next message into fewer bytes by referencing the history in the LZ77 sliding window. So, the payload of the second message will be:

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

So, 2 bytes are saved in total.

Note that even if some uncompressed messages (with the RSV1 bit unset) are inserted between the two "Hello" messages, they don't affect the LZ77 sliding window.

7.2.3.3. Using a DEFLATE Block with No Compression

A DEFLATE block with no compression may be used.

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

This is a frame constituting a text message "Hello" built 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 3rd to 13th octets consist of the payload data containing "Hello" compressed using a DEFLATE block with no compression.

7.2.3.4. Using a DEFLATE Block with "BFINAL" Set to 1

On platforms on which the flush method using an empty DEFLATE block with no compression is not available, implementors can choose to flush data using DEFLATE blocks with "BFINAL" set to 1.

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

This is the 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.

7.2.3.5. Two DEFLATE Blocks in One Message

Two or more DEFLATE blocks may be used in one 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.

7.2.3.6. Generating an Empty Fragment

Suppose that an endpoint is sending data of unknown size. The endpoint may encounter the end-of-data signal from the data source when its buffer for uncompressed data is empty. In such a case, the endpoint just needs to send the last fragment with the FIN bit set to 1 and the payload set to the DEFLATE block(s), which contains 0 bytes of data. If the compression library being used doesn't generate any data when its buffer is empty, an empty uncompressed DEFLATE block can be built and used for this purpose as follows:

```
0x00
```

The single octet 0x00 contains the header bits with "BFINAL" set to 0 and "BTYPE" set to 00, and 5 padding bits of 0.

7.3. Implementation Notes

On most common software development platforms, the DEFLATE compression library provides a method for aligning 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.

Some platforms may only provide methods to output and process compressed data with a zlib header and an Adler-32 checksum. On such platforms, developers need to write stub code to remove and complement the zlib and Adler-32 checksum by themselves.

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

If a side disallows context takeover, its endpoint can easily figure out whether or not a certain message will be shorter if compressed. Otherwise, it's not easy to know whether future messages will benefit from having a certain message compressed. Implementors may employ some heuristics to determine this.

8. Security Considerations

There is a known exploit when history-based compression is combined with a secure transport [CRIME]. Implementors should pay attention to this point when integrating this extension with other extensions or protocols.

9. IANA Considerations

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

IANA has registered the following WebSocket extension name in the "WebSocket Extension Name Registry" defined in [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.

9.2. Registration of the "Per-Message Compressed" WebSocket Framing Header Bit

IANA has registered the following WebSocket framing header bit in the "WebSocket Framing Header Bits Registry" defined in [RFC6455].

Value
: RSV1

Description
: The "Per-Message Compressed" bit, which indicates whether or not the message is compressed. RSV1 is set for compressed messages and unset for uncompressed messages.

Reference
: Section 6 of this document.

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

10. References

10.1. Normative References

[CRIME] Rizzo, J. and T. Duong, "The CRIME attack", EKOparty Security Conference, September 2012.

[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, DOI 10.1109/TIT.1977.1055714, May 1977, <https://www.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempel_1977_universal_algorithm.pdf>.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <http://www.rfc-editor.org/info/rfc1951>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <http://www.rfc-editor.org/info/rfc6455>.

10.2. Informative References

[RFC1979] Woods, J., "PPP Deflate Protocol", RFC 1979, DOI 10.17487/RFC1979, August 1996, <http://www.rfc-editor.org/info/rfc1979>.

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

Acknowledgements

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

Thanks to the following people who participated in discussions on the HyBi WG and contributed ideas and/or provided detailed reviews (the list is likely incomplete): Adam Rice, Alexander Philippou, 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, Salvatore Loreto, Simone Bordet, Tobias Oberstein, and Yutaka Hirano. Note that the people listed above didn't necessarily endorse the end result of this work.

Author's Address

Takeshi Yoshino
Google, Inc.

Email: tyoshino@google.com