HTTPbis Working Group
Internet-Draft
Intended status: Informational
Expires: December 27, 2013

R. Peon
Google, Inc
H. Ruellan
Canon CRF

June 25, 2013

HTTP Header Compression
draft-ietf-httpbis-header-compression-00

Abstract

This document describes a format adapted to efficiently represent HTTP headers in the context of HTTP/2.0.

Status of This Memo

This Internet-Draft is submitted 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 December 27, 2013.

Table of Contents

1. Introduction

This document describes a format adapted to efficiently represent HTTP headers in the context of HTTP/2.0.

2. Overview

In HTTP/1.X, HTTP headers, which are necessary for the functioning of the protocol, are transmitted with no transformations. Unfortunately, the amount of redundancy in both the keys and the values of these headers is astonishingly high, and is the cause of increased latency on lower bandwidth links. This indicates that an alternate encoding for headers would be beneficial to latency, and that is what is proposed here. As shown by SPDY [SPDY], Deflate compresses HTTP very effectively. However, the use of a compression

scheme which allows for arbitrary matches against the previously encoded data (such as Deflate) exposes users to security issues. In particular, the compression of sensitive data, together with other data controlled by an attacker, may lead to leakage of that sensitive data, even when the resultant bytes are transmitted over an encrypted channel. Another consideration is that processing and memory costs of a compressor such as Deflate may also be too high for some classes of devices, for example when doing forward or reverse proxying.

2.1. Outline

The HTTP header representation described in this document is based on indexing tables that store (name, value) pairs, called header tables in the remainder of this document. This scheme is believed to be safe for all known attacks against the compression context today. Header tables are incrementally updated during the whole HTTP/2.0 session. Two independent header tables are used during a HTTP/2.0 session, one for HTTP request headers and one for HTTP response headers.

The encoder is responsible for deciding which headers to insert as (name, value) pairs in the header table. The decoder then does exactly what the encoder prescribes, ending in a state that exactly matches the encoder's state. This enables decoders to remain simple and understand a wide variety of encoders.

A header may be represented as a literal or as an index. If represented as a literal, the representation specifies whether this header is used to update the indexing table. The different representations are described in Section 3.3.

A set of headers is coded as a difference from the previous set of headers.

An example illustrating the use these different mechanisms to represent headers is available in Appendix B.

3. Header Encoding

3.1. Encoding Components

The encoding and decoding of headers relies on a few components. First, a header table (see Section 3.2) is used to associate headers to index values. Second, a set of headers is encoded as a difference from the previous reference set of headers (see Section 3.4).

As messages are exchanged in two directions, from client to server and from server to client, there are two sets of components: one for

each direction. All the headers sent in messages from the client to the server are encoded (and decoded) using one set of components. All the headers sent in messages from the server to the client (including headers contained in PUSH_PROMISE frame) are encoded using the other set of compotents.

3.2. Header Table

A header table consists of an ordered list of (name, value) pairs. A pair is either inserted at the end of the table or replaces an existing pair depending on the chosen representation. A pair can be represented as an index which is its position in the table, starting with 0 for the first entry.

Header names are always represented as lower-case strings. An input header name matches the header name of a (name, value) pair stored in the Header Table if they are equal using a character-based, _case insensitive_ comparison. An input header value matches the header value of a (name, value) pair stored in the Header Table if they are equal using a character-based, _case sensitive_ comparison. An input header (name, value) pair matches a pair in the Header Table if both the name and value are matching as per above.

The header table is progressively updated based on headers represented as literal (as defined in Section 3.3.1). Two update mechanisms are defined:

- Incremental indexing: the represented header is inserted at the end of the header table as a (name, value) pair. The inserted pair index is set to the next free index in the table: it is equal to the number of headers in the table before its insertion.

- Substitution indexing: the represented header contains an index to an existing (name, value) pair. The existing pair value is replaced by the pair representing the new header.

Incremental and substitution indexing are optional. If none of them is selected in a header representation, the header table is not updated. In particular, no update happens on the header table when processing an indexed representation.

The header table size can be bounded so as to limit the memory requirements (see the SETTINGS_MAX_BUFFER_SIZE in Section 5). The header table size is defined as the sum of the size of each entry of the table. The size of an entry is the sum of the length in bytes (as defined in Section 4.2.2) of its name, of value's length in bytes and of 32 bytes (for accounting for the entry structure overhead).

When an entry is added to the header table, if the header table size is greater than the limit, the table size is reduced by dropping the entries at the beginning of the table until the header table size becomes lower than or equal to the limit. Dropping entries from the beginning of the table causes a renumbering of the remaining entries. [[Feedback is needed on this automatic eviction strategy.]]

To optimize the representation of the headers exchanged at the beginning of an HTTP/2.0 session, the header table is initialized with common headers. Two lists of initial headers are provided in Appendix A. One is for messages sent from a client to a server, the other is for messages sent from a server to a client.

3.3. Header Representation

3.3.1. Literal Representation

The literal representation defines a new header. A literal header is represented as:

- A header name, with two possible representations:
  - A literal string, as described in Section 4.2.2.
  - A index in the header table referencing the name of the corresponding header. The index is represented as an integer, as described in Section 4.2.1.
- The header value, represented as a literal string, as described in Section 4.2.2.

3.3.2. Indexed Representation

The indexed representation defines a header as a match to a (name, value) pair in the header table. An indexed header is represented as:

- An integer representing the index of the matching (name, value) pair, as described in Section 4.2.1.

3.4. Differential Coding

A set of headers is encoded as a difference from the previous reference set of headers. The initial reference set of headers is the empty set.

An indexed representation toggles the presence of the header in the current set of headers. If the header corresponding to the indexed

representation was not in the set, it is added to the set. If the header index was in the set, it is removed from it.

A literal representation adds a header to the current set of headers if the header is added to the header table (either by incremental indexing or by substitution indexing).

To ensure a correct decoding of a set of headers, the following steps or equivalent ones MUST be executed by the decoder.

First, upon starting the decoding of a new set of headers, the reference set of headers is interpreted into the working set of headers: for each header in the reference set, an entry is added to the working set, containing the header name, its value, and its current index in the header table.

Then, the header representations are processed in their order of occurrence in the frame.

For an indexed representation, the decoder checks whether the index is present in the working set. If true, the corresponding entry is removed from the working set. If several entries correspond to this encoded index, all these entries are removed from the working set. If the index is not present in the working set, it is used to retrieve the corresponding header from the header table, and a new entry is added to the working set representing this header.

For a literal representation, a new entry is added to the working set representing this header. If the literal representation specifies that the header is to be indexed, the header is added accordingly to the header table, and its index is included in the entry in the working set. Otherwise, the entry in the working set contains an undefined index.

When all the header representations have been processed, the working set contains all the headers of the set of headers.

The new reference set of headers is computed by removing from the working set all the headers that are not present in the header table.

It should be noted that during the decoding of the header representations, the same index may be associated to different headers in the working set and in the header table.

4. Detailed Format

4.1. Header Blocks

A header block consists of a set of header fields, which are name-value pairs. Each header field is encoded using one of the header representation.

4.2. Low-level representations

4.2.1. Integer representation

Integers are used to represent name indexes, pair indexes or string lengths. The integer representation keeps byte-alignment as much as possible as this allows various processing optimizations as well as efficient use of DEFLATE. For that purpose, an integer representation always finishes at the end of a byte.

An integer is represented in two parts: a prefix that fills the current byte and an optional list of bytes that are used if the integer value does not fit in the prefix. The number of bits of the prefix (called N) is a parameter of the integer representation.

The N-bit prefix allows filling the current byte. If the value is small enough (strictly less than 2^N-1), it is encoded within the N-bit prefix. Otherwise all the bits of the prefix are set to 1 and the value is encoded using an unsigned variable length integer [1] representation.

The algorithm to represent an integer I is as follows:

1. If I < 2^N - 1, encode I on N bits

2. Else, encode 2^N - 1 on N bits and do the following steps:

3.

    1. Set I to (I - (2^N - 1)) and Q to 1

    2. While Q > 0

    3.

        1. Compute Q and R, quotient and remainder of I divided by 2^7

        2. If Q is strictly greater than 0, write one 1 bit; otherwise, write one 0 bit

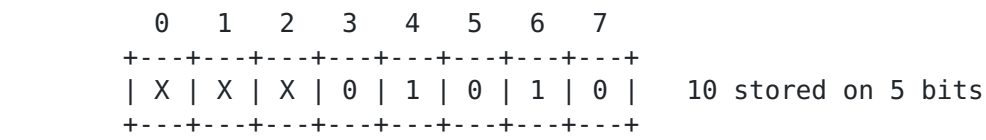
        3. Encode R on the next 7 bits

        4. I = Q

4.2.1.1. Example 1: Encoding 10 using a 5-bit prefix

The value 10 is to be encoded with a 5-bit prefix.

- 10 is less than 31 (= 2^5 - 1) and is represented using the 5-bit prefix.

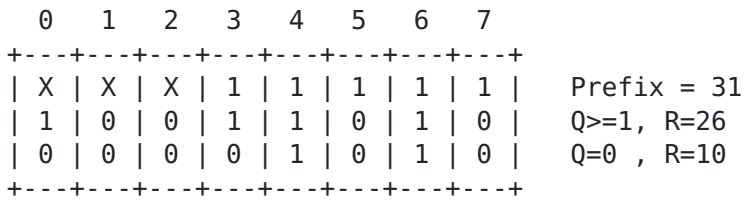


4.2.1.2. Example 2: Encoding 1337 using a 5-bit prefix

The value I=1337 is to be encoded with a 5-bit prefix.

- 1337 is greater than 31 (= 2^5 - 1).
-
  - The 5-bit prefix is filled with its max value (31).
- The value to represent on next bytes is I = 1337 - (2^5 - 1) = 1306.
-
  - 1306 = 128*10 + 26, i.e. Q=10 and R=26.
  - Q is greater than 1, bit 8 is set to 1.
  - The remainder R=26 is encoded on next 7 bits.
  - I is replaced by the quotient Q=10.
- The value to represent on next bytes is I = 10.
-
  - 10 = 128*0 + 10, i.e. Q=0 and R=10.
  - Q is equal to 0, bit 16 is set to 0.
  - The remainder R=10 is encoded on next 7 bits.
  - I is replaced by the quotient Q=0.

o The process ends.



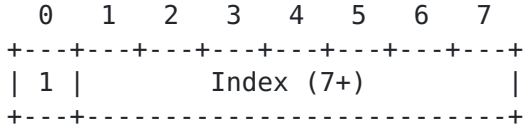
4.2.2. String literal representation

Literal strings can represent header names or header values. They are encoded in two parts:

1. The string length, defined as the number of bytes needed to store its UTF-8 representation, is represented as an integer with a zero bits prefix. If the string length is strictly less than 128, it is represented as one byte.

2. The string value represented as a list of UTF-8 characters.

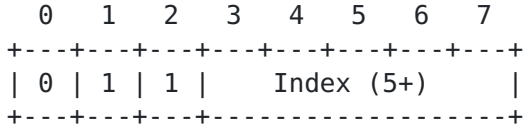
4.3. Indexed Header Representation



This representation starts with the '1' 1-bit prefix, followed by the index of the matching pair, represented as an integer with a 7-bit prefix.

4.4. Literal Header Representation

4.4.1. Literal Header without Indexing



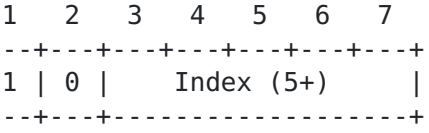
This representation, which does not involve updating the header table, starts with the '011' 3-bit pattern.

If the header name matches the header name of a (name, value) pair stored in the Header Table, the index of the pair increased by one (index + 1) is represented as an integer with a 5-bit prefix. Note that if the index is strictly below 31, one byte is used.

If the header name does not match a header name entry, the value 0 is represented on 5 bits followed by the header name, represented as a literal string.

Header name representation is followed by the header value represented as a literal string as described in Section 4.2.2.

4.4.2. Literal Header with Incremental Indexing



This representation starts with the '010' 3-bit pattern.

If the header name matches the header name of a (name, value) pair stored in the Header Table, the index of the pair increased by one (index + 1) is represented as an integer with a 5-bit prefix. Note that if the index is strictly below 31, one byte is used.

If the header name does not match a header name entry, the value 0 is represented on 5 bits followed by the header name, represented as a literal string.

Header name representation is followed by the header value represented as a literal string as described in Section 4.2.2.

4.4.3. Literal Header with Substitution Indexing

This representation starts with the '00' 2-bit pattern.

If the header name matches the header name of a (name, value) pair stored in the Header Table, the index of the pair increased by one (index + 1) is represented as an integer with a 6-bit prefix. Note that if the index is strictly below 62, one byte is used.

If the header name does not match a header name entry, the value 0 is represented on 6 bits followed by the header name, represented as a literal string.

The index of the substituted (name, value) pair is inserted after the header name representation as a 0-bit prefix integer.

This index is followed by the header value represented as a literal string as described in Section 4.2.2.

5. Parameter Negotiation

A few parameters can be used to accomodate client and server processing and memory requirements.

SETTINGS_MAX_BUFFER_SIZE: Allows the sender to inform the remote endpoint of the maximum size it accepts for the header table. The default value is 4096 bytes.
[[Is this default value OK? Do we need a maximum size? Do we want to allow infinite buffer?]]
When the remote endpoint receives a SETTINGS frame containing a SETTINGS_MAX_BUFFER_SIZE setting with a value smaller than the one currently in use, it MUST send as soon as possible a HEADER frame with a stream identifier of 0x0 containing a value smaller than or equal to the received setting value.
[[This changes slightly the behaviour of the HEADERS frame, which should be updated as follows:]]
A HEADER frame with a stream identifier of 0x0 indicates that the sender has reduced the maximum size of the header table. The new maximum size of the header table is encoded on 32-bit. The decoder MUST reduce its own header table by dropping entries from it until the size of the header table is lower than or equal to the transmitted maximum size.

6. Security Considerations

TODO?

7. IANA Considerations

This memo includes no request to IANA.

8. Informative References

[SPDY] Belshe, M. and R. Peon, "SPDY Protocol", February 2012, <http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy>.

Appendix A. Initial header names

[[The tables in this section should be updated based on statistical analysis of header names frequency and specific HTTP 2.0 header rules (like removal of some headers).]]
[[These tables are not adapted for headers contained in PUSH_PROMISE frames. Either the tables can be merged, or the table for responses can be updated.]]

A.1. Requests

The following table lists the pre-defined headers that make-up the initial header table user to represent requests sent from a client to a server.

Index	Header Name	Header Value
0	:scheme	http
1	:scheme	https
2	:host	
3	:path	/
4	:method	get
5	accept	
6	accept-charset	
7	accept-encoding	
8	accept-language	
9	cookie	
10	if-modified-since	
11	keep-alive	
12	user-agent	
13	proxy-connection	
14	referer	
15	accept-datetime	
16	authorization	
17	allow	
18	cache-control	
19	connection	
20	content-length	
21	content-md5	
22	content-type	
23	date	
24	expect	
25	from	
26	if-match	
27	if-none-match	
28	if-range	
29	if-unmodified-since	

30	max-forwards	
31	pragma	
32	proxy-authorization	
33	range	
34	te	
35	upgrade	
36	via	
37	warning	

Table 1

A.2. Responses

The following table lists the pre-defined headers that make-up the initial header table used to represent responses sent from a server to a client. The same header table is also used to represent request headers sent from a server to a client in a PUSH_PROMISE frame.

Index	Header Name	Header Value
0	:status	200
1	age	
2	cache-control	
3	content-length	
4	content-type	
5	date	
6	etag	
7	expires	
8	last-modified	
9	server	
10	set-cookie	
11	vary	
12	via	
13	access-control-allow-origin	
14	accept-ranges	
15	allow	
16	connection	
17	content-disposition	
18	content-encoding	
19	content-language	
20	content-location	
21	content-md5	
22	content-range	
23	link	
24	location	
25	p3p	

	26	pragma	
	27	proxy-authenticate	
	28	refresh	
	29	retry-after	
	30	strict-transport-security	
	31	trailer	
	32	transfer-encoding	
	33	warning	
	34	www-authenticate	

Table 2

Appendix B. Example

Here is an example that illustrates different representations and how tables are updated. [[This section needs to be updated to integrate differential coding.]]

B.1. First header set

The first header set to represent is the following:

```
:path: /my-example/index.html
user-agent: my-user-agent
x-my-header: first
```

The header table is empty, all headers are represented as literal headers with indexing. The 'x-my-header' header name is not in the header name table and is encoded literally. This gives the following representation:

```
0x44      (literal header with incremental indexing, name index = 3)
0x16      (header value string length = 22)
/my-example/index.html
0x4D      (literal header with incremental indexing, name index = 12)
0x0D      (header value string length = 13)
my-user-agent
0x40      (literal header with incremental indexing, new name)
0x0B      (header name string length = 11)
x-my-header
0x05      (header value string length = 5)
first
```

The header table is as follows after the processing of these headers:

Header table

Index	Header Name	Header Value	
0	:scheme	http	
1	:scheme	https	
...	...	...	
37	warning		
38	:path	/my-example/index.html	added header
39	user-agent	my-user-agent	added header
40	x-my-header	first	added header

As all the headers in the first header set are indexed in the header table, all are kept in the reference set of headers, which is:

Reference Set:
:path, /my-example/index.html
user-agent, my-user-agent
x-my-header, first

B.2. Second header set

The second header set to represent is the following:

:path: /my-example/resources/script.js
user-agent: my-user-agent
x-my-header: second

Comparing this second header set to the reference set, the first and third headers are from the reference set are not present in this second header set and must be removed. In addition, in this new set, the first and third headers have to be encoded. The path header is represented as a literal header with substitution indexing. The x -my-header will be represented as a literal header with incremental indexing.

0xa6 (indexed header, index = 38: removal from reference set)
0xa8 (indexed header, index = 40: removal from reference set)

```
0x04       (literal header, substitution indexing, name index = 3)
0x26       (replaced entry index = 38)
0x1f       (header value string length = 31)
/my-example/resources/script.js
0x5f 0x0a  (literal header, incremental indexing, name index = 40)
0x06       (header value string length = 6)
second
```

The header table is updated as follow:

Header table

Index	Header Name	Header Value	
0	:scheme	http	
1	:scheme	https	
...	...	...	
37	warning		
38	:path	/my-example/resources/ script.js	replaced header
39	user-agent	my-user-agent	
40	x-my-header	first	
41	x-my-header	second	added header

All the headers in this second header set are indexed in the header table, therefore, all are kept in the reference set of headers, which becomes:

Reference Set:

```
:path, /my-example/resources/script.js
user-agent, my-user-agent
x-my-header, second
```

Authors' Addresses

Roberto Peon
Google, Inc

EMail: fenix@google.com

Herve Ruellan
Canon CRF

EMail: herve.ruellan@crf.canon.fr