HTTPbis Working Group
Internet-Draft
Intended status: Informational
Expires: February 28, 2014

R. Peon
Google, Inc
H. Ruellan
Canon CRF

August 27, 2013

HPACK - Header Compression for HTTP/2.0
draft-ietf-httpbis-header-compression-03

Abstract

This document describes HPACK, 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 February 28, 2014.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
2. Overview 3
  2.1. Outline 3
3. Header Encoding 3
  3.1. Encoding Concepts 3
    3.1.1. Encoding Context 4
    3.1.2. Header Table 4
    3.1.3. Reference Set 5
    3.1.4. Header set 6
    3.1.5. Header Representation 6
    3.1.6. Header Emission 6
  3.2. Header Set Processing 7
    3.2.1. Header Representation Processing 7
    3.2.2. Reference Set Emission 7
    3.2.3. Header Set Completion 8
    3.2.4. Header Table Management 8
4. Detailed Format 8
  4.1. Low-level representations 8
    4.1.1. Integer representation 8
    4.1.2. Header Name Representation 10
    4.1.3. Header Value Representation 11
  4.2. Indexed Header Representation 11
  4.3. Literal Header Representation 11
    4.3.1. Literal Header without Indexing 11
    4.3.2. Literal Header with Incremental Indexing 12
    4.3.3. Literal Header with Substitution Indexing 13
5. Parameter Negotiation 15
6. Security Considerations 15
7. IANA Considerations 16
8. References 16
  8.1. Normative References 16
  8.2. Informative References 16
Appendix A. Change Log (to be removed by RFC Editor before publication 17
  A.1. Since draft-ietf-httpbis-header-compression-01 17
  A.2. Since draft-ietf-httpbis-header-compression-01 17
Appendix B. Initial Header Tables 18
  B.1. Requests 18
  B.2. Responses 19
Appendix C. Example 20
  C.1. First header set 20
  C.2. Second header set 22

1. Introduction

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

2. Overview

In HTTP/1.X, headers are sent without any form of compression. As web pages have grown to include dozens to hundreds of requests, the redundant headers in these requests now pose a problem of measurable latency and unnecessary bandwidth. 1 [PERF1] 2 [PERF2]

SPDY [SPDY] initially addressed this redundancy by compressing headers with Deflate, which proved very effective at eliminating the redundant headers. However, that aproach exposed a security risk as demonstrated by the CRIME [CRIME].

In this document, we propose a new header compressor which eliminates the redundant headers, is not vulnerable to the CRIME style attack, and which also has a bounded memory cost for use in small constrained environments.

2.1. Outline

The HTTP header encoding described in this document is based on a header table that map (name, value) pairs to index values. Header tables are incrementally updated during the HTTP/2.0 session.

The encoder is responsible for deciding which headers to insert as new entries 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.

As two consecutive sets of headers often have headers in common, each set of headers is coded as a difference from the previous set of headers. The goal is to only encode the changes (headers present in one of the set and not in the other) between the two sets of headers.

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

3. Header Encoding

3.1. Encoding Concepts

The encoding and decoding of headers relies on some components and concepts. The set of components used form an encoding context.

Header Table: The header table (see Section 3.1.2) is a component used to associate headers to index values.

Reference Set: The reference set (see Section 3.1.3) is a component containing a group of headers used as a reference for the differential encoding of a new set of headers.

Header Set: A header set (see Section 3.1.4) is a group of headers that are encoded jointly. A complete set of key-value pairs as encoded in an HTTP request or response is a header set.

Header Representation: A header can be represented in encoded form either as a literal or as an index (see Section 3.1.5). The indexed representation is based on the header table.

Header Emission: When decoding a set of headers, some operations emit a header (see Section 3.1.6). An emitted header is added to the set of headers that form the HTTP request or response. Once emitted, a header can't be removed from the set of headers.

3.1.1. Encoding Context

The set of components used to encode or decode a header set form an encoding context: an encoding context contains a header table and a reference set.

Using HTTP, messages are exchanged between a client and a server in both direction. To keep the encoding of headers in each direction independent from the other direction, there is one encoding context for each direction.

The headers contained in a PUSH_PROMISE frame sent by a server to a client are encoded within the same context as the headers contained in the HEADERS frame corresponding to a response sent from the server to the client.

3.1.2. Header Table

A header table consists of an ordered list of (name, value) pairs. The first entry of a header table is assigned the index 0.

A header can be represented by an entry from the header table. Rather than encoding a literal value for the header field name and value, the encoder can select an entry from the header table.

Literal header names MUST be translated to lowercase before encoding and transmission. This enables an encoder to perform direct bitwise comparisons on names and values when determining if an entry already

exists in the header table.

There is no need for the header table to contain duplicate entries. However, duplicate entries MUST NOT be treated as an error by a decoder.

Initially, a header table contains a list of common headers. Two initial lists of header are provided in Appendix B. One list is for headers transmitted from a client to a server, the other for the reverse direction.

A header table is modified by either adding a new entry at the end of the table, or by replacing an existing entry.

The encoder decides how to update the header table and as such can control how much memory is used by the header table. To limit the memory requirements on the decoder side, the header table size is bounded (see the SETTINGS_HEADER_TABLE_SIZE in Section 5).

The size of an entry is the sum of its name's length in bytes (as defined in Section 4.1.2), of its value's length in bytes (Section 4.1.3) and of 32 bytes. The 32 bytes are an accounting for the entry structure overhead. For example, an entry structure using two 64-bits pointers to reference the name and the value and the entry, and two 64-bits integer for counting the number of references to these name and value would use 32 bytes.

The size of a header table is the sum of the size of its entries.

3.1.3. Reference Set

A reference set is defined as an unordered set of references to entries of the header table.

The initial reference set is the empty set.

The reference set is updated during the processing of a set of headers.

Using the differential encoding, a header that is not present in the reference set can be encoded either with an indexed representation (if the header is present in the header table), or with a literal representation (if the header is not present in the header table).

A header that is to be removed from the reference set is encoded with an indexed representation.

3.1.4. Header set

A header set is a group of header fields that are encoded as a whole. Each header field is a (name, value) pair.

A header set is encoded using an ordered list of zero or more header representations. All the header representations describing a header set a grouped into a header block.

3.1.5. Header Representation

A header can be represented either as a literal or as an index.

Literal Representation: A literal representation defines a new header. The header name is represented either literally or as a reference to an entry of the header table. The header value is represented literally.

Three different literal representations are provided:

* A literal representation that does not add the header to the header table (see Section 4.3.1).

* A literal representation that adds the header at the end of the header table (see Section 4.3.2).

* A literal representation that uses the header to replace an existing entry of the header table (see Section 4.3.3).

Indexed Representation: The indexed representation defines a header as a reference in the header table (see Section 4.2).

3.1.6. Header Emission

The emission of header is the process of adding a header to the current set of headers. Once an header is emitted, it can't be removed from the current set of headers.

The concept of header emission allows a decoder to know when it can pass a header safely to a higher level on the receiver side. This allows a decoder to be implemented in a streaming way, and as such to only keep in memory the header table and the reference set. With such an implementation, the amount of memory used by the decoder is bounded, even in presence of a very large set of headers. The management of memory for handling very large sets of headers can therefore be deferred to the application, which may be able to emit the header to the wire and thus free up memory quickly.

3.2. Header Set Processing

The processing of an encoded header set to obtain a list of headers is defined in this section. To ensure a correct decoding of a header set, a decoder MUST obey the following rules.

3.2.1. Header Representation Processing

All the header representations contained in a header block are processed in the order in which they are presented, as specified below.

An _indexed representation_ corresponding to an entry _not present_ in the reference set entails the following actions:

- The header corresponding to the entry is emitted.

- The entry is added to the reference set.

An _indexed representation_ corresponding to an entry _present_ in the reference set entails the following actions:

- The entry is removed from the reference set.

A _literal representation_ that is _not added_ to the header table entails the following action:

- The header is emitted.

A _literal representation_ that is _added_ to the header table entails the following actions:

- The header is emitted.

- The header is added to the header table, at the location defined by the representation.

- The new entry is added to the reference set.

3.2.2. Reference Set Emission

Once all the representations contained in a header block have been processed, the headers that are in common with the previous header set are emitted, during the reference set emission.

For the reference set emission, each header contained in the reference set that has not been emitted during the processing of the header block is emitted.

3.2.3. Header Set Completion

Once all of the header representations have been processed, and the remaining items in the reference set have been emitted, the header set is complete.

3.2.4. Header Table Management

The header table can be modified by either adding a new entry to it or by replacing an existing one. Before doing such a modification, it has to be ensured that the header table size will stay lower than or equal to the SETTINGS_HEADER_TABLE_SIZE limit (see Section 5). To achieve this, repeatedly, the first entry of the header table is removed, until enough space is available for the modification.

A consequence of removing one or more entries at the beginning of the header table is that the remaining entries are renumbered. The first entry of the header table is always associated to the index 0.

When the modification of the header table is the replacement of an existing entry, the replaced entry is the one indicated in the literal representation before any entry is removed from the header table. If the entry to be replaced is removed from the header table when performing the size adjustment, the replacement entry is inserted at the beginning of the header table.

The addition of a new entry with a size greater than the SETTINGS_HEADER_TABLE_SIZE limit causes all the entries from the header table to be dropped and the new entry not to be added to the header table. The replacement of an existing entry with a new entry with a size greater than the SETTINGS_HEADER_TABLE_SIZE has the same consequences.

4. Detailed Format

4.1. Low-level representations

4.1.1. Integer representation

Integers are used to represent name indexes, pair indexes or string lengths. To allow for optimized processing, 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:

```
If I < 2^N - 1, encode I on N bits
Else
    encode 2^N - 1 on N bits
    I = I - (2^N - 1)
    While I >= 128
         Encode (I % 128 + 128) on 8 bits
         I = I / 128
    encode (I) on 8 bits
```

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

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

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

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| X | X | X | 0 | 1 | 0 | 1 | 0 |   10 stored on 5 bits
+---+---+---+---+---+---+---+---+
```

4.1.1.2. 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).

I = 1337 - (2^5 - 1) = 1306.

I (1306) is greater than or equal to 128, the while loop body executes:

I % 128 == 26

26 + 128 == 154

154 is encoded in 8 bits as: 10011010

I is set to 10 (1306 / 128 == 10)

I is no longer greater than or equal to 128, the while loop terminates.

I, now 10, is encoded on 8 bits as: 00001010

The process ends.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| X | X | X | 1 | 1 | 1 | 1 | 1 |   Prefix = 31, I = 1306
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |   1306>=128, encode(154), I = 1306/128
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |   10<128, encode(10), done
+---+---+---+---+---+---+---+---+
```

4.1.2. Header Name Representation

Header names are sequences of ASCII characters that MUST conform to the following header-name ABNF construction:

```
LOWERALPHA = %x61-7A
header-char = "!" / "#" / "$" / "%" / "&" / "'" /
              "*" / "+" / "-" / "." / "^" / "_" /
              "`" / "|" / "~" / DIGIT / LOWERALPHA
header-name = [":"] 1*header-char
```

They are encoded in two parts:

1. The length of the text, defined as the number of octets of storage required to store the text, represented as a variable-length-quantity (Section 4.1.1).

2. The specific sequence of ASCII octets

4.1.3. Header Value Representation

Header values are encoded as sequences of UTF-8 encoded text. They are encoded in two parts:

1. The length of the text, defined as the number of octets of storage required to store the text, represented as a variable-length-quantity (Section 4.1.1).

2. The specific sequence of octets representing the UTF-8 text.

Invalid UTF-8 octet sequences, "over-long" UTF-8 encodings, and UTF-8 octets that represent invalid Unicode Codepoints MUST NOT be used.

4.2. Indexed Header Representation

An indexed header representation identifies an entry in the header table. The entry is emitted and added to the reference set if it is not currently in the reference set. The entry is removed from the reference set if it is present in the reference set.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1 |        Index (7+)         |
+---+---------------------------+
```

Indexed Header

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

4.3. Literal Header Representation

Literal header representations contain a literal header field value. Header field names are either provided as a literal or by reference to an existing header table entry.

Literal representations all result in the emission of a header when decoded.

4.3.1. Literal Header without Indexing

An literal header without indexing causes the emission of a header without altering the header table.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 1 | 1 |    Index (5+)     |
+---+---+---+-------------------+
|       Value Length (8+)       |
+-------------------------------+
| Value String (Length octets)  |
+-------------------------------+
```

Literal Header without Indexing - Indexed Name

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 1 | 1 |         0         |
+---+---+---+-------------------+
|       Name Length (8+)        |
+-------------------------------+
|  Name String (Length octets)  |
+-------------------------------+
|       Value Length (8+)       |
+-------------------------------+
| Value String (Length octets)  |
+-------------------------------+
```

Literal Header without Indexing - New Name

This representation 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 (Section 4.1.2).

Header name representation is followed by the header value represented as a literal string as described in Section 4.1.3.

4.3.2. Literal Header with Incremental Indexing

A literal header with incremental indexing adds a new entry to the header table.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 1 | 0 |    Index (5+)     |
+---+---+---+-------------------+
|       Value Length (8+)       |
+-------------------------------+
| Value String (Length octets)  |
+-------------------------------+
```

Literal Header with Incremental Indexing -
Indexed Name

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 1 | 0 |         0         |
+---+---+---+-------------------+
|       Name Length (8+)        |
+-------------------------------+
|  Name String (Length octets)  |
+-------------------------------+
|       Value Length (8+)       |
+-------------------------------+
| Value String (Length octets)  |
+-------------------------------+
```

Literal Header with Incremental Indexing -
New Name

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 (Section 4.1.2).

Header name representation is followed by the header value represented as a literal string as described in Section 4.1.3.

4.3.3. Literal Header with Substitution Indexing

A literal header with substitution indexing replaces an existing header table entry.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 0 |      Index (6+)       |
+---+---+-----------------------+
|    Substituted Index (8+)     |
+-------------------------------+
|       Value Length (8+)       |
+-------------------------------+
| Value String (Length octets)  |
+-------------------------------+
```

Literal Header with Substitution Indexing -
Indexed Name

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 0 |           0           |
+---+---+-----------------------+
|       Name Length (8+)        |
+-------------------------------+
|  Name String (Length octets)  |
+-------------------------------+
|    Substituted Index (8+)     |
+-------------------------------+
|       Value Length (8+)       |
+-------------------------------+
| Value String (Length octets)  |
+-------------------------------+
```

Literal Header with Substitution Indexing -
New Name

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 63, 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 (Section 4.1.2).

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

The index of the substituted pair MUST correspond to a position in the header table containing a non-void entry. An index for the

substituted pair that corresponds to empty position in the header table MUST be treated as an error.

This index is followed by the header value represented as a literal string as described in Section 4.1.3.

5. Parameter Negotiation

A few parameters can be used to accommodate client and server processing and memory requirements. [[anchor3: These settings are currently not supported as they have not been integrated in the main specification. Therefore, the maximum buffer size for the header table is fixed at 4096 bytes.]]

SETTINGS_HEADER_TABLE_SIZE (TBD): Allows the sender to inform the remote endpoint of the maximum size it accepts for the header table.
The default value is 4096 bytes.
[[anchor4: 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_HEADER_TABLE_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.
[[anchor5: 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

This compressor exists to solve security issues present in stream compressors such as DEFLATE whereby the compression context can be efficiently probed to reveal secrets. A conformant implementation of this specification should be fairly safe against that kind of attack, as the reaping of any information from the compression context requires more work than guessing and verifying the plaintext data directly with the server. As with any secret, however, the longer the length of the secret, the more difficult the secret is to guess. It is inadvisable to have short cookies that are relied upon to remain secret for any duration of time.

A proper security-conscious implementation will also need to prevent

timing attacks by ensuring that the amount of time it takes to do string comparisons is always a function of the total length of the strings, and not a function of the number of matched characters.

Another common security problem is when the remote endpoint successfully causes the local endpoint to exhaust its memory. This compressor attempts to deal with the most obvious ways that this could occur by limiting both the peak and the steady-state amount of memory consumed in the compressor state, by providing ways for the application to consume/flush the emitted headers in small chunks, and by considering overhead in the state size calculation. Implementors must still be careful in the creation of APIs to an implementation of this compressor by ensuring that header keys and values are either emitted as a stream, or that the compression implementation have a limit on the maximum size of a key or value. Failure to implement these kinds of safeguards may still result in a scenario where the local endpoint exhausts its memory.

7. IANA Considerations

This document registers the SETTINGS_HEADER_TABLE_SIZE setting in the "HTTP/2.0 Settings" registry established by [HTTP2]. The assigned code for this setting is TBD.

8. References

8.1. Normative References

[HTTP2] Belshe, M., Peon, R., Thomson, M., and A. Melnikov, "Hypertext Transfer Protocol version 2.0", draft-ietf-httpbis-http2-06 (work in progress), February 2013.

8.2. Informative References

[CRIME] Rizzo, J. and T. Duong, "The Crime Attack", September 2012, <https://docs.google.com/a/twist.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit#slide=id.g1eb6c1b5_3_6>.

[PERF1] Belshe, M., "IETF83: SPDY and What to Consider for HTTP/2.0", March 2012, <http://www.ietf.org/proceedings/83/slides/slides-83-httpbis-3>.

[PERF2] McManus, P., "SPDY What I Like About You", September 2011, <http://bitsup.blogspot.com/2011/09/spdy-what-i-like-about-you.html>.

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

URIs

[1] <http://en.wikipedia.org/wiki/Variable-length_quantity>

Appendix A. Change Log (to be removed by RFC Editor before publication

A.1. Since draft-ietf-httpbis-header-compression-01

- o Refactored of Header Encoding Section: split definitions and processing rule.

- o Backward incompatible change: Updated reference set management as per issue #214. This changes how the interaction between the reference set and eviction works. This also changes the working of the reference set in some specific cases.

- o Backward incompatible change: modified initial header list, as per issue #188.

- o Added example of 32 bytes entry structure (issue #191).

- o Added Header Set Completion section. Reflowed some text. Clarified some writing which was akward. Added text about duplicate header entry encoding. Clarified some language w.r.t Header Set. Changed x-my-header to mynewheader. Added text in the HeaderEmission section indicating that the application may also be able to free up memory more quickly. Added information in Security Considerations section.

A.2. Since draft-ietf-httpbis-header-compression-01

Fixed bug/omission in integer representation algorithm.

Changed the document title.

Header matching text rewritten.

Changed the definition of header emission.

Changed the name of the setting which dictates how much memory the compression context should use.

Removed "specific use cases" section

Corrected erroneous statement about what index can be contained in one byte

Added descriptions of opcodes

Removed security claims from introduction.

Appendix B. Initial Header Tables

[[anchor11: 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).]]
[[anchor12: 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.]]

B.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	user-agent	
12	referer	
13	authorization	
14	allow	
15	cache-control	
16	connection	
17	content-length	
18	content-type	
19	date	
20	expect	
21	from	
22	if-match	
23	if-none-match	
24	if-range	
25	if-unmodified-since	
26	max-forwards	
27	proxy-authorization	
28	range	
29	via	

Table 1: Initial Header Table for Requests

B.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-range	
22	link	
23	location	
24	proxy-authenticate	
25	refresh	
26	retry-after	
27	strict-transport-security	
28	transfer-encoding	
29	www-authenticate	

Table 2: Initial Header Table for Responses

Appendix C. Example

Here is an example that illustrates different representations and how tables are updated. [[anchor15: This section needs to be updated to better reflect the new processing of header fields, and include more examples.]]

C.1. First header set

The first header set to represent is the following:

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

The header table is empty, all headers are represented as literal headers with indexing. The 'mynewheader' 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)
mynewheader
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	mynewheader	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
```

```
mynewheader, first
```

C.2. Second header set

The second header set to represent is the following:

```
:path: /my-example/resources/script.js
user-agent: my-user-agent
mynewheader: 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 mynewheader 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	mynewheader	first	
41	mynewheader	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
mynewheader, second

Authors' Addresses

Roberto Peon
Google, Inc

EMail: fenix@google.com

Herve Ruellan
Canon CRF

EMail: herve.ruellan@crf.canon.fr