Network Working Group
Internet-Draft
Expires: September 12, 2013

H. Ruellan
J. Fujisawa
Canon, Inc.
R. Bellessort
Y. Fablet

March 11, 2013

Header Diff: A compact HTTP header representation for HTTP/2.0
draft-ruellan-headerdiff-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 September 12, 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

2.1. Design Principles

HTTP headers can be represented in various ways. As shown by SPDY, Deflate compresses very well HTTP headers. But the use of Deflate has been found to cause security issues. In particular, the compression of sensitive data, together with other data controlled by an attacker, may lead to leakage of the sensitive data. The processing and memory costs may also be too high for some classes of devices.

Having a lightweight compact HTTP header representation is therefore useful. To design this representation, the focus was put on the following points:

- Simplicity: the representation should have a small number of options that allow handling any kind of headers; in particular, the use of dedicated codecs for each type of header value is not considered here.

- Efficiency: the representation should provide good compression at a small encoding/decoding cost for both processing and memory.

- Flexibility: the representation should be compatible with constrained devices, but also provide improved efficiency when more capable devices are used.

- Deflate-friendly: Deflate has proven its efficiency for encoding HTTP headers. A good HTTP header representation should be efficient as a pre-compression step prior to applying Deflate.

2.2. 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. 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 follows exactly

what the encoder prescribes. This enables decoders to remain simple and understand a wide variety of encoders.

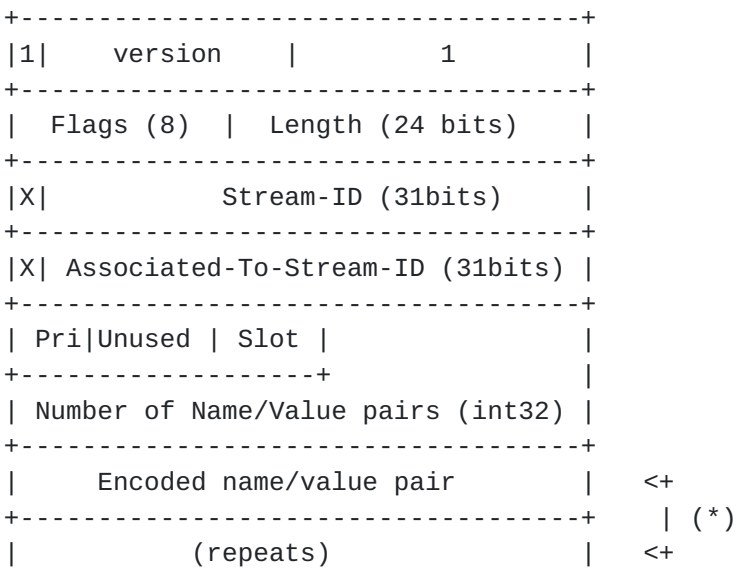
A header may be represented as a literal, an index or a delta. If represented as a literal or a delta, the representation specifies whether this header is used to update the indexing table. The different representations are described in Section 3.2.

To improve literal headers representation compactness, header names are indexed in a specific name table. Two independent name tables are used during a HTTP/2.0 session, one for HTTP request headers and another for HTTP response headers.

An example illustrating the use of the different tables to represent headers is available in Appendix A.3. Once a set of header is represented using the available representations, it can optionally be compressed with Deflate.

2.3. Integration within HTTP/2.0

The headers are inserted in the HTTP/2.0 frames at the same place as defined in SPDY (next chart was adapted from draft-ietf-httpbis-http2-00).



(*) This section is the "Name/Value Header Block", and may be compressed.

The modifications to SPDY are the following:

- The headers are not represented as string tokens but using one of the possible representation described in Section 4.

- The Deflate step is made optional.

2.3.1. Deflate Usage

The header representation described in Section 4 is amenable to Deflate compression. The Deflate algorithm improves the compression at the expense of additional processing.

At least two potential drawbacks have been identified when using Deflate. First, security issues may arise when using Deflate, like the CRIME attack [1]. Second, it may increase the workload of network intermediaries: they may need to uncompress and recompress the headers of all messages, even though they only need to process a few of them.

The use of Deflate may still be envisioned if properly set up. Several approaches are available and should be studied:

Restricting Deflate to Huffman-only coding is an option. This is supported by many Deflate implementations such as zlib. It may be used in the environments subject to CRIME attacks. This approach should be compared to the direct use of hand-tailored Huffman coding.

The use of indexing mechanisms prior Deflate may solve some security issues. More precise analysis of the security impact of using indexing mechanisms prior Deflate should be studied as described in Section 6.3.

Partial use of Deflate on a selected subset of headers may also be an option as described in Section 6.4.

Restricting the use of Deflate to safe cases, such as controlled environments (widgets, native applications), anonymous connections and so on can be envisioned. For instance, restricting the use of Deflate to HTTP response headers should not enable CRIME-like attacks.

3. Indexing Strategies

3.1. Indexing Tables

3.1.1. Header Table

A header table consists in an ordered list of (name, value) pairs. Once a header pair is inserted in the header table, its index does not change until the pair gets removed. A pair is either inserted at the end of the table or replaces an existing pair depending on the chosen representation.

Header names should be represented as lower-case strings. A header name is matching with a pair name if they are equal using a character-based, _case insensitive_ comparison. A header value is matching with a pair value if they are equal using a character-based, _case sensitive_ comparison. A header is matching with a (name,value) pair if both name and value are matching.

The header table is progressively updated based on headers represented as literal (as defined in Section 3.2.1) or delta (as defined in Section 3.2.3). 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 header value.

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. The header table size is defined as the sum of the length (as defined in Section 4.1.2) of the values of all header table pairs. Header names are not counted in the header table size.

3.1.2. Name Table

A name table is an ordered list of name entries that is used to efficiently represent header names. A header name is matching a name table entry if they are equal using a character-based, _case insensitive_ comparison.

If a header name is matching a name table entry, it is represented as an integer based on the index of the entry, as described in Section 4.1.1. If a header name is not matching any of the name table entry, it is represented as a string, as described in Section 4.1.2. A new entry containing the name is then inserted at the end of the name table. Once inserted in the name table, a header name is never removed and its index is never changing.

To optimize the representation of the headers exchanged at the beginning of the HTTP/2.0 session, the header name table is initally populated with common header names. The initial header names list is provided in Appendix A.

3.2. Header Representation

3.2.1. Literal Representation

The literal representation defines a header independentently of the header table. A literal header is represented as:

- A header name, represented using the name table, as described in Section 3.1.2.

- The header value, represented as a literal string, as described in Section 4.1.2.

3.2.2. 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 4.1.1.

3.2.3. Delta Representation

The delta representation defines a header as a reference to a (name,value) pair contained in the header table. The names must match between the represented header and the reference pair. The values should start by a common substring between the represented header and the reference pair.

A delta header is represented as:

- An integer representing the index of the reference (name, value) pair, as described in Section 4.1.1. The pair name must match the name of the header.

- An integer representing the length of the common prefix shared between the header value and the pair value, as described in Section 4.1.1.

- A string representing the suffix value to append to the common prefix to obtain the header value, as defined in Section 4.1.2.

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. 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 [2] 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

4.1.1.1. 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.

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|  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).
- The value to represent on next bytes is I = 1337 - 2^5 = 1305.
-
  - 1305 = 128*10 + 25, i.e. Q=10 and R=25.
  - Q is greater than 1, bit 8 is set to 1.
  - The remainder R=25 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.

```
 +---+---+---+---+---+---+---+---+
|  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
+-----+-----+-----+-----+-----+-----+-----+-----+
|  X  |  X  |  X  |  1  |  1  |  1  |  1  |  1  |  Prefix = 31
|  1  |  0  |  0  |  1  |  1  |  0  |  0  |  1  |  Q>=1, R=25
|  0  |  0  |  0  |  0  |  1  |  0  |  1  |  0  |  Q=0 , R=10
+-----+-----+-----+-----+-----+-----+-----+-----+
```

4.1.2. String literal representation

Literal strings can represent header names, header values or header values suffix in the case of delta coding. 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.

4.2. Indexed Header Representation

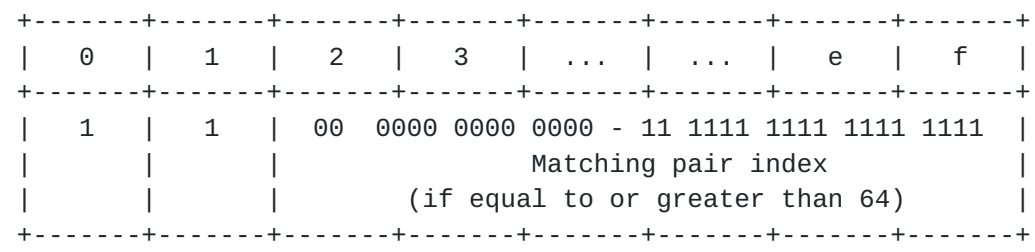
Indexed headers can be represented as short indexed header if the matching pair index is strictly below 64. Otherwise it is represented as a long indexed header.

4.2.1. Short Indexed Header

```
+-------+-------+-------+-------+-------+-------+-------+-------+
|   0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
+-------+-------+-------+-------+-------+-------+-------+-------+
|   1   |   0   |            00 0000 - 11 1111                 |
|       |       |           Matching pair index                 |
|       |       |         (if strictly lower than 64)           |
+-------+-------+-------+-------+-------+-------+-------+-------+
```

This representation starts with the '10' 2-bit pattern, followed by the index of the matching pair, represented on 6 bits. A short indexed header is always coded in one byte.

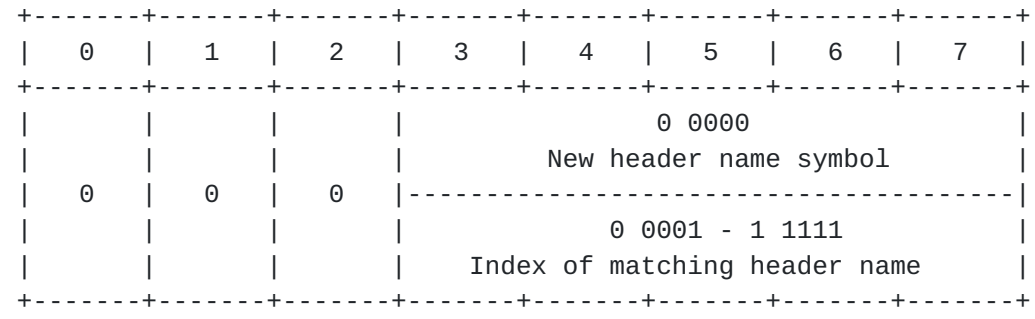
4.2.2. Long Indexed Header



This representation starts with the '11' 2-bit pattern, followed by the value of the index of the matching pair minus 64, represented as an integer with a 14-bit prefix A long indexed header is coded in two bytes if the index minus 64 is strictly below 16383.

4.3. Literal Header Representation

4.3.1. Literal Header without Indexing



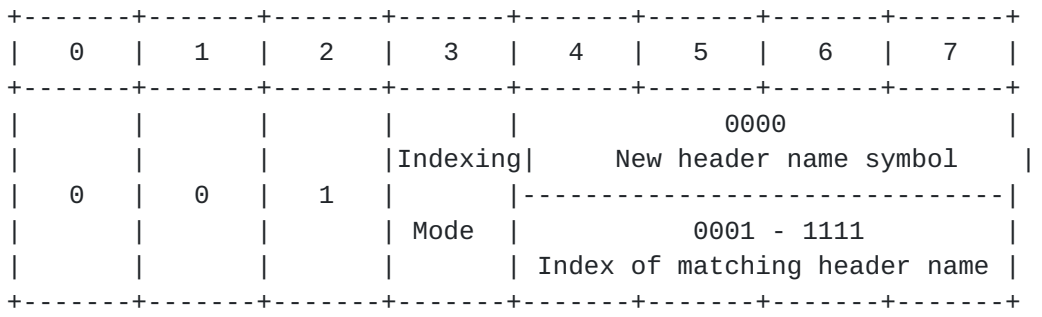
This representation, which does not involve updating the header table, starts with the '000' 3-bit pattern.

If the header name matches a header name entry whose index is IN, the value (IN+1) is represented as an integer with a 5-bit prefix. Note that if the index is strictly below 30, 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.

4.3.2. Literal Header with Indexing



This representation starts with the '001' 3-bit pattern. The fourth bit sets the indexing mode: 0 for incremental indexing, 1 for substitution indexing.

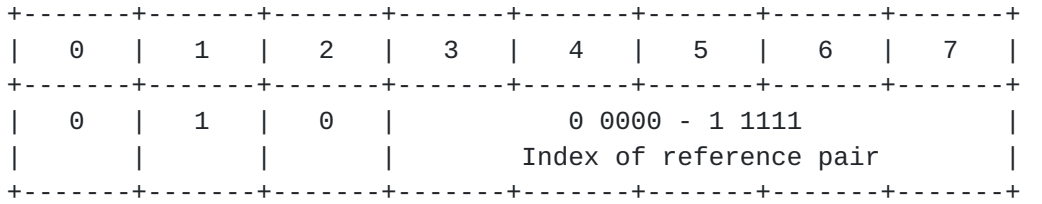
If the header name matches a header name entry whose index is IN, the value (IN+1) is represented as an integer with a 4-bit prefix. Note that if the index is strictly below 14, one byte is used.

If the header name does not match a header name entry, the value 0 is represented on 4 bits followed by the header name, represented as a literal string.

Header name representation is followed by the header value represented as a string as described in Section 4.1.2. In the case of substitution indexing, the substituted (name,value) pair index is inserted before the header value as a zero-bit prefix integer. The header value is represented as a literal string.

4.4. Delta Header Representation

4.4.1. Delta Header without Indexing



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

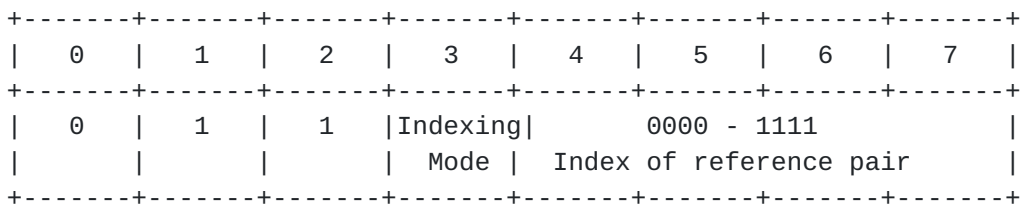
It continues with the index IR of the reference header pair. The value IR is represented as an integer with a 5-bit prefix. Note that if the index is strictly below 31, one byte is used.

Index value is followed by:

1. the length of the common prefix shared between the header value and the pair value, represented as an integer with a zero-bit prefix.

2. the header value suffix represented as a literal string.

4.4.2. Delta Header with Indexing



This representation starts with the '011' 3-bit pattern. The fourth bit sets the indexing mode, 0 for incremental indexing and 1 for substitution indexing.

It continues with the index IR of the reference header pair. The value IR is represented as an integer with a 4-bit prefix. Note that if the index is strictly below 15, one byte is used.

Index value is followed by:

1. the length of the common prefix shared between the header value and the pair value, represented as an integer with a zero-bit prefix.

2. the header value suffix, represented as a literal string.

5. Parameter Negotiation

Two parameters may be used to accomodate the client and server processing and memory requirements:

- A parameter Nh that configures the size of the header table. The size can be computed as 2^Nh. Nh is exchanged as an unsigned integer.

- A parameter Nd that configures the Deflate step. If Nd is equal to zero, no Deflate step is used. Otherwise, Deflate is used with a sliding window equal to 2^Nd. Huffman-only coding is advertised using the Deflate block initial bits. Nd is exchanged as an unsigned integer.

This section should be further completed, including but not limited to the following points:

- Define default values?
- Define when negotiation happens: at the beginning only, any time during the session...
- Define how are exchanged these parameters, probably using SETTINGS frames, through the definition of 4 settings (outgoing-Nd, outgoing-Nh, preferred-incomingNd, preferred-incoming-Nh).

6. Open Questions

6.1. Typed Codecs

Typed codecs may be useful to represent header values, especially on response side. An additional typed header representation could be defined, adding support for a small number of codecs such as:

- An Integer codec: may be useful for headers such as 'Age' and 'Content-Length'.
- A Date codec: may be useful for headers such as 'Date', 'Expires', 'If-Modified-Since', 'Last-Modified'.

6.2. Specific header processing

Some (name,value) pairs may be singled out to improve network nodes processing, such as:

- The request line (verb, version and URL) for HTTP requests.
- The response line (status and version) for HTTP responses.

For those headers, the specification may define specific rules that can improve the processing cost, at the expense of some compression loss:

- All those headers are placed as the first headers in the SYN_STREAM frame, with a predefined order.
- Verb, version and status are represented as integers with zero-bit prefix.
- Indexed, delta or literal representation may be used for URL values. In the case of delta and literal representation, only the substitution mode is used so that a processor only needs to store

the URL of the previous message to compute the URL of the current message.

6.3. Security Issues

Adequate use of delta and indexed representation before using of Deflate are supposed to solve security issues such as the CRIME attack. For instance, if cookie headers are represented as indexed headers as much as possible, attackers may be prevented from progressively learning its value.

This point should be confirmed with deeper analysis. Additional study should also be done to evaluate whether the proposed indexed and delta representation create any new security issue.

6.4. Deflate Partial Usage

To circumvent Deflate issues related to both security and network intermediaries, the header set of a given message can be split in two buckets. A first bucket would be sent without using Deflate, while the second bucket would be further compressed using Deflate. The decision would be done by the encoder. The specification could define a minimum set of headers that SHOULD never be compressed using Deflate, for instance URLs and cookies.

Another possibility would be to define Deflate as a specific representation. The use of Deflate would then be decided on per-header basis. That would enable excluding any header that may contain sensitive data. The overall scheme would be less efficient though (padding bits and so on).

Additional analysis of the complexity and benefit of these approaches would be needed to go further. For instance, these approaches should be compared to the use of Deflate restricted to Huffman-coding in terms of simplicity and compression benefits.

6.5. Max length and entry numbers

The integer representation allows representation of unbounded values. If bounding the table entries number or string lengths, the integer encoding may be further optimized.

- Decide whether to limit the length of strings to a max value and if so which value, 32768?

- Decide whether to limit name table entries to 256?

- Decide whether to limit header table entries to 16384?

7. Security Considerations

This section should be completed according the previous sections.

8. IANA Considerations

This memo includes no request to IANA.

9. References

Appendix A. Initial header names

A.1. Requests

Indexes strictly lower than 14 are always encoded on 1 byte. Hence, the 14 most frequent names should be set in the 14 first positions. This table may be updated based on statistical analysis of header names frequency and specific HTTP 2.0 header rules (like removal of 'proxy-connection', url being split or not...).

Index	Header Name
0	accept
1	accept-charset
2	accept-encoding
3	accept-language
4	cookie
5	method
6	host
7	if-modified-since
8	keep-alive
9	url
10	user-agent
11	version
12	proxy-connection

13	referer
14	accept-datetime
15	authorization
16	allow
17	cache-control
18	connection
19	content-length
20	content-md5
21	content-type
22	date
23	expect
24	from
25	if-match
26	if-none-match
27	if-range
28	if-unmodified-since
29	max-forwards
30	pragma
31	proxy-authorization
32	range
33	te
34	upgrade
35	via
36	warning

```
+---------+-------------------------------------+
```

A.2. Responses

Indexes strictly lower than 14 are always encoded on 1 byte. Hence, the 14 most frequent names should be set in the 14 first positions. This table may be updated based on statistical analysis of header names frequency and specific HTTP 2.0 header rules.

Index	Header Name
0	age
1	cache-control
2	content-length
3	content-type
4	date
5	etag
6	expires
7	last-modified
8	server
9	set-cookie
10	status
11	vary
12	version
13	via
14	access-control-allow-origin
15	accept-ranges
16	allow
17	connection

18	content-disposition
19	content-encoding
20	content-language
21	content-location
22	content-md5
23	content-range
24	link
25	location
26	p3p
27	pragma
28	proxy-authenticate
29	refresh
30	retry-after
31	strict-transport-security
32	trailer
33	transfer-encoding
34	warning
35	www-authenticate

A.3. Example

Here is an example that illustrates different representations and how tables are updated.

A.3.1. First header set

The first header set to represent is the following:

```
url: http://www.example.org/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:

```
0x2A      (literal header with indexing, name index = 9)
0x2C      (header value string length = 44)
http://www.example.org/my-example/index.html
0x2B      (literal header with indexing, name index = 10)
0x0D      (header value string length = 43)
my-user-agent
0x20      (literal header with indexing, new name)
0x0B      (header name string length = 11)
x-my-header
0x05      (header value string length = 5)
first
```

The header tables are as follow after the processing of these headers:

Name table

Index	Header Name	
0	accept	
1	accept-charset	
...	...	
36	warning	
37	x-my-header	added name

Header table

0	url	http://www.example.org/ my-example/index.html	added pair
1	user-agent	my-user-agent	added pair

```
+----+-------------+-----------------------------------+
| 2  | x-my-header | first                             | added pair
+----+-------------+-----------------------------------+
```

A.3.2. Second header set

The second header set to represent is the following:

```
url: http://www.example.org/my-example/resources/script.js
user-agent: my-user-agent
x-my-header: second
```

The url header is represented as a delta header with substitution. The user-agent header will be represented as a short header. The x -my-header will be represented as a literal header with indexing.

```
0x70       (delta header with substitution, header index = 0)
0x22       (common prefix length = 32)
0x13       (suffix value length = 19)
resources/script.js
0x81       (indexed header, index = 1)
0x2f 0x17  (literal header with indexing, name index = 37)
0x05       (header value string length = 5)
second
```

The name table remains unchanged. The header table is updated as follow:

```
+----+-------------+-----------------------------------+
| 0  |     url     | http://www.example.org/           | substituted
|    |             |     my-example/resources/script.js | pair
+----+-------------+-----------------------------------+
| 1  |  user-agent | my-user-agent                     |
+----+-------------+-----------------------------------+
| 2  | x-my-header | first                             |
+----+-------------+-----------------------------------+
| 3  | x-my-header | second                            | added pair
+----+-------------+-----------------------------------+
```

Authors' Addresses

Herve Ruellan

Email: herve.ruellan@crf.canon.fr

Jun Fujisawa
Canon, Inc.
3-30-2 Shimomaruko
Ohta-ku, Tokyo 146-8501
Japan

Email: fujisawa.jun@canon.co.jp

Romain Bellessort

Email: romain.bellessort@crf.canon.fr

Youenn Fablet

Email: youenn.fablet@crf.canon.fr