HTTPbis Working Group R. Peon
Internet-Draft Google, Inc
Intended status: Standards Track H. Ruellan
Expires: August 17, 2014 Canon CRF
February 13, 2014

# HPACK - Header Compression for HTTP/2
## draft-ietf-httpbis-header-compression-06

Abstract

This specification defines HPACK, a compression format for efficiently representing HTTP header fields in the context of HTTP/2.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at [1].

Working Group information and related documents can be found at [2] (Wiki) and [3] (source code and issues tracker).

The changes in this draft are summarized in Appendix A.1.

Table of Contents

## 1. Introduction

This specification defines HPACK, a compression format for efficiently representing HTTP header fields in the context of HTTP/2 (see [HTTP2]).

## 2. Overview

In HTTP/1.1 (see [HTTP-p1]), header fields are encoded without any form of compression. As web pages have grown to include dozens to hundreds of requests, the redundant header fields in these requests now measurably increase latency and unnecessarily consume bandwidth (see [PERF1] and [PERF2]).

SPDY [SPDY] initially addressed this redundancy by compressing header fields using the DEFLATE format [DEFLATE], which proved very effective at efficiently representing the redundant header fields. However, that approach exposed a security risk as demonstrated by the CRIME attack (see [CRIME]).

This document describes HPACK, a new compressor for header fields which eliminates redundant header fields, is not vulnerable to known security attacks, and which also has a bounded memory requirement for use in constrained environments.

## 2.1. Outline

The HTTP header field encoding defined in this document is based on a header table that maps name-value pairs to index values. The header table is incrementally updated during the HTTP/2 connection.

A set of header fields is treated as an unordered collection of name-value pairs. Names and values are considered to be opaque sequences of octets. The order of header fields is not guaranteed to be preserved after being compressed and decompressed.

As two consecutive sets of header fields often have header fields in common, each set is coded as a difference from the previous set. The goal is to only encode the changes (header fields present in one of the sets that are absent from the other) between the two sets of header fields.

A header field is represented either literally or as a reference to a name-value pair in the header table. A set of header fields is stored as a set of references to entries in the header table (possibly keeping only a subset of it, as some header fields may be missing a corresponding entry in the header table). Differences between consecutive sets of header fields are encoded as changes to the set of references.

The encoder is responsible for deciding which header fields to insert as new entries in the header table. The decoder executes the modifications to the header table and reference set prescribed by the encoder, reconstructing the set of header fields in the process. This enables decoders to remain simple and understand a wide variety of encoders.

Examples illustrating the use of these different mechanisms to represent header fields are available in Appendix D.

# 3. Header Field Encoding

## 3.1. Encoding Concepts

The encoding and decoding of header fields relies on some components and concepts:

Header Field: A name-value pair. Both the name and value are treated as opaque sequences of octets.

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

Static Table: The static table (see Appendix B) is a component used to associate static header fields to index values. This data is ordered, read-only, always accessible, and may be shared amongst all encoding contexts.

Reference Set: The reference set (see Section 3.1.3) is a component containing an unordered set of references to entries in the header table. This is used for the differential encoding of a new header set.

Header Set: A header set is an unordered group of header fields that are encoded jointly. A complete set of key-value pairs contained in a HTTP request or response is a header set.

Header Field Representation: A header field can be represented in encoded form either as a literal or as an index (see Section 3.1.4).

Header Block: The entire set of encoded header field representations which, when decoded, yield a complete header set.

Header Field Emission: When decoding a set of header field representations, some operations emit a header field (see Section 3.1.5). Emitted header fields are added to the current header set and cannot be removed.

#### 3.1.1. Encoding Context

The set of mutable structures used within an encoding context include a header table and a reference set. Everything else is either immutable or conceptual.

HTTP messages are exchanged between a client and a server in both directions. The encoding of header fields in each direction is independent from the other direction. There is a single encoding context for each direction used to encode all header fields sent in that direction.

#### 3.1.2. Header Table

A header table consists of a list of header fields maintained in first-in, first-out order. The first and newest entry in a header

table is always at index 1, and the oldest entry of a header table is at the index len(header table).

The header table is initially empty.

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

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 of the decoder, the header table size is strictly bounded (see Section 3.3.1).

The header table is updated during the processing of a set of header field representations (see Section 3.2.1).

### 3.1.3. Reference Set

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

The reference set is initially empty.

The reference set is updated during the processing of a set of header field representations (see Section 3.2.1).

The reference set enables differential encoding, whereby only differences between the previous header set and the current header set need to be encoded. The use of differential encoding is optional for any header set.

When an entry is evicted from the header table, if it was referenced from the reference set, its reference is removed from the reference set.

To limit the memory requirements on the decoder side for handling the reference set, only entries within the header table can be contained in the reference set. To still allow entries from the static table to take advantage of the differential encoding, when a header field is represented as a reference to an entry of the static table, this entry is inserted into the header table (see Section 3.2.1).

### 3.1.4. Header Field Representation

An encoded header field can be represented either as a literal or as an index.

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

Two different literal representations are provided:

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

* A literal representation that adds the header field as a new entry at the beginning of the header table (see Section 4.3.2).

Indexed Representation: The indexed representation defines a header field as a reference to an entry in either the header table or the static table (see Section 4.2).

```
        <----------  Index Address Space ---------->
        <-- Header  Table -->  <-- Static  Table -->
        +---+-----------+---+  +---+-----------+---+
        | 1 |    ...    | k |  |k+1|    ...    | n |
        +---+-----------+---+  +---+-----------+---+
        ^                   |
        |                   V
 Insertion Point       Drop Point
```

Index Address Space

Indices between 1 and len(header table), inclusive, refer to elements in the header table, with index 1 referring to the beginning of the table.

Indices between len(header table)+1 and len(header table)+len(static table), inclusive, refer to elements in the static table, where the index len(header table)+1 refers to the first entry in the static table.

Index 0 signals a modification of the encoding context: either the reference set is emptied, or the maximum size of the header table is updated (see Section 4.4).

Any other indices MUST be treated as erroneous, and the compression context considered corrupt and unusable.

### 3.1.5. Header Field Emission

The emission of a header field is the process of marking a header field as belonging to the current header set. Once a header has been emitted, it cannot be removed from the current header set.

On the decoding side, an emitted header field can be safely passed to the upper processing layer as part of the current header set. The decoder MAY pass the emitted header fields to the upper processing layer in any order.

By emitting header fields instead of emitting header sets, the decoder can be implemented in a streaming way, and as such has only to keep in memory the header table and the reference set. This bounds the amount of memory used by the decoder, even in presence of a very large set of header fields. The management of memory for handling very large sets of header fields can therefore be deferred to the upper processing layers.

## 3.2. Header Block Decoding

The processing of a header block to obtain a header set is defined in this section. To ensure that the decoding will successfully produce a header set, a decoder MUST obey the following rules.

### 3.2.1. Header Field Representation Processing

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

An _indexed representation_ with an index value of 0 entails one of the following actions, depending on what is encoded next:

- The reference set is emptied.

- The maximum size of the header table is updated.

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

- The entry is removed from the reference set.

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

- If referencing an element of the static table:

   * The header field corresponding to the referenced entry is emitted.

   * The referenced static entry is inserted at the beginning of the header table.

   * A reference to this new header table entry is added to the reference set (except if this new entry didn't fit in the header table).

- If referencing an element of the header table:

   * The header field corresponding to the referenced entry is emitted.

   * The referenced header table entry is added to the reference set.

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

- The header field is emitted.

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

- The header field is emitted.

- The header field is inserted at the beginning of the header table.

- A reference to the new entry is added to the reference set (except if this new entry didn't fit in the header table).

#### 3.2.2. Reference Set Emission

Once all the representations contained in a header block have been processed, the header fields referenced in the reference set which have not previously been emitted during this processing are emitted.

#### 3.2.3. Header Set Completion

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

### 3.3. Header Table Management

### 3.3.1. Maximum Table Size

To limit the memory requirements on the decoder side, the size of the header table is bounded. The size of the header table MUST stay lower than or equal to its maximum size.

By default, the maximum size of the header table is equal to the value of the HTTP/2 setting SETTINGS_HEADER_TABLE_SIZE defined by the decoder (see [HTTP2]). The encoder can change this maximum size (see Section 4.4), but it must stay lower than or equal to the value of SETTINGS_HEADER_TABLE_SIZE.

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

The size of an entry is the sum of its name's length in octets (as defined in Section 4.1.2), of its value's length in octets (Section 4.1.2) and of 32 octets.

The lengths are measured on the non-encoded entry name and entry value (for the case when a Huffman encoding is used to transmit string values).

The 32 octets 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 octets.

### 3.3.2. Entry Eviction When Header Table Size Changes

Whenever an entry is evicted from the header table, any reference to that entry contained by the reference set is removed.

Whenever the maximum size for the header table is made smaller, entries are evicted from the end of the header table until the size of the header table is less than or equal to the maximum size.

The eviction of an entry from the header table causes the index of the entries in the static table to be reduced by one.

### 3.3.3. Entry Eviction when Adding New Entries

Whenever a new entry is to be added to the table, any name referenced by the representation of this new entry is cached, and then entries are evicted from the end of the header table until the size of the header table is less than or equal to (maximum size - new entry size), or until the table is empty.

If the size of the new entry is less than or equal to the maximum size, that entry is added to the table. It is not an error to attempt to add an entry that is larger than the maximum size.

# 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 an octet.

An integer is represented in two parts: a prefix that fills the current octet and an optional list of octets that are used if the integer value does not fit within 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 octet. 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 [4] representation. N is always between 1 and 8 bits. An integer starting at an octet-boundary will have an 8-bit prefix.

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
```

For informational purpose, the algorithm to decode an integer I is as follows:

```
decode I from the next N bits
if I < 2^N - 1, return I
else
    M = 0
    repeat
        B = next octet
        I = I + (B & 127) * 2^M
```

```
        M = M + 7
    while B & 128 == 128
    return I
```

This integer representation allows for values of indefinite size. It is also possible for an encoder to send a large number of zero values, which can waste octets and could be used to overflow integer values. Excessively large integer encodings - in value or octet length - MUST be treated as a decoding error. Different limits can be set for each of the different uses of integers, based on implementation constraints.

#### 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&lt;128, encode(10), done
+---+---+---+---+---+---+---+---+
```

#### 4.1.1.3. Example 3: Encoding 42 starting at an octet-boundary

The value 42 is to be encoded starting at an octet-boundary. This implies that a 8-bit prefix is used.

o 42 is less than 255 (= 2^8 - 1) and is represented using the 8-bit prefix.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |   42 stored on 8 bits
+---+---+---+---+---+---+---+---+
```

### 4.1.2. String Literal Representation

Header field names and header field values are encoded as sequences of octets. A header field name or a header field value is encoded in three parts:

1. One bit, H, indicating whether or not the octets are Huffman encoded.

2. The number of octets required to hold the result of the next step, represented as an integer with a 7-bit prefix (see Section 4.1.1), immediately following the first bit.

3. The encoded data of the string:

* If H is '1', then the encoded string data is the bitwise concatenation of the canonical [CANON] Huffman code [HUFF] corresponding to each octet of the data, followed by between 0-7 bits of padding.

* If H is '0', then the encoded string is the octets of the field value without modification.

Padding is necessary when doing Huffman encoding to ensure that the remaining bits between the actual end of the data and the next octet boundary are not misinterpreted as part of the input data.

When padding for Huffman encoding, the bits from the EOS (end-of-string) entry in the Huffman table are used, starting with the MSB (most significant bit). This entry is guaranteed to be at least 8 bits long.

String literals which use Huffman encoding are encoded with the Huffman Codes Appendix C (see examples in Request Examples with Huffman Appendix D.3 and in Response Examples with Huffman Appendix D.5).

The EOS symbol is represented with value 256, and is used solely to signal the end of the Huffman-encoded key data or the end of the Huffman-encoded value data. Given that only between 0-7 bits of the EOS symbol is included in any Huffman-encoded string, and given that the EOS symbol is at least 8 bits long, it is expected that it should never be successfully decoded.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1 |  Value Length Prefix (7)  |
+---+---+---+---+---+---+---+---+
|   Value Length (0-N octets)   |
+---+---+---+---+---+---+---+---+
...
+---+---+---+---+---+---+---+---+
| Huffman Encoded Data  |Padding|
+---+---+---+---+---+---+---+---+
```

String Literal with Huffman Encoding

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 |  Value Length Prefix (7)  |
+---+---+---+---+---+---+---+---+
|   Value Length (0-N octets)   |
+---+---+---+---+---+---+---+---+
```

```
...
+---+---+---+---+---+---+---+---+
|  Field Bytes without Encoding |
+---+---+---+---+---+---+---+---+
```

String Literal without Huffman Encoding

### 4.2. Indexed Header Field Representation

An indexed header field representation either identifies an entry in the header table or static table. The processing of an indexed header field representation is described in Section 3.2.1.

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

Indexed Header Field

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.

The index value of 0 is reserved for signalling changes in the encoding context (see Section 4.4).

### 4.3. Literal Header Field Representation

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

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

#### 4.3.1. Literal Header Field without Indexing

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

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

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

        Literal Header Field without Indexing - Indexed Name

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

          Literal Header Field without Indexing - New Name
```

This representation starts with the '01' 2-bit pattern.

If the header field name matches the header field name of a (name, value) pair stored in the Header Table or Static Table, the header field name can be represented using the index of that entry. In this case, the index of the entry, index (which is strictly greater than 0), is represented as an integer with a 6-bit prefix (see Section 4.1.1).

Otherwise, the header field name is represented as a literal. The value 0 is represented on 6 bits followed by the header field name (see Section 4.1.2).

The header field name representation is followed by the header field value represented as a literal string as described in Section 4.1.2.

### 4.3.2. Literal Header Field with Incremental Indexing

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

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

```
       Literal Header Field with Incremental Indexing - Indexed Name

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

         Literal Header Field with Incremental Indexing - New Name
```

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

If the header field name matches the header field name of a (name, value) pair stored in the Header Table or Static Table, the header field name can be represented using the index of that entry. In this case, the index of the entry, index (which is strictly greater than 0), is represented as an integer with a 6-bit prefix (see Section 4.1.1).

Otherwise, the header field name is represented as a literal. The value 0 is represented on 6 bits followed by the header field name (see Section 4.1.2).

The header field name representation is followed by the header field value represented as a literal string as described in Section 4.1.2.

### 4.4. Encoding Context Update

An indexed value of 0 is reserved for signalling changes in the encoding context. The type of the change is encoded on the following octet(s). Any change in the encoding context is applied immediately.

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

                         Reference Set Emptying
```

An octet with its high bit set to '1' signals that the reference set is emptied. The remaining bits are set to '0'.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 |   New maximum size (7+)   |
+---+---------------------------+
```

Maximum Header Table Size Change

An octet with its high bit set to '0' signals the new maximum size of the header table. This new maximum size MUST be lower than or equal to the value of the setting SETTINGS_HEADER_TABLE_SIZE (see [HTTP2]).

The new maximum size is encoded as an integer with a 7-bit prefix.

Change in the maximum size of the header table can trigger entry evictions (see Section 3.3.2).

## 5. 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 plain text 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.

A decoder needs to ensure that larger values or encodings of integers do not permit exploitation. Decoders MUST limit the size of integers, both in value and encoded length, that it accepts (see Section 4.1.1).

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 header fields 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 field 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.

A particular care should be used for the maximum size of the header table. While an endpoint can fully control the maximum size of its header table for the decoding size, by using SETTINGS_HEADER_TABLE_SIZE, the maximum size of the encoding size is controlled by the remote peer. The endpoint should check the SETTINGS_HEADER_TABLE_SIZE defined by the remote peer, and decrease the maximum size for the encoding size if needed.

## 6. Acknowledgements

This document includes substantial editorial contributions from the following individuals: Mike Bishop, Jeff Pinner, Julian Reschke, Martin Thomson.

## 7. References

### 7.1. Normative References

[HTTP-p1] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", draft-ietf-httpbis-p1-messaging-26 (work in progress), February 2014.

[HTTP2] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-10 (work in progress), February 2014.

### 7.2. Informative References

[CANON] Schwartz, E. and B. Kallick, "Generating a canonical prefix encoding", Communications of the ACM Volume 7 Issue 3, pp. 166-169, March 1964, <http://dl.acm.org/citation.cfm?id=363991>.

[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>.

[DEFLATE] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[HUFF] Huffman, D., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the Institute of Radio Engineers Volume 40, Number 9, pp. 1098-1101, September 1952, <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4051119>.

[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", draft-mbelshe-httpbis-spdy-00 (work in progress), February 2012.

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

## A.1. Since draft-ietf-httpbis-header-compression-05

- Regenerated examples.
- Only one Huffman table for requests and responses.
- Added maximum size for header table, independent of SETTINGS_HEADER_TABLE_SIZE.
- Added pseudo-code for integer decoding.
- Improved examples (removing unnecessary removals).

## A.2. Since draft-ietf-httpbis-header-compression-04

- Updated examples: take into account changes in the spec, and show more features.
- Use 'octet' everywhere instead of having both 'byte' and 'octet'.
- Added reference set emptying.
- Editorial changes and clarifications.
- Added "host" header to the static table.
- Ordering for list of values (either NULL- or comma-separated).

### A.3. Since draft-ietf-httpbis-header-compression-03

- A large number of editorial changes; changed the description of evicting/adding new entries.

- Removed substitution indexing

- Changed 'initial headers' to 'static headers', as per issue #258

- Merged 'request' and 'response' static headers, as per issue #259

- Changed text to indicate that new headers are added at index 0 and expire from the largest index, as per issue #233

### A.4. Since draft-ietf-httpbis-header-compression-02

- Corrected error in integer encoding pseudocode.

### A.5. Since draft-ietf-httpbis-header-compression-01

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

- 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.

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

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

- 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.6. Since draft-ietf-httpbis-header-compression-00

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 octet

Added descriptions of opcodes

Removed security claims from introduction.

## Appendix B. Static Table

The static table consists of an unchangeable ordered list of (name, value) pairs. The first entry in the table is always represented by the index len(header table)+1, and the last entry in the table is represented by the index len(header table)+len(static table).

[[The ordering of these tables is currently arbitrary. The tables in this section should be updated and ordered such that the table entries with the smallest indices are those which, based on a statistical analysis of the frequency of use weighted by size, achieve the largest decrease in octets transmitted subject to HTTP 2 header field rules (like removal of some header fields). This set of header fields is currently very likely incomplete, and should be made complete. ]]

The following table lists the pre-defined header fields that make-up the static header table.

| Index | Header Name | Header Value |
|---|---|---|
| 1 | :authority | |
| 2 | :method | GET |
| 3 | :method | POST |
| 4 | :path | / |
| 5 | :path | /index.html |
| 6 | :scheme | http |
| 7 | :scheme | https |
| 8 | :status | 200 |
| 9 | :status | 500 |
| 10 | :status | 404 |
| 11 | :status | 403 |
| 12 | :status | 400 |

| | | | |
|---|---|---|---|
| 13 | :status | 401 | |
| 14 | accept-charset | | |
| 15 | accept-encoding | | |
| 16 | accept-language | | |
| 17 | accept-ranges | | |
| 18 | accept | | |
| 19 | access-control-allow-origin | | |
| 20 | age | | |
| 21 | allow | | |
| 22 | authorization | | |
| 23 | cache-control | | |
| 24 | content-disposition | | |
| 25 | content-encoding | | |
| 26 | content-language | | |
| 27 | content-length | | |
| 28 | content-location | | |
| 29 | content-range | | |
| 30 | content-type | | |
| 31 | cookie | | |
| 32 | date | | |
| 33 | etag | | |
| 34 | expect | | |
| 35 | expires | | |
| 36 | from | | |
| 37 | host | | |
| 38 | if-match | | |
| 39 | if-modified-since | | |
| 40 | if-none-match | | |
| 41 | if-range | | |
| 42 | if-unmodified-since | | |
| 43 | last-modified | | |
| 44 | link | | |
| 45 | location | | |
| 46 | max-forwards | | |
| 47 | proxy-authenticate | | |
| 48 | proxy-authorization | | |
| 49 | range | | |
| 50 | referer | | |
| 51 | refresh | | |
| 52 | retry-after | | |
| 53 | server | | |
| 54 | set-cookie | | |
| 55 | strict-transport-security | | |
| 56 | transfer-encoding | | |
| 57 | user-agent | | |
| 58 | vary | | |
| 59 | via | | |
| 60 | www-authenticate | | |

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

Table 1: Static Table Entries

The table give the index of each entry in the static table. The full index of each entry, to be used for encoding a reference to this entry, is computed by adding the number of entries in the header table to this index.

## Appendix C. Huffman Codes

The following Huffman codes are used when encoding string literals.

[[This table will be regenerated. ]]

| sym | aligned to MSB as bits | len in bits | aligned to LSB as hex | len in bits |
|---|---|---|---|---|
| ( 0) | \|11111111\|11111111\|11110111\|010 | [27] | 7ffffba | [27] |
| ( 1) | \|11111111\|11111111\|11110111\|011 | [27] | 7ffffbb | [27] |
| ( 2) | \|11111111\|11111111\|11110111\|100 | [27] | 7ffffbc | [27] |
| ( 3) | \|11111111\|11111111\|11110111\|101 | [27] | 7ffffbd | [27] |
| ( 4) | \|11111111\|11111111\|11110111\|110 | [27] | 7ffffbe | [27] |
| ( 5) | \|11111111\|11111111\|11110111\|111 | [27] | 7ffffbf | [27] |
| ( 6) | \|11111111\|11111111\|11111000\|000 | [27] | 7ffffc0 | [27] |
| ( 7) | \|11111111\|11111111\|11111000\|001 | [27] | 7ffffc1 | [27] |
| ( 8) | \|11111111\|11111111\|11111000\|010 | [27] | 7ffffc2 | [27] |
| ( 9) | \|11111111\|11111111\|11111000\|011 | [27] | 7ffffc3 | [27] |
| ( 10) | \|11111111\|11111111\|11111000\|100 | [27] | 7ffffc4 | [27] |
| ( 11) | \|11111111\|11111111\|11111000\|101 | [27] | 7ffffc5 | [27] |
| ( 12) | \|11111111\|11111111\|11111000\|110 | [27] | 7ffffc6 | [27] |
| ( 13) | \|11111111\|11111111\|11111000\|111 | [27] | 7ffffc7 | [27] |
| ( 14) | \|11111111\|11111111\|11111001\|000 | [27] | 7ffffc8 | [27] |
| ( 15) | \|11111111\|11111111\|11111001\|001 | [27] | 7ffffc9 | [27] |
| ( 16) | \|11111111\|11111111\|11111001\|010 | [27] | 7ffffca | [27] |
| ( 17) | \|11111111\|11111111\|11111001\|011 | [27] | 7ffffcb | [27] |
| ( 18) | \|11111111\|11111111\|11111001\|100 | [27] | 7ffffcc | [27] |
| ( 19) | \|11111111\|11111111\|11111001\|101 | [27] | 7ffffcd | [27] |
| ( 20) | \|11111111\|11111111\|11111001\|110 | [27] | 7ffffce | [27] |
| ( 21) | \|11111111\|11111111\|11111001\|111 | [27] | 7ffffcf | [27] |
| ( 22) | \|11111111\|11111111\|11111010\|000 | [27] | 7ffffd0 | [27] |
| ( 23) | \|11111111\|11111111\|11111010\|001 | [27] | 7ffffd1 | [27] |
| ( 24) | \|11111111\|11111111\|11111010\|010 | [27] | 7ffffd2 | [27] |
| ( 25) | \|11111111\|11111111\|11111010\|011 | [27] | 7ffffd3 | [27] |
| ( 26) | \|11111111\|11111111\|11111010\|100 | [27] | 7ffffd4 | [27] |
| ( 27) | \|11111111\|11111111\|11111010\|101 | [27] | 7ffffd5 | [27] |
| ( 28) | \|11111111\|11111111\|11111010\|110 | [27] | 7ffffd6 | [27] |

```
       ( 29) |11111111|11111111|11111010|111 [27]      7ffffd7 [27]
       ( 30) |11111111|11111111|11111011|000 [27]      7ffffd8 [27]
       ( 31) |11111111|11111111|11111011|001 [27]      7ffffd9 [27]
   ' ' ( 32) |11101000| [8]                                 e8 [8]
   '!' ( 33) |11111111|1100 [12]                           ffc [12]
   '"' ( 34) |11111111|111010 [14]                        3ffa [14]
   '#' ( 35) |11111111|1111100 [15]                       7ffc [15]
   '$' ( 36) |11111111|1111101 [15]                       7ffd [15]
   '%' ( 37) |100100 [6]                                    24 [6]
   '&' ( 38) |1101110 [7]                                   6e [7]
   ''' ( 39) |11111111|1111110 [15]                       7ffe [15]
   '(' ( 40) |11111111|010 [11]                            7fa [11]
   ')' ( 41) |11111111|011 [11]                            7fb [11]
   '*' ( 42) |11111110|10 [10]                             3fa [10]
   '+' ( 43) |11111111|100 [11]                            7fc [11]
   ',' ( 44) |11101001| [8]                                 e9 [8]
   '-' ( 45) |100101 [6]                                    25 [6]
   '.' ( 46) |00100 [5]                                      4 [5]
   '/' ( 47) |0000 [4]                                       0 [4]
   '0' ( 48) |00101 [5]                                      5 [5]
   '1' ( 49) |00110 [5]                                      6 [5]
   '2' ( 50) |00111 [5]                                      7 [5]
   '3' ( 51) |100110 [6]                                    26 [6]
   '4' ( 52) |100111 [6]                                    27 [6]
   '5' ( 53) |101000 [6]                                    28 [6]
   '6' ( 54) |101001 [6]                                    29 [6]
   '7' ( 55) |101010 [6]                                    2a [6]
   '8' ( 56) |101011 [6]                                    2b [6]
   '9' ( 57) |101100 [6]                                    2c [6]
   ':' ( 58) |11110110|0 [9]                               1ec [9]
   ';' ( 59) |11101010| [8]                                 ea [8]
   '<' ( 60) |11111111|11111111|10 [18]                  3fffe [18]
   '=' ( 61) |101101 [6]                                    2d [6]
   '>' ( 62) |11111111|11111110|0 [17]                   1fffc [17]
   '?' ( 63) |11110110|1 [9]                               1ed [9]
   '@' ( 64) |11111111|111011 [14]                        3ffb [14]
   'A' ( 65) |1101111 [7]                                   6f [7]
   'B' ( 66) |11101011| [8]                                 eb [8]
   'C' ( 67) |11101100| [8]                                 ec [8]
   'D' ( 68) |11101101| [8]                                 ed [8]
   'E' ( 69) |11101110| [8]                                 ee [8]
   'F' ( 70) |1110000 [7]                                   70 [7]
   'G' ( 71) |11110111|0 [9]                               1ee [9]
   'H' ( 72) |11110111|1 [9]                               1ef [9]
   'I' ( 73) |11111000|0 [9]                               1f0 [9]
   'J' ( 74) |11111000|1 [9]                               1f1 [9]
   'K' ( 75) |11111110|11 [10]                             3fb [10]
   'L' ( 76) |11111001|0 [9]                               1f2 [9]
```

```
    'M' ( 77) |11101111| [8]                                 ef [8]
    'N' ( 78) |11111001|1 [9]                               1f3 [9]
    'O' ( 79) |11111010|0 [9]                               1f4 [9]
    'P' ( 80) |11111010|1 [9]                               1f5 [9]
    'Q' ( 81) |11111011|0 [9]                               1f6 [9]
    'R' ( 82) |11111011|1 [9]                               1f7 [9]
    'S' ( 83) |11110000| [8]                                 f0 [8]
    'T' ( 84) |11110001| [8]                                 f1 [8]
    'U' ( 85) |11111100|0 [9]                               1f8 [9]
    'V' ( 86) |11111100|1 [9]                               1f9 [9]
    'W' ( 87) |11111101|0 [9]                               1fa [9]
    'X' ( 88) |11111101|1 [9]                               1fb [9]
    'Y' ( 89) |11111110|0 [9]                               1fc [9]
    'Z' ( 90) |11111111|00 [10]                            3fc [10]
    '[' ( 91) |11111111|111100 [14]                       3ffc [14]
    '\' ( 92) |11111111|11111111|11111011|010 [27]     7ffffda [27]
    ']' ( 93) |11111111|11100 [13]                        1ffc [13]
    '^' ( 94) |11111111|111101 [14]                       3ffd [14]
    '_' ( 95) |101110 [6]                                    2e [6]
    '`' ( 96) |11111111|11111111|110 [19]                7fffe [19]
    'a' ( 97) |01000 [5]                                      8 [5]
    'b' ( 98) |101111 [6]                                    2f [6]
    'c' ( 99) |01001 [5]                                      9 [5]
    'd' (100) |110000 [6]                                    30 [6]
    'e' (101) |0001 [4]                                       1 [4]
    'f' (102) |110001 [6]                                    31 [6]
    'g' (103) |110010 [6]                                    32 [6]
    'h' (104) |110011 [6]                                    33 [6]
    'i' (105) |01010 [5]                                      a [5]
    'j' (106) |1110001 [7]                                   71 [7]
    'k' (107) |1110010 [7]                                   72 [7]
    'l' (108) |01011 [5]                                      b [5]
    'm' (109) |110100 [6]                                    34 [6]
    'n' (110) |01100 [5]                                      c [5]
    'o' (111) |01101 [5]                                      d [5]
    'p' (112) |01110 [5]                                      e [5]
    'q' (113) |11110010| [8]                                 f2 [8]
    'r' (114) |01111 [5]                                      f [5]
    's' (115) |10000 [5]                                     10 [5]
    't' (116) |10001 [5]                                     11 [5]
    'u' (117) |110101 [6]                                    35 [6]
    'v' (118) |1110011 [7]                                   73 [7]
    'w' (119) |110110 [6]                                    36 [6]
    'x' (120) |11110011| [8]                                 f3 [8]
    'y' (121) |11110100| [8]                                 f4 [8]
    'z' (122) |11110101| [8]                                 f5 [8]
    '{' (123) |11111111|11111110|1 [17]                  1fffd [17]
    '|' (124) |11111111|101 [11]                           7fd [11]
```

```
    '}' (125)  |11111111|11111111|0                         1fffe  [17]
    '~' (126)  |11111111|1101                                 ffd  [12]
        (127)  |11111111|11111111|11111011|011            7ffffdb  [27]
        (128)  |11111111|11111111|11111011|100            7ffffdc  [27]
        (129)  |11111111|11111111|11111011|101            7ffffdd  [27]
        (130)  |11111111|11111111|11111011|110            7ffffde  [27]
        (131)  |11111111|11111111|11111011|111            7ffffdf  [27]
        (132)  |11111111|11111111|11111100|000            7ffffe0  [27]
        (133)  |11111111|11111111|11111100|001            7ffffe1  [27]
        (134)  |11111111|11111111|11111100|010            7ffffe2  [27]
        (135)  |11111111|11111111|11111100|011            7ffffe3  [27]
        (136)  |11111111|11111111|11111100|100            7ffffe4  [27]
        (137)  |11111111|11111111|11111100|101            7ffffe5  [27]
        (138)  |11111111|11111111|11111100|110            7ffffe6  [27]
        (139)  |11111111|11111111|11111100|111            7ffffe7  [27]
        (140)  |11111111|11111111|11111101|000            7ffffe8  [27]
        (141)  |11111111|11111111|11111101|001            7ffffe9  [27]
        (142)  |11111111|11111111|11111101|010            7ffffea  [27]
        (143)  |11111111|11111111|11111101|011            7ffffeb  [27]
        (144)  |11111111|11111111|11111101|100            7ffffec  [27]
        (145)  |11111111|11111111|11111101|101            7ffffed  [27]
        (146)  |11111111|11111111|11111101|110            7ffffee  [27]
        (147)  |11111111|11111111|11111101|111            7ffffef  [27]
        (148)  |11111111|11111111|11111110|000            7fffff0  [27]
        (149)  |11111111|11111111|11111110|001            7fffff1  [27]
        (150)  |11111111|11111111|11111110|010            7fffff2  [27]
        (151)  |11111111|11111111|11111110|011            7fffff3  [27]
        (152)  |11111111|11111111|11111110|100            7fffff4  [27]
        (153)  |11111111|11111111|11111110|101            7fffff5  [27]
        (154)  |11111111|11111111|11111110|110            7fffff6  [27]
        (155)  |11111111|11111111|11111110|111            7fffff7  [27]
        (156)  |11111111|11111111|11111111|000            7fffff8  [27]
        (157)  |11111111|11111111|11111111|001            7fffff9  [27]
        (158)  |11111111|11111111|11111111|010            7fffffa  [27]
        (159)  |11111111|11111111|11111111|011            7fffffb  [27]
        (160)  |11111111|11111111|11111111|100            7fffffc  [27]
        (161)  |11111111|11111111|11111111|101            7fffffd  [27]
        (162)  |11111111|11111111|11111111|110            7fffffe  [27]
        (163)  |11111111|11111111|11111111|111            7ffffff  [27]
        (164)  |11111111|11111111|11100000|00             3ffff80  [26]
        (165)  |11111111|11111111|11100000|01             3ffff81  [26]
        (166)  |11111111|11111111|11100000|10             3ffff82  [26]
        (167)  |11111111|11111111|11100000|11             3ffff83  [26]
        (168)  |11111111|11111111|11100001|00             3ffff84  [26]
        (169)  |11111111|11111111|11100001|01             3ffff85  [26]
        (170)  |11111111|11111111|11100001|10             3ffff86  [26]
        (171)  |11111111|11111111|11100001|11             3ffff87  [26]
        (172)  |11111111|11111111|11100010|00             3ffff88  [26]
```

```
    (173) |11111111|11111111|11100010|01 [26]      3ffff89 [26]
    (174) |11111111|11111111|11100010|10 [26]      3ffff8a [26]
    (175) |11111111|11111111|11100010|11 [26]      3ffff8b [26]
    (176) |11111111|11111111|11100011|00 [26]      3ffff8c [26]
    (177) |11111111|11111111|11100011|01 [26]      3ffff8d [26]
    (178) |11111111|11111111|11100011|10 [26]      3ffff8e [26]
    (179) |11111111|11111111|11100011|11 [26]      3ffff8f [26]
    (180) |11111111|11111111|11100100|00 [26]      3ffff90 [26]
    (181) |11111111|11111111|11100100|01 [26]      3ffff91 [26]
    (182) |11111111|11111111|11100100|10 [26]      3ffff92 [26]
    (183) |11111111|11111111|11100100|11 [26]      3ffff93 [26]
    (184) |11111111|11111111|11100101|00 [26]      3ffff94 [26]
    (185) |11111111|11111111|11100101|01 [26]      3ffff95 [26]
    (186) |11111111|11111111|11100101|10 [26]      3ffff96 [26]
    (187) |11111111|11111111|11100101|11 [26]      3ffff97 [26]
    (188) |11111111|11111111|11100110|00 [26]      3ffff98 [26]
    (189) |11111111|11111111|11100110|01 [26]      3ffff99 [26]
    (190) |11111111|11111111|11100110|10 [26]      3ffff9a [26]
    (191) |11111111|11111111|11100110|11 [26]      3ffff9b [26]
    (192) |11111111|11111111|11100111|00 [26]      3ffff9c [26]
    (193) |11111111|11111111|11100111|01 [26]      3ffff9d [26]
    (194) |11111111|11111111|11100111|10 [26]      3ffff9e [26]
    (195) |11111111|11111111|11100111|11 [26]      3ffff9f [26]
    (196) |11111111|11111111|11101000|00 [26]      3ffffa0 [26]
    (197) |11111111|11111111|11101000|01 [26]      3ffffa1 [26]
    (198) |11111111|11111111|11101000|10 [26]      3ffffa2 [26]
    (199) |11111111|11111111|11101000|11 [26]      3ffffa3 [26]
    (200) |11111111|11111111|11101001|00 [26]      3ffffa4 [26]
    (201) |11111111|11111111|11101001|01 [26]      3ffffa5 [26]
    (202) |11111111|11111111|11101001|10 [26]      3ffffa6 [26]
    (203) |11111111|11111111|11101001|11 [26]      3ffffa7 [26]
    (204) |11111111|11111111|11101010|00 [26]      3ffffa8 [26]
    (205) |11111111|11111111|11101010|01 [26]      3ffffa9 [26]
    (206) |11111111|11111111|11101010|10 [26]      3ffffaa [26]
    (207) |11111111|11111111|11101010|11 [26]      3ffffab [26]
    (208) |11111111|11111111|11101011|00 [26]      3ffffac [26]
    (209) |11111111|11111111|11101011|01 [26]      3ffffad [26]
    (210) |11111111|11111111|11101011|10 [26]      3ffffae [26]
    (211) |11111111|11111111|11101011|11 [26]      3ffffaf [26]
    (212) |11111111|11111111|11101100|00 [26]      3ffffb0 [26]
    (213) |11111111|11111111|11101100|01 [26]      3ffffb1 [26]
    (214) |11111111|11111111|11101100|10 [26]      3ffffb2 [26]
    (215) |11111111|11111111|11101100|11 [26]      3ffffb3 [26]
    (216) |11111111|11111111|11101101|00 [26]      3ffffb4 [26]
    (217) |11111111|11111111|11101101|01 [26]      3ffffb5 [26]
    (218) |11111111|11111111|11101101|10 [26]      3ffffb6 [26]
    (219) |11111111|11111111|11101101|11 [26]      3ffffb7 [26]
    (220) |11111111|11111111|11101110|00 [26]      3ffffb8 [26]
```

```
     (221) |11111111|11111111|11101110|01 [26]       3ffffb9 [26]
     (222) |11111111|11111111|11101110|10 [26]       3ffffba [26]
     (223) |11111111|11111111|11101110|11 [26]       3ffffbb [26]
     (224) |11111111|11111111|11101111|00 [26]       3ffffbc [26]
     (225) |11111111|11111111|11101111|01 [26]       3ffffbd [26]
     (226) |11111111|11111111|11101111|10 [26]       3ffffbe [26]
     (227) |11111111|11111111|11101111|11 [26]       3ffffbf [26]
     (228) |11111111|11111111|11110000|00 [26]       3ffffc0 [26]
     (229) |11111111|11111111|11110000|01 [26]       3ffffc1 [26]
     (230) |11111111|11111111|11110000|10 [26]       3ffffc2 [26]
     (231) |11111111|11111111|11110000|11 [26]       3ffffc3 [26]
     (232) |11111111|11111111|11110001|00 [26]       3ffffc4 [26]
     (233) |11111111|11111111|11110001|01 [26]       3ffffc5 [26]
     (234) |11111111|11111111|11110001|10 [26]       3ffffc6 [26]
     (235) |11111111|11111111|11110001|11 [26]       3ffffc7 [26]
     (236) |11111111|11111111|11110010|00 [26]       3ffffc8 [26]
     (237) |11111111|11111111|11110010|01 [26]       3ffffc9 [26]
     (238) |11111111|11111111|11110010|10 [26]       3ffffca [26]
     (239) |11111111|11111111|11110010|11 [26]       3ffffcb [26]
     (240) |11111111|11111111|11110011|00 [26]       3ffffcc [26]
     (241) |11111111|11111111|11110011|01 [26]       3ffffcd [26]
     (242) |11111111|11111111|11110011|10 [26]       3ffffce [26]
     (243) |11111111|11111111|11110011|11 [26]       3ffffcf [26]
     (244) |11111111|11111111|11110100|00 [26]       3ffffd0 [26]
     (245) |11111111|11111111|11110100|01 [26]       3ffffd1 [26]
     (246) |11111111|11111111|11110100|10 [26]       3ffffd2 [26]
     (247) |11111111|11111111|11110100|11 [26]       3ffffd3 [26]
     (248) |11111111|11111111|11110101|00 [26]       3ffffd4 [26]
     (249) |11111111|11111111|11110101|01 [26]       3ffffd5 [26]
     (250) |11111111|11111111|11110101|10 [26]       3ffffd6 [26]
     (251) |11111111|11111111|11110101|11 [26]       3ffffd7 [26]
     (252) |11111111|11111111|11110110|00 [26]       3ffffd8 [26]
     (253) |11111111|11111111|11110110|01 [26]       3ffffd9 [26]
     (254) |11111111|11111111|11110110|10 [26]       3ffffda [26]
     (255) |11111111|11111111|11110110|11 [26]       3ffffdb [26]
 EOS (256) |11111111|11111111|11110111|00 [26]       3ffffdc [26]
```

## Appendix D. Examples

A number of examples are worked through here, for both requests and responses, and with and without Huffman coding.

## D.1. Header Field Representation Examples

This section show several independent representation examples.

### D.1.1. Literal Header Field with Indexing

The header field representation uses a literal name and a literal value.

Header set to encode:

```
custom-key: custom-header
```

Reference set: empty.

Hex dump of encoded data:

```
000a 6375 7374 6f6d 2d6b 6579 0d63 7573 | ..custom-key.cus
746f 6d2d 6865 6164 6572                | tom-header
```

Decoding process:

```
00                                      | == Literal indexed ==
0a                                      |   Literal name (len = 10)
6375 7374 6f6d 2d6b 6579                | custom-key
0d                                      |   Literal value (len = 13)
6375 7374 6f6d 2d68 6561 6465 72        | custom-header
                                        | -> custom-key: custom-head\
                                        |   er
```

Header Table (after decoding):

```
[  1] (s =  55) custom-key: custom-header
      Table size:  55
```

Decoded header set:

```
custom-key: custom-header
```

### D.1.2. Literal Header Field without Indexing

The header field representation uses an indexed name and a literal value.

Header set to encode:

```
:path: /sample/path
```

Reference set: empty.

Hex dump of encoded data:

```
440c 2f73 616d 706c 652f 7061 7468      | D./sample/path
```

Decoding process:

```
44                                      | == Literal not indexed ==
                                        |   Indexed name (idx = 4)
                                        |     :path
0c                                      |   Literal value (len = 12)
2f73 616d 706c 652f 7061 7468           | /sample/path
                                        | -> :path: /sample/path
```

Header table (after decoding): empty.

Decoded header set:

```
:path: /sample/path
```

### D.1.3. Indexed Header Field

The header field representation uses an indexed header field, from the static table. Upon using it, the static table entry is copied into the header table.

Header set to encode:

```
:method: GET
```

Reference set: empty.

Hex dump of encoded data:

```
82                                      | .
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
```

Header Table (after decoding):

```
[  1] (s =  42) :method: GET
      Table size:  42
```

Decoded header set:

```
:method: GET
```

### D.1.4. Indexed Header Field from Static Table

The header field representation uses an indexed header field, from the static table. In this example, the SETTINGS_HEADER_TABLE_SIZE is set to 0, therefore, the entry is not copied into the header table.

Header set to encode:

```
:method: GET
```

Reference set: empty.

Hex dump of encoded data:

```
82                                      | .
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
```

Header table (after decoding): empty.

Decoded header set:

```
:method: GET
```

## D.2. Request Examples without Huffman

This section shows several consecutive header sets, corresponding to HTTP requests, on the same connection.

### D.2.1. First request

Header set to encode:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
8287 8604 0f77 7777 2e65 7861 6d70 6c65 | .....www.example
2e63 6f6d                               | .com
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
87                                      | == Indexed - Add ==
                                        |   idx = 7
                                        | -> :scheme: http
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :path: /
04                                      | == Literal indexed ==
                                        |   Indexed name (idx = 4)
                                        |     :authority
0f                                      |   Literal value (len = 15)
7777 772e 6578 616d 706c 652e 636f 6d   | www.example.com
                                        | -> :authority: www.example\
                                        |   .com
```

```
Header Table (after decoding):

[  1] (s =  57) :authority: www.example.com
[  2] (s =  38) :path: /
[  3] (s =  43) :scheme: http
[  4] (s =  42) :method: GET
      Table size: 180


Decoded header set:

:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

### D.2.2. Second request

This request takes advantage of the differential encoding of header sets.

```
Header set to encode:

:method: GET
:scheme: http
:path: /
:authority: www.example.com
cache-control: no-cache


Reference set:

[  1] :authority: www.example.com
[  2] :path: /
[  3] :scheme: http
[  4] :method: GET


Hex dump of encoded data:

1b08 6e6f 2d63 6163 6865                | ..no-cache


Decoding process:
```

```
1b                                      | == Literal indexed ==
                                        |   Indexed name (idx = 27)
                                        |     cache-control
08                                      |   Literal value (len = 8)
6e6f 2d63 6163 6865                     | no-cache
                                        | -> cache-control: no-cache
```

Header Table (after decoding):

```
[  1] (s =  53) cache-control: no-cache
[  2] (s =  57) :authority: www.example.com
[  3] (s =  38) :path: /
[  4] (s =  43) :scheme: http
[  5] (s =  42) :method: GET
      Table size: 233
```

Decoded header set:

```
cache-control: no-cache
:authority: www.example.com
:path: /
:scheme: http
:method: GET
```

### D.2.3. Third request

This request has not enough headers in common with the previous request to take advantage of the differential encoding. Therefore, the reference set is emptied before encoding the header fields.

Header set to encode:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

Reference set:

```
[  1] cache-control: no-cache
[  2] :authority: www.example.com
[  3] :path: /
[  4] :scheme: http
[  5] :method: GET
```

Hex dump of encoded data:

```
8080 858c 8b84 000a 6375 7374 6f6d 2d6b | ........custom-k
6579 0c63 7573 746f 6d2d 7661 6c75 65   | ey.custom-value
```

Decoding process:

```
80 80                                   | == Empty reference set ==
                                        |   idx = 0
                                        |   flag = 1
85                                      | == Indexed - Add ==
                                        |   idx = 5
                                        | -> :method: GET
8c                                      | == Indexed - Add ==
                                        |   idx = 12
                                        | -> :scheme: https
8b                                      | == Indexed - Add ==
                                        |   idx = 11
                                        | -> :path: /index.html
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :authority: www.example\
                                        |   .com
00                                      | == Literal indexed ==
0a                                      |   Literal name (len = 10)
6375 7374 6f6d 2d6b 6579                | custom-key
0c                                      |   Literal value (len = 12)
6375 7374 6f6d 2d76 616c 7565           | custom-value
                                        | -> custom-key: custom-valu\
                                        |   e
```

Header Table (after decoding):

```
[  1] (s =  54) custom-key: custom-value
[  2] (s =  48) :path: /index.html
[  3] (s =  44) :scheme: https
[  4] (s =  53) cache-control: no-cache
[  5] (s =  57) :authority: www.example.com
[  6] (s =  38) :path: /
[  7] (s =  43) :scheme: http
[  8] (s =  42) :method: GET
      Table size: 379
```

Decoded header set:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

## D.3. Request Examples with Huffman

This section shows the same examples as the previous section, but using Huffman encoding for the literal values.

### D.3.1. First request

Header set to encode:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
8287 8604 8bdb 6d88 3e68 d1cb 1225 ba7f | ......m..h...%..
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
87                                      | == Indexed - Add ==
                                        |   idx = 7
                                        | -> :scheme: http
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :path: /
04                                      | == Literal indexed ==
                                        |   Indexed name (idx = 4)
                                        |     :authority
8b                                      |   Literal value (len = 15)
                                        |     Huffman encoded:
db6d 883e 68d1 cb12 25ba 7f             | .m..h...%..
                                        |     Decoded:
                                        | www.example.com
                                        | -> :authority: www.example\
                                        |   .com
```

Header Table (after decoding):

```
[  1] (s =  57) :authority: www.example.com
[  2] (s =  38) :path: /
[  3] (s =  43) :scheme: http
[  4] (s =  42) :method: GET
      Table size: 180
```

Decoded header set:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

### D.3.2. Second request

This request takes advantage of the differential encoding of header sets.

Header set to encode:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
cache-control: no-cache
```

Reference set:

```
[  1] :authority: www.example.com
[  2] :path: /
[  3] :scheme: http
[  4] :method: GET
```

Hex dump of encoded data:

```
1b86 6365 4a13 98ff                     | ..ceJ...
```

Decoding process:

```
1b                                      | == Literal indexed ==
                                        |   Indexed name (idx = 27)
                                        |     cache-control
86                                      |   Literal value (len = 8)
                                        |     Huffman encoded:
6365 4a13 98ff                          | ceJ...
                                        |     Decoded:
                                        | no-cache
                                        | -> cache-control: no-cache
```

Header Table (after decoding):

```
[  1] (s =  53) cache-control: no-cache
[  2] (s =  57) :authority: www.example.com
[  3] (s =  38) :path: /
[  4] (s =  43) :scheme: http
[  5] (s =  42) :method: GET
      Table size: 233
```

Decoded header set:

```
cache-control: no-cache
:authority: www.example.com
:path: /
:scheme: http
:method: GET
```

### D.3.3. Third request

This request has not enough headers in common with the previous request to take advantage of the differential encoding. Therefore, the reference set is emptied before encoding the header fields.

Header set to encode:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

Reference set:

```
[  1] cache-control: no-cache
[  2] :authority: www.example.com
[  3] :path: /
[  4] :scheme: http
[  5] :method: GET
```

Hex dump of encoded data:

```
8080 858c 8b84 0088 4eb0 8b74 9790 fa7f | ........N..t....
894e b08b 7497 9a17 a8ff                | .N..t.....
```

Decoding process:

```
80 80                                   | == Empty reference set ==
                                        |   idx = 0
                                        |   flag = 1
85                                      | == Indexed - Add ==
                                        |   idx = 5
                                        | -> :method: GET
8c                                      | == Indexed - Add ==
                                        |   idx = 12
                                        | -> :scheme: https
8b                                      | == Indexed - Add ==
                                        |   idx = 11
                                        | -> :path: /index.html
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :authority: www.example\
                                        |   .com
00                                      | == Literal indexed ==
88                                      |   Literal name (len = 10)
                                        |     Huffman encoded:
4eb0 8b74 9790 fa7f                     | N..t....
                                        |     Decoded:
                                        | custom-key
89                                      |   Literal value (len = 12)
                                        |     Huffman encoded:
4eb0 8b74 979a 17a8 ff                  | N..t.....
                                        |     Decoded:
                                        | custom-value
                                        | -> custom-key: custom-valu\
                                        |   e
```

Header Table (after decoding):

```
[  1] (s =  54) custom-key: custom-value
[  2] (s =  48) :path: /index.html
[  3] (s =  44) :scheme: https
[  4] (s =  53) cache-control: no-cache
[  5] (s =  57) :authority: www.example.com
[  6] (s =  38) :path: /
[  7] (s =  43) :scheme: http
[  8] (s =  42) :method: GET
      Table size: 379
```

Decoded header set:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

## D.4. Response Examples without Huffman

This section shows several consecutive header sets, corresponding to HTTP responses, on the same connection. SETTINGS_HEADER_TABLE_SIZE is set to the value of 256 octets, causing some evictions to occur.

### D.4.1. First response

Header set to encode:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
0803 3330 3218 0770 7269 7661 7465 221d | ..302..private".
4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
2032 303a 3133 3a32 3120 474d 5430 1768 |  20:13:21 GMT0.h
7474 7073 3a2f 2f77 7777 2e65 7861 6d70 | ttps://www.examp
6c65 2e63 6f6d                          | le.com
```

Decoding process:

```
08                                      | == Literal indexed ==
                                        |   Indexed name (idx = 8)
                                        |     :status
03                                      |   Literal value (len = 3)
3330 32                                 | 302
                                        | -> :status: 302
18                                      | == Literal indexed ==
                                        |   Indexed name (idx = 24)
                                        |     cache-control
07                                      |   Literal value (len = 7)
7072 6976 6174 65                       | private
                                        | -> cache-control: private
22                                      | == Literal indexed ==
                                        |   Indexed name (idx = 34)
                                        |     date
1d                                      |   Literal value (len = 29)
4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
2032 303a 3133 3a32 3120 474d 54        |  20:13:21 GMT
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:21 GMT
30                                      | == Literal indexed ==
                                        |   Indexed name (idx = 48)
                                        |     location
17                                      |   Literal value (len = 23)
6874 7470 733a 2f2f 7777 772e 6578 616d | https://www.exam
706c 652e 636f 6d                       | ple.com
                                        | -> location: https://www.e\
                                        |   xample.com
```

Header Table (after decoding):

```
[  1] (s =  63) location: https://www.example.com
[  2] (s =  65) date: Mon, 21 Oct 2013 20:13:21 GMT
[  3] (s =  52) cache-control: private
[  4] (s =  42) :status: 302
      Table size: 222
```

Decoded header set:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

### D.4.2. Second response

The (":status", "302") header field is evicted from the header table to free space to allow adding the (":status", "200") header field, copied from the static table into the header table. The (":status", "302") header field doesn't need to be removed from the reference set as it is evicted from the header table.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set:

```
[  1] location: https://www.example.com
[  2] date: Mon, 21 Oct 2013 20:13:21 GMT
[  3] cache-control: private
[  4] :status: 302
```

Hex dump of encoded data:

```
8c                                      | .
```

Decoding process:

```
8c                                      | == Indexed - Add ==
                                        |   idx = 12
                                        | - evict: :status: 302
                                        | -> :status: 200
```

Header Table (after decoding):

```
[  1] (s =  42) :status: 200
[  2] (s =  63) location: https://www.example.com
[  3] (s =  65) date: Mon, 21 Oct 2013 20:13:21 GMT
[  4] (s =  52) cache-control: private
      Table size: 222
```

Decoded header set:

```
:status: 200
location: https://www.example.com
date: Mon, 21 Oct 2013 20:13:21 GMT
cache-control: private
```

### D.4.3. Third response

Several header fields are evicted from the header table during the processing of this header set. Before evicting a header belonging to the reference set, it is emitted, by coding it twice as an Indexed Representation. The first representation removes the header field from the reference set, the second one adds it again to the reference set, also emitting it.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
location: https://www.example.com
content-encoding: gzip
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age=3600; version=1
```

Reference set:

```
[  1] :status: 200
[  2] location: https://www.example.com
[  3] date: Mon, 21 Oct 2013 20:13:21 GMT
[  4] cache-control: private
```

Hex dump of encoded data:

```
8484 031d 4d6f 6e2c 2032 3120 4f63 7420 | ....Mon, 21 Oct
3230 3133 2032 303a 3133 3a32 3220 474d | 2013 20:13:22 GM
541d 0467 7a69 7084 8483 833a 3866 6f6f | T..gzip....:8foo
3d41 5344 4a4b 4851 4b42 5a58 4f51 5745 | =ASDJKHQKBZXOQWE
4f50 4955 4158 5157 454f 4955 3b20 6d61 | OPIUAXQWEOIU; ma
782d 6167 653d 3336 3030 3b20 7665 7273 | x-age=3600; vers
696f 6e3d 31                            | ion=1
```

Decoding process:

```
84                                      | == Indexed - Remove ==
                                        |   idx = 4
```

```
                                        | -> cache-control: private
   84                                   | == Indexed - Add ==
                                        |   idx = 4
                                        | -> cache-control: private
   03                                   | == Literal indexed ==
                                        |   Indexed name (idx = 3)
                                        |     date
   1d                                   |   Literal value (len = 29)
   4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
   2032 303a 3133 3a32 3220 474d 54     |  20:13:22 GMT
                                        | - evict: cache-control: pr\
                                        |   ivate
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:22 GMT
   1d                                   | == Literal indexed ==
                                        |   Indexed name (idx = 29)
                                        |     content-encoding
   04                                   |   Literal value (len = 4)
   677a 6970                            | gzip
                                        | - evict: date: Mon, 21 Oct\
                                        |    2013 20:13:21 GMT
                                        | -> content-encoding: gzip
   84                                   | == Indexed - Remove ==
                                        |   idx = 4
                                        | -> location: https://www.e\
                                        |   xample.com
   84                                   | == Indexed - Add ==
                                        |   idx = 4
                                        | -> location: https://www.e\
                                        |   xample.com
   83                                   | == Indexed - Remove ==
                                        |   idx = 3
                                        | -> :status: 200
   83                                   | == Indexed - Add ==
                                        |   idx = 3
                                        | -> :status: 200
   3a                                   | == Literal indexed ==
                                        |   Indexed name (idx = 58)
                                        |     set-cookie
   38                                   |   Literal value (len = 56)
   666f 6f3d 4153 444a 4b48 514b 425a 584f | foo=ASDJKHQKBZXO
   5157 454f 5049 5541 5851 5745 4f49 553b | QWEOPIUAXQWEOIU;
   206d 6178 2d61 6765 3d33 3630 303b 2076 |  max-age=3600; v
   6572 7369 6f6e 3d31                  | ersion=1
                                        | - evict: location: https:/\
                                        |   /www.example.com
                                        | - evict: :status: 200
                                        | -> set-cookie: foo=ASDJKHQ\
```

```
                                         |    KBZXOQWEOPIUAXQWEOIU; ma\
                                         |    x-age=3600; version=1
```

Header Table (after decoding):

```
[  1] (s =  98) set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age\
                 =3600; version=1
[  2] (s =  52) content-encoding: gzip
[  3] (s =  65) date: Mon, 21 Oct 2013 20:13:22 GMT
      Table size: 215
```

Decoded header set:

```
cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
content-encoding: gzip
location: https://www.example.com
:status: 200
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age=3600; version=1
```

## D.5. Response Examples with Huffman

This section shows the same examples as the previous section, but using Huffman encoding for the literal values. The eviction mechanism uses the length of the decoded literal values, so the same evictions occurs as in the previous section.

### D.5.1. First response

Header set to encode:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
0882 98a7 1885 73d5 cd11 1f22 98ef 6b3a | ......s....."..k:
7a0e 6e8f a263 d072 9a6e 8397 d869 bd87 | z.n..c.r.n...i..
3747 bbbf c730 90ce 3174 3d80 1b6d b107 | 7G...0..1t=..m..
cd1a 3962 44b7 4f                       | ..9bD.O
```

Decoding process:

```
08                                      | == Literal indexed ==
                                        |   Indexed name (idx = 8)
                                        |     :status
82                                      |   Literal value (len = 3)
                                        |     Huffman encoded:
98a7                                    | ..
                                        |     Decoded:
                                        | 302
                                        | -> :status: 302
18                                      | == Literal indexed ==
                                        |   Indexed name (idx = 24)
                                        |     cache-control
85                                      |   Literal value (len = 7)
                                        |     Huffman encoded:
73d5 cd11 1f                            | s....
                                        |     Decoded:
                                        | private
                                        | -> cache-control: private
22                                      | == Literal indexed ==
                                        |   Indexed name (idx = 34)
                                        |     date
98                                      |   Literal value (len = 29)
                                        |     Huffman encoded:
ef6b 3a7a 0e6e 8fa2 63d0 729a 6e83 97d8 | .k:z.n..c.r.n...
69bd 8737 47bb bfc7                     | i..7G...
                                        |     Decoded:
                                        | Mon, 21 Oct 2013 20:13:21 \
                                        | GMT
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:21 GMT
30                                      | == Literal indexed ==
                                        |   Indexed name (idx = 48)
                                        |     location
90                                      |   Literal value (len = 23)
                                        |     Huffman encoded:
ce31 743d 801b 6db1 07cd 1a39 6244 b74f | .1t=..m....9bD.O
                                        |     Decoded:
                                        | https://www.example.com
                                        | -> location: https://www.e\
                                        |   xample.com
```

Header Table (after decoding):

```
[  1] (s =  63) location: https://www.example.com
[  2] (s =  65) date: Mon, 21 Oct 2013 20:13:21 GMT
[  3] (s =  52) cache-control: private
[  4] (s =  42) :status: 302
      Table size: 222
```

Decoded header set:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

### D.5.2. Second response

The (":status", "302") header field is evicted from the header table to free space to allow adding the (":status", "200") header field, copied from the static table into the header table. The (":status", "302") header field doesn't need to be removed from the reference set as it is evicted from the header table.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set:

```
[  1] location: https://www.example.com
[  2] date: Mon, 21 Oct 2013 20:13:21 GMT
[  3] cache-control: private
[  4] :status: 302
```

Hex dump of encoded data:

```
8c                                      | .
```

Decoding process:

```
8c                                       | == Indexed - Add ==
                                         |   idx = 12
                                         | - evict: :status: 302
                                         | -> :status: 200
```

Header Table (after decoding):

```
[  1] (s =  42) :status: 200
[  2] (s =  63) location: https://www.example.com
[  3] (s =  65) date: Mon, 21 Oct 2013 20:13:21 GMT
[  4] (s =  52) cache-control: private
      Table size: 222
```

Decoded header set:

```
:status: 200
location: https://www.example.com
date: Mon, 21 Oct 2013 20:13:21 GMT
cache-control: private
```

### D.5.3. Third response

Several header fields are evicted from the header table during the processing of this header set. Before evicting a header belonging to the reference set, it is emitted, by coding it twice as an Indexed Representation. The first representation removes the header field from the reference set, the second one adds it again to the reference set, also emitting it.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
location: https://www.example.com
content-encoding: gzip
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age=3600; version=1
```

Reference set:

```
[  1] :status: 200
[  2] location: https://www.example.com
[  3] date: Mon, 21 Oct 2013 20:13:21 GMT
[  4] cache-control: private
```

Hex dump of encoded data:

```
8484 0398 ef6b 3a7a 0e6e 8fa2 63d0 729a | .....k:z.n..c.r.
6e83 97d8 69bd 873f 47bb bfc7 1d83 cbd5 | n...i..?G.......
4e84 8483 833a b3c5 adb7 7f87 6fc7 fbf7 | N....:......o...
fdbf bebf f3f7 f4fb 7ebb be9f 5f87 e37f | ..........._...
efed faee fa7c 3f1d 5d1a 23ce 5464 36cd | .....|?.].#.Td6.
494b d5d1 cc5f 0535 969b                | IK..._.5..
```

Decoding process:

```
84                                      | == Indexed - Remove ==
                                        |   idx = 4
                                        | -> cache-control: private
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> cache-control: private
03                                      | == Literal indexed ==
                                        |   Indexed name (idx = 3)
                                        |     date
98                                      |   Literal value (len = 29)
                                        |     Huffman encoded:
ef6b 3a7a 0e6e 8fa2 63d0 729a 6e83 97d8 | .k:z.n..c.r.n...
69bd 873f 47bb bfc7                     | i..?G...
                                        |     Decoded:
                                        | Mon, 21 Oct 2013 20:13:22 \
                                        | GMT
                                        | - evict: cache-control: pr\
                                        |   ivate
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:22 GMT
1d                                      | == Literal indexed ==
                                        |   Indexed name (idx = 29)
                                        |     content-encoding
83                                      |   Literal value (len = 4)
                                        |     Huffman encoded:
cbd5 4e                                 | ..N
                                        |     Decoded:
                                        | gzip
```

```
                                         | - evict: date: Mon, 21 Oct\
                                         |    2013 20:13:21 GMT
                                         | -> content-encoding: gzip
84                                       | == Indexed - Remove ==
                                         |   idx = 4
                                         | -> location: https://www.e\
                                         |   xample.com
84                                       | == Indexed - Add ==
                                         |   idx = 4
                                         | -> location: https://www.e\
                                         |   xample.com
83                                       | == Indexed - Remove ==
                                         |   idx = 3
                                         | -> :status: 200
83                                       | == Indexed - Add ==
                                         |   idx = 3
                                         | -> :status: 200
3a                                       | == Literal indexed ==
                                         |   Indexed name (idx = 58)
                                         |     set-cookie
b3                                       |   Literal value (len = 56)
                                         |     Huffman encoded:
c5ad b77f 876f c7fb f7fd bfbe bff3 f7f4 | .....o..........
fb7e bbbe 9f5f 87e3 7fef edfa eefa 7c3f | ....._........|?
1d5d 1a23 ce54 6436 cd49 4bd5 d1cc 5f05 | .].#.Td6.IK..._.
3596 9b                                  | 5..
                                         |     Decoded:
                                         | foo=ASDJKHQKBZXOQWEOPIUAXQ\
                                         | WEOIU; max-age=3600; versi\
                                         | on=1
                                         | - evict: location: https:/\
                                         |   /www.example.com
                                         | - evict: :status: 200
                                         | -> set-cookie: foo=ASDJKHQ\
                                         |   KBZXOQWEOPIUAXQWEOIU; ma\
                                         |   x-age=3600; version=1
```

Header Table (after decoding):

```
[  1] (s =  98) set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age\
                 =3600; version=1
[  2] (s =  52) content-encoding: gzip
[  3] (s =  65) date: Mon, 21 Oct 2013 20:13:22 GMT
      Table size: 215
```

Decoded header set:

```
cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
content-encoding: gzip
location: https://www.example.com
:status: 200
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age=3600; version=1
```

## Authors' Addresses

Roberto Peon
Google, Inc

EMail: fenix@google.com

Herve Ruellan
Canon CRF

EMail: herve.ruellan@crf.canon.fr