HTTPbis Working Group R. Peon
Internet-Draft Google, Inc
Intended status: Standards Track H. Ruellan
Expires: June 2, 2015 Canon CRF
November 29, 2014

HPACK - Header Compression for HTTP/2
draft-ietf-httpbis-header-compression-10

Abstract

This specification defines HPACK, a compression format for efficiently representing HTTP header fields, to be used in 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 can be found at [2]; that specific to HTTP/2 are at [3].

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

In HTTP/1.1 (see [RFC7230]), header fields are not compressed. As Web pages have grown to include dozens to hundreds of requests, the redundant header fields in these requests unnecessarily consume bandwidth, measurably increasing latency.

SPDY [SPDY] initially addressed this redundancy by compressing header fields using the DEFLATE [DEFLATE] format, 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 specification defines HPACK, a new compressor for header fields which eliminates redundant header fields, limits vulnerability to known security attacks, and which has a bounded memory requirement for use in constrained environments.

1.1. Overview

The format defined in this specification treats a list of header fields as an ordered collection of name-value pairs that can include duplicates. Names and values are considered to be opaque sequences of octets, and the order of header fields is preserved after being compressed and decompressed.

Encoding is informed by header tables that map name-value pairs to indexed values. These header tables can be incrementally updated as new pairs are encoded or decoded.

In the encoded form, a header field is represented either literally or as a reference to a name-value pair in one of the header tables. Therefore, a list of header fields can be encoded using a mixture of references and literal values.

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

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

1.2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

All numeric values are in network byte order. Values are unsigned unless otherwise indicated. Literal values are provided in decimal or hexadecimal as appropriate.

1.3. Terminology

This specification uses the following terms:

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

Dynamic Table: The dynamic table (see Section 2.3.2) is a header table used to associate stored header fields to index values. This table is dynamic and specific to an encoding or decoding context.

Static Table: The static table (see Section 2.3.1) is a header table used to associate static header fields to index values. This table is ordered, read-only, always accessible, and may be shared amongst all encoding or decoding contexts.

Header List: A header list is an ordered collection of header fields that are encoded jointly, and can contain duplicate header fields. A complete list of key-value pairs contained in an HTTP/2 header block is a header list.

Header Field Representation: A header field can be represented in encoded form either as a literal or as an index (see Section 2.4).

Header Block: An ordered list of header field representations which, when decoded, yields a complete header list.

2. Compression Process Overview

This specification does not describe a specific algorithm for an encoder. Instead, it defines precisely how a decoder is expected to operate, allowing encoders to produce any encoding that this definition permits.

2.1. Header List Ordering

HPACK preserves the ordering of header fields inside the header list. An encoder MUST order header field representations in the header block according to their ordering in the original header list. A decoder MUST order header fields in the decoded header list according to their ordering in the header block.

2.2. Encoding and Decoding Contexts

To decompress header blocks, a decoder only needs to maintain a dynamic table (see Section 2.3.2) as a decoding context. No other state is needed.

When used for bidirectional communication, such as in HTTP, the encoding and decoding dynamic tables maintained by an endpoint are completely independent. I.e., the request and response dynamic tables are separate.

2.3. Indexing Tables

HPACK uses two tables for associating header fields to indexes. The static table (see Section 2.3.1) is predefined and contains common header fields (most of them with an empty value). The dynamic table (see Section 2.3.2) is dynamic and can be used by the encoder to index header fields repeated in the encoded header lists.

These two tables are combined into a single address space for defining index values (see Section 2.3.3).

2.3.1. Static Table

The static table consists of a predefined static list of header fields. Its entries are defined in Appendix A.

2.3.2. Dynamic Table

The dynamic table consists of a list of header fields maintained in first-in, first-out order. The first and newest entry in a dynamic table is at the lowest index, and the oldest entry of a dynamic table is at the highest index.

The dynamic table is initially empty. Entries are added as each header block is decompressed.

The dynamic table can contain duplicate entries. Therefore, duplicate entries MUST NOT be treated as an error by a decoder.

The encoder decides how to update the dynamic table and as such can control how much memory is used by the dynamic table. To limit the memory requirements of the decoder, the dynamic table size is strictly bounded (see Section 4.2).

The decoder updates the dynamic table during the processing of a list of header field representations (see Section 3.2).

2.3.3. Index Address Space

The static table and the dynamic table are combined into a single index address space.

Indices between 1 and the length of the static table (inclusive) refer to elements in the static table (see Section 2.3.1).

Indices strictly greater than the length of the static table refer to elements in the dynamic table (see Section 2.3.2). The length of the static table is subtracted to find the index into the dynamic table.

Indices strictly greater than the sum of the lengths of both tables MUST be treated as a decoding error.

For a static table size of s and a dynamic table size of k, the following diagram shows the entire valid index address space.

```
<----------  Index Address Space ---------->
<-- Static  Table -->  <-- Dynamic Table -->
+---+-----------+---+  +---+-----------+---+
| 1 |    ...    | s |  |s+1|    ...    |s+k|
+---+-----------+---+  +---+-----------+---+
                       ^                   |
                       |                   V
                Insertion Point      Dropping Point
```

Figure 1: Index Address Space

2.4. Header Field Representation

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

A literal representation defines a header field by specifying its name and value. The header field name can be represented literally or as a reference to an entry in either the static table or the dynamic table. The header field value is represented literally.

Three different literal representations are defined:

- o A literal representation that does not add the header field to the dynamic table (see Section 6.2.2).

- o A literal representation that does not add the header field to the dynamic table, with the additional stipulation that this header field always use a literal representation, in particular when re-encoded by an intermediary (see Section 6.2.3).

- o A literal representation that adds the header field as a new entry at the beginning of the dynamic table (see Section 6.2.1).

An indexed representation defines a header field as a reference to an entry in either the static table or the dynamic table (see Section 6.1).

3. Header Block Decoding

3.1. Header Block Processing

A decoder processes a header block sequentially to reconstruct the original header list.

Once a header field is decoded and added to the reconstructed header list, it cannot be removed from it. A header field added to the header list can be safely passed to the application.

By passing the resulting header fields to the application, a decoder can be implemented with minimal transitory memory commitment in addition to the dynamic table.

3.2. Header Field Representation Processing

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

All the header field representations contained in a header block are processed in the order in which they appear, as specified below. Details on the formatting of the various header field representations, and some additional processing instructions are found in Section 6.

An _indexed representation_ entails the following actions:

- o The header field corresponding to the referenced entry in either the static table or dynamic table is appended to the decoded header list.

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

o The header field is appended to the decoded header list.

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

o The header field is appended to the decoded header list.

o The header field is inserted at the beginning of the dynamic table. This insertion could result in the eviction of previous entries in the dynamic table (see Section 4.4).

4. Dynamic Table Management

To limit the memory requirements on the decoder side, the dynamic table is constrained in size.

4.1. Calculating Table Size

The size of the dynamic 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 5.2), its value's length in octets (see Section 5.2), plus 32.

The size of an entry is calculated using the length of the name and value without any Huffman encoding applied.

NOTE: The additional 32 octets account for the overhead associated with an entry. For example, an entry structure using two 64-bit pointers to reference the name and the value of the entry, and two 64-bit integers for counting the number of references to the name and value would have 32 octets of overhead.

4.2. Maximum Table Size

Protocols that use HPACK determine the maximum size that the encoder is permitted to use for the dynamic table. In HTTP/2, this value is determined by the SETTINGS_HEADER_TABLE_SIZE setting (see Section 6.5.2 of [HTTP2]).

An encoder can choose to use less capacity than this maximum size (see Section 6.3), but the chosen size MUST stay lower than or equal to the maximum set by the protocol.

The maximum size of the dynamic table can be changed by the protocol at any time other than during the encoding of a header block. After changing the maximum size of the dynamic table used by the encoder, the encoder MUST signal this change via an encoding context update (see Section 6.3). This encoding context update MUST occur at the beginning of the first header block following the change to the header table size. In HTTP/2, this follows a settings acknowledgement (see Section 6.5.3 of [HTTP2]).

Multiple updates to the maximum table size can occur between the sending of two header blocks. In the case that the value of this parameter is changed more than once, if any changed value is smaller than the new maximum size, the smallest value for the parameter MUST be sent in an encoding context update. The altered maximum size is always sent, resulting in at most two encoding context updates. This ensures that the decoder is able to perform eviction based on the decoder table size (see Section 4.3).

This mechanism can be used to completely clear entries from the dynamic table by setting a maximum size of 0, which can subsequently be restored.

4.3. Entry Eviction when Dynamic Table Size Changes

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

4.4. Entry Eviction when Adding New Entries

Whenever a new entry is to be added to the dynamic table, entries are evicted from the end of the dynamic table until the size of the dynamic table is less than or equal to (maximum size - new entry size), or until the table is empty.

If the representation of the added entry references the name of an entry in the dynamic table, the referenced name is cached prior to performing eviction to avoid having the name inadvertently evicted.

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; an attempt to add an entry larger than the entire table causes the table to be emptied of all existing entries, and results in an empty table.

5. Primitive Type Representations

HPACK encoding uses two primitive types: unsigned variable length integers, and strings of octets.

5.1. 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.

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

Figure 2: Integer Value Encoded within the Prefix (shown for N = 5)

If the integer value is small enough, i.e. strictly less than 2^N-1, it is encoded within the N-bit prefix.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| ? | ? | ? | 1   1   1   1   1 |
+---+---+---+-------------------+
| 1 |   Value-(2^N-1) LSB (7)   |
+---+---------------------------+
               ...
+---+---------------------------+
| 0 |   Value-(2^N-1) MSB (7)   |
+---+---------------------------+
```

Figure 3: Integer Value Encoded after the Prefix (shown for N = 5)

Otherwise, all the bits of the prefix are set to 1 and the value, decreased by 2^N-1, is encoded using a list of one or more octets. The most significant bit of each octet is used as a continuation flag: its value is set to 1 except for the last octet in the list. The remaining bits of the octets are used to encode the decreased value.

Decoding the integer value from the list of octets starts by reversing the order of the octets in the list. Then, for each octet,

its most significant bit is removed. The remaining bits of the octets are concatenated and the resulting value is increased by 2^N-1 to obtain the integer value.

The prefix size, N, is always between 1 and 8 bits. An integer starting at an octet-boundary will have an 8-bit prefix.

Pseudo-code 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
```

Pseudo-code 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
```

Examples illustrating the encoding of integers are available in Appendix C.1.

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.

5.2. String Literal Representation

Header field names and header field values can be represented as literal string. A literal string is encoded as a sequence of octets, either by directly encoding the literal string's octets, or by using a Huffman code (see [HUFFMAN]).

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

Figure 4: String Literal Representation

A literal string representation contains the following fields:

H: A one bit flag, H, indicating whether or not the octets of the string are Huffman encoded.

String Length: The number of octets used to encode the string literal, encoded as an integer with 7-bit prefix (see Section 5.1).

String Data: The encoded data of the string literal. If H is '0', then the encoded data is the raw octets of the string literal. If H is '1', then the encoded data is the Huffman encoding of the string literal.

String literals which use Huffman encoding are encoded with the Huffman code defined in Appendix B (see examples for requests in Appendix C.4 and for responses in Appendix C.6). The encoded data is the bitwise concatenation of the codes corresponding to each octet of the string literal.

As the Huffman encoded data doesn't always end at an octet boundary, some padding is inserted after it, up to the next octet boundary. To prevent this padding to be misinterpreted as part of the string literal, the most significant bits of the code corresponding to the EOS (end-of-string) symbol are used.

Upon decoding, an incomplete code at the end of the encoded data is to be considered as padding and discarded. A padding strictly longer than 7 bits MUST be treated as a decoding error. A padding not corresponding to the most significant bits of the code for the EOS symbol MUST be treated as a decoding error. A Huffman encoded string literal containing the EOS symbol MUST be treated as a decoding error.

6. Binary Format

This section describes the detailed format of each of the different header field representations, plus the encoding context update instruction.

6.1. Indexed Header Field Representation

An indexed header field representation identifies an entry in either the static table or the dynamic table (see Section 2.3).

An indexed header field representation causes a header field to be added to the decoded header list, as described in Section 3.2.

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

Figure 5: Indexed Header Field

An indexed header field starts with the '1' 1-bit pattern, followed by the index of the matching pair, represented as an integer with a 7-bit prefix (see Section 5.1).

The index value of 0 is not used. It MUST be treated as a decoding error if found in an indexed header field representation.

6.2. Literal Header Field Representation

A literal header field representation contains a literal header field value. Header field names are either provided as a literal or by reference to an existing table entry, either from the static table or the dynamic table (see Section 2.3).

This specification defines three forms of literal header field representations; with indexing, without indexing, and never indexed.

6.2.1. Literal Header Field with Incremental Indexing

A literal header field with incremental indexing representation results in appending a header field to the decoded header list and inserting it as a new entry into the dynamic table.

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

Figure 6: Literal Header Field with Incremental 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)  |
+-------------------------------+
```

Figure 7: Literal Header Field with Incremental Indexing - New Name

A literal header field with incremental indexing representation starts with the '01' 2-bit pattern.

If the header field name matches the header field name of an entry stored in the static table or the dynamic table, the header field name can be represented using the index of that entry. In this case, the index of the entry is represented as an integer with a 6-bit prefix (see Section 5.1). This value is always non-zero.

Otherwise, the header field name is represented as a literal string (see Section 5.2). A value 0 is used in place of the 6-bit index, followed by the header field name.

Either form of header field name representation is followed by the header field value represented as a literal string (see Section 5.2).

6.2.2. Literal Header Field without Indexing

A literal header field without indexing representation results in appending a header field to the decoded header list without altering the dynamic table.

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

Figure 8: Literal Header Field without Indexing - Indexed Name

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

Figure 9: Literal Header Field without Indexing - New Name

A literal header field without indexing representation starts with the '0000' 4-bit pattern.

If the header field name matches the header field name of an entry stored in the static table or the dynamic table, the header field name can be represented using the index of that entry. In this case, the index of the entry is represented as an integer with a 4-bit prefix (see Section 5.1). This value is always non-zero.

Otherwise, the header field name is represented as a literal string (see Section 5.2). A value 0 is used in place of the 4-bit index, followed by the header field name.

Either form of header field name representation is followed by the header field value represented as a literal string (see Section 5.2).

6.2.3. Literal Header Field never Indexed

A literal header field never indexed representation results in appending a header field to the decoded header list without altering the dynamic table. Intermediaries MUST use the same representation for encoding this header field.

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

Figure 10: Literal Header Field never Indexed - Indexed Name

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

Figure 11: Literal Header Field never Indexed - New Name

A literal header field never indexed representation starts with the '0001' 4-bit pattern.

When a header field is represented as a literal header field never indexed, it MUST always be encoded with this specific literal representation. In particular, when a peer sends a header field that it received represented as a literal header field never indexed, it MUST use the same representation to forward this header field.

This representation is intended for protecting header field values that are not to be put at risk by compressing them (see Section 7.1 for more details).

The encoding of the representation is identical to the literal header field without indexing (see Section 6.2.2).

6.3. Dynamic Table Size Update

A dynamic table size update signals a change to the size of the dynamic table.

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

Figure 12: Maximum Dynamic Table Size Change

A dynamic table size update starts with the '001' 3-bit pattern, followed by the new maximum size, represented as an integer with a 5-bit prefix (see Section 5.1).

The new maximum size MUST be lower than or equal to the last value of the SETTINGS_HEADER_TABLE_SIZE parameter (see Section 6.5.2 of [HTTP2]) received from the decoder and acknowledged by the encoder (see Section 6.5.3 of [HTTP2]).

Reducing the maximum size of the dynamic table can cause entries to be evicted (see Section 4.3).

7. Security Considerations

This section describes potential areas of security concern with HPACK:

- o Use of compression as a length-based oracle for verifying guesses about secrets that are compressed into a shared compression context.

- o Denial of service resulting from exhausting processing or memory capacity at a decoder.

7.1. Probing Dynamic Table State

HPACK reduces the length of header field encodings by exploiting the redundancy inherent in protocols like HTTP. The ultimate goal of this is to reduce the amount of data that is required to send HTTP requests or responses.

The compression context used to encode header fields can be probed by an attacker who can both define header fields to be encoded and transmitted and observe the length of those fields once they are encoded. When an attacker can do both, they can adaptively modify requests in order to confirm guesses about the dynamic table state. If a guess is compressed into a shorter length, the attacker can observe the encoded length and infer that the guess was correct.

This is possible even over TLS, because while TLS provides confidentiality protection for content, it only provides a limited amount of protection for the length of that content.

Note: Padding schemes only provide limited protection against an attacker with these capabilities, potentially only forcing an increased number of guesses to learn the length associated with a given guess. Padding schemes also work directly against compression by increasing the number of bits that are transmitted.

Attacks like CRIME [CRIME] demonstrated the existence of these general attacker capabilities. The specific attack exploited the fact that DEFLATE [DEFLATE] removes redundancy based on prefix matching. This permitted the attacker to confirm guesses a character at a time, reducing an exponential-time attack into a linear-time attack.

7.1.1. Applicability to HPACK and HTTP

HPACK mitigates but does not completely prevent attacks modelled on CRIME [CRIME] by forcing a guess to match an entire header field value, rather than individual characters. An attacker can only learn whether a guess is correct or not, so is reduced to a brute force guess for the header field values.

The viability of recovering specific header field values therefore depends on the entropy of values. As a result, values with high entropy are unlikely to be recovered successfully. However, values with low entropy remain vulnerable.

Attacks of this nature are possible any time that two mutually distrustful entities control requests or responses that are placed onto a single HTTP/2 connection. If the shared HPACK compressor permits one entity to add entries to the dynamic table, and the other to access those entries, then the state of the table can be learned.

Having requests or responses from mutually distrustful entities occurs when an intermediary either:

- o sends requests from multiple clients on a single connection toward an origin server, or

- o takes responses from multiple origin servers and places them on a shared connection toward a client.

Web browsers also need to assume that requests made on the same connection by different web origins [ORIGIN] are made by mutually distrustful entities.

7.1.2. Mitigation

Users of HTTP that require confidentiality for header fields can use values with entropy sufficient to make guessing infeasible. However, this is impractical as a general solution because it forces all users of HTTP to take steps to mitigate attacks. It would impose new constraints on how HTTP is used.

Rather than impose constraints on users of HTTP, an implementation of HPACK can instead constrain how compression is applied in order to limit the potential for dynamic table probing.

An ideal solution segregates access to the dynamic table based on the entity that is constructing header fields. Header field values that are added to the table are attributed to an entity, and only the entity that created a particular value can extract that value.

To improve compression performance of this option, certain entries might be tagged as being public. For example, a web browser might make the values of the Accept-Encoding header field available in all requests.

An encoder without good knowledge of the provenance of header fields might instead introduce a penalty for bad guesses, such that attempts to guess a header field value results in all values being removed from consideration in all future requests, effectively preventing further guesses.

Note: Simply removing values from the dynamic table can be ineffectual if the attacker has a reliable way of causing values to be reinstalled. For example, a request to load an image in a web browser typically includes the Cookie header field (a potentially highly valued target for this sort of attack), and web sites can easily force an image to be loaded, thereby refreshing the entry in the dynamic table.

This response might be made inversely proportional to the length of the header field. Marking as inaccessible might occur for shorter values more quickly or with higher probability than for longer values.

Implementations might also choose to protect certain header fields that are known to be highly valued, such as the Authorization or Cookie header fields, by disabling or further limiting compression.

7.1.3. Never Indexed Literals

Refusing to generate an indexed representation for a header field is only effective if compression is avoided on all hops. The never indexed literal (see Section 6.2.3) can be used to signal to intermediaries that a particular value was intentionally sent as a literal. An intermediary MUST NOT re-encode a value that uses the never indexed literal with a representation that would index it.

7.2. Static Huffman Encoding

There is no currently known attack against a fixed Huffman encoding. A study has shown that using a fixed Huffman encoding table created an information leakage, however this same study concluded that an attacker could not take advantage of this information leakage to recover any meaningful amount of information (see [PETAL]).

7.3. Memory Consumption

An attacker can try to cause an endpoint to exhaust its memory. HPACK is designed to limit both the peak and state amounts of memory allocated by an endpoint.

The amount of memory used by the compressor state is limited by the decoder using the value of the HTTP/2 setting parameter SETTINGS_HEADER_TABLE_SIZE (see Section 6.5.2 of [HTTP2]). This limit takes into account both the size of the data stored in the dynamic table, plus a small allowance for overhead.

A decoder can limit the amount of state memory used by setting an appropriate value for the SETTINGS_HEADER_TABLE_SIZE parameter. An encoder can limit the amount of state memory it uses by signalling lower dynamic table size than the decoder allows (see Section 6.3).

The amount of temporary memory consumed by an encoder or decoder can be limited by processing header fields sequentially. An implementation does not need to retain a complete list of header fields. Note however that it might be necessary for an application to retain a complete header list for other reasons; even though HPACK does not force this to occur, application constraints might make this necessary.

7.4. Implementation Limits

An implementation of HPACK needs to ensure that large values for integers, long encoding for integers, or long string literals do not create security weaknesses.

An implementation has to set a limit for the values it accepts for integers, as well as for the encoded length (see Section 5.1). In the same way, it has to set a limit to the length it accepts for string literals (see Section 5.2).

8. Acknowledgements

This specification includes substantial input from the following individuals:

- o Mike Bishop, Jeff Pinner, Julian Reschke, Martin Thomson (substantial editorial contributions).

- o Johnny Graettinger (Huffman code statistics).

9. References

9.1. Normative References

[HTTP2] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-16 (work in progress), October 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

9.2. Informative References

[CANONICAL] Schwartz, E. and B. Kallick, "Generating a canonical prefix encoding", Communications of the ACM Volume 7 Issue 3, pp. 166-169, March 1964, <https://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.

[HUFFMAN] 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, <https://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4051119>.

[ORIGIN] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[PETAL] Tan, J. and J. Nahata, "PETAL: Preset Encoding Table Information Leakage", April 2013, <http://www.pdl.cmu.edu/PDL-FTP/associated/CMU-PDL-13-106.pdf>.

[SPDY] Belshe, M. and R. Peon, "SPDY Protocol", draft-mbelshe-httpbis-spdy-00 (work in progress), February 2012.

Appendix A. Static Table Definition

The static table (see Section 2.3.1) consists of a predefined and unchangeable list of header fields.

The static table was created by listing the most common header fields that are valid for messages exchanged inside a HTTP/2 connection. For header fields with a few frequent values, an entry was added for each of these frequent values. For other header fields, an entry was added with an empty value.

The following table lists the pre-defined header fields that make-up the static 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	204
10	:status	206
11	:status	304
12	:status	400
13	:status	404
14	:status	500
15	accept-charset	
16	accept-encoding	gzip, deflate
17	accept-language	
18	accept-ranges	
19	accept	
20	access-control-allow-origin	
21	age	
22	allow	
23	authorization	
24	cache-control	
25	content-disposition	
26	content-encoding	
27	content-language	
28	content-length	
29	content-location	
30	content-range	
31	content-type	

32	cookie		
33	date		
34	etag		
35	expect		
36	expires		
37	from		
38	host		
39	if-match		
40	if-modified-since		
41	if-none-match		
42	if-range		
43	if-unmodified-since		
44	last-modified		
45	link		
46	location		
47	max-forwards		
48	proxy-authenticate		
49	proxy-authorization		
50	range		
51	referer		
52	refresh		
53	retry-after		
54	server		
55	set-cookie		
56	strict-transport-security		
57	transfer-encoding		
58	user-agent		
59	vary		
60	via		
61	www-authenticate		

Table 1: Static Table Entries

Table 1 gives the index of each entry in the static table.

Appendix B. Huffman Code

The following Huffman code is used when encoding string literals with a Huffman coding (see Section 5.2).

This Huffman code was generated from statistics obtained on a large sample of HTTP headers. It is a canonical Huffman code (see [CANONICAL]) with some tweaking to ensure that no symbol has a unique code length.

Each row in the table defines the code used to represent a symbol:

sym: The symbol to be represented. It is the decimal value of an octet, possibly prepended with its ASCII representation. A specific symbol, "EOS", is used to indicate the end of a string literal.

code as bits: The Huffman code for the symbol represented as a base-2 integer, aligned on the most significant bit (MSB).

code as hex: The Huffman code for the symbol, represented as a hexadecimal integer, aligned on the least significant bit (LSB).

len: The number of bits for the code representing the symbol.

As an example, the code for the symbol 47 (corresponding to the ASCII character "/") consists in the 6 bits "0", "1", "1", "0", "0", "0". This corresponds to the value 0x18 (in hexadecimal) encoded on 6 bits.

```
                                                        code
                          code as bits                 as hex   len
        sym              aligned to MSB                aligned   in
                                                       to LSB   bits
       (  0)  |11111111|11000                             1ff8  [13]
       (  1)  |11111111|11111111|1011000                7fffd8  [23]
       (  2)  |11111111|11111111|11111110|0010         fffffe2  [28]
       (  3)  |11111111|11111111|11111110|0011         fffffe3  [28]
       (  4)  |11111111|11111111|11111110|0100         fffffe4  [28]
       (  5)  |11111111|11111111|11111110|0101         fffffe5  [28]
       (  6)  |11111111|11111111|11111110|0110         fffffe6  [28]
       (  7)  |11111111|11111111|11111110|0111         fffffe7  [28]
       (  8)  |11111111|11111111|11111110|1000         fffffe8  [28]
       (  9)  |11111111|11111111|11101010               ffffea  [24]
       ( 10)  |11111111|11111111|11111111|111100      3ffffffc  [30]
       ( 11)  |11111111|11111111|11111110|1001         fffffe9  [28]
       ( 12)  |11111111|11111111|11111110|1010         fffffea  [28]
       ( 13)  |11111111|11111111|11111111|111101      3ffffffd  [30]
       ( 14)  |11111111|11111111|11111110|1011         fffffeb  [28]
       ( 15)  |11111111|11111111|11111110|1100         fffffec  [28]
       ( 16)  |11111111|11111111|11111110|1101         fffffed  [28]
       ( 17)  |11111111|11111111|11111110|1110         fffffee  [28]
       ( 18)  |11111111|11111111|11111110|1111         fffffef  [28]
       ( 19)  |11111111|11111111|11111111|0000         ffffff0  [28]
       ( 20)  |11111111|11111111|11111111|0001         ffffff1  [28]
       ( 21)  |11111111|11111111|11111111|0010         ffffff2  [28]
       ( 22)  |11111111|11111111|11111111|111110      3ffffffe  [30]
       ( 23)  |11111111|11111111|11111111|0011         ffffff3  [28]
       ( 24)  |11111111|11111111|11111111|0100         ffffff4  [28]
       ( 25)  |11111111|11111111|11111111|0101         ffffff5  [28]
```

```
       ( 26)  |11111111|11111111|11111111|0110         ffffff6  [28]
       ( 27)  |11111111|11111111|11111111|0111         ffffff7  [28]
       ( 28)  |11111111|11111111|11111111|1000         ffffff8  [28]
       ( 29)  |11111111|11111111|11111111|1001         ffffff9  [28]
       ( 30)  |11111111|11111111|11111111|1010         ffffffa  [28]
       ( 31)  |11111111|11111111|11111111|1011         ffffffb  [28]
   ' ' ( 32)  |010100                                       14  [ 6]
   '!' ( 33)  |11111110|00                                 3f8  [10]
   '"' ( 34)  |11111110|01                                 3f9  [10]
   '#' ( 35)  |11111111|1010                               ffa  [12]
   '$' ( 36)  |11111111|11001                             1ff9  [13]
   '%' ( 37)  |010101                                       15  [ 6]
   '&' ( 38)  |11111000                                     f8  [ 8]
   ''' ( 39)  |11111111|010                                7fa  [11]
   '(' ( 40)  |11111110|10                                 3fa  [10]
   ')' ( 41)  |11111110|11                                 3fb  [10]
   '*' ( 42)  |11111001                                     f9  [ 8]
   '+' ( 43)  |11111111|011                                7fb  [11]
   ',' ( 44)  |11111010                                     fa  [ 8]
   '-' ( 45)  |010110                                       16  [ 6]
   '.' ( 46)  |010111                                       17  [ 6]
   '/' ( 47)  |011000                                       18  [ 6]
   '0' ( 48)  |00000                                         0  [ 5]
   '1' ( 49)  |00001                                         1  [ 5]
   '2' ( 50)  |00010                                         2  [ 5]
   '3' ( 51)  |011001                                       19  [ 6]
   '4' ( 52)  |011010                                       1a  [ 6]
   '5' ( 53)  |011011                                       1b  [ 6]
   '6' ( 54)  |011100                                       1c  [ 6]
   '7' ( 55)  |011101                                       1d  [ 6]
   '8' ( 56)  |011110                                       1e  [ 6]
   '9' ( 57)  |011111                                       1f  [ 6]
   ':' ( 58)  |1011100                                      5c  [ 7]
   ';' ( 59)  |11111011                                     fb  [ 8]
   '<' ( 60)  |11111111|1111100                           7ffc  [15]
   '=' ( 61)  |100000                                       20  [ 6]
   '>' ( 62)  |11111111|1011                               ffb  [12]
   '?' ( 63)  |11111111|00                                 3fc  [10]
   '@' ( 64)  |11111111|11010                             1ffa  [13]
   'A' ( 65)  |100001                                       21  [ 6]
   'B' ( 66)  |1011101                                      5d  [ 7]
   'C' ( 67)  |1011110                                      5e  [ 7]
   'D' ( 68)  |1011111                                      5f  [ 7]
   'E' ( 69)  |1100000                                      60  [ 7]
   'F' ( 70)  |1100001                                      61  [ 7]
   'G' ( 71)  |1100010                                      62  [ 7]
   'H' ( 72)  |1100011                                      63  [ 7]
   'I' ( 73)  |1100100                                      64  [ 7]
```

```
    'J' ( 74)  |1100101                                      65  [ 7]
    'K' ( 75)  |1100110                                      66  [ 7]
    'L' ( 76)  |1100111                                      67  [ 7]
    'M' ( 77)  |1101000                                      68  [ 7]
    'N' ( 78)  |1101001                                      69  [ 7]
    'O' ( 79)  |1101010                                      6a  [ 7]
    'P' ( 80)  |1101011                                      6b  [ 7]
    'Q' ( 81)  |1101100                                      6c  [ 7]
    'R' ( 82)  |1101101                                      6d  [ 7]
    'S' ( 83)  |1101110                                      6e  [ 7]
    'T' ( 84)  |1101111                                      6f  [ 7]
    'U' ( 85)  |1110000                                      70  [ 7]
    'V' ( 86)  |1110001                                      71  [ 7]
    'W' ( 87)  |1110010                                      72  [ 7]
    'X' ( 88)  |11111100                                     fc  [ 8]
    'Y' ( 89)  |1110011                                      73  [ 7]
    'Z' ( 90)  |11111101                                     fd  [ 8]
    '[' ( 91)  |11111111|11011                             1ffb  [13]
    '\' ( 92)  |11111111|11111110|000                     7fff0  [19]
    ']' ( 93)  |11111111|11100                             1ffc  [13]
    '^' ( 94)  |11111111|111100                            3ffc  [14]
    '_' ( 95)  |100010                                       22  [ 6]
    '`' ( 96)  |11111111|1111101                           7ffd  [15]
    'a' ( 97)  |00011                                         3  [ 5]
    'b' ( 98)  |100011                                       23  [ 6]
    'c' ( 99)  |00100                                         4  [ 5]
    'd' (100)  |100100                                       24  [ 6]
    'e' (101)  |00101                                         5  [ 5]
    'f' (102)  |100101                                       25  [ 6]
    'g' (103)  |100110                                       26  [ 6]
    'h' (104)  |100111                                       27  [ 6]
    'i' (105)  |00110                                         6  [ 5]
    'j' (106)  |1110100                                      74  [ 7]
    'k' (107)  |1110101                                      75  [ 7]
    'l' (108)  |101000                                       28  [ 6]
    'm' (109)  |101001                                       29  [ 6]
    'n' (110)  |101010                                       2a  [ 6]
    'o' (111)  |00111                                         7  [ 5]
    'p' (112)  |101011                                       2b  [ 6]
    'q' (113)  |1110110                                      76  [ 7]
    'r' (114)  |101100                                       2c  [ 6]
    's' (115)  |01000                                         8  [ 5]
    't' (116)  |01001                                         9  [ 5]
    'u' (117)  |101101                                       2d  [ 6]
    'v' (118)  |1110111                                      77  [ 7]
    'w' (119)  |1111000                                      78  [ 7]
    'x' (120)  |1111001                                      79  [ 7]
    'y' (121)  |1111010                                      7a  [ 7]
```

```
    'z' (122)  |1111011                                      7b  [ 7]
    '{' (123)  |11111111|1111110                           7ffe  [15]
    '|' (124)  |11111111|100                                7fc  [11]
    '}' (125)  |11111111|111101                            3ffd  [14]
    '~' (126)  |11111111|11101                             1ffd  [13]
        (127)  |11111111|11111111|11111111|1100         ffffffc  [28]
        (128)  |11111111|11111110|0110                    fffe6  [20]
        (129)  |11111111|11111111|010010                 3fffd2  [22]
        (130)  |11111111|11111110|0111                    fffe7  [20]
        (131)  |11111111|11111110|1000                    fffe8  [20]
        (132)  |11111111|11111111|010011                 3fffd3  [22]
        (133)  |11111111|11111111|010100                 3fffd4  [22]
        (134)  |11111111|11111111|010101                 3fffd5  [22]
        (135)  |11111111|11111111|1011001                7fffd9  [23]
        (136)  |11111111|11111111|010110                 3fffd6  [22]
        (137)  |11111111|11111111|1011010                7fffda  [23]
        (138)  |11111111|11111111|1011011                7fffdb  [23]
        (139)  |11111111|11111111|1011100                7fffdc  [23]
        (140)  |11111111|11111111|1011101                7fffdd  [23]
        (141)  |11111111|11111111|1011110                7fffde  [23]
        (142)  |11111111|11111111|11101011               ffffeb  [24]
        (143)  |11111111|11111111|1011111                7fffdf  [23]
        (144)  |11111111|11111111|11101100               ffffec  [24]
        (145)  |11111111|11111111|11101101               ffffed  [24]
        (146)  |11111111|11111111|010111                 3fffd7  [22]
        (147)  |11111111|11111111|1100000                7fffe0  [23]
        (148)  |11111111|11111111|11101110               ffffee  [24]
        (149)  |11111111|11111111|1100001                7fffe1  [23]
        (150)  |11111111|11111111|1100010                7fffe2  [23]
        (151)  |11111111|11111111|1100011                7fffe3  [23]
        (152)  |11111111|11111111|1100100                7fffe4  [23]
        (153)  |11111111|11111110|11100                  1fffdc  [21]
        (154)  |11111111|11111111|011000                 3fffd8  [22]
        (155)  |11111111|11111111|1100101                7fffe5  [23]
        (156)  |11111111|11111111|011001                 3fffd9  [22]
        (157)  |11111111|11111111|1100110                7fffe6  [23]
        (158)  |11111111|11111111|1100111                7fffe7  [23]
        (159)  |11111111|11111111|11101111               ffffef  [24]
        (160)  |11111111|11111111|011010                 3fffda  [22]
        (161)  |11111111|11111110|11101                  1fffdd  [21]
        (162)  |11111111|11111110|1001                    fffe9  [20]
        (163)  |11111111|11111111|011011                 3fffdb  [22]
        (164)  |11111111|11111111|011100                 3fffdc  [22]
        (165)  |11111111|11111111|1101000                7fffe8  [23]
        (166)  |11111111|11111111|1101001                7fffe9  [23]
        (167)  |11111111|11111110|11110                  1fffde  [21]
        (168)  |11111111|11111111|1101010                7fffea  [23]
        (169)  |11111111|11111111|011101                 3fffdd  [22]
```

```
    (170)  |11111111|11111111|011110                 3fffde  [22]
    (171)  |11111111|11111111|11110000               fffff0  [24]
    (172)  |11111111|11111110|11111                  1fffdf  [21]
    (173)  |11111111|11111111|011111                 3fffdf  [22]
    (174)  |11111111|11111111|1101011                7fffeb  [23]
    (175)  |11111111|11111111|1101100                7fffec  [23]
    (176)  |11111111|11111111|00000                  1fffe0  [21]
    (177)  |11111111|11111111|00001                  1fffe1  [21]
    (178)  |11111111|11111111|100000                 3fffe0  [22]
    (179)  |11111111|11111111|00010                  1fffe2  [21]
    (180)  |11111111|11111111|1101101                7fffed  [23]
    (181)  |11111111|11111111|100001                 3fffe1  [22]
    (182)  |11111111|11111111|1101110                7fffee  [23]
    (183)  |11111111|11111111|1101111                7fffef  [23]
    (184)  |11111111|11111110|1010                    fffea  [20]
    (185)  |11111111|11111111|100010                 3fffe2  [22]
    (186)  |11111111|11111111|100011                 3fffe3  [22]
    (187)  |11111111|11111111|100100                 3fffe4  [22]
    (188)  |11111111|11111111|1110000                7ffff0  [23]
    (189)  |11111111|11111111|100101                 3fffe5  [22]
    (190)  |11111111|11111111|100110                 3fffe6  [22]
    (191)  |11111111|11111111|1110001                7ffff1  [23]
    (192)  |11111111|11111111|11111000|00           3ffffe0  [26]
    (193)  |11111111|11111111|11111000|01           3ffffe1  [26]
    (194)  |11111111|11111110|1011                    fffeb  [20]
    (195)  |11111111|11111110|001                     7fff1  [19]
    (196)  |11111111|11111111|100111                 3fffe7  [22]
    (197)  |11111111|11111111|1110010                7ffff2  [23]
    (198)  |11111111|11111111|101000                 3fffe8  [22]
    (199)  |11111111|11111111|11110110|0            1ffffec  [25]
    (200)  |11111111|11111111|11111000|10           3ffffe2  [26]
    (201)  |11111111|11111111|11111000|11           3ffffe3  [26]
    (202)  |11111111|11111111|11111001|00           3ffffe4  [26]
    (203)  |11111111|11111111|11111011|110          7ffffde  [27]
    (204)  |11111111|11111111|11111011|111          7ffffdf  [27]
    (205)  |11111111|11111111|11111001|01           3ffffe5  [26]
    (206)  |11111111|11111111|11110001               fffff1  [24]
    (207)  |11111111|11111111|11110110|1            1ffffed  [25]
    (208)  |11111111|11111110|010                     7fff2  [19]
    (209)  |11111111|11111111|00011                  1fffe3  [21]
    (210)  |11111111|11111111|11111001|10           3ffffe6  [26]
    (211)  |11111111|11111111|11111100|000          7ffffe0  [27]
    (212)  |11111111|11111111|11111100|001          7ffffe1  [27]
    (213)  |11111111|11111111|11111001|11           3ffffe7  [26]
    (214)  |11111111|11111111|11111100|010          7ffffe2  [27]
    (215)  |11111111|11111111|11110010               fffff2  [24]
    (216)  |11111111|11111111|00100                  1fffe4  [21]
    (217)  |11111111|11111111|00101                  1fffe5  [21]
```

```
      (218)  |11111111|11111111|11111010|00           3ffffe8  [26]
      (219)  |11111111|11111111|11111010|01           3ffffe9  [26]
      (220)  |11111111|11111111|11111111|1101         ffffffd  [28]
      (221)  |11111111|11111111|11111100|011          7ffffe3  [27]
      (222)  |11111111|11111111|11111100|100          7ffffe4  [27]
      (223)  |11111111|11111111|11111100|101          7ffffe5  [27]
      (224)  |11111111|11111110|1100                    fffec  [20]
      (225)  |11111111|11111111|11110011               fffff3  [24]
      (226)  |11111111|11111110|1101                    fffed  [20]
      (227)  |11111111|11111111|00110                  1fffe6  [21]
      (228)  |11111111|11111111|101001                 3fffe9  [22]
      (229)  |11111111|11111111|00111                  1fffe7  [21]
      (230)  |11111111|11111111|01000                  1fffe8  [21]
      (231)  |11111111|11111111|1110011                7ffff3  [23]
      (232)  |11111111|11111111|101010                 3fffea  [22]
      (233)  |11111111|11111111|101011                 3fffeb  [22]
      (234)  |11111111|11111111|11110111|0            1ffffee  [25]
      (235)  |11111111|11111111|11110111|1            1ffffef  [25]
      (236)  |11111111|11111111|11110100               fffff4  [24]
      (237)  |11111111|11111111|11110101               fffff5  [24]
      (238)  |11111111|11111111|11111010|10           3ffffea  [26]
      (239)  |11111111|11111111|1110100                7ffff4  [23]
      (240)  |11111111|11111111|11111010|11           3ffffeb  [26]
      (241)  |11111111|11111111|11111100|110          7ffffe6  [27]
      (242)  |11111111|11111111|11111011|00           3ffffec  [26]
      (243)  |11111111|11111111|11111011|01           3ffffed  [26]
      (244)  |11111111|11111111|11111100|111          7ffffe7  [27]
      (245)  |11111111|11111111|11111101|000          7ffffe8  [27]
      (246)  |11111111|11111111|11111101|001          7ffffe9  [27]
      (247)  |11111111|11111111|11111101|010          7ffffea  [27]
      (248)  |11111111|11111111|11111101|011          7ffffeb  [27]
      (249)  |11111111|11111111|11111111|1110         ffffffe  [28]
      (250)  |11111111|11111111|11111101|100          7ffffec  [27]
      (251)  |11111111|11111111|11111101|101          7ffffed  [27]
      (252)  |11111111|11111111|11111101|110          7ffffee  [27]
      (253)  |11111111|11111111|11111101|111          7ffffef  [27]
      (254)  |11111111|11111111|11111110|000          7fffff0  [27]
      (255)  |11111111|11111111|11111011|10           3ffffee  [26]
   EOS (256)  |11111111|11111111|11111111|111111      3fffffff  [30]
```

Appendix C. Examples

A number of examples are worked through here, covering integer encoding, header field representation, and the encoding of whole lists of header fields, for both requests and responses, and with and without Huffman coding.

C.1. Integer Representation Examples

This section shows the representation of integer values in details (see Section 5.1).

C.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
+---+---+---+---+---+---+---+---+
```

C.1.2. Example 2: Encoding 1337 Using a 5-bit Prefix

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

1337 is greater than 31 (2^5 - 1).

The 5-bit prefix is filled with its max value (31).

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

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

I % 128 == 26

26 + 128 == 154

154 is encoded in 8 bits as: 10011010

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

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

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

The process ends.

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

C.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.

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

C.2. Header Field Representation Examples

This section shows several independent representation examples.

C.2.1. Literal Header Field with Indexing

The header field representation uses a literal name and a literal value. The header field is added to the dynamic table.

Header list to encode:

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

Hex dump of encoded data:

```
400a 6375 7374 6f6d 2d6b 6579 0d63 7573 | @.custom-key.cus
746f 6d2d 6865 6164 6572                | tom-header
```

Decoding process:

```
40                                      | == 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
```

Dynamic Table (after decoding):

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

Decoded header list:

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

C.2.2. Literal Header Field without Indexing

The header field representation uses an indexed name and a literal value. The header field is not added to the dynamic table.

Header list to encode:

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

Hex dump of encoded data:

```
040c 2f73 616d 706c 652f 7061 7468      | ../sample/path
```

Decoding process:

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

Dynamic table (after decoding): empty.

Decoded header list:

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

C.2.3. Literal Header Field never Indexed

The header field representation uses a literal name and a literal value. The header field is not added to the dynamic table, and must use the same representation if re-encoded by an intermediary.

Header list to encode:

```
password: secret
```

Hex dump of encoded data:

```
1008 7061 7373 776f 7264 0673 6563 7265 | ..password.secre
74                                      | t
```

Decoding process:

```
10                                      | == Literal never indexed ==
08                                      |   Literal name (len = 8)
7061 7373 776f 7264                     | password
06                                      |   Literal value (len = 6)
7365 6372 6574                          | secret
                                        | -> password: secret
```

Dynamic table (after decoding): empty.

Decoded header list:

```
password: secret
```

C.2.4. Indexed Header Field

The header field representation uses an indexed header field, from the static table.

Header list to encode:

```
:method: GET
```

Hex dump of encoded data:

```
82                                      | .
```

Decoding process:

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

Dynamic table (after decoding): empty.

Decoded header list:

:method: GET

C.3. Request Examples without Huffman Coding

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

C.3.1. First Request

Header list to encode:

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

Hex dump of encoded data:

```
8286 8441 0f77 7777 2e65 7861 6d70 6c65 | ...A.www.example
2e63 6f6d                               | .com
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :scheme: http
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :path: /
41                                      | == Literal indexed ==
                                        |   Indexed name (idx = 1)
                                        |     :authority
0f                                      |   Literal value (len = 15)
7777 772e 6578 616d 706c 652e 636f 6d   | www.example.com
                                        | -> :authority: www.example\
                                        |   .com
```

Dynamic Table (after decoding):

```
[  1] (s =  57) :authority: www.example.com
      Table size:  57
```

Decoded header list:

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

C.3.2. Second Request

Header list to encode:

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

Hex dump of encoded data:

```
8286 84be 5808 6e6f 2d63 6163 6865      | ....X.no-cache
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :scheme: http
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :path: /
be                                      | == Indexed - Add ==
                                        |   idx = 62
                                        | -> :authority: www.example\
                                        |   .com
58                                      | == Literal indexed ==
                                        |   Indexed name (idx = 24)
                                        |     cache-control
08                                      |   Literal value (len = 8)
6e6f 2d63 6163 6865                     | no-cache
                                        | -> cache-control: no-cache
```

Dynamic Table (after decoding):

```
[  1] (s =  53) cache-control: no-cache
[  2] (s =  57) :authority: www.example.com
      Table size: 110
```

Decoded header list:

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

C.3.3. Third Request

Header list to encode:

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

Hex dump of encoded data:

```
8287 85bf 400a 6375 7374 6f6d 2d6b 6579 | ....@.custom-key
0c63 7573 746f 6d2d 7661 6c75 65        | .custom-value
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
87                                      | == Indexed - Add ==
                                        |   idx = 7
                                        | -> :scheme: https
85                                      | == Indexed - Add ==
                                        |   idx = 5
                                        | -> :path: /index.html
bf                                      | == Indexed - Add ==
                                        |   idx = 63
                                        | -> :authority: www.example\
                                        |   .com
40                                      | == 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
```

Dynamic Table (after decoding):

```
[  1] (s =  54) custom-key: custom-value
[  2] (s =  53) cache-control: no-cache
[  3] (s =  57) :authority: www.example.com
      Table size: 164
```

Decoded header list:

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

C.4. Request Examples with Huffman Coding

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

C.4.1. First Request

Header list to encode:

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

Hex dump of encoded data:

```
8286 8441 8cf1 e3c2 e5f2 3a6b a0ab 90f4 | ...A......:k....
ff                                      | .
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :scheme: http
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :path: /
41                                      | == Literal indexed ==
                                        |   Indexed name (idx = 1)
                                        |     :authority
8c                                      |   Literal value (len = 12)
                                        |     Huffman encoded:
f1e3 c2e5 f23a 6ba0 ab90 f4ff           | .....:k.....
                                        |     Decoded:
                                        | www.example.com
                                        | -> :authority: www.example\
                                        |   .com
```

Dynamic Table (after decoding):

```
[  1] (s =  57) :authority: www.example.com
      Table size:  57
```

Decoded header list:

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

C.4.2. Second Request

Header list to encode:

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

Hex dump of encoded data:

```
8286 84be 5886 a8eb 1064 9cbf           | ....X....d..
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :scheme: http
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :path: /
be                                      | == Indexed - Add ==
                                        |   idx = 62
                                        | -> :authority: www.example\
                                        |   .com
58                                      | == Literal indexed ==
                                        |   Indexed name (idx = 24)
                                        |     cache-control
86                                      |   Literal value (len = 6)
                                        |     Huffman encoded:
a8eb 1064 9cbf                          | ...d..
                                        |     Decoded:
                                        | no-cache
                                        | -> cache-control: no-cache
```

Dynamic Table (after decoding):

```
[  1] (s =  53) cache-control: no-cache
[  2] (s =  57) :authority: www.example.com
      Table size: 110
```

Decoded header list:

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

C.4.3. Third Request

Header list to encode:

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

Hex dump of encoded data:

```
8287 85bf 4088 25a8 49e9 5ba9 7d7f 8925 | ....@.%.I.[.}..%
a849 e95b b8e8 b4bf                     | .I.[....
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
87                                      | == Indexed - Add ==
                                        |   idx = 7
                                        | -> :scheme: https
85                                      | == Indexed - Add ==
                                        |   idx = 5
                                        | -> :path: /index.html
bf                                      | == Indexed - Add ==
                                        |   idx = 63
                                        | -> :authority: www.example\
                                        |   .com
40                                      | == Literal indexed ==
88                                      |   Literal name (len = 8)
                                        |     Huffman encoded:
25a8 49e9 5ba9 7d7f                     | %.I.[.}.
                                        |     Decoded:
                                        | custom-key
89                                      |   Literal value (len = 9)
                                        |     Huffman encoded:
25a8 49e9 5bb8 e8b4 bf                  | %.I.[....
                                        |     Decoded:
                                        | custom-value
                                        | -> custom-key: custom-valu\
                                        |   e
```

Dynamic Table (after decoding):

```
[  1] (s =  54) custom-key: custom-value
[  2] (s =  53) cache-control: no-cache
[  3] (s =  57) :authority: www.example.com
      Table size: 164
```

Decoded header list:

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

C.5. Response Examples without Huffman Coding

This section shows several consecutive header lists, corresponding to HTTP responses, on the same connection. The HTTP/2 setting parameter SETTINGS_HEADER_TABLE_SIZE is set to the value of 256 octets, causing some evictions to occur.

C.5.1. First Response

Header list to encode:

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

Hex dump of encoded data:

```
4803 3330 3258 0770 7269 7661 7465 611d | H.302X.privatea.
4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
2032 303a 3133 3a32 3120 474d 546e 1768 |  20:13:21 GMTn.h
7474 7073 3a2f 2f77 7777 2e65 7861 6d70 | ttps://www.examp
6c65 2e63 6f6d                          | le.com
```

Decoding process:

```
48                                      | == Literal indexed ==
                                        |   Indexed name (idx = 8)
                                        |     :status
03                                      |   Literal value (len = 3)
3330 32                                 | 302
                                        | -> :status: 302
58                                      | == Literal indexed ==
                                        |   Indexed name (idx = 24)
                                        |     cache-control
07                                      |   Literal value (len = 7)
7072 6976 6174 65                       | private
                                        | -> cache-control: private
61                                      | == Literal indexed ==
                                        |   Indexed name (idx = 33)
                                        |     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
6e                                      | == Literal indexed ==
                                        |   Indexed name (idx = 46)
                                        |     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
```

Dynamic 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 list:

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

C.5.2. Second Response

The (":status", "302") header field is evicted from the dynamic table to free space to allow adding the (":status", "307") header field.

Header list to encode:

```
:status: 307
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Hex dump of encoded data:

```
4803 3330 37c1 c0bf                     | H.307...
```

Decoding process:

```
48                                      | == Literal indexed ==
                                        |   Indexed name (idx = 8)
                                        |     :status
03                                      |   Literal value (len = 3)
3330 37                                 | 307
                                        | - evict: :status: 302
                                        | -> :status: 307
c1                                      | == Indexed - Add ==
                                        |   idx = 65
                                        | -> cache-control: private
c0                                      | == Indexed - Add ==
                                        |   idx = 64
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:21 GMT
bf                                      | == Indexed - Add ==
                                        |   idx = 63
                                        | -> location: https://www.e\
                                        |   xample.com
```

Dynamic Table (after decoding):

```
[  1] (s =  42) :status: 307
[  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 list:

```
:status: 307
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

C.5.3. Third Response

Several header fields are evicted from the dynamic table during the processing of this header list.

Header list 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
```

Hex dump of encoded data:

```
88c1 611d 4d6f 6e2c 2032 3120 4f63 7420 | ..a.Mon, 21 Oct
3230 3133 2032 303a 3133 3a32 3220 474d | 2013 20:13:22 GM
54c0 5a04 677a 6970 7738 666f 6f3d 4153 | T.Z.gzipw8foo=AS
444a 4b48 514b 425a 584f 5157 454f 5049 | DJKHQKBZXOQWEOPI
5541 5851 5745 4f49 553b 206d 6178 2d61 | UAXQWEOIU; max-a
6765 3d33 3630 303b 2076 6572 7369 6f6e | ge=3600; version
3d31                                    | =1
```

Decoding process:

```
88                                      | == Indexed - Add ==
                                        |   idx = 8
                                        | -> :status: 200
c1                                      | == Indexed - Add ==
                                        |   idx = 65
                                        | -> cache-control: private
61                                      | == Literal indexed ==
                                        |   Indexed name (idx = 33)
                                        |     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
c0                                      | == Indexed - Add ==
                                        |   idx = 64
                                        | -> location: https://www.e\
                                        |   xample.com
5a                                      | == Literal indexed ==
                                        |   Indexed name (idx = 26)
                                        |     content-encoding
04                                      |   Literal value (len = 4)
677a 6970                               | gzip
                                        | - evict: date: Mon, 21 Oct\
                                        |    2013 20:13:21 GMT
                                        | -> content-encoding: gzip
77                                      | == Literal indexed ==
                                        |   Indexed name (idx = 55)
                                        |     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: 307
                                        | -> set-cookie: foo=ASDJKHQ\
                                        |   KBZXOQWEOPIUAXQWEOIU; ma\
                                        |   x-age=3600; version=1
```

Dynamic 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 list:

```
: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
```

C.6. Response Examples with Huffman Coding

This section shows the same examples as the previous section, but using Huffman encoding for the literal values. The HTTP/2 setting parameter SETTINGS_HEADER_TABLE_SIZE is set to the value of 256 octets, causing some evictions to occur. The eviction mechanism uses the length of the decoded literal values, so the same evictions occurs as in the previous section.

C.6.1. First Response

Header list to encode:

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

Hex dump of encoded data:

```
4882 6402 5885 aec3 771a 4b61 96d0 7abe | H.d.X...w.Ka..z.
9410 54d4 44a8 2005 9504 0b81 66e0 82a6 | ..T.D. .....f...
2d1b ff6e 919d 29ad 1718 63c7 8f0b 97c8 | -..n..)...c.....
e9ae 82ae 43d3                          | ....C.
```

Decoding process:

```
48                                      | == Literal indexed ==
                                        |   Indexed name (idx = 8)
                                        |     :status
82                                      |   Literal value (len = 2)
                                        |     Huffman encoded:
6402                                    | d.
                                        |     Decoded:
                                        | 302
                                        | -> :status: 302
58                                      | == Literal indexed ==
                                        |   Indexed name (idx = 24)
                                        |     cache-control
85                                      |   Literal value (len = 5)
                                        |     Huffman encoded:
aec3 771a 4b                            | ..w.K
                                        |     Decoded:
                                        | private
                                        | -> cache-control: private
61                                      | == Literal indexed ==
                                        |   Indexed name (idx = 33)
                                        |     date
96                                      |   Literal value (len = 22)
                                        |     Huffman encoded:
d07a be94 1054 d444 a820 0595 040b 8166 | .z...T.D. .....f
e082 a62d 1bff                          | ...-..
                                        |     Decoded:
                                        | Mon, 21 Oct 2013 20:13:21 \
                                        | GMT
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:21 GMT
6e                                      | == Literal indexed ==
                                        |   Indexed name (idx = 46)
                                        |     location
91                                      |   Literal value (len = 17)
                                        |     Huffman encoded:
9d29 ad17 1863 c78f 0b97 c8e9 ae82 ae43 | .)...c.........C
d3                                      | .
                                        |     Decoded:
                                        | https://www.example.com
                                        | -> location: https://www.e\
                                        |   xample.com
```

Dynamic 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 list:

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

C.6.2. Second Response

The (":status", "302") header field is evicted from the dynamic table to free space to allow adding the (":status", "307") header field.

Header list to encode:

```
:status: 307
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Hex dump of encoded data:

```
4883 640e ffc1 c0bf                     | H.d.....
```

Decoding process:

```
48                                      | == Literal indexed ==
                                        |   Indexed name (idx = 8)
                                        |     :status
83                                      |   Literal value (len = 3)
                                        |     Huffman encoded:
640e ff                                 | d..
                                        |     Decoded:
                                        | 307
                                        | - evict: :status: 302
                                        | -> :status: 307
c1                                      | == Indexed - Add ==
                                        |   idx = 65
                                        | -> cache-control: private
c0                                      | == Indexed - Add ==
                                        |   idx = 64
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:21 GMT
bf                                      | == Indexed - Add ==
                                        |   idx = 63
                                        | -> location: https://www.e\
                                        |   xample.com
```

Dynamic Table (after decoding):

```
[  1] (s =  42) :status: 307
[  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 list:

```
:status: 307
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

C.6.3. Third Response

Several header fields are evicted from the dynamic table during the processing of this header list.

Header list 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
```

Hex dump of encoded data:

```
88c1 6196 d07a be94 1054 d444 a820 0595 | ..a..z...T.D. ..
040b 8166 e084 a62d 1bff c05a 839b d9ab | ...f...-...Z....
77ad 94e7 821d d7f2 e6c7 b335 dfdf cd5b | w..........5...[
3960 d5af 2708 7f36 72c1 ab27 0fb5 291f | 9`..'..6r..'..).
9587 3160 65c0 03ed 4ee5 b106 3d50 07   | ..1`e...N...=P.
```

Decoding process:

```
88                                      | == Indexed - Add ==
                                        |   idx = 8
                                        | -> :status: 200
c1                                      | == Indexed - Add ==
                                        |   idx = 65
                                        | -> cache-control: private
61                                      | == Literal indexed ==
                                        |   Indexed name (idx = 33)
                                        |     date
96                                      |   Literal value (len = 22)
                                        |     Huffman encoded:
d07a be94 1054 d444 a820 0595 040b 8166 | .z...T.D. .....f
e084 a62d 1bff                          | ...-..
                                        |     Decoded:
                                        | Mon, 21 Oct 2013 20:13:22 \
                                        | GMT
                                        | - evict: cache-control: pr\
                                        |   ivate
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:22 GMT
c0                                      | == Indexed - Add ==
                                        |   idx = 64
                                        | -> location: https://www.e\
                                        |   xample.com
5a                                      | == Literal indexed ==
                                        |   Indexed name (idx = 26)
                                        |     content-encoding
83                                      |   Literal value (len = 3)
                                        |     Huffman encoded:
```

```
9bd9 ab                                 | ...
                                        |     Decoded:
                                        | gzip
                                        | - evict: date: Mon, 21 Oct\
                                        |    2013 20:13:21 GMT
                                        | -> content-encoding: gzip
77                                      | == Literal indexed ==
                                        |   Indexed name (idx = 55)
                                        |     set-cookie
ad                                      |   Literal value (len = 45)
                                        |     Huffman encoded:
94e7 821d d7f2 e6c7 b335 dfdf cd5b 3960 | .........5...[9`
d5af 2708 7f36 72c1 ab27 0fb5 291f 9587 | ..'..6r..'..)...
3160 65c0 03ed 4ee5 b106 3d50 07        | 1`e...N...=P.
                                        |     Decoded:
                                        | foo=ASDJKHQKBZXOQWEOPIUAXQ\
                                        | WEOIU; max-age=3600; versi\
                                        | on=1
                                        | - evict: location: https:/\
                                        |   /www.example.com
                                        | - evict: :status: 307
                                        | -> set-cookie: foo=ASDJKHQ\
                                        |   KBZXOQWEOPIUAXQWEOIU; ma\
                                        |   x-age=3600; version=1

Dynamic 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 list:

: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
```

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

D.1. Since draft-ietf-httpbis-header-compression-09

- o Renamed header table to dynamic table.
- o Updated integer representation.
- o Editorial corrections.

D.2. Since draft-ietf-httpbis-header-compression-08

- o Removed the reference set.
- o Removed header emission.
- o Explicit handling of several SETTINGS_HEADER_TABLE_SIZE parameter changes.
- o Changed header set to header list, and forced ordering.
- o Updated examples.
- o Exchanged header and static table positions.

D.3. Since draft-ietf-httpbis-header-compression-07

- o Removed old text on index value of 0.
- o Added clarification for signalling of maximum table size after a SETTINGS_HEADER_TABLE_SIZE update.
- o Rewrote security considerations.
- o Many editorial clarifications or improvements.
- o Added convention section.
- o Reworked document's outline.
- o Updated static table. Entry 16 has now "gzip, deflate" for value.
- o Updated Huffman table, using data set provided by Google.

D.4. Since draft-ietf-httpbis-header-compression-06

- o Updated format to include literal headers that must never be compressed.
- o Updated security considerations.

- Moved integer encoding examples to the appendix.
- Updated Huffman table.
- Updated static header table (adding and removing status values).
- Updated examples.

D.5. Since draft-ietf-httpbis-header-compression-05

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

D.6. 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).

D.7. 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

D.8. Since draft-ietf-httpbis-header-compression-02

- Corrected error in integer encoding pseudocode.

D.9. 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.

D.10. 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.

Authors' Addresses

Roberto Peon
Google, Inc

EMail: fenix@google.com

Herve Ruellan
Canon CRF

EMail: herve.ruellan@crf.canon.fr