HTTPbis Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 19, 2014

R. Peon
Google, Inc
H. Ruellan
Canon CRF

June 17, 2014

HPACK - Header Compression for HTTP/2
draft-ietf-httpbis-header-compression-08

Abstract

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

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <https://lists.w3.org/Archives/Public/ietf-http-wg/>.

Working Group information can be found at <http://tools.ietf.org/wg/httpbis/>; that specific to HTTP/2 are at <http://http2.github.io/>.

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

Copyright Notice

Table of Contents

1. Introduction

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

2. HPACK Overview

In HTTP/1.1 (see [RFC7230]), header fields are encoded without any form of compression. As web pages have grown to include dozens to hundreds of requests, the redundant header fields in these requests now measurably increase latency and unnecessarily consume bandwidth (see [SPDY-DESC-1] and [SPDY-DESC-2]).

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

This document describes HPACK, a new compressor for header fields which eliminates redundant header fields, limits vulnerability to known security attacks, and which has a bounded memory requirement for use in constrained environments.

2.1. Outline

The HTTP header field encoding defined in this document is based on a header table that maps name-value pairs to index values. The header table is incrementally updated as new values are encoded or decoded.

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

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

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

A subset of the header fields that are encoded as references to the header table are maintained in a reference set that is used as the initial set of header fields for the next encoding.

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

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

2.2. 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. Hexadecimal literals are prefixed with "0x" to distinguish them from decimal literals.

2.3. Terminology

This document uses the following terms:

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

Header Table: The header table (see Section 3.2) is a component used to associate stored header fields to index values.

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

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

Reference Set: The reference set (see Section 3.3) is a component containing an unordered set of references to entries in the header table. It doesn't contain duplicate references. The reference set is used for the differential encoding of a new header set.

Header Field Representation: A header field can be represented in encoded form either as a literal or as an index (see Section 3.4).

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

Header Field Emission: When decoding a set of header field representations, some operations emit a header field (see Section 3.5). Emitted header fields are added to the output header set and cannot be removed.

3. Decoding 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.

3.1. Encoding and Decoding Contexts

HPACK requires that a decoder maintains both a header table and a reference set. No other state information is needed to decode messages. An encoder that wishes to reference entries in the header table, reference set, or static table needs to maintain a copy of the information a decoder holds.

When used for bidirectional communication, such as in HTTP, the encoding and decoding contexts maintained by an endpoint are completely independent. Header fields are encoded without any reference to the local decoding state; and header fields are decoded without reference to the encoding state.

Each endpoint maintains a header table and a reference set in order to decode header blocks, and optionally a copy of the information maintained by their peer.

3.2. Header Table

A header table consists of a list of header fields maintained in first-in, first-out order. The first and newest entry in a header table is always at index 1, and the oldest entry of a header table is at the index corresponding to the number of entries in the header table.

The header table is initially empty.

The header 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 header table and as such can control how much memory is used by the header table. To limit the memory requirements of the decoder, the header table size is strictly bounded (see Section 5.1).

The header table is updated during the processing of a set of header field representations (see Section 4.1).

3.3. Reference Set

A reference set is an unordered set of references to entries of the header table. It never contains duplicate references.

The reference set is initially empty.

The reference set is updated during the processing of a set of header field representations (see Section 4.1).

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

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

To limit the memory requirements on the decoder side for handling the reference set, only entries within the header table can be contained in the reference set. To still allow entries from the static table to take advantage of the differential encoding, when a header field is represented as a reference to an entry of the static table, this entry is inserted into the header table (see Section 4.1).

3.4. Header Field Representation

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

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

Three different literal representations are provided:

- A literal representation that does not add the header field to the header table (see Section 7.2.2).

- A literal representation that does not add the header field to the header table, with the additional stipulation that this header field always use a literal representation, in particular when re-encoded by an intermediary (see Section 7.2.3).

- A literal representation that adds the header field as a new entry at the beginning of the header table (see Section 7.2.1).

An indexed representation defines a header field as a reference to an entry in either the header table or the static table (see Section 7.1).

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

Indices between one higher than the length of the header table represent indexes into the static table. The length of the header table is subtracted to find the index into the static table.

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

An indexed representation using an entry of the static table induces a copy of this entry into the header table (see Section 4.1) for bounding memory requirements on the decoder side (see Section 5.1). For this reason, the header table is accessed more frequently than the static table and has the lower indices.

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

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

Index Address Space

3.5. Header Field Emission

A decoder processes an encoded header block sequentially. As different instructions are processed, some might specify that a header field is emitted.

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

An emitted header field can be safely passed to the upper processing layer as part of the current header set. The decoder can pass emitted header fields to the upper processing layer in any order.

By emitting header fields instead of emitting header sets, a decoder can be implemented with minimal memory commitment in addition to the header table and the reference set. The management of memory for handling very large sets of header fields can therefore be deferred to the upper processing layers.

4. Header Block Decoding

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

4.1. Header Field Representation Processing

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 7.

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

o The entry is removed from the reference set.

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

o If referencing an element of the static table:

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

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

      * A reference to this new header table entry is added to the reference set, unless this new entry didn't fit in the header table.

   o If referencing an element of the header table:

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

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

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

o The header field is emitted.

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

o The header field is emitted.

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

o A reference to the new entry is added to the reference set, unless this new entry didn't fit in the header table.

4.2. Reference Set Emission

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

After the emission of these remaining header fields, the header set is complete.

5. Header Table Management

5.1. Maximum Table Size

To limit the memory requirements on the decoder side, the mutable structures used in an encoding context are constrained in size. These mutable structures are the header table and the reference set.

The size of the header table is bounded by a maximum size defined by the decoder. The size of the header table MUST always be lower than or equal to this maximum size.

The reference set can only contain references to entries of the header table, and can't contain references to entries of the static table. In addition, it can't contain duplicate references. Therefore, its maximum size is bounded by the size of the header table.

By default, the maximum size of the header table is equal to the value of the HTTP/2 setting SETTINGS_HEADER_TABLE_SIZE defined by the decoder (see Section 6.5.2 of [HTTP2]). The encoder can change this maximum size (see Section 7.3), but it MUST stay lower than or equal to the value of SETTINGS_HEADER_TABLE_SIZE.

After applying an updated value of the HTTP/2 setting SETTINGS_HEADER_TABLE_SIZE that changes the maximum size of the header table used by the encoder, the encoder MUST signal this change via an encoding context update (see Section 7.3). This encoding context update MUST occur at the beginning of the first header block following the SETTINGS frame sent to acknowledge the application of the updated settings.

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

The size of an entry is the sum of its name's length in octets (as defined in Section 6.2), its value's length in octets (Section 6.2), plus 32.

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

The additional 32 octets account for 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.

5.2. Entry Eviction When Header Table Size Changes

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

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

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

5.3. Entry Eviction when Adding New Entries

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

If the representation of the added entry references the name of an entry in the header 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.

6. Primitive Type Representations

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

6.1. Integer representation

Integers are used to represent name indexes, pair indexes or string lengths. To allow for optimized processing, an integer representation always finishes at the end of an octet.

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

The N-bit prefix allows filling the current octet. If the value is small enough (strictly less than 2^N-1), it is encoded within the N-bit prefix. Otherwise all the bits of the prefix are set to 1 and the value is encoded using an unsigned variable length integer representation (see <http://en.wikipedia.org/wiki/Variable-length_quantity>). N is always between 1 and 8 bits. An integer starting at an octet-boundary will have an 8-bit prefix.

The algorithm to represent an integer I is as follows:

```
if I < 2^N - 1, encode I on N bits
else
    encode (2^N - 1) on N bits
    I = I - (2^N - 1)
    while I >= 128
         encode (I % 128 + 128) on 8 bits
         I = I / 128
    encode I on 8 bits
```

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

```
decode I from the next N bits
if I < 2^N - 1, return I
else
    M = 0
    repeat
        B = next octet
        I = I + (B & 127) * 2^M
        M = M + 7
    while B & 128 == 128
    return I
```

Examples illustrating the encoding of integers are available in Appendix D.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.

6.2. 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 [HUFFMAN].

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

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

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 6.1).

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 C (see examples in Request Examples with Huffman Coding (Appendix D.4) and in Response Examples with Huffman Coding (Appendix D.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 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.

7. Binary Format

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

7.1. Indexed Header Field Representation

An indexed header field representation identifies an entry in either the header table or the static table.

An indexed header field representation can either causes a header field to be emitted or to be removed from the reference set, as described in Section 4.1.

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

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.

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

7.2. 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 header table or the static table.

A literal representation always result in the emission of a header field when decoded.

7.2.1. Literal Header Field with Incremental Indexing

A literal header field with incremental indexing representation causes the emission of a header field, adding it as a new entry to the header table.

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

Literal Header Field 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)  |
+-------------------------------+
```

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 header table or the static 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 6.1). This value is always non-zero.

Otherwise, the header field name is represented as a literal. A value 0 is used in place of the 6-bit index, followed by the header field name (see Section 6.2).

Either form of header field name representation is followed by the header field value represented as a literal string as described in Section 6.2.

7.2.2. Literal Header Field without Indexing

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

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

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)  |
+-------------------------------+
```

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 header table or the static 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 6.1). This value is always non-zero.

Otherwise, the header field name is represented as a literal. A value 0 is used in place of the 4-bit index, followed by the header field name (see Section 6.2).

Either form of header field name representation is followed by the header field value represented as a literal string as described in Section 6.2.

7.2.3. Literal Header Field Never Indexed

A literal header field never indexed representation causes the emission of a header field without altering the header 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)  |
+-------------------------------+
```

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)  |
+-------------------------------+
```

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 8.1 for more details).

The encoding of the representation is identical to the literal header field without indexing (see Section 7.2.2).

7.3. Encoding Context Update

An encoding context update causes the immediate application of a change to the encoding context.

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

Context Update

An encoding context update starts with the '001' 3-bit pattern.

It is followed by a flag specifying the type of the change, and by any data necessary to describe the change itself.

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

Reference Set Emptying

The flag bit being set to '1' signals that the reference set is emptied. The remaining bits MUST be set to '0', non-zero values MUST be treated as a decoding error.

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

Maximum Header Table Size Change

The flag bit being set to '0' signals that a change to the maximum size of the header table. This new maximum size MUST be lower than or equal to the maximum set by the decoder. That is, the value of the HTTP/2 setting SETTINGS_HEADER_TABLE_SIZE, defined in Section 6.5.2 of [HTTP2].

The new maximum size is encoded as an integer with a 4-bit prefix (see Section 6.1).

Reducing the maximum size of the header table causes entries to be evicted (see Section 5.2).

8. 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.

8.1. Probing Header 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 that has the following capabilities: to define header fields to be encoded and transmitted; and to observe the length of those fields once they are encoded. This allows an attacker to adaptively modify requests in order to confirm guesses about the header 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 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] demonstrated the existence of these general attacker capabilities. The specific attack exploited the fact that [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 constant time attack.

8.1.1. Applicability to HPACK and HTTP

HPACK mitigates but does not completely prevent attacks modelled on [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 header 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.

8.1.2. 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 header table probing.

An ideal solution segregates access to the header 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 an 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 header 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 header 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.

8.1.3. 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 (Section 7.2.3) 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 as an indexed representation.

8.2. Static Huffman Encoding

There is currently no known threat taking advantage of the use of 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]).

8.3. 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 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 header table, plus a small allowance for overhead.

A decoder can limit the amount of state memory used by setting an appropriate value for the setting SETTINGS_HEADER_TABLE_SIZE. An encoder can limit the amount of state memory it uses by signaling lower header table size than the decoder allows (see Section 7.3).

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 set of header fields. Note however that it might be necessary for an application to retain a complete header set for other reasons; even though HPACK does not force this to occur, application constraints might make this necessary.

8.4. 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 6.1). In the same way, it has to set a limit to the length it accepts for string literals (see Section 6.2).

9. Acknowledgements

This document 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).

10. References

10.1. Normative References

[HTTP2] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-13 (work in progress), June 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.

10.2. 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-DESC-1] Belshe, M., "IETF83: SPDY and What to Consider for HTTP/2.0 ", March 2012, <https://www.ietf.org/proceedings/83/slides/slides-83-httpbis-3>.

[SPDY-DESC-2] McManus, P., "SPDY: What I Like About You", September 2011, <https://bitsup.blogspot.com/2011/09/spdy-what-i-like-about-you.html>.

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

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

A.1. Since draft-ietf-httpbis-header-compression-07

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

A.2. Since draft-ietf-httpbis-header-compression-06

- Updated format to include literal headers that must never be compressed.
- Updated security considerations.
- Moved integer encoding examples to the appendix.
- Updated Huffman table.
- Updated static header table (adding and removing status values).
- Updated examples.

A.3. Since draft-ietf-httpbis-header-compression-05

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

A.4. Since draft-ietf-httpbis-header-compression-04

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

A.5. Since draft-ietf-httpbis-header-compression-03

- o A large number of editorial changes; changed the description of evicting/adding new entries.
- o Removed substitution indexing
- o Changed 'initial headers' to 'static headers', as per issue #258
- o Merged 'request' and 'response' static headers, as per issue #259
- o Changed text to indicate that new headers are added at index 0 and expire from the largest index, as per issue #233

A.6. Since draft-ietf-httpbis-header-compression-02

- o Corrected error in integer encoding pseudocode.

A.7. Since draft-ietf-httpbis-header-compression-01

- o Refactored of Header Encoding Section: split definitions and processing rule.
- o Backward incompatible change: Updated reference set management as per issue #214. This changes how the interaction between the reference set and eviction works. This also changes the working of the reference set in some specific cases.
- o Backward incompatible change: modified initial header list, as per issue #188.
- o Added example of 32 octets entry structure (issue #191).

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

A.8. Since draft-ietf-httpbis-header-compression-00

Fixed bug/omission in integer representation algorithm.

Changed the document title.

Header matching text rewritten.

Changed the definition of header emission.

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

Removed "specific use cases" section

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

Added descriptions of opcodes

Removed security claims from introduction.

Appendix B. Static Table

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

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

Appendix C. Huffman Code

The following Huffman code is used when encoding string literals with a Huffman coding (see Section 6.2).

This Huffman code was generated from statistics obtained on a large sample of HTTP headers. It is a canonical Huffman code [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	ffff1	[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 D. Examples

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

D.1. Integer Representation Examples

This section shows the representation of integer values in details (see Section 6.1).

D.1.1. Example 1: Encoding 10 Using a 5-bit Prefix

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

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

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

D.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
+---+---+---+---+---+---+---+---+
```

D.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
+---+---+---+---+---+---+---+---+
```

D.2. Header Field Representation Examples

This section shows several independent representation examples.

D.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 header table.

Header set to encode:

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

Reference set: empty.

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
```

```
Header Table (after decoding):

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


Decoded header set:

custom-key: custom-header
```

D.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 header table.

```
Header set to encode:

:path: /sample/path


Reference set: empty.

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


Header table (after decoding): empty.

Decoded header set:

:path: /sample/path
```

D.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 header table, and must use the same representation if re-encoded by an intermediary.

Header set to encode:

```
password: secret
```

Reference set: empty.

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
```

Header table (after decoding): empty.

Decoded header set:

```
password: secret
```

D.2.4. Indexed Header Field

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

Header set to encode:

```
:method: GET
```

Reference set: empty.

Hex dump of encoded data:

```
82                                      | .
```

Decoding process:

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

Header Table (after decoding):

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

Decoded header set:

```
:method: GET
```

D.2.5. Indexed Header Field from Static Table

The header field representation uses an indexed header field, from the static table. In this example, the HTTP/2 setting SETTINGS_HEADER_TABLE_SIZE is set to 0, therefore, the entry is not copied into the header table.

Header set to encode:

```
:method: GET
```

Reference set: empty.

Hex dump of encoded data:

```
82                                      | .
```

Decoding process:

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

Header table (after decoding): empty.

Decoded header set:

:method: GET

D.3. Request Examples without Huffman Coding

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

D.3.1. First Request

Header set to encode:

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

Reference set: empty.

Hex dump of encoded data:

```
8287 8644 0f77 7777 2e65 7861 6d70 6c65 | ...D.www.example
2e63 6f6d                               | .com
```

Decoding process:

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

Header Table (after decoding):

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

Decoded header set:

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

D.3.2. Second Request

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

Header set to encode:

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

Reference set:

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

Hex dump of encoded data:

```
5c08 6e6f 2d63 6163 6865                | \.no-cache
```

Decoding process:

```
5c                                      | == Literal indexed ==
                                        |   Indexed name (idx = 28)
                                        |     cache-control
08                                      |   Literal value (len = 8)
6e6f 2d63 6163 6865                     | no-cache
                                        | -> cache-control: no-cache
```

Header Table (after decoding):

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

Decoded header set:

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

D.3.3. Third Request

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

Header set to encode:

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

Reference set:

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

Hex dump of encoded data:

```
3085 8c8b 8440 0a63 7573 746f 6d2d 6b65 | 0....@.custom-ke
790c 6375 7374 6f6d 2d76 616c 7565      | y.custom-value
```

Decoding process:

```
30                                      | == Empty reference set ==
                                        |   idx = 0
                                        |   flag = 1
85                                      | == Indexed - Add ==
                                        |   idx = 5
                                        | -> :method: GET
8c                                      | == Indexed - Add ==
                                        |   idx = 12
                                        | -> :scheme: https
8b                                      | == Indexed - Add ==
                                        |   idx = 11
                                        | -> :path: /index.html
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :authority: www.example\
                                        |   .com
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
```

Header Table (after decoding):

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

Decoded header set:

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

D.4. Request Examples with Huffman Coding

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

D.4.1. First Request

Header set to encode:

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

Reference set: empty.

Hex dump of encoded data:

```
8287 8644 8cf1 e3c2 e5f2 3a6b a0ab 90f4 | ...D......:k....
ff                                      | .
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
87                                      | == Indexed - Add ==
                                        |   idx = 7
                                        | -> :scheme: http
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :path: /
44                                      | == Literal indexed ==
                                        |   Indexed name (idx = 4)
                                        |     :authority
8c                                      |   Literal value (len = 15)
                                        |     Huffman encoded:
f1e3 c2e5 f23a 6ba0 ab90 f4ff           | .....:k.....
                                        |     Decoded:
                                        | www.example.com
                                        | -> :authority: www.example\
                                        |   .com
```

Header Table (after decoding):

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

Decoded header set:

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

D.4.2. Second Request

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

Header set to encode:

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

Reference set:

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

Hex dump of encoded data:

```
5c86 a8eb 1064 9cbf                     | \....d..
```

Decoding process:

```
5c                                      | == Literal indexed ==
                                        |   Indexed name (idx = 28)
                                        |     cache-control
86                                      |   Literal value (len = 8)
                                        |     Huffman encoded:
a8eb 1064 9cbf                          | ...d..
                                        |     Decoded:
                                        | no-cache
                                        | -> cache-control: no-cache
```

Header Table (after decoding):

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

Decoded header set:

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

D.4.3. Third Request

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

Header set to encode:

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

Reference set:

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

Hex dump of encoded data:

```
3085 8c8b 8440 8825 a849 e95b a97d 7f89 | 0....@.%.I.[.}..
25a8 49e9 5bb8 e8b4 bf                  | %.I.[....
```

Decoding process:

```
30                                      | == Empty reference set ==
                                        |   idx = 0
                                        |   flag = 1
85                                      | == Indexed - Add ==
                                        |   idx = 5
                                        | -> :method: GET
8c                                      | == Indexed - Add ==
                                        |   idx = 12
                                        | -> :scheme: https
8b                                      | == Indexed - Add ==
                                        |   idx = 11
                                        | -> :path: /index.html
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :authority: www.example\
                                        |   .com
40                                      | == Literal indexed ==
88                                      |   Literal name (len = 10)
                                        |     Huffman encoded:
25a8 49e9 5ba9 7d7f                     | %.I.[.}.
                                        |     Decoded:
                                        | custom-key
89                                      |   Literal value (len = 12)
                                        |     Huffman encoded:
25a8 49e9 5bb8 e8b4 bf                  | %.I.[....
                                        |     Decoded:
                                        | custom-value
                                        | -> custom-key: custom-valu\
                                        |   e
```

Header Table (after decoding):

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

Decoded header set:

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

D.5. Response Examples without Huffman Coding

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

D.5.1. First Response

Header set to encode:

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

Reference set: empty.

Hex dump of encoded data:

```
4803 3330 3259 0770 7269 7661 7465 631d | H.302Y.privatec.
4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
2032 303a 3133 3a32 3120 474d 5471 1768 |  20:13:21 GMTq.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
59                                      | == Literal indexed ==
                                        |   Indexed name (idx = 25)
                                        |     cache-control
07                                      |   Literal value (len = 7)
7072 6976 6174 65                       | private
                                        | -> cache-control: private
63                                      | == Literal indexed ==
                                        |   Indexed name (idx = 35)
                                        |     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
71                                      | == Literal indexed ==
                                        |   Indexed name (idx = 49)
                                        |     location
17                                      |   Literal value (len = 23)
6874 7470 733a 2f2f 7777 772e 6578 616d | https://www.exam
706c 652e 636f 6d                       | ple.com
                                        | -> location: https://www.e\
                                        |   xample.com
```

Header Table (after decoding):

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

Decoded header set:

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

D.5.2. Second Response

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

Header set to encode:

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

Reference set:

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

Hex dump of encoded data:

```
8c                                      | .
```

Decoding process:

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

Header Table (after decoding):

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

Decoded header set:

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

D.5.3. Third Response

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

Header set to encode:

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

Reference set:

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

Hex dump of encoded data:

```
8484 431d 4d6f 6e2c 2032 3120 4f63 7420 | ..C.Mon, 21 Oct
3230 3133 2032 303a 3133 3a32 3220 474d | 2013 20:13:22 GM
545e 0467 7a69 7084 8483 837b 3866 6f6f | T^.gzip....{8foo
3d41 5344 4a4b 4851 4b42 5a58 4f51 5745 | =ASDJKHQKBZXOQWE
4f50 4955 4158 5157 454f 4955 3b20 6d61 | OPIUAXQWEOIU; ma
782d 6167 653d 3336 3030 3b20 7665 7273 | x-age=3600; vers
696f 6e3d 31                            | ion=1
```

Decoding process:

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

```
                                        | -> cache-control: private
   84                                   | == Indexed - Add ==
                                        |   idx = 4
                                        | -> cache-control: private
   43                                   | == Literal indexed ==
                                        |   Indexed name (idx = 3)
                                        |     date
   1d                                   |   Literal value (len = 29)
   4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
   2032 303a 3133 3a32 3220 474d 54     |  20:13:22 GMT
                                        | - evict: cache-control: pr\
                                        |   ivate
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:22 GMT
   5e                                   | == Literal indexed ==
                                        |   Indexed name (idx = 30)
                                        |     content-encoding
   04                                   |   Literal value (len = 4)
   677a 6970                            | gzip
                                        | - evict: date: Mon, 21 Oct\
                                        |    2013 20:13:21 GMT
                                        | -> content-encoding: gzip
   84                                   | == Indexed - Remove ==
                                        |   idx = 4
                                        | -> location: https://www.e\
                                        |   xample.com
   84                                   | == Indexed - Add ==
                                        |   idx = 4
                                        | -> location: https://www.e\
                                        |   xample.com
   83                                   | == Indexed - Remove ==
                                        |   idx = 3
                                        | -> :status: 200
   83                                   | == Indexed - Add ==
                                        |   idx = 3
                                        | -> :status: 200
   7b                                   | == Literal indexed ==
                                        |   Indexed name (idx = 59)
                                        |     set-cookie
   38                                   |   Literal value (len = 56)
   666f 6f3d 4153 444a 4b48 514b 425a 584f | foo=ASDJKHQKBZXO
   5157 454f 5049 5541 5851 5745 4f49 553b | QWEOPIUAXQWEOIU;
   206d 6178 2d61 6765 3d33 3630 303b 2076 |  max-age=3600; v
   6572 7369 6f6e 3d31                  | ersion=1
                                        | - evict: location: https:/\
                                        |   /www.example.com
                                        | - evict: :status: 200
                                        | -> set-cookie: foo=ASDJKHQ\
```

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

Header Table (after decoding):

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

Decoded header set:

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

D.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 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.

D.6.1. First Response

Header set to encode:

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

Reference set: empty.

Hex dump of encoded data:

```
4882 6402 5985 aec3 771a 4b63 96d0 7abe | H.d.Y...w.Kc..z.
9410 54d4 44a8 2005 9504 0b81 66e0 82a6 | ..T.D. .....f...
2d1b ff71 919d 29ad 1718 63c7 8f0b 97c8 | -..q..)...c.....
e9ae 82ae 43d3                          | ....C.
```

Decoding process:

```
48                                      | == Literal indexed ==
                                        |   Indexed name (idx = 8)
                                        |     :status
82                                      |   Literal value (len = 3)
                                        |     Huffman encoded:
6402                                    | d.
                                        |     Decoded:
                                        | 302
                                        | -> :status: 302
59                                      | == Literal indexed ==
                                        |   Indexed name (idx = 25)
                                        |     cache-control
85                                      |   Literal value (len = 7)
                                        |     Huffman encoded:
aec3 771a 4b                            | ..w.K
                                        |     Decoded:
                                        | private
                                        | -> cache-control: private
63                                      | == Literal indexed ==
                                        |   Indexed name (idx = 35)
                                        |     date
96                                      |   Literal value (len = 29)
                                        |     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
71                                      | == Literal indexed ==
                                        |   Indexed name (idx = 49)
                                        |     location
91                                      |   Literal value (len = 23)
                                        |     Huffman encoded:
9d29 ad17 1863 c78f 0b97 c8e9 ae82 ae43 | .)...c.........C
d3                                      | .
                                        |     Decoded:
                                        | https://www.example.com
                                        | -> location: https://www.e\
                                        |   xample.com
```

Header Table (after decoding):

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

Decoded header set:

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

D.6.2. Second Response

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

Header set to encode:

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

Reference set:

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

Hex dump of encoded data:

```
8c                                      | .
```

Decoding process:

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

Header Table (after decoding):

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

Decoded header set:

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

D.6.3. Third Response

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

Header set to encode:

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

Reference set:

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

Hex dump of encoded data:

```
8484 4396 d07a be94 1054 d444 a820 0595 | ..C..z...T.D. ..
040b 8166 e084 a62d 1bff 5e83 9bd9 ab84 | ...f...-..^.....
8483 837b ad94 e782 1dd7 f2e6 c7b3 35df | ...{.........5.
dfcd 5b39 60d5 af27 087f 3672 c1ab 270f | ..[9`..'..6r..'.
b529 1f95 8731 6065 c003 ed4e e5b1 063d | .)...1`e...N...=
5007                                    | P.
```

Decoding process:

```
84                                      | == Indexed - Remove ==
                                        |   idx = 4
                                        | -> cache-control: private
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> cache-control: private
43                                      | == Literal indexed ==
                                        |   Indexed name (idx = 3)
                                        |     date
96                                      |   Literal value (len = 29)
                                        |     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
5e                                      | == Literal indexed ==
                                        |   Indexed name (idx = 30)
                                        |     content-encoding
83                                      |   Literal value (len = 4)
                                        |     Huffman encoded:
9bd9 ab                                 | ...
                                        |     Decoded:
                                        | gzip
```

```
                                         | - evict: date: Mon, 21 Oct\
                                         |    2013 20:13:21 GMT
                                         | -> content-encoding: gzip
84                                       | == Indexed - Remove ==
                                         |   idx = 4
                                         | -> location: https://www.e\
                                         |   xample.com
84                                       | == Indexed - Add ==
                                         |   idx = 4
                                         | -> location: https://www.e\
                                         |   xample.com
83                                       | == Indexed - Remove ==
                                         |   idx = 3
                                         | -> :status: 200
83                                       | == Indexed - Add ==
                                         |   idx = 3
                                         | -> :status: 200
7b                                       | == Literal indexed ==
                                         |   Indexed name (idx = 59)
                                         |     set-cookie
ad                                       |   Literal value (len = 56)
                                         |     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: 200
                                         | -> set-cookie: foo=ASDJKHQ\
                                         |   KBZXOQWEOPIUAXQWEOIU; ma\
                                         |   x-age=3600; version=1
```

Header Table (after decoding):

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

Decoded header set:

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

Authors' Addresses

Roberto Peon
Google, Inc

EMail: fenix@google.com

Herve Ruellan
Canon CRF

EMail: herve.ruellan@crf.canon.fr