Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 11, 2013

J. Snell

June 09, 2013

HTTP/2.0 Discussion: Stored Header Encoding draft-snell-httpbis-bohe-10

Abstract

This memo describes a proposed alternative encoding for headers that combines the best concepts from the proposed Delta and HeaderDiff options with the typed value codecs introduced by previous versions of this draft.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 11, 2013.

Table of Contents

1. Stored Header Encoding

The Stored Header Encoding is an alternative "binary header encoding" for HTTP/2.0 that combines the best elements from three other proposed encodings, including:

- The "Header Delta Compression" scheme proposed by Roberto Peon in http://tools.ietf.org/html/draft-rpeon-httpbis-header-compression-03

- The "Header Diff" encoding proposed by Herve Reullan, Jun Fujisawa, Romain Bellessort, and Youenn Fablet in http://tools.ietf.org/html/draft-ruellan-headerdiff-00

- The "Binary Optimized Header Encoding" proposed by James Snell (me) in http://tools.ietf.org/html/draft-snell-httpbis-bohe-03

The Stored Header Encoding seeks to find an elegant, efficient and simple marriage of the best concepts from each of these separate proposals.

2. State Model

The compressor and decompressor each maintain a cache of header value pairs. There is a static cache, prepopulated by the specification, and a dynamic cache, populated through the compression and decompression process. Each cache contains a maximum of 128 individual key+value pairs.

Each item in the index is referenced by an 8-bit identifier. The most significant bit identifies whether an item from the static or dynamic cache is being referenced. Note: the Nil byte (0x00) is a valid identifier for the dynamic cache.

0xxxxxxx: Dynamic Cache

1xxxxxxx: Static Cache

The dynamic cache is managed in a "least recently written" style, that is, as the cache fills to capacity in both number of entries and maximum stored byte size, the least recently written items are dropped and those index positions are reused.

Index positions from the dynamic cache are assigned in "encounter order", beginning from 0x00 and increasing monotonically to 0x7F. That is to say, the positions are assigned in precisely the same order that they are serialized, and thereby encountered by the decompressor upon reading and processing the block.

Every item in the store consists of a Header Name and a Value. The Name is a lower-case ASCII character sequence. The Value is either a UTF-8 string, a number, a Timestamp or an arbitrary sequence of binary octets.

The available size of the stored compression state can be capped by the decompressor. Each stored value contributes to the accumulated size of the storage state. As new key+value pairs are assigned positions in the dynamic cache, the least-recently assigned items must be removed if necessary to free up the required space.

The size of string values is measured by the number of UTF-8 bytes required for the character sequence.

The size for number and timestamp values is measured by the number of unsigned variable length integer (uvarint) encoded bytes it takes to represent the value (see the section of value types below).

The size of raw binary values is measured by the number of octets.

Duplicate header values MUST be counted individually.

Header names also count towards the stored state size but are only counted once. That is, for instance, the header name "foo" would contribute three bytes to the stored state size regardless of how many distinct instances of the "foo" header appear within the stored state.

Header names MUST NOT exceed 255 octets in length.

3. Header Serialization

Headers are serialized into four typed header groups, each represented by a two-bit identifier. These groups are serialized sequentially. A serialized header block can contain, at most 256 header groups. The first byte of the serialized block is an unsigned, 0-based counter indicating the number of groups.

Header Group Prefix Codes:

00 -- Index Header Group
01 -- Index Range Header Group
10 -- Cloned Index Header Group
11 -- Literal Header Group

The Cloned Index (10) and Literal (11) header group types have an additional "ephemeral" flag indicating whether the group modifies the compression state.

Each header group contains a single 8-bit prefix and up to 32 distinct header instances.

If a particular serialization block contains more than 32 intances of a given type, then multiple instances of the Header Group Type can be included in the serialized block. For instance, if a given message contains 33 index references, the serialized block may contain two separate Index Header Groups.

Wire Format

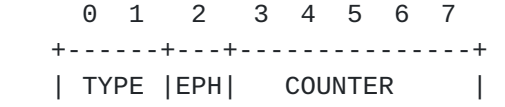
```
header-block              = count *(index-header-group  /
```

```
                                index-range-header-group  /
                                cloned-index-header-group /
                                literal-header-group)

     count                    = OCTET

     ; Header Group Prefix = 8 bits ...
     ;  First two bits = header-group-type
     ;  Third bit = ephemeral flag
     ;  Final five bits = instance counter
     ;
     index-header-group-type    = 00
     index-range-group-type     = 01
     cloned-index-group-type    = 10
     literal-group-type         = 11
     count                      = 5bit

     index-header-prefix        = index-header-group-type
                                  unset count ; 000xxxxx
     index-range-header-prefix  = index-header-group-type
                                  unset count ; 010xxxxx
     cloned-index-header-prefix = cloned-index-group-type
                                  bit   count ; 10?xxxxx
     literal-header-prefix      = literal-group-type
                                  bit   count ; 11?xxxxx

     ; Cache Index Identifier = 8 bits ...
     ;  0xxxxxxx = Dynamic Cache Identifier
     ;  1xxxxxxx = Static Cache Identifier
     cache-index                = %x00-FF

     ; Index Header Group
     index-header-group         = index-header-prefix
                                  1*32cache-index

     ; Index-Range Header Group
     ;  Contains a pair of cache-index values, second MUST
     ;  be strictly higher in value than the first...
     index-range-header-group   = index-range-header-prefix
                                  1*32(cache-index cache-index)

     ; Cloned-Index Header Group
     cloned-index-header-group  = cloned-index-header-prefix
                                  1*32(cache-index value)

     ; Literal Header Group
     literal-header-group       = literal-header-prefix
                                  1*32(name value)
```

```
value                     = text-value /
                            number-value /
                            timestamp-value /
                            binary-value

text-value-type           = 000 ; three bits
number-value-type         = 001
timestamp-value-type      = 010
binary-value-type         = 011

text-value-prefix         = text-value-type count
number-value-prefix       = number-value-type count
timestamp-value-prefix    = timestamp-value-type count
binary-value-prefix       = binary-value-type count

text-value                = text-value-prefix 1*32string
number-value              = number-value-type 1*32uvarint
timestamp-value           = timestamp-value-prefix 1*32uvarint
binary-value              = binary-value-prefix uvarint *OCTET

uvarint                   = *uvarint-continuation uvarint-final
uvarint-continuation      = %x80-FF
uvarint-final             = %x00-7F

name                      = OCTET 1*namechar

namechar                  = ":" / "!" / "#" / "$" / "%" / "&" /
                            "'" / "*" / "+" / "-" / "." / "^" /
                            "_" / "`" / "|" / "~" / DIGIT / %x61-7A

string                    = uvarint *(HUFFMAN-ENCODED-CHAR)
                            HUFFMAN-EOF
                            padding-to-nearest-byte;
padding-to-nearest-byte   = *7unset

bit                       = set / unset
unset                     = 0
set                       = 1
```

3.1. Header Group Prefix

The Header Group Prefix is a single octet that provides three distinct pieces of information:



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

TYPE:
The two most significant bits identify the group type.

EPH:
The next bit is the "ephemeral flag" and is used only for Cloned and Literal group types. This bit indicates whether or not the group alters the stored compression state.

COUNTER:
The remaining five bits specify the number of header instances in the group, with 00000 indicating that the group contains 1 instance and 11111 contains 32. A header group MUST contain at least one instance.

The remaining serialization of the header group depends entirely on the group type.

3.2. Index Header Group

The serialization of the Index Header Group consists of the Header Group Prefix and up to 32 additional octets, each referencing a single 8-bit storage index identifier for items in either the Static or Dynamic Cache.

For instance

```
  00000000 00000000 = References item #0 from
                      the dynamic cache

  00000001 00000000 10000000 = References item #0 from the
                               dynamic cache and item #0
                               from the static cache
```

Index Header Groups do not affect the stored compression state. If an Index Header Group references a header index that has not yet been allocated, the deserialization MUST terminate with an error. This likely means that the compression state has become out of sync and needs to be reestablished.

3.3. Index Range Header Group

The serialization of the Index Range Header Group consists of the Header Group Prefix and up to 32 additional 2-octet (16 bits) pairs of 8-bit storage index identifiers. Each pair specifies a sequential range of adjacent ranges.

For instance:

```
01000000 00000000 00000100 = References items #0-#4 from
                             the dynamic cache.
                             (five distinct items total)
```

A range MAY span dynamic and static index values. Index values are treated as unsigned byte values, so indices from the static cache are numerically greater than dynamic cache values.. e.g.

```
01000000 01111111 10000001 = References item #127 from the
                             dynamic cache, and items #0
                             and #1 from the static cache.
```

Index Range Header Groups do not affect the stored compression state. If a range references a header index that has not yet been allocated, the deserialization MUST terminate with an error. This likely means that the compression state has become out of sync and needs to be reestablished.

3.4. Cloned Index Header Group

The serialization of the Cloned Index Header Group consists of the Header Group Prefix and up to 32 Index+Value pairs. Each Index+Value pair consists of a leading 8-bit storage index of an existing stored header followed by a new serialized value. The serialization of the value depends on the value type (see discussion of Value serialization below).

The Cloned Header Group affects the stored compression state if, and only if, the "ephemeral" flag in the Header Group Prefix is NOT set. If the header group is not marked as being ephemeral, then the specified value is stored in the next available storage index using the key name from the referenced storage index.

For instance, assume the dynamic cache currently contains an item at index #1 with key name "foo" and value "bar", the following causes a new item to be added to the storage with key name "foo" and value "baz":

```
10000000 00000001 00000000 00000100
10111000 01001111 10110101 00100000
```

An explanation of the value syntax is given a bit later.

If a Cloned Header Group references a header index that has not yet been allocated, the deserialization MUST terminate with an error. This likely means that the compression state has become out of sync and needs to be reestablished.

3.5. Literal Header Group

The serialization of the Literal Header Group consists of the Header Group Prefix and up to 32 Name+Value pairs. Each Name+Value pair consists of a length-prefixed sequence of ASCII bytes specifying the Header Name followed by the serialized value. The header name length prefix is encoded as a single unsigned 8-bit integer. The serialization of the value depends on the value type. The value length prefix is encoded as an unsigned variable length integer (uvarint). The length prefix SHOULD NOT be longer than five octets and SHOULD NOT specify a value larger than 0xFFFF.

The Literal Header Group affects the stored compression state if, and only if, the "ephemeral" flag in the Header Group Prefix is NOT set. If the header group is not marked as being ephemeral, then the specified key name and value are stored in the next available storage index.

For instance:

```
11000000 00000011 01100110 01101111
01101111 00000000 00000010 10111000
01000100 11010010
```

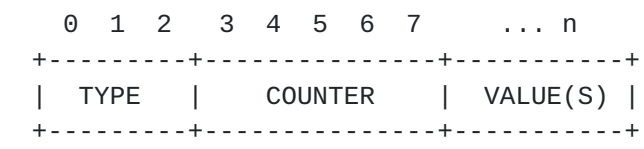
4. Header Values

Header Values can be one of four types, each identified by a three-bit identifier:

- 000 -- UTF-8 Text
- 001 -- Numeric
- 010 -- Timetamp
- 011 -- Raw Binary Octets

A single value can contain up to 32 discreet "value instances".

Each serialized value is preceded by an 8-bit Value Prefix.



TYPE:
The three most significant bits specify the value type.

COUNTER:
The remaining bits specify the number of discreet instances in the value. 00000 indicates that one instance is included, 11111 indicates that 32 instances are included. The value MUST contain at least one instance.

The remaining serialization depends entirely on the type.

4.1. UTF-8 Text Values

UTF-8 Text is encoded as a length-prefixed sequence of Huffman-encoded UTF-8 octets. The length prefix is encoded as an unsigned variable-length integer (Section 4.5) specifying the number of octets after applying the Huffman-encoding.

One text value "foo":

```
00000000 00000011 10000100 11100111 10100100
```

Two text values "foo" and "bar":

```
00000001 00000011 10000100 11100111 10100100
00000011 10111000 01000100 11010010
```

4.2. Numeric Values

Numeric values are encoded as unsigned variable-length integers (uvarint) (Section 4.5) of up to a maximum of 10-octets. Negative values cannot be represented using this syntax.

One numeric value "100"

00100000 01100100

One numeric value "1234"

00100000 11010010 00001001

Two numeric values "100" and "1234"

00100001 01100100 11010010 00001001

4.3. Timestamp Values

Timestamp values are encoded as unsigned variable-length integers (Section 4.5) specifying the number of milliseconds that have passed since the standard Epoch (1970-01-01T00:00:00 GMT). The syntax is identical that used for Numeric Values. Dates prior to the epoch cannot be represented using this syntax.

Representing timestamps in this manner ensures that timestamps will always encode using six bytes up and until 2109-05-15T07:35:00 GMT, then as seven bytes up to and until 19809-03-05T11:03:41 GMT.

A single timestamp value (1370729066123 milliseconds since the epoch):

01000000 10001011 11011101 11000110 10101110 11110010 00100111

4.4. Raw Binary Octet Values

Binary values are encoded as a length prefixed sequence of arbitrary octets. The length prefix is encoded as an unsigned variable length integer.

A single binary value (0x55AA0F):

01100000 00000011 01010101 10101010 00001111

4.5. Unsigned Variable Length Integer Syntax

Unsigned variable length integers are serialized with the least-significant bytes first in batches of 7-bits, with the most

significant bit per byte reserved as a continuation bit. Values less than or equal to 127 are serialized using at most one byte; values less than or equal to 16383 are serialized using at most two bytes; values less than or equal to 2097151 are serialized using at most three bytes; and so on.

UVarInt Psuedocode:

```
def uvarint(num):
  return [] if num = 0
  ret = []
  while(num != 0):
    m = num >>> 7    ; unsigned shift left 7 bits
    ret.push (byte)((num & ~0x80) | ( m >  0 ? 0x80 : 0x00 ));
    num = m;
  return ret;
```

The variable length encoding of the value 217 is:

11011001 00000001

The variable length encoding of the value 1386210052 is:

10000100 11000110 11111111 10010100 00000101

4.6. Huffman Coding

All UTF-8 text values are compressed using a modified static Huffman code. "Modified" because the encoded version may contain compact-representations of raw, arbitrary UTF-8 bytes that are not covered by the static Huffman code table.

There are two Huffman tables in use, one for HTTP Requests and another for HTTP Responses, each covers UTF-8 codepoints strictly less than 128 as well the fifty possible UTF-8 leading octets.

The encoded result MUST end with a specific terminal sequence of bits called the "HUFFMAN_EOF". Currently, the HUFFMAN_EOF is the same for both the Request and Response tables, but that could change if the tables are regenerated. Currently, the HUFFMAN_EOF sequence is 101001.

Codepoints >= 128 are handled by first taking the leading octet of the UTF-8 representation and serializing its associated Huffman code from the table to the output stream, then, depending on the octets

value, serializing the six least significant bits from each of the remaining trailing octets.

For instance, the UTF-8 character U+00D4 (LATIN CAPITAL LETTER O WITH CIRCUMFLEX), with UTF-8 representation of C394 (hex) is encoded as:

11000100 01010010 10010000

The first 8-bits represents the Huffman-table prefix, the six most significant bytes of the second octet are taken directly from the six least significant bits of the second UTF-8 byte (0x94). Following those six bits are the six bits of the HUFFMAN_EOF 101001, followed by four unset padding bits.

The number of raw UTF-8 bits to write depends on the value of the leading octet. If the value is between 0xC2 and 0xDF (inclusive), six bits from the second continuation byte is encoded. If the value is between 0xE0 and 0xEF (inclusive), six bits from the second and third continuation bytes are encoded. If the value is between 0xF0 and 0xF4 (inclusive), six bits from the second, third and fourth continuation bytes are encoded. UTF-8 codepoints that require greater than four bytes to encode cannot be represented.

5. Implementation Considerations

When implementing the Stored Header Encoding, it is important to note that the compression state is managed in encounter order. This means that the compressor must delay storing items in the compression state until the order in which frames are to be serialized out to the network has been determined. This is a potential performance bottleneck that needs to be fully tested out to determine its impact.

6. Security Considerations

TBD

7. References

7.1. Normative References

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

7.2. Informational References

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

Appendix A. Huffman Tables

Request Table

```
    (  0)  |11111111|11111111|11111111|0 [25]        1fffffe [25]
    (  1)  |11111111|11111111|11111111|1 [25]        1ffffff [25]
    (  2)  |11111111|11111111|11100000 [24]          ffffe0 [24]
    (  3)  |11111111|11111111|11100001 [24]          ffffe1 [24]
    (  4)  |11111111|11111111|11100010 [24]          ffffe2 [24]
    (  5)  |11111111|11111111|11100011 [24]          ffffe3 [24]
    (  6)  |11111111|11111111|11100100 [24]          ffffe4 [24]
    (  7)  |11111111|11111111|11100101 [24]          ffffe5 [24]
    (  8)  |11111111|11111111|11100110 [24]          ffffe6 [24]
    (  9)  |11111111|11111111|11100111 [24]          ffffe7 [24]
    ( 10)  |11111111|11111111|11101000 [24]          ffffe8 [24]
    ( 11)  |11111111|11111111|11101001 [24]          ffffe9 [24]
    ( 12)  |11111111|11111111|11101010 [24]          ffffea [24]
    ( 13)  |11111111|11111111|11101011 [24]          ffffeb [24]
    ( 14)  |11111111|11111111|11101100 [24]          ffffec [24]
    ( 15)  |11111111|11111111|11101101 [24]          ffffed [24]
    ( 16)  |11111111|11111111|11101110 [24]          ffffee [24]
    ( 17)  |11111111|11111111|11101111 [24]          fffef [24]
    ( 18)  |11111111|11111111|11110000 [24]          fffff0 [24]
    ( 19)  |11111111|11111111|11110001 [24]          fffff1 [24]
    ( 20)  |11111111|11111111|11110010 [24]          fffff2 [24]
    ( 21)  |11111111|11111111|11110011 [24]          fffff3 [24]
    ( 22)  |11111111|11111111|11110100 [24]          fffff4 [24]
    ( 23)  |11111111|11111111|11110101 [24]          fffff5 [24]
    ( 24)  |11111111|11111111|11110110 [24]          fffff6 [24]
    ( 25)  |11111111|11111111|11110111 [24]          fffff7 [24]
    ( 26)  |11111111|11111111|11111000 [24]          fffff8 [24]
    ( 27)  |11111111|11111111|11111001 [24]          fffff9 [24]
    ( 28)  |11111111|11111111|11111010 [24]          fffffa [24]
    ( 29)  |11111111|11111111|11111011 [24]          fffffb [24]
    ( 30)  |11111111|11111111|11111100 [24]          fffffc [24]
    ( 31)  |11111111|11111111|11111101 [24]          fffffd [24]
' ' ( 32)  |11111111|0110 [12]                          ff6 [12]
'!' ( 33)  |11111111|0111 [12]                          ff7 [12]
'"' ( 34)  |11111111|111010 [14]                       3ffa [14]
'#' ( 35)  |11111111|1111100 [15]                      7ffc [15]
'$' ( 36)  |11111111|1111101 [15]                      7ffd [15]
'%' ( 37)  |011000 [6]                                   18 [6]
'&' ( 38)  |1010100 [7]                                  54 [7]
''' ( 39)  |11111111|1111110 [15]                      7ffe [15]
'(' ( 40)  |11111111|1000 [12]                          ff8 [12]
')' ( 41)  |11111111|1001 [12]                          ff9 [12]
'*' ( 42)  |11111111|1010 [12]                          ffa [12]
'+' ( 43)  |11111111|1011 [12]                          ffb [12]
```

```
   ',' ( 44)  |11111011|10 [10]                          3ee [10]
   '-' ( 45)  |011001 [6]                                 19 [6]
   '.' ( 46)  |00010 [5]                                   2 [5]
   '/' ( 47)  |00011 [5]                                   3 [5]
   '0' ( 48)  |011010 [6]                                 1a [6]
   '1' ( 49)  |011011 [6]                                 1b [6]
   '2' ( 50)  |011100 [6]                                 1c [6]
   '3' ( 51)  |011101 [6]                                 1d [6]
   '4' ( 52)  |1010101 [7]                                55 [7]
   '5' ( 53)  |1010110 [7]                                56 [7]
   '6' ( 54)  |1010111 [7]                                57 [7]
   '7' ( 55)  |1011000 [7]                                58 [7]
   '8' ( 56)  |1011001 [7]                                59 [7]
   '9' ( 57)  |1011010 [7]                                5a [7]
   ':' ( 58)  |011110 [6]                                 1e [6]
   ';' ( 59)  |11111011|11 [10]                          3ef [10]
   '<' ( 60)  |11111111|11111111|10 [18]               3fffe [18]
   '=' ( 61)  |011111 [6]                                 1f [6]
   '>' ( 62)  |11111111|11111110|0 [17]                1fffc [17]
   '?' ( 63)  |11110110|0 [9]                            1ec [9]
   '@' ( 64)  |11111111|11100 [13]                      1ffc [13]
   'A' ( 65)  |10111010 [8]                               ba [8]
   'B' ( 66)  |11110110|1 [9]                            1ed [9]
   'C' ( 67)  |10111011 [8]                               bb [8]
   'D' ( 68)  |10111100 [8]                               bc [8]
   'E' ( 69)  |11110111|0 [9]                            1ee [9]
   'F' ( 70)  |10111101 [8]                               bd [8]
   'G' ( 71)  |11111100|00 [10]                          3f0 [10]
   'H' ( 72)  |11111100|01 [10]                          3f1 [10]
   'I' ( 73)  |11110111|1 [9]                            1ef [9]
   'J' ( 74)  |11111100|10 [10]                          3f2 [10]
   'K' ( 75)  |11111111|010 [11]                         7fa [11]
   'L' ( 76)  |11111100|11 [10]                          3f3 [10]
   'M' ( 77)  |11111000|0 [9]                            1f0 [9]
   'N' ( 78)  |11111101|00 [10]                          3f4 [10]
   'O' ( 79)  |11111101|01 [10]                          3f5 [10]
   'P' ( 80)  |11111000|1 [9]                            1f1 [9]
   'Q' ( 81)  |11111101|10 [10]                          3f6 [10]
   'R' ( 82)  |11111001|0 [9]                            1f2 [9]
   'S' ( 83)  |11111001|1 [9]                            1f3 [9]
   'T' ( 84)  |11111010|0 [9]                            1f4 [9]
   'U' ( 85)  |11111101|11 [10]                          3f7 [10]
   'V' ( 86)  |11111110|00 [10]                          3f8 [10]
   'W' ( 87)  |11111110|01 [10]                          3f9 [10]
   'X' ( 88)  |11111110|10 [10]                          3fa [10]
   'Y' ( 89)  |11111110|11 [10]                          3fb [10]
   'Z' ( 90)  |11111111|00 [10]                          3fc [10]
   '[' ( 91)  |11111111|111011 [14]                     3ffb [14]
```

```
   '\' ( 92)  |11111111|11111111|11111110 [24]          fffffe [24]
   ']' ( 93)  |11111111|111100 [14]                       3ffc [14]
   '^' ( 94)  |11111111|111101 [14]                       3ffd [14]
   '_' ( 95)  |1011011 [7]                                  5b [7]
   '`' ( 96)  |11111111|11111111|110 [19]                 7fffe [19]
   'a' ( 97)  |00100 [5]                                    4 [5]
   'b' ( 98)  |1011100 [7]                                  5c [7]
   'c' ( 99)  |00101 [5]                                    5 [5]
   'd' (100)  |100000 [6]                                  20 [6]
   'e' (101)  |0000 [4]                                     0 [4]
   'f' (102)  |100001 [6]                                  21 [6]
   'g' (103)  |100010 [6]                                  22 [6]
   'h' (104)  |100011 [6]                                  23 [6]
   'i' (105)  |00110 [5]                                    6 [5]
   'j' (106)  |10111110 [8]                                be [8]
   'k' (107)  |10111111 [8]                                bf [8]
   'l' (108)  |100100 [6]                                  24 [6]
   'm' (109)  |100101 [6]                                  25 [6]
   'n' (110)  |100110 [6]                                  26 [6]
   'o' (111)  |00111 [5]                                    7 [5]
   'p' (112)  |01000 [5]                                    8 [5]
   'q' (113)  |11111010|1 [9]                              1f5 [9]
   'r' (114)  |01001 [5]                                    9 [5]
   's' (115)  |01010 [5]                                    a [5]
   't' (116)  |01011 [5]                                    b [5]
   'u' (117)  |100111 [6]                                  27 [6]
   'v' (118)  |11000000 [8]                                c0 [8]
   'w' (119)  |101000 [6]                                  28 [6]
   'x' (120)  |11000001 [8]                                c1 [8]
   'y' (121)  |11000010 [8]                                c2 [8]
   'z' (122)  |11111011|0 [9]                              1f6 [9]
   '{' (123)  |11111111|11111110|1 [17]                 1fffd [17]
   '|' (124)  |11111111|1100 [12]                          ffc [12]
   '}' (125)  |11111111|11111111|0 [17]                  1fffe [17]
   '~' (126)  |11111111|1101 [12]                          ffd [12]
      (127)  |101001 [6]                                  29 [6]
      (0xC2) |11000011 [8]                                c3 [8]
      (0xC3) |11000100 [8]                                c4 [8]
      (0xC4) |11000101 [8]                                c5 [8]
      (0xC5) |11000110 [8]                                c6 [8]
      (0xC6) |11000111 [8]                                c7 [8]
      (0xC7) |11001000 [8]                                c8 [8]
      (0xC8) |11001001 [8]                                c9 [8]
      (0xC9) |11001010 [8]                                ca [8]
      (0xCA) |11001011 [8]                                cb [8]
      (0xCB) |11001100 [8]                                cc [8]
      (0xCC) |11001101 [8]                                cd [8]
      (0xCD) |11001110 [8]                                ce [8]
```

```
      (0xCE) |11001111 [8]                                  cf [8]
      (0xCF) |11010000 [8]                                  d0 [8]
      (0xD0) |11010001 [8]                                  d1 [8]
      (0xD1) |11010010 [8]                                  d2 [8]
      (0xD2) |11010011 [8]                                  d3 [8]
      (0xD3) |11010100 [8]                                  d4 [8]
      (0xD4) |11010101 [8]                                  d5 [8]
      (0xD5) |11010110 [8]                                  d6 [8]
      (0xD6) |11010111 [8]                                  d7 [8]
      (0xD7) |11011000 [8]                                  d8 [8]
      (0xD8) |11011001 [8]                                  d9 [8]
      (0xD9) |11011010 [8]                                  da [8]
      (0xDA) |11011011 [8]                                  db [8]
      (0xDB) |11011100 [8]                                  dc [8]
      (0xDC) |11011101 [8]                                  dd [8]
      (0xDD) |11011110 [8]                                  de [8]
      (0xDE) |11011111 [8]                                  df [8]
      (0xDF) |11100000 [8]                                  e0 [8]
      (0xE0) |11100001 [8]                                  e1 [8]
      (0xE1) |11100010 [8]                                  e2 [8]
      (0xE2) |11100011 [8]                                  e3 [8]
      (0xE3) |11100100 [8]                                  e4 [8]
      (0xE4) |11100101 [8]                                  e5 [8]
      (0xE5) |11100110 [8]                                  e6 [8]
      (0xE6) |11100111 [8]                                  e7 [8]
      (0xE7) |11101000 [8]                                  e8 [8]
      (0xE8) |11101001 [8]                                  e9 [8]
      (0xE9) |11101010 [8]                                  ea [8]
      (0xEA) |11101011 [8]                                  eb [8]
      (0xEB) |11101100 [8]                                  ec [8]
      (0xEC) |11101101 [8]                                  ed [8]
      (0xED) |11101110 [8]                                  ee [8]
      (0xEE) |11101111 [8]                                  ef [8]
      (0xEF) |11110000 [8]                                  f0 [8]
      (0xF0) |11110001 [8]                                  f1 [8]
      (0xF1) |11110010 [8]                                  f2 [8]
      (0xF2) |11110011 [8]                                  f3 [8]
      (0xF3) |11110100 [8]                                  f4 [8]
      (0xF4) |11110101 [8]                                  f5 [8]
```

Response Table:

```
      (  0)  |11111111|11111111|11111111|0 [25]        1fffffe [25]
      (  1)  |11111111|11111111|11111111|1 [25]        1ffffff [25]
      (  2)  |11111111|11111111|11100000 [24]          ffffe0 [24]
      (  3)  |11111111|11111111|11100001 [24]          ffffe1 [24]
      (  4)  |11111111|11111111|11100010 [24]          ffffe2 [24]
```

```
      (  5)  |11111111|11111111|11100011 [24]          fffe3 [24]
      (  6)  |11111111|11111111|11100100 [24]          fffe4 [24]
      (  7)  |11111111|11111111|11100101 [24]          fffe5 [24]
      (  8)  |11111111|11111111|11100110 [24]          fffe6 [24]
      (  9)  |11111111|11111111|11100111 [24]          fffe7 [24]
      ( 10)  |11111111|11111111|11101000 [24]          fffe8 [24]
      ( 11)  |11111111|11111111|11101001 [24]          fffe9 [24]
      ( 12)  |11111111|11111111|11101010 [24]          fffea [24]
      ( 13)  |11111111|11111111|11101011 [24]          fffeb [24]
      ( 14)  |11111111|11111111|11101100 [24]          fffec [24]
      ( 15)  |11111111|11111111|11101101 [24]          fffed [24]
      ( 16)  |11111111|11111111|11101110 [24]          fffee [24]
      ( 17)  |11111111|11111111|11101111 [24]          fffef [24]
      ( 18)  |11111111|11111111|11110000 [24]          fffff0 [24]
      ( 19)  |11111111|11111111|11110001 [24]          fffff1 [24]
      ( 20)  |11111111|11111111|11110010 [24]          fffff2 [24]
      ( 21)  |11111111|11111111|11110011 [24]          fffff3 [24]
      ( 22)  |11111111|11111111|11110100 [24]          fffff4 [24]
      ( 23)  |11111111|11111111|11110101 [24]          fffff5 [24]
      ( 24)  |11111111|11111111|11110110 [24]          fffff6 [24]
      ( 25)  |11111111|11111111|11110111 [24]          fffff7 [24]
      ( 26)  |11111111|11111111|11111000 [24]          fffff8 [24]
      ( 27)  |11111111|11111111|11111001 [24]          fffff9 [24]
      ( 28)  |11111111|11111111|11111010 [24]          ffffffa [24]
      ( 29)  |11111111|11111111|11111011 [24]          ffffffb [24]
      ( 30)  |11111111|11111111|11111100 [24]          ffffffc [24]
      ( 31)  |11111111|11111111|11111101 [24]          ffffffd [24]
  ' ' ( 32)  |11111111|0110 [12]                          ff6 [12]
  '!' ( 33)  |11111111|0111 [12]                          ff7 [12]
  '"' ( 34)  |11111111|111010 [14]                       3ffa [14]
  '#' ( 35)  |11111111|1111100 [15]                      7ffc [15]
  '$' ( 36)  |11111111|1111101 [15]                      7ffd [15]
  '%' ( 37)  |011000 [6]                                   18 [6]
  '&' ( 38)  |1010100 [7]                                  54 [7]
  ''' ( 39)  |11111111|1111110 [15]                      7ffe [15]
  '(' ( 40)  |11111111|1000 [12]                          ff8 [12]
  ')' ( 41)  |11111111|1001 [12]                          ff9 [12]
  '*' ( 42)  |11111111|1010 [12]                          ffa [12]
  '+' ( 43)  |11111111|1011 [12]                          ffb [12]
  ',' ( 44)  |11111011|10 [10]                           3ee [10]
  '-' ( 45)  |011001 [6]                                   19 [6]
  '.' ( 46)  |00010 [5]                                    2 [5]
  '/' ( 47)  |00011 [5]                                    3 [5]
  '0' ( 48)  |011010 [6]                                   1a [6]
  '1' ( 49)  |011011 [6]                                   1b [6]
  '2' ( 50)  |011100 [6]                                   1c [6]
  '3' ( 51)  |011101 [6]                                   1d [6]
  '4' ( 52)  |1010101 [7]                                  55 [7]
```

```
   '5' ( 53)  |1010110 [7]                                    56 [7]
   '6' ( 54)  |1010111 [7]                                    57 [7]
   '7' ( 55)  |1011000 [7]                                    58 [7]
   '8' ( 56)  |1011001 [7]                                    59 [7]
   '9' ( 57)  |1011010 [7]                                    5a [7]
   ':' ( 58)  |011110 [6]                                     1e [6]
   ';' ( 59)  |11111011|11 [10]                              3ef [10]
   '<' ( 60)  |11111111|11111111|10 [18]                   3fffe [18]
   '=' ( 61)  |011111 [6]                                     1f [6]
   '>' ( 62)  |11111111|11111110|0 [17]                    1fffc [17]
   '?' ( 63)  |11110110|0 [9]                                1ec [9]
   '@' ( 64)  |11111111|11100 [13]                          1ffc [13]
   'A' ( 65)  |10111010 [8]                                   ba [8]
   'B' ( 66)  |11110110|1 [9]                                1ed [9]
   'C' ( 67)  |10111011 [8]                                   bb [8]
   'D' ( 68)  |10111100 [8]                                   bc [8]
   'E' ( 69)  |11110111|0 [9]                                1ee [9]
   'F' ( 70)  |10111101 [8]                                   bd [8]
   'G' ( 71)  |11111100|00 [10]                              3f0 [10]
   'H' ( 72)  |11111100|01 [10]                              3f1 [10]
   'I' ( 73)  |11110111|1 [9]                                1ef [9]
   'J' ( 74)  |11111100|10 [10]                              3f2 [10]
   'K' ( 75)  |11111111|010 [11]                             7fa [11]
   'L' ( 76)  |11111100|11 [10]                              3f3 [10]
   'M' ( 77)  |11111000|0 [9]                                1f0 [9]
   'N' ( 78)  |11111101|00 [10]                              3f4 [10]
   'O' ( 79)  |11111101|01 [10]                              3f5 [10]
   'P' ( 80)  |11111000|1 [9]                                1f1 [9]
   'Q' ( 81)  |11111101|10 [10]                              3f6 [10]
   'R' ( 82)  |11111001|0 [9]                                1f2 [9]
   'S' ( 83)  |11111001|1 [9]                                1f3 [9]
   'T' ( 84)  |11111010|0 [9]                                1f4 [9]
   'U' ( 85)  |11111101|11 [10]                              3f7 [10]
   'V' ( 86)  |11111110|00 [10]                              3f8 [10]
   'W' ( 87)  |11111110|01 [10]                              3f9 [10]
   'X' ( 88)  |11111110|10 [10]                              3fa [10]
   'Y' ( 89)  |11111110|11 [10]                              3fb [10]
   'Z' ( 90)  |11111111|00 [10]                              3fc [10]
   '[' ( 91)  |11111111|111011 [14]                         3ffb [14]
   '\' ( 92)  |11111111|11111111|11111110 [24]           fffffe [24]
   ']' ( 93)  |11111111|111100 [14]                         3ffc [14]
   '^' ( 94)  |11111111|111101 [14]                         3ffd [14]
   '_' ( 95)  |1011011 [7]                                   5b [7]
   '`' ( 96)  |11111111|11111111|110 [19]                  7fffe [19]
   'a' ( 97)  |00100 [5]                                     4 [5]
   'b' ( 98)  |1011100 [7]                                   5c [7]
   'c' ( 99)  |00101 [5]                                     5 [5]
   'd' (100)  |100000 [6]                                   20 [6]
```

'e' (101)	|0000 [4]	0 [4]
'f' (102)	|100001 [6]	21 [6]
'g' (103)	|100010 [6]	22 [6]
'h' (104)	|100011 [6]	23 [6]
'i' (105)	|00110 [5]	6 [5]
'j' (106)	|10111110 [8]	be [8]
'k' (107)	|10111111 [8]	bf [8]
'l' (108)	|100100 [6]	24 [6]
'm' (109)	|100101 [6]	25 [6]
'n' (110)	|100110 [6]	26 [6]
'o' (111)	|00111 [5]	7 [5]
'p' (112)	|01000 [5]	8 [5]
'q' (113)	|11111010|1 [9]	1f5 [9]
'r' (114)	|01001 [5]	9 [5]
's' (115)	|01010 [5]	a [5]
't' (116)	|01011 [5]	b [5]
'u' (117)	|100111 [6]	27 [6]
'v' (118)	|11000000 [8]	c0 [8]
'w' (119)	|101000 [6]	28 [6]
'x' (120)	|11000001 [8]	c1 [8]
'y' (121)	|11000010 [8]	c2 [8]
'z' (122)	|11111011|0 [9]	1f6 [9]
'{' (123)	|11111111|11111110|1 [17]	1fffd [17]
'|' (124)	|11111111|1100 [12]	ffc [12]
'}' (125)	|11111111|11111111|0 [17]	1fffe [17]
'~' (126)	|11111111|1101 [12]	ffd [12]
(127)	|101001 [6]	29 [6]
(0xC2)	|11000011 [8]	c3 [8]
(0xC3)	|11000100 [8]	c4 [8]
(0xC4)	|11000101 [8]	c5 [8]
(0xC5)	|11000110 [8]	c6 [8]
(0xC6)	|11000111 [8]	c7 [8]
(0xC7)	|11001000 [8]	c8 [8]
(0xC8)	|11001001 [8]	c9 [8]
(0xC9)	|11001010 [8]	ca [8]
(0xCA)	|11001011 [8]	cb [8]
(0xCB)	|11001100 [8]	cc [8]
(0xCC)	|11001101 [8]	cd [8]
(0xCD)	|11001110 [8]	ce [8]
(0xCE)	|11001111 [8]	cf [8]
(0xCF)	|11010000 [8]	d0 [8]
(0xD0)	|11010001 [8]	d1 [8]
(0xD1)	|11010010 [8]	d2 [8]
(0xD2)	|11010011 [8]	d3 [8]
(0xD3)	|11010100 [8]	d4 [8]
(0xD4)	|11010101 [8]	d5 [8]
(0xD5)	|11010110 [8]	d6 [8]
(0xD6)	|11010111 [8]	d7 [8]

```
      (0xD7) |11011000 [8]                                d8 [8]
      (0xD8) |11011001 [8]                                d9 [8]
      (0xD9) |11011010 [8]                                da [8]
      (0xDA) |11011011 [8]                                db [8]
      (0xDB) |11011100 [8]                                dc [8]
      (0xDC) |11011101 [8]                                dd [8]
      (0xDD) |11011110 [8]                                de [8]
      (0xDE) |11011111 [8]                                df [8]
      (0xDF) |11100000 [8]                                e0 [8]
      (0xE0) |11100001 [8]                                e1 [8]
      (0xE1) |11100010 [8]                                e2 [8]
      (0xE2) |11100011 [8]                                e3 [8]
      (0xE3) |11100100 [8]                                e4 [8]
      (0xE4) |11100101 [8]                                e5 [8]
      (0xE5) |11100110 [8]                                e6 [8]
      (0xE6) |11100111 [8]                                e7 [8]
      (0xE7) |11101000 [8]                                e8 [8]
      (0xE8) |11101001 [8]                                e9 [8]
      (0xE9) |11101010 [8]                                ea [8]
      (0xEA) |11101011 [8]                                eb [8]
      (0xEB) |11101100 [8]                                ec [8]
      (0xEC) |11101101 [8]                                ed [8]
      (0xED) |11101110 [8]                                ee [8]
      (0xEE) |11101111 [8]                                ef [8]
      (0xEF) |11110000 [8]                                f0 [8]
      (0xF0) |11110001 [8]                                f1 [8]
      (0xF1) |11110010 [8]                                f2 [8]
      (0xF2) |11110011 [8]                                f3 [8]
      (0xF3) |11110100 [8]                                f4 [8]
      (0xF4) |11110101 [8]                                f5 [8]
```

Appendix B. Static Storage Cache

```
   0x80 ":date"                        = NIL
   0x81 ":scheme"                      = "https"
   0x82 ":scheme"                      = "http"
   0x83 ":scheme"                      = "ftp"
   0x84 ":method"                      = "get"
   0x85 ":method"                      = "post"
   0x86 ":method"                      = "put"
   0x87 ":method"                      = "delete"
   0x88 ":method"                      = "options"
   0x89 ":method"                      = "patch"
   0x8A ":method"                      = "connect"
   0x8B ":path"                        = "/"
   0x8C ":host"                        = NIL
   0x8D "cookie"                       = NIL
```

```
   0x8E ":status"                    = NIL
   0x8F ":status-text"               = NIL
   0x90 ":version"                   = NIL
   0x91 "accept"                     = NIL
   0x92 "accept-charset"             = NIL
   0x93 "accept-encoding"            = NIL
   0x94 "accept-language"            = NIL
   0x95 "accept-ranges"              = NIL
   0x96 "allow"                      = NIL
   0x97 "authorization"              = NIL
   0x98 "cache-control"              = NIL
   0x99 "content-base"               = NIL
   0x9A "content-encoding"           = NIL
   0x9B "content-length"             = NIL
   0x9C "content-location"           = NIL
   0x9D "content-md5"                = NIL
   0x9E "content-range"              = NIL
   0x9F "content-type"               = NIL
   0xA0 "content-disposition"        = NIL
   0xA1 "content-language"           = NIL
   0xA2 "etag"                       = NIL
   0xA3 "expect"                     = NIL
   0xA4 "expires"                    = NIL
   0xA5 "from"                       = NIL
   0xA6 "if-match"                   = NIL
   0xA7 "if-modified-since"          = NIL
   0xA8 "if-none-match"              = NIL
   0xA9 "if-range"                   = NIL
   0xAA "if-unmodified-since"        = NIL
   0xAB "last-modified"              = NIL
   0xAC "location"                   = NIL
   0xAD "max-forwards"               = NIL
   0xAE "origin"                     = NIL
   0xAF "pragma"                     = NIL
   0xB0 "proxy-authenticate"         = NIL
   0xB1 "proxy-authorization"        = NIL
   0xB2 "range"                      = NIL
   0xB3 "referer"                    = NIL
   0xB4 "retry-after"                = NIL
   0xB5 "server"                     = NIL
   0xB6 "set-cookie"                 = NIL
   0xB7 "status"                     = NIL
   0xB8 "te"                         = NIL
   0xB9 "trailer"                    = NIL
   0xBA "transfer-encoding"          = NIL
   0xBB "upgrade"                    = NIL
   0xBC "user-agent"                 = NIL
   0xBD "vary"                       = NIL
```

```
   0xBE "via"                          = NIL
   0xBF "warning"                      = NIL
   0xC0 "www-authenticate"             = NIL
   0xC1 "access-control-allow-origin" = NIL
   0xC2 "get-dictionary"               = NIL
   0xC3 "p3p"                          = NIL
   0xC4 "link"                         = NIL
   0xC5 "prefer"                       = NIL
   0xC6 "preference-applied"           = NIL
   0xC7 "accept-patch"                 = NIL
   0xC8 NIL
   0xC9 NIL
   0xCA NIL
   0xCB NIL
   0xCC NIL
   0xCD NIL
   0xCE NIL
   0xCF NIL
   0xD0 NIL
   0xD1 NIL
   0xD2 NIL
   0xD3 NIL
   0xD4 NIL
   0xD5 NIL
   0xD6 NIL
   0xD7 NIL
   0xD8 NIL
   0xD9 NIL
   0xDA NIL
   0xDB NIL
   0xDC NIL
   0xDD NIL
   0xDE NIL
   0xDF NIL
   0xE0 NIL
   0xE1 NIL
   0xE2 NIL
   0xE3 NIL
   0xE4 NIL
   0xE5 NIL
   0xE6 NIL
   0xE7 NIL
   0xE8 NIL
   0xE9 NIL
   0xEA NIL
   0xEB NIL
   0xEC NIL
   0xED NIL
```

0xEE NIL
0xEF NIL
0xF0 NIL
0xF1 NIL
0xF2 NIL
0xF3 NIL
0xF4 NIL
0xF5 NIL
0xF6 NIL
0xF7 NIL
0xF8 NIL
0xF9 NIL
0xFA NIL
0xFB NIL
0xFC NIL
0xFD NIL
0xFE NIL
0xFF NIL

Appendix C. Updated Standard Header Definitions

In order to properly deal with the backwards compatibility concerns for HTTP/1, there are several important rules for use of Typed Codecs in HTTP headers:

- All header fields MUST be explicitly defined to use the new header types. All existing HTTP/1 header fields, then, will continue to be represented as ISO-8859-1 Text unless their standard definitions are updated. The HTTP/2 specification would update the definition of specific known header fields (e.g. content-length, date, if-modified-since, etc).

- Extension header fields that use the typed codecs will have specific normative transformations to ISO-8859-1 defined.

  * UTF-8 Text will be converted to ISO-8859-1 with extended characters pct-encoded

  * Numbers will be converted to their ASCII equivalent values.

  * Date Times will be converted to their HTTP-Date equivalent values.

  * Binary fields will be Base64-encoded.

- There will be no normative transformation from ISO-8859-1 values into the typed codecs. Implementations are free to apply

transformation where those impls determine it is appropriate, but it will be perfectly legal for an implementation to pass a text value through even if it is known that a given header type has a typed codec equivalent (for instance, Content-Length may come through as a number or a text value, either will be valid). This means that when translating from HTTP/1 -> HTTP/2, receiving implementations need to be prepared to handle either value form.

A Note of warning: Individual header fields MAY be defined such that they can be represented using multiple types. Numeric fields, for instance, can be represented using either the uvarint encoding or using the equivalent sequence of ASCII numbers. Implementers will need to be capable of supporting each of the possible variations. Designers of header field definitions need to be aware of the additional complexity and possible issues that allowing for such alternatives can introduce for implementers.

Based on an initial survey of header fields currently defined by the HTTPbis specification documents, the following header field definitions can be updated to make use of the new types

Field	Type	Description
content-length	Numeric or Text	Can be represented as either an unsigned, variable-length integer or a sequence of ASCII numbers.
date	Timestamp or Text	Can be represented as either a uvarint encoded timestamp or as text (HTTP-date).
max-forwards	Numeric or Text	Can be represented as either an unsigned, variable-length integer or a sequence of ASCII numbers.
retry-after	Timestamp, Numeric or Text	Can be represented as either a uvarint encoded timestamp, an unsigned, variable-length integer, or the text equivalents of either (HTTP-date or sequence of ASCII numbers)
if-modified-since	Timestamp or Text	Can be represented as either a uvarint encoded timestamp or as text (HTTP-

		date).
if-unmodified-since	Timestamp or Text	Can be represented as either a uvarint encoded timestamp or as text (HTTP-date).
last-modified	Timestamp or Text	Can be represented as either a uvarint encoded timestamp or as text (HTTP-date).
age	Numeric or Text	Can be represented as either an unsigned, variable-length integer or a sequence of ASCII numbers.
expires	Timestamp or Text	Can be represented as either a uvarint encoded timestamp or as text (HTTP-date).
etag	Binary or Text	Can be represented as either a sequence of binary octets or using the currently defined text format. When represented as binary octets, the Entity Tag MUST be considered to be a Strong Entity tag. Weak Entity Tags cannot be represented using the binary octet option.

Appendix D. Alternative Timestamp encodings

This specification currently uses the number of milliseconds from the UNIX Epoch to represent timestamps. This 64-bit number is encoded using the same uvarint encoding as Numeric fields. This means that the timestamp is encoded using a variable width that, right now (for about the next 100 years or so), will encode in six bytes, then seven bytes for the reasonable future.

One possible alternative approach we can take is similar to NTP's handling of Era's. We can take the current timestamp and generate an Era value, with a maximum of 255 (0xFF). This is used as a multiplier for the timestamp value. The two values are calculated using the following formula:

```
m   = 4294967296000
now = milliseconds since UNIX Epoch
era = now / m
ts  = now % m
```

The allowable values for "era" would be capped at 255. This value is encoded as a single byte, followed by a uvarint encoding of ts. This ensures that the timestamp will never be encoded using more than 7-bytes total, though it may be encoded in as few as two bytes on extremely rare occasions (specifically, immediately following each era rollover).

Wire Syntax:

```
era       = %x00-FF
ts        = uvarint
date-time = era ts
```

To convert back to the Epoch, the formula is equally simple:

```
now = era * m + ts
```

The largest date we can encode using this format is "36812-02-20T00:36:15.999Z". Dates prior to the epoch cannot be represented.

The significant drawback with this approach is that current dates would encode in 7-bytes until the next era rollover, which will occur at approximately 2106-02-07T06:28:15.999Z (give or take a few leap seconds thrown in here and there).

Note: The byte lengths assume we want millisecond precision. If we opted to keep the second precision currently in HTTP/1, then this alternative encoding ensures that our timestamps never exceed six-bytes in length.

Appendix E. Alternative uvarint encodings

The uvarint encoding currently specified by this specification is certainly not the only possible option we can use. I chose it simply because it is drop dead simple to implement. There are quite a few other approaches we can take, each of which can be used as drop-in replacements for the current approach. Below are just a couple alternatives. There are plenty others. We just need to pick the one that we feel will work the best.

It ought to be noted that while each of these schemes vary in details such as endianess, specific wire-format, etc, each will typically encode the same numbers using the same number of bytes with variations of only a single byte only in the edge cases. Processing time for each is also equivalent when dealing with any number less or equal to 64-bits in length. This means that the choice is largely a matter of style than substance.

E.1. Option 1:

With this option, leading bits are used to indicate the total number of bytes used to encode the number value, with no fixed upper limit. Values strictly less than 128 are encoded using a single byte.

```
0 xxxxxxx
10 xxxxxx OCTET
110 xxxxx 2OCTET
1110 xxxx 3OCTET
11110 xxx 4OCTET
111110 xx 5OCTET
1111110 x 6OCTET
11111110  7OCTET
11111111  0xxxxxxx 7OCTET
11111111  10xxxxxx 8OCTET
...
```

The number of leading 1 bits specify the number of additional bytes used to serialize the value. A single 0 bit is used to mark the end of this prefix, the remaining bits are used to encode the minimum bits necessary to encode the value (with appropriate leading 0 bits to ensure proper byte-alignment).

For instance, the integer value 500, which is represented in binary as 00000001 11110100, can be encoded using two bytes, 10000001 11110100

The integer value 9770098, which is represented in binary as 10010101 00010100 01110010, can be encoded using four bytes: 11100000 10010101 00010100 01110010

E.2. Option 2:

This option is generally identical to the previous with the exception of being capped at a maximum of nine encoded octets total. Rather than growing indefinitely, the encoded value must never require more then eight continuation bytes to encode. Because of this restriction, there is no need for a trailing 0-bit spilling over past the first leading byte.

```
0 xxxxxxx
10 xxxxxx OCTET
110 xxxxx 2OCTET
1110 xxxx 3OCTET
11110 xxx 4OCTET
111110 xx 5OCTET
1111110 x 6OCTET
11111110  7OCTET
11111111  8OCTET
...
```

The number of leading 1 bits specify the number of additional bytes used to serialize the value. If the number of bytes required is less than 8, a single 0 bit is used to mark the end of this prefix, the remaining bits are used to encode the minimum bits necessary to encode the value (with appropriate leading 0 bits to ensure proper byte-alignment).

For instance, the integer value 500, which is represented in binary as 00000001 11110100, can be encoded using two bytes, 10000001 11110100

The integer value 9770098, which is represented in binary as 10010101 00010100 01110010, can be encoded using four bytes: 11100000 10010101 00010100 01110010

This format is not capable of encoding any number requiring more than 64-bits.

Author's Address

James M Snell

Email: jasnell@gmail.com