Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 11, 2014

J. Snell

July 10, 2013

HTTP/2.0 Discussion: Stored Header Encoding
draft-snell-httpbis-bohe-11

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 January 11, 2014.

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. Model

A "header" is a (name,value) pair. The name is a sequence of lower-case ASCII characters. The value is either a UTF-8 string, an integer, a Date-Time, or an arbitrary sequence of binary octets.

The compressor and decompressor each maintain a synchronized cache of up to 256 headers. Every header in the cache is referenced by an 8-bit identifier. Note that the Nil byte (0x00) is a valid identifier.

The 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 cleared and their index positions are reused.

Index positions from the cache are assigned in "encounter order", beginning from 0x00 and increasing monotonically to 0xFF. 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.

The available size of the stored compression state can be capped by the decompressor using the SETTINGS_MAX_BUFFER_SIZE setting. Each stored header contributes to the accumulated size of the storage state. As new haeder pairs are assigned positions in the cache, the least-recently assigned items must be cleared, if necessary, to free up the required space. Clearing existing items does not change the index positions of the other items in the cache.

The size of a header is calculated as: The number of ASCII octets required for the name plus the number of octets required for the value plus 32-bytes to account for any internal storage overhead. The number of octets required for the value depends on the value type:

- String values are measured by the number of UTF-8 encoded octets required to represent the character sequence.

- Number and Date-Time values are measured by the number of unsigned variable length integer (uvarint) encoded bytes required to represent the value using a 5-bit prefix.

- Binary values are measured by the number of octets contained by the sequence.

3. Header Encoding and Decoding

The set of headers is encoded for transmission using the following process:

1. For each header, determine if the (name,value) pair already exists in the table.

   * If an exact match is found in the header table, encode the indexed position of the header as an Indexed Reference and advance to the next header (name,value) pair.

   * Otherwise, move to the step #2.

2. Determine if a header (name,value) pair with the same name already exists in the table. If a matching name is found, make note of the indexed position of the matching name and continue to step #3.

3. Determine whether the new header (name,value) pair ought to be added to the header table or not.

   * If the header is not to be added to the header table, encode the header as a Non-Indexed Literal Representation and continue to the next header (name,value) pair.

   * Otherwise, continue to step #4.

4. If an existing indexed header using the same name was found in the header table in step #2, determine if the new header (name,value) pair ought to replace that existing entry or if it ought to be added as a new entry.

   * If the header is to replace the existing entry, encode the header as an Indexed Literal Replacement Representation.

   * Otherwise encode the header as an Indexed Literal Representation.

Following these steps, headers are serialized into one of four representation types, each represented by a two-bit prefix code. The types and their codes are:

o Indexed - 10

o Non-Indexed Literal - 00

o Indexed Literal Replacement - 01

o Indexed Literal - 10

Each representation type is encoded into groups of up to 64 instances. Each group is prefixed by a single octet prefix. The two

most significant bits identify the representation type, the six least significant bits specify the number of instances in the group, with 000000 indicating a single instance and 111111 indicating 64.

If a particular serialization block requires more than 64 intances of a given type, then multiple instances of the group type can be encoded. For instance, if a given message contains 65 Indexed Representations, the encoded block would contain two separate Indexed Representation groups.

Decoding simply reverses the encoding steps:

1. First initialize an empty working set of headers.

2. Begin iterating through each representation group:

   * If it is an Indexed group, iterate through each index included in the group, look up the corresponding (name,value) pair in the header table and add that to the working set. If no matching (name,value) is found, terminate and report an error.

   * If it's a Non-Indexed Literal group, iterate through each (name,value) pair included in the group and add that to the working set.

   * If it's an Indexed Literal group, iterate through each (name,value) pair included in the group and add that to both the header table and the working set.

   * If it's an Indexed Literal Replacement group, iterate through each (name,value) pair included in the group, replace the existing entry in the header table at the identified index, and add the new (name,value) to the working set. If no matching (name,value) is found, terminate and report an error.

3. Continue with each representation group until the complete block has been decoded.

3.1. Literal (name,value) Representation

The structure of an encoded (name,value) pair consists of:

- A 3-bit value type identifier,

- The name, encoded either as a literal string or as the header table index position of another existing header sharing the same name, and

o The encoded value.

The three most-sigificant bits of the first octet identify the value type.

This design allows for a maximum of 7 value types, only four of which are defined by this specification. The three remaining value types are reserved for future use. The currently defined value types are:

UTF-8 Text (000)

Integer (001)

Timestamp (010)

Raw Binary (111)

If the name is encoded using an index reference to another existing (name,value) pair in the header table, the remaining five least significant bits of the first octet are set to zero and the next byte identifies the referenced header table index position.

If the name is encoded as a literal string, the number of ASCII bytes required to represent the name is encoded as a unsigned variable length integer with a five-bit prefix, filling the 5-remaining least significant bits of the first octet.

The encoding of the value depends on the value type.

UTF-8 Text:
First, the number of UTF-8 encoded bytes required to represent the value is encoded as an unsigned variable length integer with a 0-bit prefix, followed by the full sequence of UTF-8 bytes.

Integer
The integer's value is encoded as an unsigned variable length integer with a 0-bit prefix. Negative or fractional numbers cannot be represented.

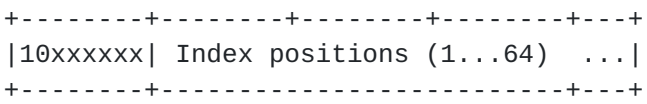
Timestamp
The timestamp is represented as the number of milliseconds ellapsed since the standard Epoch (1970-01-01T00:00:00 GMT), encoded as an unsigned variable length integer with a 0-bit prefix. Timestamps that predate the Epoch cannot be represented.

Raw Binary

The number of octets in the sequence is encoded as an unsigned variable length integer with a 0-bit prefix, followed by the full sequence of octets.

3.2. Indexed Representation

The serialization of an Indexed Representation group consists of the one-octet header group prefix followed by up to 64 single-octet header table index references.



For instance:

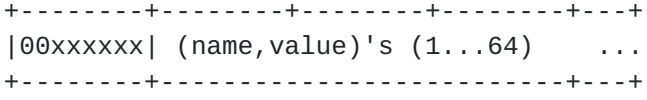
0x80 0x00
References item #0 from the header table.

0x81 0x00 0x01
References items #0 and #1 from the header table.

Indexed Representations do not modify the header table state in any way. If an Indexed References specifies a header index that has not yet been allocated or whose value has been cleared, decoding MUST terminate with an error.

3.3. Non-Indexed Literal Representation

The serialization of a group of Non-Indexed Literal representations consists of the one-octet header prefix followed by up to 64 Literal (name,value) Representations.

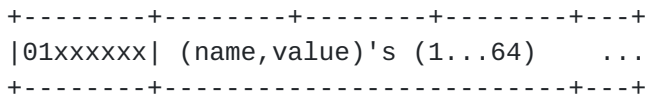


For instance:

0x00 0x01 0x61 0x01 0x62
Specifies a single header with name "a" and a UTF-8 String value of "b" is to be handled as a Non-Indexed header (it is not added to the header table).

3.4. Indexed Literal Representation

The serialization of a group of Indexed Literal representations consists of the one-octet header prefix followed by up to 64 Literal (name,value) Representations.



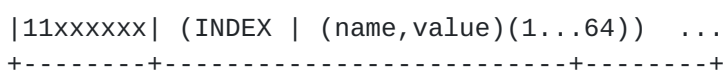
For instance:

0x40 0x01 0x61 0x01 0x62
Specifies a single header with name "a" and a UTF-8 String value of "b" is to be handled as an Indexed header (it will be added to the header table).

0x40 0x21 0x61 0x03
Specifies a single header with name "a" and Integer value of 3 is to be handled as an Indexed header (it will be added to the header table).

3.5. Indexed Literal Replacement Representation

The serialization of a group of Indexed Literal representations consists of the one-octet header prefix followed by up to 64 single octet index references identifying an existing header table entry followed by the new Literal (name,value) representation meant to replace it.



For instance:

0xC0 0x03 0x01 0x61 0x01 0x62
Specifies that a single header with name "a" and a UTF-8 String value of "b" is to replace the existing (name,value) entry in the header table located at index position #3.

0xC0 0x03 0x21 0x61 0x03
Specifies that a single header with name "a" and Integer value of 3 is to replace the existing (name,value) entry in the header table located at index position #3.

4. Unsigned Variable Length Integer Syntax

Unsigned integers are encoded as defined in [I-D.ietf-httpbis-header-compression].

5. Security Considerations

TBD

6. References

6.1. Normative References

[I-D.ietf-httpbis-header-compression]
Peon, R. and H. Ruellan, "HTTP/2.0 Header Compression", draft-ietf-httpbis-header-compression-01 (work in progress), July 2013.

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

6.2. Informational References

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

Appendix A. Initial Header Table Entries

Index	Name	Value	Type
0	:scheme	http	Text
1	:scheme	https	Text
2	:host		
3	:path	/	
4	:method	GET	Text
5	accept		
6	accept-charset		
7	accept-encoding		
8	accept-language		
9	cookie		
10	if-modified-since		
11	keep-alive		
12	user-agent		
13	proxy-connection		
14	referer		
15	accept-datetime		
16	authorization		
17	allow		
18	cache-control		

19	connection			
20	content-length			
21	content-md5			
22	content-type			
23	date			
24	expect			
25	from			
26	if-match			
27	if-none-match			
28	if-range			
29	if-unmodified-since			
30	max-forwards			
31	pragma			
32	proxy-authorization			
33	range			
34	te			
35	upgrade			
36	via			
37	warning			
38	:status	200	Integer	
39	age			
40	cache-control			
41	content-length			
42	content-type			
43	date			
44	etag			
45	expires			
46	last-modified			
47	server			
48	set-cookie			
49	vary			
50	via			
51	access-control-allow-origin			
52	accept-ranges			
53	allow			
54	connection			
55	content-disposition			
56	content-encoding			
57	content-language			
58	content-location			
59	content-md5			
60	content-range			
61	link			
62	location			
63	p3p			
64	pragma			
65	proxy-authenticate			
66	refresh			

67	retry-after			
68	strict-transport-security			
69	trailer			
70	transfer-encoding			
71	warning			
72	www-authenticate			
73	user-agent			

Appendix B. 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	Can be represented as

	Text	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 C. Example

C.1. First Header Set:

The first header set to represent is the following:

```
:path: /my-example/index.html
user-agent: my-user-agent
x-my-header: first
```

The header table is prefilled as defined in Appendix A, however, none of the values represented in the initial set can be found in the header table. All headers, then, are encoding using the Indexed Literal Representation:

```
43 00 03 16 2f 6d 79 2d 65 78
61 6d 70 6c 65 2f 69 6e 64 65
78 2e 68 74 6d 6c 00 49 6d 79
2d 75 73 65 72 2d 61 67 65 6e
74 0b 78 2d 6d 79 2d 68 65 61
64 65 72 05 66 69 72 73 74
```

Three new entries are added to the header table:

Index	Name	Value
74	:path	/my-example/index.html
75	user-agent	my-user-agent
76	x-my-header	first

C.2. Second Header Set:

The second header set to represent is the following:

```
:path: /my-example/resources/script.js
user-agent: my-user-agent
x-my-header: second
```

Comparing this second header set to the first, we see that the :path and x-my-header headers have new values, while the user-agent value remains unchanged. For the sake of the example let's encode the :path and x-my-header headers using Indexed Literal Replacement representations. The user-agent header will be encoded as an Indexed Representation.

```
80 4b a3 4a 00 4a 1f 2f 6d 79
2d 65 78 61 6d 70 6c 65 2f 72
65 73 6f 75 72 63 65 73 2f 73
63 72 69 70 74 2e 6a 73 00 4c
06 73 65 63 6f 6e 64
```

Items #74 and #76 added by the previous header set are replaced:

Index	Name	Value
74	:path	/my-example/resources/script.js
75	user-agent	my-user-agent
76	x-my-header	second

C.3. Third Header Set:

Let's suppose a third header set that is identical to the second is sent:

82 4b 4c 4d

Author's Address

James M Snell

Email: jasnell@gmail.com