Network Working Group J. Snell
Internet-Draft
Intended status: Informational August 14, 2013
Expires: February 15, 2014

HTTP/2.0 Discussion: Stored Header Encoding
draft-snell-httpbis-bohe-13

Abstract

This memo describes a proposed alternative optimized encoding for ordered sets of header field (name,value) pairs in HTTP/2 HEADERS and PUSH_PROMISE frames.

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 February 15, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Stored Header Encoding

The Stored Header Encoding is a proposed alternative "compressed header encoding" for HTTP/2.0 that offers reasonably good compression ratios, support for a range of compressor strategies, efficient value type codecs, constrained state requirements, routing-friendly header value ordering, and easier implementation relative to the current

header compression proposal.

2. Model

A "header" is a (name,value) pair. The name is a sequence of lower-case ASCII characters. The value is either an HTTP 1.1 defined field-value (see [I-D.ietf-httpbis-p1-messaging]), a sequence of UTF-8 encoded octets, an integer, a timestamp, or an opaque sequence of binary octets.

The compressor and decompressor each maintain a synchronized cache of up to 256 headers. Every header stored in the cache is referenced by an 8-bit identifier ranging from 0x00-FF (inclusive).

The cache is managed in a "least recently written" manner, 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 can be reused.

The specific index position to which a header is assigned is determined by the encoder using any algorithm the encoder determines to be appropriate. If a specific index position already has an assigned header, the existing header is replaced.

The maximum total size of the cache can be limited by the decompressor using the SETTINGS_MAX_BUFFER_SIZE setting. Each stored header contributes a certain number of octets to the total accumulated size of the cache. If adding a new header to the cache will cause the total size to grow beyond the set limit, the least-recently written items are removed in the order written until enough space is available to add the new item. Clearing existing items does not change the index positions of the remaining items in the cache.

The SETTINGS_MAX_BUFFER_SIZE setting has an initial default value of 4096 bytes. The decompressor can establish a new maximum buffer size at any time, possibly causing header (name,value) pairs to be evicted from the cache if the newly established limit is less than the current total size.

The decompressor can disable use of the storage cache completely by setting the SETTINGS_MAX_BUFFER_SIZE setting to 0, forcing the cache to empty completely and making it impossible to add new headers.

The size of a header is calculated as: The number of octets required for the name plus the number of octets required for the value plus 32-octets 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 octets required to represent the value using a 5-bit prefix.

- Legacy (HTTP/1.1) values are measured by the number of octets required to represent the value.

- 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 cache.

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

   * Otherwise, move to step #2.

2. Determine if a header (name,value) pair with the same name already exists in the cache. 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 assigned to the cache.

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

   * Otherwise, assign an index position for the header (name,value) pair and 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:

- 10 - Indexed
- 00 - Non-Indexed Literal
- 01 - Indexed Literal

Headers can be encoded into groups of up to 64 instances. Each group is prefixed by a single octet. The two most significant bits of this prefix identify the representation type, the six remaining bits specify the number of instances, with 000000 indicating a single instance and 111111 indicating 64.

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 cache 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, assign it to the specified position in the cache and add it to the working set.

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

When a single header name is used multiple times with different values, the order in which those values are serialized and processed is significant. The working set created by the decoding process above MUST preserve the ordering of those values as received.

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 sequence of ASCII octets or as the cache index position of another existing header sharing the same name, and
- 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, five of which are defined by this specification. The two remaining types are reserved for future use. The currently defined value types are:

UTF-8 (000)

Integer (001)

Timestamp (010)

Legacy (100)

Opaque Binary (111)

Of the five types, the Legacy type is reserved for encoding header values conforming to the field-value construct defined by [I-D.ietf-httpbis-p1-messaging], and is used specifically for backwards compatibility with header fields that have not yet been updated to use a more specific type value (see Appendix B).

If the name is encoded using an index reference to another existing (name,value) pair in the cache, the remaining five least significant bits of the first octet are set to zero and the next octet identifies the referenced cache index position. This octet MUST NOT reference a cache index position that is not currently assigned.

If the name is encoded as a sequence of ASCII octets, the number of octets 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, followed by the sequence of ASCII octets conforming to the following header-name construct:

Header name ABNF:

```
header-name = [":"] 1*header-char

header-char = "!" / "#" / "$" / "%" / "&" / "'" /
              "*" / "+" / "-" / "." / "^" / "_" /
              "`" / "|" / "~" / DIGIT / LOWERALPHA

LOWERALPHA = %x61-7A
```

The encoding of the value depends on the value type.

UTF-8:
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 octets.

Integer
Integer values ranging from 0 to 2^64-1 are encoded as unsigned variable length integers with a 0-bit prefix. Negative or fractional numbers cannot be represented.

Timestamp

Timestamps 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.

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

Opaque
: 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.

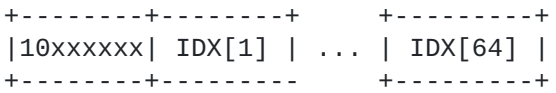
3.1.1. Dealing with invalid name or value encodings

Implementations encountering invalid name or value encodings MUST signal an error and terminate processing of the header block. Examples of such errors include:

- Header names that include any octets not explicitly permitted by the above header-name ABNF construction;
- UTF-8 values that include a byte order mark, over-long or invalid octet sequences, or octets representing invalid Unicode codepoints;
- Integer or Date-Time values that encode numbers strictly larger than 2^64-1;

3.2. Indexed Representation

The serialization of an Indexed Representation consists of a single octet prefix followed by up to 64 single-octet cache index position references.



For instance:

0x80 0x00
: References item #0 from the cache.

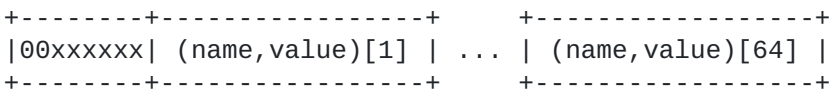
0x81 0x00 0x01

References items #0 and #1 from the cache.

Indexed Representations do not cause the cache state to be modified in any way. If an Indexed References specifies an index position that has not yet been assigned 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 a single-octet prefix followed by up to 64 Literal (name,value) Representations.

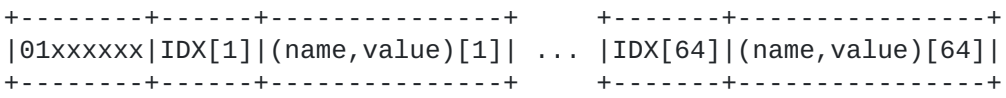


For instance:

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

3.4. Indexed Literal Representation

The serialization of a group of Indexed Literal representations consists of a single-octet prefix followed by up to 64 (index position, Literal (name,value) representation) pairs.



For instance:

0x40 0x03 0x01 0x61 0x01 0x62
Specifies that a single header with name "a" and a UTF-8 String value of "b" is to be assigned to the cache at index position #3.

0x40 0x03 0x21 0x61 0x04
Specifies that a single header with name "a" and Integer value of 3 is to be assigned to the cache at index position #4.

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.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

6.2. Informational References

[I-D.ietf-httpbis-p1-messaging]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", draft-ietf-httpbis-p1-messaging-23 (work in progress), July 2013.

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

Appendix A. Initial Cache 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

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

o All header fields MUST be explicitly defined to use the new header types. All existing HTTP/1 header fields will continue to be represented in conformance to the field-value construct defined by [I-D.ietf-httpbis-p1-messaging] unless their specific definitions are updated. Such fields MUST specify the Legacy value type when serialized. The HTTP/2 specification would update the definitions of specific known header fields (e.g. content-length, date, if-modified-since, etc).

o For translation to HTTP/1.1, header fields that use the typed codecs will have specific normative transformations defined.

* UTF-8 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.

* Opaque fields will be Base64-encoded.

* Legacy fields are passed through untranslated.

- There will be no normative transformation from legacy values into the typed codecs. Implementations are free to apply transformation where they determine it to be appropriate, but it will be perfectly acceptable for an implementation to pass a text value through as a Legacy type even if it is known that a given header has a typed codec equivalent.

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

```
43 4a 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 4B 00 49
6d 79 2d 75 73 65 72 2d 61 67
65 6e 74 4c 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 cache:

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.

```
80 4b 41 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 4c 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