Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 3, 2013

J. Snell
August 2, 2012

HTTP/2.0 Discussion: Binary Optimized Header Encoding
draft-snell-httpbis-bohe-00

Abstract

This memo describes a proposed alternative encoding for headers within SPDY SYN_STREAM, SYN_REPLY and HEADERS frames.

Status of this Memo

This Internet-Draft is submitted to IETF 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."

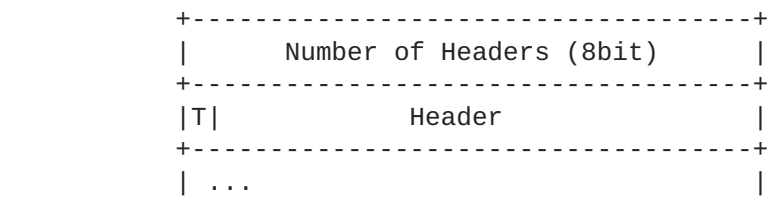
This Internet-Draft will expire on February 3, 2013.

Table of Contents

1. Binary Optimized Header Encoding

Binary Optimized Header Encoding is a proposed alternative serialization for headers within SPDY SYN_STREAM, SYN_REPLY and HEADERS frames that is designed to optimize generation, consumption and processing of the most commonly used HTTP headers.

Alternate Header Block Serialization:



Within the existing SPDY Header Block, a 32-bit value is used to identify the number of headers within the block. For all practical purposes, it is exceedingly unlikely that a single block of headers will contain anywhere near 4,294,967,295 distinct headers. Obviously a 32-bit integer is significant overkill for this purpose. As an alternative, an 8-bit value suggested.

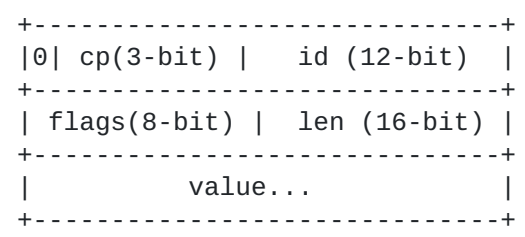
The header block consists of zero or more distinct headers, each of which begin with a single Type-bit whose value indicates the type of header. There are two header types: Registered and Extension. The specific structure of the remaining header depends on the header type.

The header block MAY be compressed as described within [draft-montenegro-httpbis-speed-mobility-02].

1.1. Registered Headers

Registered Headers are well-known and well-defined headers for which there is a published RFC and IANA registration. Each is assigned an unsigned 12-bit integer identifier and an unsigned 3-bit integer codepage. If the codepage is 0, the implication is that the header MUST be understood in order for the request or response message to be handled properly. Codepages 1-5 represent "MUST-IGNORE" headers; that is, such headers MUST be ignored by processors if they are unrecognized by the processing application. Codepages 6 and 7 are reserved for "Private Use", with Codepage 6 being used for "MUST UNDERSTAND PRIVATE USE" headers.

The structure of Registered Headers:



The first single bit within the structure is the Type-bit. When this bit is off, the header is a Registered Header.

The next three bits identify the headers codepage. The value is interpreted as an unsigned integer in the range 0-7.

The next twelve bits specify the header's specific numeric identifier within the codepage.

Following the identifier are 8 reserved flag bits.

- Bit 0x1 indicates that the header value contains UTF-8 encoded character content. If the bit is not set, the value is assumed to contain non-character-based binary data.
- Bit 0x2 indicates that the header specifies multiple NUL (0) separated values. When set, processors MUST treat NUL (0) octets within the value as a delimiter and not as part of the value itself.

The remaining content of the structure consists of a 16-bit unsigned integer specifying the remaining length of the header value. The value MAY be zero length.

The minimum length of a registered header is 5-octets (40-bits).

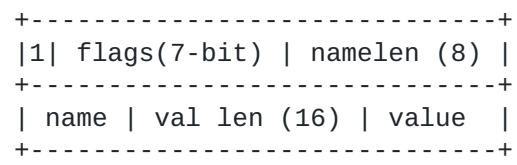
When Flag 0x2 is set, the header may contain multiple values separated by a single NUL (0) byte. Each distinct value MUST NOT be zero-length. When Flag 0x2 is not set, and Flag 0x1 is also not set, NUL bytes contained within the value are to be considered part of the value. The use of NUL bytes within character-based values is not permitted except when used as a delimiter separating multiple values.

When multiple values are included, the value length field MUST specify the total length, in octets, of all values plus the number of NUL (0) byte separators. For example, for a header value consisting of the two strings "foo" and "bar", the total value length would be 7.

1.2. Extension Headers

Extension Headers are simple name+value pairs essentially as they exist today, but with a number of important modifications.

The structure of Extension Headers



The first single bit is the Type-bit. When this bit is on, the header is an Extension Header.

The next seven bits are reserved flags.

- Bit 0x1 indicates that the header value contains UTF-8 encoded character content. If the bit is not set, the value is assumed to contain non-character-based binary data.
- Bit 0x2 indicates that the header specifies multiple NUL (0) separated values. When set, processors MUST treat NUL (0) octets within the value as a value-separated and not as part of the value itself.

The next 8-bits specify the length in octets of the ASCII-encoded header name as unsigned integer, followed by the name itself. The name MUST conform to the field-name construction as defined in [draft-ietf-httpbis-p1-messaging-2].

The length of the remaining value is specified as an unsigned 16-bit integer, followed by the value itself. Zero length values are permitted.

When Flag 0x2 is set, the header may contain multiple values separated by a single NUL (0) byte. Each distinct value MUST be zero-length.

When multiple values are included, the value length field MUST specify the total length, in octets, of all values plus the number of NUL (0) byte separators. For example, for a header value consisting of the two strings "foo" and "bar", the total value length would be 7.

1.3. Binary vs. Character Values

Specific header values can be encoded as either a stream of binary octets or as UTF-8 encoded character data.

For example, within the existing SPDY specification, the HTTP Version is represented as a header using the field-name ":version" with the version number represented as an ASCII string, consuming 19-bytes in all.

Version Header using the existing SPDY encoding:

```
  00 00 00 08 3a 76 65 72  |....:ver|
  73 69 6f 6e 00 00 00 03  |sion....|
  31 2e 31                 |2.0|
```

Using the Binary Optimized Header Encoding, this can be reduced to a compact 7 or 8 bytes using either binary or character data:

Version Header using Character Data:

```
  00 01 01 00 03 31 2e 31  |.....2.0|
```

Version Header using Binary Data:

```
  00 01 00 00 02 02 00     |.......|
```

Likewise, SPDY uses a ":method" header to specify the HTTP Method used for a particular request, with the value represented as an ASCII string, consuming 18 bytes for GET requests.

Method Header using the existing SPDY encoding:

```
  00 00 00 07 3a 6d 65 74  |....:met|
  68 6f 64 00 00 00 03 47  |hod....G|
  45 54                    |GET|
```

Using optimized encoding, this can be reduced to a compact 6 or 8 bytes using either binary or character data:

Method Header using Character Data:

```
  00 02 01 00 03 47 45 54  |.....GET|
```

Method Header using Binary Data, assuming the value 0x1 is defined to represent the GET method:

```
  00 02 00 00 01 01       |......|
```

There are many headers used within HTTP applications for which binary encodings would be difficult or unnecessary. For those, utilizing the character encoding option would be appropriate. With some work it should be possible to define optimized binary encodings for many

of the existing complex headers.

1.4. Example Headers

Assume the following registered headers:

HTTP Header	Codepage	ID
Version	0	1
Method	0	2
Host	0	3
Path (Request URI)	0	4
Accept-Language	1	1

And the following values representing known HTTP Methods:

Method	Value
GET	1
POST	2
PUT	3
DELETE	4
PATCH	5
HEAD	6
OPTIONS	7
CONNECT	8

The Version header can be encoded as (7-bytes):

```
  00 01 00 00 02 02 00    |.......|
```

The GET Method header can be encoded as (6-bytes):

```
  00 02 00 00 01 01       |......|
```

The Host Header can be encoded as (20-bytes):

```
  00 03 01 00 0f 77 77 77 |.....www|
  2e 65 78 61 6d 70 6c 65 |.example|
  2e 6f 72 67             |.org|
```

A simple Accept-Lang header would be encoded as (10-bytes):

```
  10 01 01 00 05 65 6e 2d |.....en-|
  55 53                   |US|
```

A Path header encoding the request URI (45-bytes):

```
  00 04 01 00 28 2f 74 68  |...../th|
  69 73 2f 69 73 2f 74 68  |is/is/th|
  65 2f 72 65 71 75 65 73  |e/reques|
  74 3f 69 73 3d 69 74 26  |t?is=it&|
  6e 6f 74 3d 62 65 61 75  |not=beau|
  74 69 66 75 6c           |tiful|
```

The combined serialization of the five headers into a single block requires a total of 89 bytes. By comparison, the equivalent serialization using the existing SPDY encoding requires 150 bytes sans compression (28 bytes of which are wasted by the unnecessary use of int32).

The equivalent SPDY encoding:

```
  00 00 00 05 00 00 00 08  |........|
  3a 76 65 72 73 69 6f 6e  |:version|
  00 00 00 03 31 2e 31 00  |....1.1.|
  00 00 07 3a 6d 65 74 68  |...:meth|
  6f 64 00 00 00 03 47 45  |od....GE|
  54 00 00 00 05 3a 68 6f  |T....:ho|
  73 74 00 00 00 0f 77 77  |st....ww|
  77 2e 65 78 61 6d 70 6c  |w.exampl|
  65 2e 6f 72 67 00 00 00  |e.org...|
  0f 41 63 63 65 70 74 2d  |.Accept-|
  4c 61 6e 67 75 61 67 65  |Language|
  00 00 00 05 65 6e 2d 55  |....en-U|
  53 00 00 00 05 3a 70 61  |S....:pa|
  74 68 00 00 00 28 2f 74  |th..../t|
  68 69 73 2f 69 73 2f 74  |his/is/t|
  68 65 2f 72 65 71 75 65  |he/reque|
  73 74 3f 69 73 3d 69 74  |st?is=it|
  26 6e 6f 74 3d 62 65 61  |&not=bea|
  75 74 69 66 75 6c        |utiful|
```

Note that the equivalent information encoded within an HTTP/1.1 request message requires 102 bytes.

2. Security Considerations

TBD

3. Normative References

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

Appendix A. Additional Examples

Assuming the following (intentionally incomplete) header registrations adapted from the existing http-bis specifications.

HTTP Header	Codepage	ID
Version	0	1
Method	0	2
Host	0	3
Path (Request URI)	0	4
Status	0	5
Status-Text	0	6
Content-Length	0	7
Content-Type	0	8
Content-Encoding	0	9
Expect	0	10
Location	0	11
Last-Modified	0	12
ETag	0	13
If-Match	0	14
If-None-Match	0	15
If-Modified-Since	0	16
If-Unmodified-Since	0	17
Age	0	18
Cache-Control	0	19
Expires	0	20
Vary	0	21
Accept	1	1
Accept-Language	1	2
Accept-Charset	1	3
Accept-Encoding	1	4
Allow	1	5
Content-Language	1	6
Content-Location	1	7
Date	1	8
From	1	9
Warning	1	10

And the following values representing known HTTP Methods:

Method	Value
GET	1
POST	2
PUT	3
DELETE	4
PATCH	5
HEAD	6
OPTIONS	7
CONNECT	8

We can derive the following optimized encodings:

Version Header:

```
  00 01 00 00 02 02 00    |.......|
```

Method Header (GET Request)

```
  00 02 00 00 01 01       |......|
```

Method Header (PATCH Request)

```
  00 02 00 00 01 05       |......|
```

Method Header (Custom "FOO" Method)

```
  00 02 01 00 03 46 4F 4F |.....FOO|
```

Host Header:

```
  00 03 01 00 0f 77 77 77 |.....www|
  2e 65 78 61 6d 70 6c 65 |.example|
  2e 6f 72 67             |.org|
```

Representation of HTTP Response Status ("200 OK"):

```
  00 05 00 00 01 C8 00 06 |........|
  01 00 02 4F 4B          |...OK|
```

The status above is represented as two separate headers, one containing the status code, the other containing the status text.

Content-Length Header (value encoded as uint32):

```
  00 07 00 00 04 00 00 00 |........|
  C8                      |.|
```

Content-Type Header:

```
  00 08 01 00 0A 69 6d 61 |.....ima|
  67 65 2f 6a 70 65 67    |ge/jpeg|
```

Expect Header (Expect: 100):

```
  00 0A 00 00 01 64       |......|
```

Last-Modified (Using RFC3339 Format):

```
  00 0C 01 00 19 32 30 31 |.....201|
  32 2d 30 38 2d 30 31 54 |2-08-01T|
  30 34 3a 32 33 3a 31 32 |04:23:12|
  2e 31 32 33 34 5a       |.1234Z|
```

ETag (Strong Entity-Tag, String-format):

```
  00 0D 01 00 07 22 61 62 |....."ab|
  63 64 65 22             |cde"|
```

If-None-Match:

```
  00 0F 01 00 07 22 61 62 |....."ab|
  63 64 65 22             |cde"|
```

If-None-Match (Multiple values)

```
  00 0F 03 00 0F 22 61 62 |....."ab|
  63 64 65 22 00 22 61 62 |cde"."ab|
  63 64 66 22             |cdf"|
```

Allow (GET, POST, FOO):

```
  10 05 02 00 07 01 00 02 |........|
  00 46 4f 4f             |.FOO|
```

Author's Address

James M Snell

Email: jasnell@gmail.com