Binary Representation of HTTP Messages

Abstract

   This document defines a binary format for representing HTTP
   messages.

About This Document

   This note is to be removed before publishing as an RFC.

   Status information for this document may be found at https://
   datatracker.ietf.org/doc/draft-ietf-httpbis-binary-message/.

   Discussion of this document takes place on the HTTP Working Group
   mailing list (mailto:ietf-http-wg@w3.org), which is archived at
   https://lists.w3.org/Archives/Public/ietf-http-wg/. Working Group
   information can be found at https://httpwg.org/.

   Source for this draft and an issue tracker can be found at https://
   github.com/httpwg/http-extensions/labels/binary-messages.

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 https://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 30 October 2022.

Table of Contents

1. Introduction

   This document defines a simple format for representing an HTTP
   message ([HTTP]), either request or response. This allows for the
   encoding of HTTP messages that can be conveyed outside of an HTTP

protocol. This enables the transformation of entire messages, including the application of authenticated encryption.

The design of this format is informed by the framing structure of HTTP/2 ([H2]) and HTTP/3 ([H3]). Rules for constructing messages rely on the rules defined in HTTP/2, but the format itself is distinct; see Section 6.

This format defines message/bhttp, a binary alternative to the message/http content type defined in [MESSAGING]. A binary format permits more efficient encoding and processing of messages. A binary format also reduces exposure to security problems related to processing of HTTP messages.

Two modes for encoding are described:

  *a known-length encoding includes length prefixes for all major
   message components; and

  *an indefinite-length encoding enables efficient generation of
   messages where lengths are not known when encoding starts.

This format is designed to convey the semantics of valid HTTP messages as simply and efficiently as possible. It is not designed to capture all of the details of the encoding of messages from specific HTTP versions ([MESSAGING], [H2], [H3]). As such, this format is unlikely to be suitable for applications that depend on an exact recording of the encoding of messages.

2. Conventions and Definitions

The key words **"MUST"**, **"MUST NOT"**, **"REQUIRED"**, **"SHALL"**, **"SHALL NOT"**, **"SHOULD"**, **"SHOULD NOT"**, **"RECOMMENDED"**, **"NOT RECOMMENDED"**, **"MAY"**, and **"OPTIONAL"** in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses terminology from HTTP ([HTTP]) and notation from QUIC (Section 1.3 of [QUIC]).

3. Format

Section 6 of [HTTP] defines five distinct parts to HTTP messages. A framing indicator is added to signal how these parts are composed:

  1. Framing indicator. This format uses a single integer to
     describe framing, which describes whether the message is a
     request or response and how subsequent sections are formatted;
     see Section 3.3.

2. For a response, any number of interim responses, each
    consisting of an informational status code and header section.

3. Control data. For a request, this contains the request method
    and target. For a response, this contains the status code.

4. Header section. This contains zero or more header fields.

5. Content. This is a sequence of zero or more bytes.

6. Trailer section. This contains zero or more trailer fields.

7. Optional padding. Any amount of zero-valued bytes.

All lengths and numeric values are encoded using the variable-length
integer encoding from Section 16 of [QUIC]. Integer values do not
need to be encoded on the minimum number of bytes necessary.

3.1. Known Length Messages

A message that has a known length at the time of construction uses
the format shown in Figure 1.

```
Message with Known-Length {
  Framing Indicator (i) = 0..1,
  Known-Length Informational Response (..) ...,
  Control Data (..),
  Known-Length Field Section (..),
  Known-Length Content (..),
  Known-Length Field Section (..),
  Padding (..),
}

Known-Length Field Section {
  Length (i),
  Field Line (..) ...,
}

Known-Length Content {
  Content Length (i),
  Content (..)
}

Known-Length Informational Response {
  Informational Response Control Data (..),
  Known-Length Field Section (..),
}
```

Figure 1: Known-Length Message

That is, a known-length message consists of a framing indicator, a block of control data that is formatted according to the value of the framing indicator, a header section with a length prefix, binary content with a length prefix, and a trailer section with a length prefix.

For a known-length encoding, the length prefix on field sections and content is a variable-length encoding of an integer. This integer is the number of bytes in the field section or content.

Response messages that contain informational status codes result in a different structure; see Section 3.5.1. Note that while the Known-Length Informational Response field is shown in Figure 1, it can only appear in response messages.

Fields in the header and trailer sections consist of a length-prefixed name and length-prefixed value; see Section 3.6.

The format allows for the message to be truncated before any of the length prefixes that precede the field sections or content; see Section 3.8.

The variable-length integer encoding means that there is a limit of 2^62-1 bytes for each field section and the message content.

3.2. Indeterminate Length Messages

A message that is constructed without encoding a known length for each section uses the format shown in Figure 2:

```
Indeterminate-Length Message  {
  Framing Indicator (i) = 2..3,
  Indeterminate-Length Informational Response (..) ...,
  Control Data (..),
  Indeterminate-Length Field Section (..),
  Indeterminate-Length Content (..) ...,
  Indeterminate-Length Field Section (..),
  Padding (..),
}

Indeterminate-Length Content {
  Indeterminate-Length Content Chunk (..) ...,
  Content Terminator (i) = 0,
}

Indeterminate-Length Content Chunk {
  Chunk Length (i) = 1..,
  Chunk (..)
}

Indeterminate-Length Field Section {
  Field Line (..) ...,
  Content Terminator (i) = 0,
}

Indeterminate-Length Informational Response {
  Informational Response Control Data (..),
  Indeterminate-Length Field Section (..),
}
```

Figure 2: Indeterminate-Length Message

That is, an indeterminate length consists of a framing indicator, a
block of control data that is formatted according to the value of
the framing indicator, a header section that is terminated by a zero
value, any number of non-zero-length chunks of binary content, a
zero value, and a trailer section that is terminated by a zero
value.

The indeterminate-length encoding only uses length prefixes for
content blocks. Multiple length-prefixed portions of content can be
included, each prefixed by a non-zero Chunk Length integer
describing the number of bytes in the block. The Chunk Length is
encoded as a variable-length integer.

Each Field Line in an Indeterminate-Length Field Section starts with
a Name Length field. An Indeterminate-Length Field Section ends with
a Content Terminator field. The zero value of the Content Terminator
distinguishes it from the Name Length field, which cannot contain a
value of 0.

Response messages that contain informational status codes result in a different structure; see Section 3.5.1. Note that while the Indeterminate-Length Informational Response field is shown in Figure 2, it can only appear in response messages.

Indeterminate-length messages can be truncated in a similar way as known-length messages; see Section 3.8.

Indeterminate-length messages use the same encoding for field lines as known-length messages; see Section 3.6.

3.3. Framing Indicator

The start of each binary message is a framing indicator that is a single integer that describes the structure of the subsequent sections. The framing indicator can take just four values:

  *A value of 0 describes a request of known length.

  *A value of 1 describes a response of known length.

  *A value of 2 describes a request of indeterminate length.

  *A value of 3 describes a response of indeterminate length.

Other values cause the message to be invalid; see Section 4.

3.4. Request Control Data

The control data for a request message contains the method and request target. That information is encoded as an ordered sequence of fields: Method, Scheme, Authority, Path. Each of these fields is prefixed with a length.

The values of these fields follow the rules in HTTP/2 (Section 8.3.1 of [H2]) that apply to the :method, :scheme, :authority, and :path pseudo-header fields respectively. However, where the :authority pseudo-header field might be omitted in HTTP/2, a zero-length value is encoded instead.

The format of request control data is shown in Figure 3.

```
Request Control Data {
  Method Length (i),
  Method (..),
  Scheme Length (i),
  Scheme (..),
  Authority Length (i),
  Authority (..),
  Path Length (i),
  Path (..),
}
```

Figure 3: Format of Request Control Data

3.5. Response Control Data

The control data for a response message consists of the status code.
The status code is encoded as a variable length integer, not a
length-prefixed decimal string.

The format of final response control data is shown in Figure 4.

```
Final Response Control Data {
  Status Code (i) = 200..599,
}
```

Figure 4: Format of Final Response Control Data

3.5.1. Informational Status Codes

Responses that include information status codes (see Section 15.2 of
[HTTP]) are encoded by repeating the response control data and
associated header section until the final status code is encoded.

The format of the informational response control data is shown in
Figure 5.

```
Informational Response Control Data {
  Status Code (i) = 100..199,
}
```

Figure 5: Format of Informational Response Control Data

A response message can include any number of informational responses
that precede a final status code. These convey an information status
code and a header block.

If the response control data includes an informational status code
(that is, a value between 100 and 199 inclusive), the control data
```

is followed by a header section (encoded with known- or
indeterminate- length according to the framing indicator) and
another block of control data. This pattern repeats until the
control data contains a final status code (200 to 599 inclusive).

## 3.6.  Header and Trailer Field Lines

Header and trailer sections consist of zero or more field lines; see
Section 5 of [HTTP]. The format of a field section depends on
whether the message is known- or intermediate-length.

Each field line includes a name and a value. Both the name and value
are length-prefixed sequences of bytes. The field name length is at
least one byte. The format of a field line is shown in Figure 6.

```
Field Line {
  Name Length (i) = 1..,
  Name (..),
  Value Length (i),
  Value (..),
}
```

Figure 6: Format of a Field Line

For field names, byte values that are not permitted in an HTTP field
name cause the message to be invalid; see Section 5.1 of [HTTP] for
a definition of what is valid and Section 4 for handling of invalid
messages. A recipient **MUST** treat a message that contains field
values that would cause an HTTP/2 message to be malformed according
to Section 8.2.1 of [H2] as invalid; see Section 4.

The same field name can be repeated in multiple field lines; see
Section 5.2 of [HTTP] for the semantics of repeated field names and
rules for combining values.

Fields that relate to connections (Section 7.6.1 of [HTTP]) cannot
be used to produce the effect on a connection in this context. These
fields **SHOULD** be removed when constructing a binary message.
However, they do not cause a message to be invalid (Section 4);
permitting these fields allows a binary message to capture the
content of a messages that are exchanged in a protocol context.

Like HTTP/2, this format has an exception for the combination of
multiple instances of the Cookie field. Instances of fields with the
ASCII-encoded value of cookie are combined using a semicolon octet
(0x3b) rather than a comma; see Section 8.2.3 of [H2].

3.7. Content

The content of messages is a sequence of bytes of any length. Though a known-length message has a limit, this limit is large enough that it is unlikely to be a practical limitation. There is no limit to the size of content in an indeterminate length message.

3.8. Padding and Truncation

Messages can be padded with any number of zero-valued bytes. Non-zero padding bytes cause a message to be invalid (see Section 4). Unlike other parts of a message, a processor **MAY** decide not to validate the value of padding bytes.

Truncation can be used to reduce the size of messages that have no data in trailing field sections or content. If the trailers of a message is empty, it **MAY** be omitted by the encoder in place of adding a length field equal to zero. An encoder **MAY** omit empty content in the same way if the trailers are also empty. A message that is truncated at any other point is invalid; see Section 4.

Decoders **MUST** treat missing truncated fields as equivalent to having been sent with the length field sent to zero.

Padding is compatible with truncation of empty parts of the messages. Zero-valued bytes will be interpreted as zero-length part, which is semantically equivalent to the part being absent.

4. Invalid Messages

This document describes a number of ways that a message can be invalid. Invalid messages **MUST NOT** be processed except to log an error and produce an error response.

The format is designed to allow incremental processing. Implementations need to be aware of the possibility that an error might be detected after performing incremental processing.

5. Examples

This section includes example requests and responses encoded in both known-length and indefinite-length forms.

5.1. Request Example

The example HTTP/1.1 message in Figure 7 shows the content in the message/http format.

Valid HTTP/1.1 messages require lines terminated with CRLF (the two bytes 0x0a and 0x0d). For simplicity and consistency, the content of

these examples is limited to text, which also uses CRLF for line
endings.

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
```

Figure 7: Sample HTTP Request

This can be expressed as a binary message (type message/bhttp) using
a known-length encoding as shown in hexadecimal in Figure 8. Figure
8 view includes some of the text alongside to show that most of the
content is not modified.

```
00034745 54056874 74707300 0a2f6865   ..GET.https../he
6c6c6f2e 74787440 6c0a7573 65722d61   llo.txt@l.user-a
67656e74 34637572 6c2f372e 31362e33   gent4curl/7.16.3
206c6962 6375726c 2f372e31 362e3320    libcurl/7.16.3
4f70656e 53534c2f 302e392e 376c207a   OpenSSL/0.9.7l z
6c69622f 312e322e 3304686f 73740f77   lib/1.2.3.host.w
77772e65 78616d70 6c652e63 6f6d0f61   ww.example.com.a
63636570 742d6c61 6e677561 67650665   ccept-language.e
6e2c206d 690000                       n, mi..
```

Figure 8: Known-Length Binary Encoding of Request

This example shows that the Host header field is not replicated in
the :authority field, as is required for ensuring that the request
is reproduced accurately; see Section 8.3.1 of [H2].

The same message can be truncated with no effect on interpretation.
In this case, the last two bytes - corresponding to content and a
trailer section - can each be removed without altering the semantics
of the message.

The same message, encoded using an indefinite-length encoding is
shown in Figure 9. As the content of this message is empty, the
difference in formats is negligible.

```
02034745 54056874 74707300 0a2f6865   ..GET.https../he
6c6c6f2e 7478740a 75736572 2d616765   llo.txt.user-age
6e743463 75726c2f 372e3136 2e33206c   nt4curl/7.16.3 l
69626375 726c2f37 2e31362e 33204f70   ibcurl/7.16.3 Op
656e5353 4c2f302e 392e376c 207a6c69   enSSL/0.9.7l zli
622f312e 322e3304 686f7374 0f777777   b/1.2.3.host.www
2e657861 6d706c65 2e636f6d 0f616363   .example.com.acc
6570742d 6c616e67 75616765 06656e2c   ept-language.en,
206d6900 00000000 00000000 00000000    mi............
```

Figure 9: Indefinite-Length Binary Encoding of Request

This indefinite-length encoding contains 10 bytes of padding. As two
additional bytes can be truncated in the same way as the known-
length example, anything up to 12 bytes can be removed from this
message without affecting its meaning.

5.2. Response Example

Response messages can contain interim (1xx) status codes as the
message in Figure 10 shows. Figure 10 includes examples of
informational status codes defined in [RFC2518] and [RFC8297].

```
HTTP/1.1 102 Processing
Running: "sleep 15"

HTTP/1.1 103 Early Hints
Link: </style.css>; rel=preload; as=style
Link: </script.js>; rel=preload; as=script

HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain

Hello World! My content includes a trailing CRLF.
```

Figure 10: Sample HTTP Response

As this is a longer example, only the indefinite-length encoding is
shown in Figure 11. Note here that the specific text used in the
reason phrase is not retained by this encoding.

```
03406607 72756e6e 696e670a 22736c65   .@f.running."sle
65702031 35220040 67046c69 6e6b233c   ep 15".@g.link#<
2f737479 6c652e63 73733e3b 2072656c   /style.css>; rel
3d707265 6c6f6164 3b206173 3d737479   =preload; as=sty
6c65046c 696e6b24 3c2f7363 72697074   le.link$</script
2e6a733e 3b207265 6c3d7072 656c6f61   .js>; rel=preloa
643b2061 733d7363 72697074 0040c804   d; as=script.@..
64617465 1d4d6f6e 2c203237 204a756c   date.Mon, 27 Jul
20323030 39203132 3a32383a 35332047    2009 12:28:53 G
4d540673 65727665 72064170 61636865   MT.server.Apache
0d6c6173 742d6d6f 64696669 65641d57   .last-modified.W
65642c20 3232204a 756c2032 30303920   ed, 22 Jul 2009
31393a31 353a3536 20474d54 04657461   19:15:56 GMT.eta
67142233 34616133 38372d64 2d313536   g."34aa387-d-156
38656230 30220d61 63636570 742d7261   8eb00".accept-ra
6e676573 05627974 65730e63 6f6e74656   nges.bytes.conte
6e742d6c 656e6774 68023531 04766172   nt-length.51.var
790f4163 63657074 2d456e63 6f64696e   y.Accept-Encodin
670c636f 6e74656e 742d7479 70650a74   g.content-type.t
6578742f 706c6169 6e003348 656c6c6f   ext/plain.3Hello
20576f72 6c642120 4d792063 6f6e7465    World! My conte
6e742069 6e636c75 64657320 61207472   nt includes a tr
61696c69 6e672043 524c462e 0d0a0000   ailing CRLF.....
```

                 Figure 11: Binary Response including Interim Responses

   A response that uses the chunked encoding (see Section 7.1 of
   [MESSAGING]) as shown for Figure 12 can be encoded using indefinite-
   length encoding, which minimizes buffering needed to translate into
   the binary format. However, chunk boundaries do not need to be
   retained and any chunk extensions cannot be conveyed using the
   binary format; see Section 6.


```
HTTP/1.1 200 OK
Transfer-Encoding: chunked

4
This
6
 conte
13;chunk-extension=foo
nt contains CRLF.

0
Trailer: text
```

                      Figure 12: Chunked Encoding Example

Figure 13 shows this message using the known-length coding. Note
that the transfer-encoding header field is removed.


```
0140c800 1d546869 7320636f 6e74656e   .@...This conten
7420636f 6e746169 6e732043 524c462e   t contains CRLF.
0d0a0d07 74726169 6c657204 74657874   ....trailer.text
```

                 Figure 13: Known-Length Encoding of Response

6. Notable Differences with HTTP Protocol Messages

   This format is designed to carry HTTP semantics just like HTTP/1.1,
   HTTP/2, or HTTP/3 ([MESSAGING], [H2], [H3]). However, there are some
   notable differences between this format and the format used in an
   interactive protocol version.

   In particular, as a standalone representation, this format lacks the
   following features of the formats used in those protocols:

     *chunk extensions (Section 7.1.1 of [MESSAGING]) and transfer
      encoding (Section 6.1 of [MESSAGING]) from HTTP/1.1

     *generic framing and extensibility capabilities

     *field blocks other than a single header and trailer field block

     *carrying reason phrases in responses (Section 4 of [MESSAGING])

     *header compression ([HPACK], [QPACK])

     *framing of responses that depends on the corresponding request
      (such as HEAD) or the value of the status code (such as 204 or
      304); these responses use the same framing as all other messages

   Some of these features are also absent in HTTP/2 and HTTP/3.

   Unlike HTTP/2 and HTTP/3, this format uses a a fixed format for
   control data rather than using pseudo-fields. Messages are invalid
   (Section 4) if they contain fields
   named :method, :scheme, :authority, :path, or :status. Other pseudo-
   fields that are defined by protocol extensions **MAY** be included;
   pseudo-fields cannot be included in trailers (see Section 8.1 of
   [H2]). Field lines containing pseudo-fields **MUST** precede other field
   lines. A message that contains a pseudo-field after any other field
   is invalid; see Section 4.

   Note that while some messages - CONNECT or upgrade requests in
   particular - can be represented using this format, doing so serves

no purpose as these requests are used to affect protocol behavior, which this format cannot do without additional mechanisms.

7. "message/bhttp" Media Type

The message/bhttp media type can be used to enclose a single HTTP request or response message, provided that it obeys the MIME restrictions for all "message" types regarding line length and encodings.

Type name: message

Subtype name: bhttp

Required parameters: N/A

Optional parameters: None

Encoding considerations: only "8bit" or "binary" is permitted

Security considerations: see [Section 8](#)

Interoperability considerations: N/A

Published specification: this specification

Applications that use this media type: N/A

Fragment identifier considerations: N/A

Additional information:

   Magic number(s): N/A

   Deprecated alias names for this type: N/A

   File extension(s): N/A

   Macintosh file type code(s): N/A

Person and email address to contact for further information: see
   Authors' Addresses section

Intended usage: COMMON

Restrictions on usage: N/A

Author: see Authors' Addresses section

Change controller: IESG

8. Security Considerations

Many of the considerations that apply to HTTP message handling apply
to this format; see Section 17 of [HTTP] and Section 11 of
[MESSAGING] for common issues in handling HTTP messages.

Strict parsing of the format with no tolerance for errors can help
avoid a number of attacks. However, implementations still need to be
aware of the possibility of resource exhaustion attacks that might
arise from receiving large messages, particularly those with large
numbers of fields.

Implementations need to ensure that they aren't subject to resource
exhaustion attack from a maliciously crafted message. Overall, the
format is designed to allow for minimal state when processing
messages. However, producing a combined field value (Section 5.2 of
[HTTP]) for fields might require the commitment of resources. In
particular, combining might be necessary for the Cookie field when
translating this format for use in other contexts, such as use in an
API or translation to HTTP/1.1 [MESSAGING], where the recipient of
the field might not expect multiple values.

9. IANA Considerations

IANA is requested to add the "Media Types" registry at https://
www.iana.org/assignments/media-types with the registration
information in Section 7 for the media type "message/bhttp".

10. References

10.1. Normative References

[H2]   Thomson, M. and C. Benfield, "HTTP/2", Work in Progress,
           Internet-Draft, draft-ietf-httpbis-http2bis-07, 24
           January 2022, <https://datatracker.ietf.org/doc/html/
           draft-ietf-httpbis-http2bis-07>.

[H3]   Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/
           3)", Work in Progress, Internet-Draft, draft-ietf-quic-
           http-34, 2 February 2021, <https://datatracker.ietf.org/
           doc/html/draft-ietf-quic-http-34>.

[HTTP]   Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP
           Semantics", Work in Progress, Internet-Draft, draft-ietf-
           httpbis-semantics-19, 12 September 2021, <https://
           datatracker.ietf.org/doc/html/draft-ietf-httpbis-
           semantics-19>.

[MESSAGING] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/
           1.1", Work in Progress, Internet-Draft, draft-ietf-

httpbis-messaging-19, 12 September 2021, <https://
datatracker.ietf.org/doc/html/draft-ietf-httpbis-
messaging-19>.

[QUIC]   Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based
            Multiplexed and Secure Transport", RFC 9000, DOI
            10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/
            rfc/rfc9000>.

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
            Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/
            RFC2119, March 1997, <https://www.rfc-editor.org/rfc/
            rfc2119>.

[RFC8174]   Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
            2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
            May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

10.2. Informative References

[HPACK]   Peon, R. and H. Ruellan, "HPACK: Header Compression for
            HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015,
            <https://www.rfc-editor.org/rfc/rfc7541>.

[QPACK]   Krasic, C. '., Bishop, M., and A. Frindell, "QPACK:
            Header Compression for HTTP/3", Work in Progress,
            Internet-Draft, draft-ietf-quic-qpack-21, 2 February
            2021, <https://datatracker.ietf.org/doc/html/draft-ietf-
            quic-qpack-21>.

[RFC2518]   Goland, Y., Whitehead, E., Faizi, A., Carter, S., and D.
            Jensen, "HTTP Extensions for Distributed Authoring --
            WEBDAV", RFC 2518, DOI 10.17487/RFC2518, February 1999,
            <https://www.rfc-editor.org/rfc/rfc2518>.

[RFC8297]   Oku, K., "An HTTP Status Code for Indicating Hints", RFC
            8297, DOI 10.17487/RFC8297, December 2017, <https://
            www.rfc-editor.org/rfc/rfc8297>.

Acknowledgments

Authors' Addresses

Martin Thomson
Mozilla

Email: mt@lowentropy.net

Christopher A. Wood
Cloudflare

Email: caw@heapingbits.net