Structured Headers

Mark Nottingham, Fastly @mnot

HTTP headers are horrible.

```
Cache-Control: max-age=3600, must-revalidate
```

```
Accept: text/html;q=0.9, image/*
```

```
Cache-Control: max-age=3600
Cache-Control: must-revalidate
```

```
Cache-Control: max-age=3600, must-revalidate
```

```
Set-Cookie: foo=bar; Expires=Wed, 09 Jun 2021 10:18:14 GMT
Set-Cookie: baz=bat
```

```
Cache-Control   = 1#cache-directive
cache-directive = token [ "=" ( token / quoted-string ) ]
```

Cache-Control: max-age =3600

Invalid, right? ^

Chrome	Firefox	Safari	Edge	nginx	Squid	ATS	httpd	Varnish	Fastly
●	○	○	○	●	●	○	●	○	○

IMPLIED *LWS

The grammar described by this specification is word-based. Except where noted otherwise, linear white space (LWS) can be included between any two adjacent words (token or quoted-string), and between adjacent words and separators, without changing the interpretation of a field.

```
Cache-Control   = 1#cache-directive
cache-directive = token [ "=" ( token / quoted-string ) ]
```

Cache-Control: max-age=3600a

Will the cache use it? ^

Chrome	Firefox	Safari	Edge	nginx	Squid	ATS	httpd	Varnish	Fastly
●	●	○	●	○	●	●	○	●	●

4.1. Policy Syntax

A Content Security Policy consists of a U+003B SEMICOLON (`;`) delimited list of directives. Each directive consists of a directive name and (optionally) a directive value, defined by the following ABNF:

```
policy-token     = [ directive-token *( ";" [ directive-token ] ) ]
directive-token  = *WSP [ directive-name [ WSP directive-value ] ]
directive-name   = 1*( ALPHA / DIGIT / "-" )
directive-value  = *( WSP / <VCHAR except ";" and ","> )
```

4.1.1. Parsing Policies

To ***parse the policy*** *policy*, the user agent MUST use an algorithm equivalent to the following:

1. Let the *set of directives* be the empty set.
2. For each non-empty token returned by strictly splitting the string *policy* on the character U+003B SEMICOLON (`;`):
    1. Skip whitespace.
    2. Collect a sequence of characters that are not space characters. The collected characters are the *directive name*.
    3. If there are characters remaining in *token*, skip ahead exactly one character (which must be a space character).
    4. The remaining characters in *token* (if any) are the *directive value*.
    5. If the *set of directives* already contains a directive whose name is a case insensitive match for *directive*

Why Headers are Horrible

- Headers specification is difficult:
  - ABNF has extremely sharp edges
  - Writing and implementing parsing algorithms is painful
    - Error handling is often forgotten
- Headers are ill-defined and don’t leverage common syntax
- Header parsing and serialisation is usually one-off
  - Security and performance suffer
- Interoperability sucks

HTTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 13, 2020

M. Nottingham
Fastly
P-H. Kamp
The Varnish Cache Project

February 10, 2020

Structured Headers for HTTP

draft-ietf-httpbis-header-structure-latest

Abstract

This document describes a set of data types and associated algorithms that are intended to make it easier and safer to define and handle HTTP header fields. It is intended for use by specifications of new HTTP header fields that wish to use a common syntax that is more restrictive than traditional HTTP field values.

Item Types

- Integer `15`
- Decimal `3.5`
- String `"foo bar"`
- Token `foo`
- Boolean `?1`
- Byte Sequence `:cHJldGVuZCB0aGlzIGlzIGJpbmFyeSBjb250ZW50Lg==:`

All Items can be parameterised `15; a=b;c=5`

Container Types

- List `3.5, 4, foo, "hello world"`
  - Inner lists `(3.5, 4.5), other-thing`
- Dictionary `foo= 3.5, bar=?0`
  - Inner lists `thing=(1,2), other=(3,4)`

4.2.2. Parsing a Dictionary

Given an ASCII string as input_string, return an ordered map whose values are (item_or_inner_list, parameters) tuples. input_string is modified to remove the parsed value.

1. Let dictionary be an empty, ordered map.
2. While input_string is not empty:
    1. Let this_key be the result of running Parsing a Key (Section 4.2.3.3) with input_string.
    2. If the first character of input_string is "=":
        1. Consume the first character of input_string.
        2. Let member be the result of running Parsing an Item or Inner List (Section 4.2.1.1) with input_string.
    3. Otherwise:
        1. Let value be Boolean true.
        2. Let parameters be an empty, ordered map.
        3. Let member be the tuple (value, parameters).
    4. Add name this_key with value member to dictionary. If dictionary already contains a name this_key (comparing character-for-character), overwrite its value.
    5. Discard any leading SP characters from input_string.
    6. If input_string is empty, return dictionary.
    7. Consume the first character of input_string; if it is not ",", fail parsing.

```
Example-IntegerHeader: 42

Example-StringHeader: "hello world"

Example-BinaryHdr: :cHJldGVuZCB0aGlzIGmFyeSBjb250ZW50Lg==:

Example-DictHeader: max-age=60, must-revalidate

Example-ListHeader: foo, bar;q=0.1
```

- Syntax is defined in terms of rich, well-understood types
  - Error handling is taken care of
- Common, generic libraries for parsing and serialisation
  - Test corpus to ensure interoperability
- Implementations can concentrate efforts on security and perf

- Python 3
  `pip install shhh`

- Ruby
  https://github.com/phluid61/http-structured-headers

- JavaScript
  `npm install structured-header`

- C++ in Chrome
  https://bugs.chromium.org/p/chromium/issues/detail?id=1011101

- Erlang
  https://github.com/ninenines/cowlib/blob/master/src/cow_http_struct_hd.erl

- Test corpus
  https://github.com/httpwg/structured-header-tests

- Variants
- Gateway-Error
- Signature
- Accept-Signature
- Sec-Metadata
- ...

Opportunities

1) Backporting

```
HEADERMAP = {
    "accept": "list",
    "accept-encoding": "list",
    "accept-language": "list",
    "accept-patch": "list",
    "accept-ranges": "list",
    "access-control-allow-credentials": "item",
    "access-control-allow-headers": "list",
    "access-control-allow-methods": "list",
    "access-control-allow-origin": "item",
    "access-control-max-age": "item",
    "access-control-request-headers": "list",
    "access-control-request-method": "item",
    "age": "item",
    "allow": "list",
    "alpn": "list",
    "alt-svc": "dictionary",
    "alt-used": "item",
    "cache-control": "dictionary",
    "connection": "list",
    "content-encoding": "item",
    "content-language": "list",
    "content-length": "item",
    "content-type": "item",
    "expect": "item",
    "expect-ct": "dictionary",
    "forwarded": "dictionary",
    "host": "item",
    "keep-alive": "dictionary",
    "origin": "item",
    "pragma": "dictionary",
    "prefer": "dictionary",
    "preference-applied": "dictionary",
    "retry-after": "item",
    "strict-transport-security": "dictionary",
    "surrogate-control": "dictionary",
    "te": "list",
    "trailer": "list",
    "transfer-encoding": "list",
    "vary": "list",
    "x-content-type-options": "item",
    "x-xss-protection": "list"
}
```

```
import http.cookies
def parse_cookie(value):
    cookies = http.cookies.SimpleCookie()
    cookies.load(value)
    cookies = {c:cookies[c].value for c in cookies}
    return cookies

import calendar
from email.utils import parsedate as lib_parsedate
def parse_date(value):
    date_tuple = lib_parsedate(value)
    if date_tuple is None:
        raise ValueError
    if date_tuple[0] < 100:
        if date_tuple[0] > 68:
            date_tuple = (date_tuple[0]+1900,) + date_tuple[1:]
        else:
            date_tuple = (date_tuple[0]+2000,) + date_tuple[1:]
    return calendar.timegm(date_tuple)

backport_funcs = {
  b'cookie': parse_cookie,
  b'date': parse_date,
  b'last-modified': parse_date,
  b'expires': parse_date
}
```

2) Binary Serialisation

- HTTP/2 extension negotiated with a SETTING
  - “I will properly handle Structure Headers you send me...”
    - “Native” structured headers as-is
    - “Backported” structured headers (with appropriate processing, e.g., Date and Cookie)
      - Any input that results in a parse failure gets sent as an unstructured header
- Hop-by-hop negotiation, but structured headers can be forwarded if the next hop understands them

- Can fall back to non-structured headers for errors
- More (much more) efficient serialisation and parsing
- Sometimes, more efficient on the wire - e.g., integer, binary
- Can choose where/when to parse, header-by-header
- Once it's parsed, it's parsed.

3) Structured Compression

Internet Engineering Task Force (IETF)
Request for Comments: 7541
Category: Standards Track
ISSN: 2070-1721

R. Peon
Google, Inc
H. Ruellan
Canon CRF

May 2015

HPACK: Header Compression for HTTP/2

Abstract

This specification defines HPACK, a compression format for efficiently representing HTTP header fields, to be used in HTTP/2.

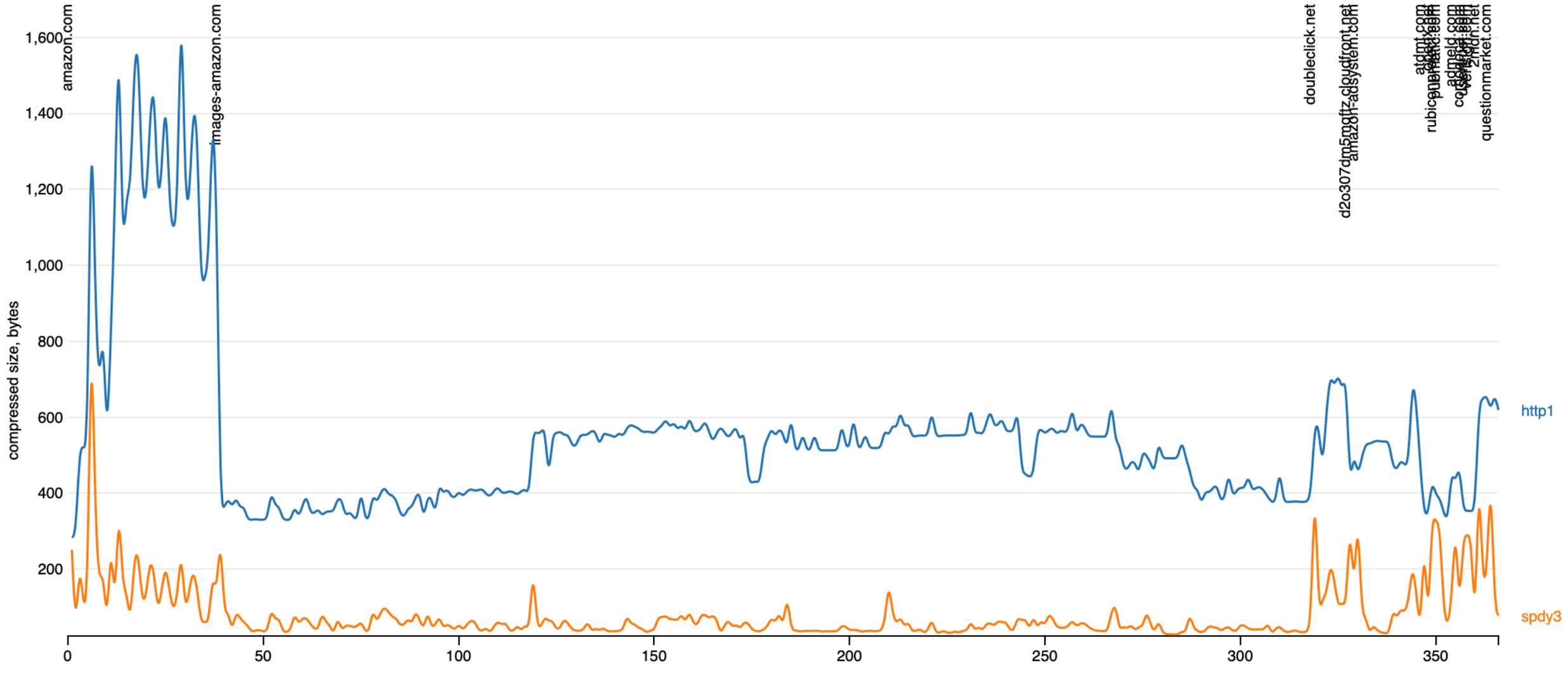
Status of This Memo

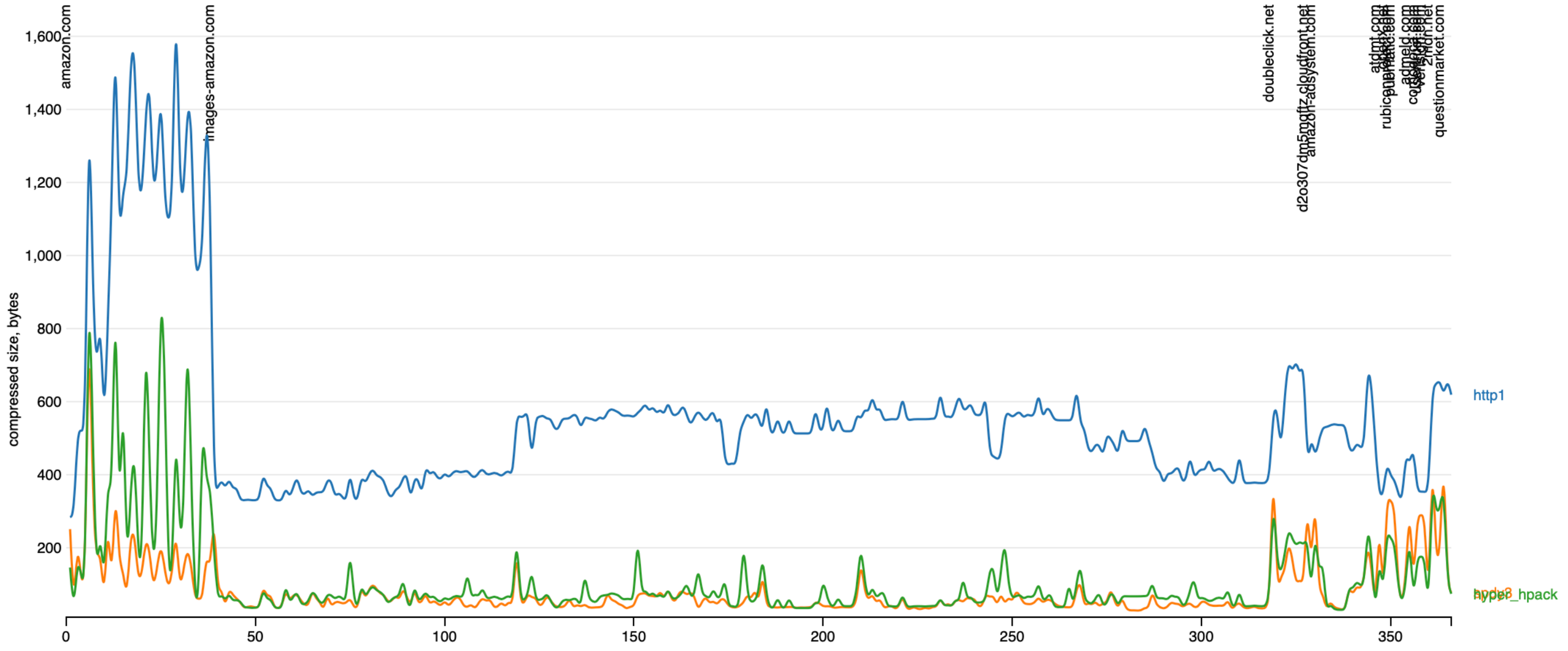
PROPOSED STANDARD
This document has errata.

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7541.


amazon.com
images-amazon.com
doubleclick.net
d2o307dm5mgftz.cloudfront.net
ama28ir-adsystem.com
questionmarket.com
compressed size, bytes
1,600
1,400
1,200
1,000
800
600
400
200
0
50
100
150
200
250
300
350
http1
spdy3

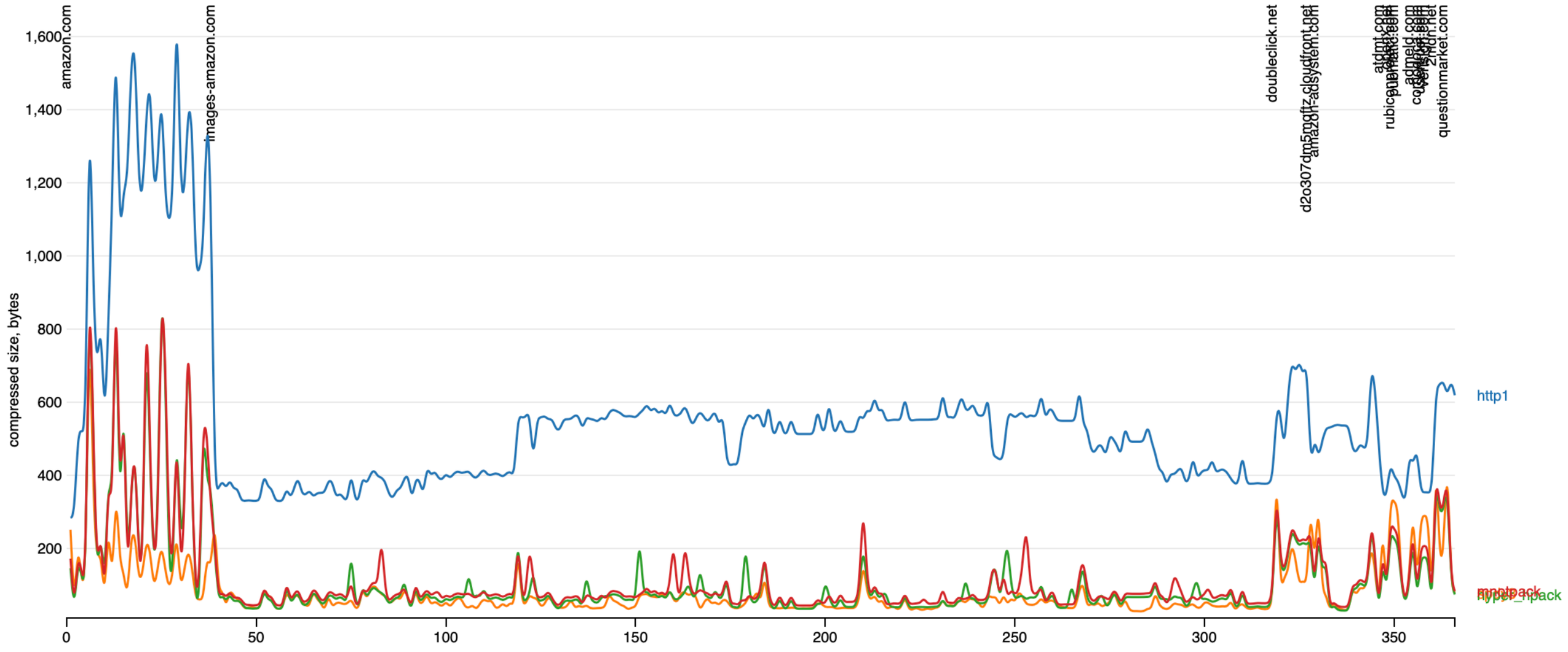

amazon.com
images-amazon.com
doubleclick.net
d2o3o7dm5mqftz.cloudfront.net
ama28ll-adsystem.com
rubiconproject.com
questionmarket.com
compressed size, bytes
1,600
1,400
1,200
1,000
800
600
400
200
0
50
100
150
200
250
300
350
http1
hpack

HPACK

```
Cache-Control: max-age=3600, s-maxage=7200, must-revalidate
```

HPACK+Structure

```
Cache-Control: max-age=3600, s-maxage=7200, must-revalidate
```


amazon.com
images-amazon.com
doubleclick.net
d2o3o7dm5mqftz.cloudfront.net
ama28ln-adsystem.com
rubiconproject.com
questionmarket.com
compressed size, bytes
1,600
1,400
1,200
1,000
800
600
400
200
0
50
100
150
200
250
300
350
http1
spdy3_1pack

Structured Headers gives us...

- Easier and more complete header specification
- Common implementations of parsers and serialisers
- Better focus for security and performance engineering
- Opportunities for future efficiencies

Challenges

- Syntax is purposefully limited, so some headers may shy away
  - ... but initial adoption is promising
- Getting full benefits requires end-to-end support
  - e.g., browser JS APIs, server-side integration
  - ... but partial benefits are still attractive
- Compression benefits still unproven

- https://httpwg.org/http-extensions/
- https://httpwg.org/http-core/
- https://github.com/httpwg/structured-header-tests/
- https://mnot.github.io/I-D/binary-structured-headers/
- https://cache-tests.fyi/