Structured Headers

HTTP ~~Headers~~ Fields

are defined using three layers of abstraction

1) Characters

```
header-field   = field-name ":" OWS field-value OWS
field-name     = token
field-value    = *( field-content / obs-fold )
field-content  = field-vchar [ 1*( SP / HTAB / field-vchar) field-vchar ]
field-vchar    = VCHAR / obs-text
obs-text       = %x80-FF
```

Historically, HTTP has allowed field content with text in the ISO-8859-1 charset, supporting other charsets only through use of [RFC2047] encoding. In practice, most HTTP header field values use only a subset of the US-ASCII charset. Newly defined header fields SHOULD limit their field values to US-ASCII octets. A recipient SHOULD treat other octets in field content (obs-text) as opaque data.

RFC7230, Section 3.2.4

2) Arrays. Sort of.

A recipient *MAY* combine multiple header fields with the same field name into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field value to the combined field value in order, separated by a comma.

RFC7230

3) A word-based grammar

> [T]his specification uses ABNF rules that are named according to each registered field name, wherein the rule defines the valid grammar for that field's corresponding field values (i.e., after the field-value has been extracted by a generic field parser).

‘When I use a word,’ ABNF said, in rather a scornful tone, ‘it means just what I choose it to mean — neither more nor less.’

```
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

Surely that fails? ^

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*

In a Nutshell

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

A Typical HTTP WG Meeting

HTTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 24, 2019

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

January 20, 2019

Structured Headers for HTTP

draft-ietf-httpbis-header-structure-latest

Abstract

This document describes a set of data types and algorithms associated with them that are intended to make it easier and safer to define and handle HTTP header fields. It is intended for use by new specifications of HTTP header fields as well as revisions of existing header field specifications when doing so does not cause interoperability issues.

Item Types

- Integer
- Float
- String
- Token
- Boolean
- Byte Sequence

Container Types

- List
- Dictionary
- List of Lists
- Parameterised List

```
Example-IntegerHeader: 42

Example-StringHeader: "hello world"

Example-BinaryHdr: *cHJldGVuZCB0aGlzIGmFyeSBjb250ZW50Lg==*
```

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

Example-ParamListHeader: text/plain; q=0.1;encoding=utf-8
```

4.2.1. Parsing a Dictionary from Text

Given an ASCII string input_string, return an ordered map of (key, item). 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 Parse a Key from Text (Section 4.2.2) with input_string.
    2. If dictionary already contains this_key, fail parsing.
    3. Consume the first character of input_string; if it is not "=", fail parsing.
    4. Let this_value be the result of running Parse Item from Text (Section 4.2.7) with input_string.
    5. Add key this_key with value this_value to dictionary.
    6. Discard any leading OWS from input_string.
    7. If input_string is empty, return dictionary.
    8. Consume the first character of input_string; if it is not COMMA, fail parsing.
    9. Discard any leading OWS from input_string.
    10. If input_string is empty, fail parsing.
3. No structured data has been found; fail parsing.

- Syntax is defined in terms of rich, well-understood types
- Error handling is taken care of
- Common, generic libraries for parsing and serialisation
- Common test corpus to ensure interoperability
- Tantalising possibility of back porting to existing headers

- Python 3: `pip install shhh`
- JavaScript: `npm install structured-header`
- C++: in Chrome (partial)
- Test corpus: https://github.com/httpwg/structured-header-tests

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


NOW, THINGS ARE GOING TO GET A LITTLE WEIRD.

Backporting

```
backportmap = {
  b':status':                         ITEM,
  b'accept':                          PARAMLIST,
  b'accept-encoding':                 PARAMLIST,
  b'accept-language':                 PARAMLIST,
  b'alt-svc':                         PARAMLIST,
  b'content-type':                    PARAMLIST,
  b'forwarded':                       PARAMLIST,
  b'te':                              PARAMLIST,
  b'cache-control':                   DICT,
  b'pragma':                          DICT,
  b'surrogate-control':               DICT,
  b'prefer':                          DICT,
  b'preference-applied':              DICT,
  b'digest':                          DICT,
  b'age':                             ITEM,
  b'content-length':                  ITEM,
  b'access-control-max-age':          ITEM,
  b'alt-used':                        ITEM,
  b'host':                            ITEM,
  b'content-encoding':                ITEM,
  b'expect':                          ITEM,
  b'origin':                          ITEM,
  b'access-control-allow-credentials': ITEM,
  b'access-control-request-method':   ITEM,
  b'x-content-type-options':          ITEM,
  b'access-control-allow-origin':     ITEM,
  b'accept-patch':                    LIST,
  b'accept-ranges':                   LIST,
  b'allow':                           LIST,
  b'alpn':                            LIST,
  b'content-language':                LIST,
  b'transfer-encoding':               LIST,
  b'vary':                            LIST,
  b'trailer':                         LIST,
  b'access-control-allow-headers':    LIST,
  b'access-control-allow-methods':    LIST,
  b'access-control-request-headers':  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
}
```

Alternative Serialisations

- 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.

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.

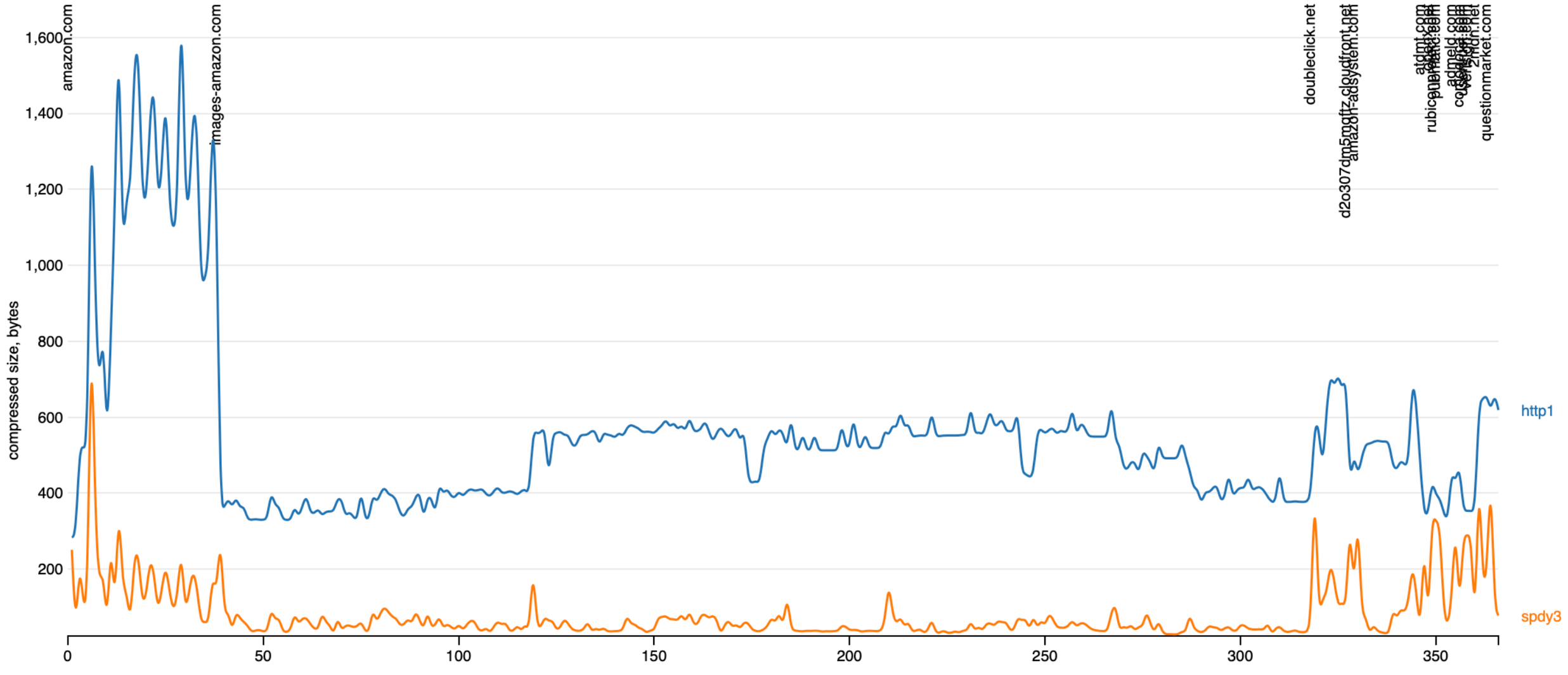
Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7541.

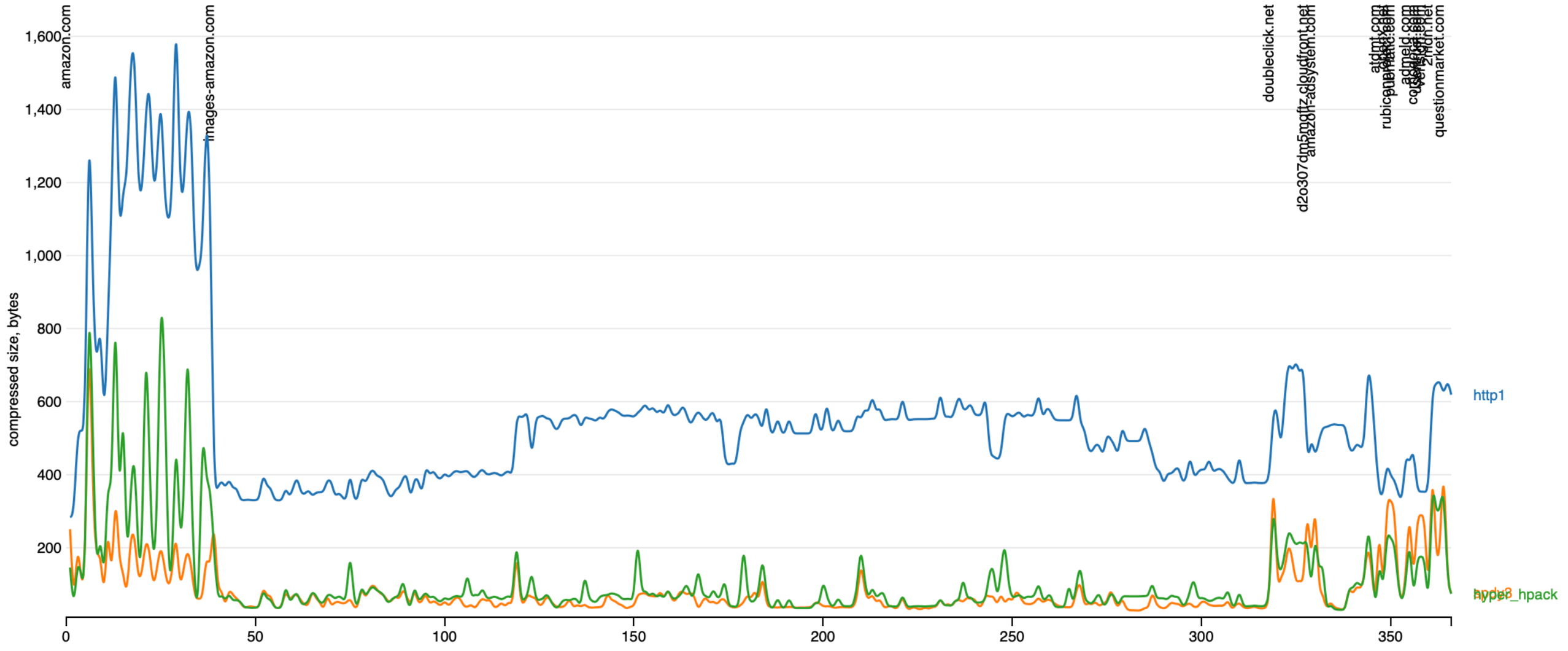
HPACK

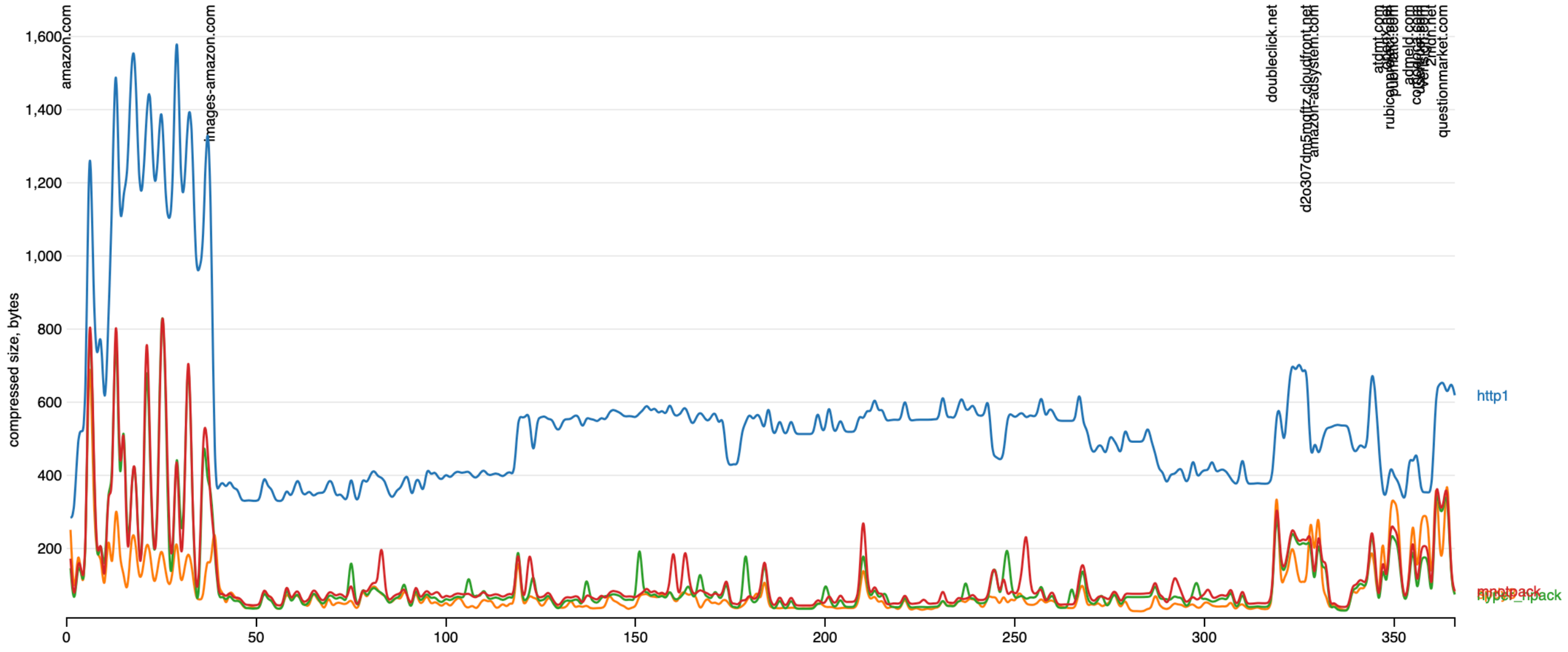
```
Cache-Control: max-age=3600, s-maxage=7200, must-revalidate
```

HPACK+Structure

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


amazon.com
images-amazon.com
doubleclick.net
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


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

Challenges

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