HTTP/2

Frederik Deweerdt - @fdeweerdt
Nathan Taylor - @dijkstracula

Why?

RFC 2616 - 1999

Network Working Group
Request for Comments: 2616
Obsoletes: 2068
Category: Standards Track

R. Fielding
UC Irvine
J. Gettys
Compaq/W3C
J. Mogul
Compaq
H. Frystyk
W3C/MIT
L. Masinter
Xerox
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT
June 1999

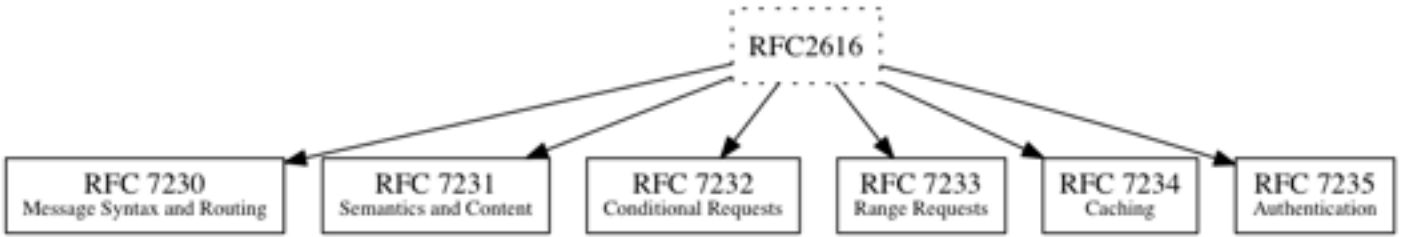
Hypertext Transfer Protocol -- HTTP/1.1

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

HTTP refresh - 2014



HTTP and TCP

- HTTP/1.0: open - fetch - close
- HTTP/1.1: open - fetch - wait - fetch - wait - … - close

HTTP/1.1 concurrency

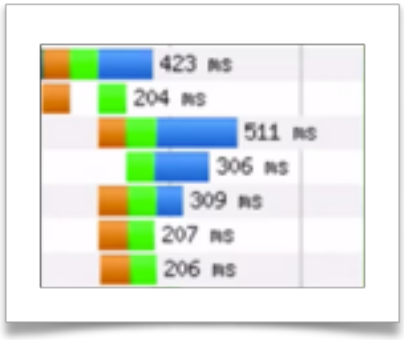
RFC 2616 - Section 8.1.4

Clients that use persistent connections SHOULD limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. A proxy SHOULD use up to 2*N connections to another server or proxy, where N is the number of simultaneously active users. These guidelines are intended to improve HTTP response times and avoid congestion.

HTTP/1.1 concurrency

RFC 2616 - Section 8.1.4

Clients that use persistent connections SHOULD limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. A proxy SHOULD use up to 2*N connections to another server or proxy, where N is the number of simultaneously active users. These guidelines are intended to improve HTTP response times and avoid congestion.



HTTP/1.1 concurrency

gophertiles?latency=1000	200	http/1.1	document	Other	18.1 KB	404 ms	94 ms
gophertiles?x=1&y=0&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	815 B	1.33 s	
gophertiles?x=5&y=0&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	821 B	1.35 s	
gophertiles?x=2&y=0&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	829 B	1.35 s	
gophertiles?x=4&y=0&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	834 B	1.35 s	
gophertiles?x=3&y=0&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	823 B	1.35 s	
gophertiles?x=0&y=0&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	816 B	1.38 s	
gophertiles?x=9&y=1&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	828 B	2.16 s	
gophertiles?x=6&y=5&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	1.1 KB	2.17 s	
gophertiles?x=7&y=2&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	1.0 KB	2.17 s	
gophertiles?x=4&y=6&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	1.1 KB	2.18 s	
gophertiles?x=10&y=2&cachebust=14715389...	200	http/1.1	jpeg	gophertiles?latency=10...	826 B	2.18 s	
gophertiles?x=10&y=5&cachebust=14715389...	200	http/1.1	jpeg	gophertiles?latency=10...	829 B	2.19 s	
gophertiles?x=9&y=2&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	929 B	3.24 s	
gophertiles?x=13&y=5&cachebust=14715389...	200	http/1.1	jpeg	gophertiles?latency=10...	830 B	3.24 s	
gophertiles?x=4&y=4&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	1.0 KB	3.24 s	
gophertiles?x=0&y=8&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	909 B	3.24 s	
gophertiles?x=3&y=2&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	960 B	3.25 s	
gophertiles?x=9&y=0&cachebust=147153899...	200	http/1.1	jpeg	gophertiles?latency=10...	837 B	3.25 s	

HTTP/1.1 pipelining

8.1.2.2 Pipelining

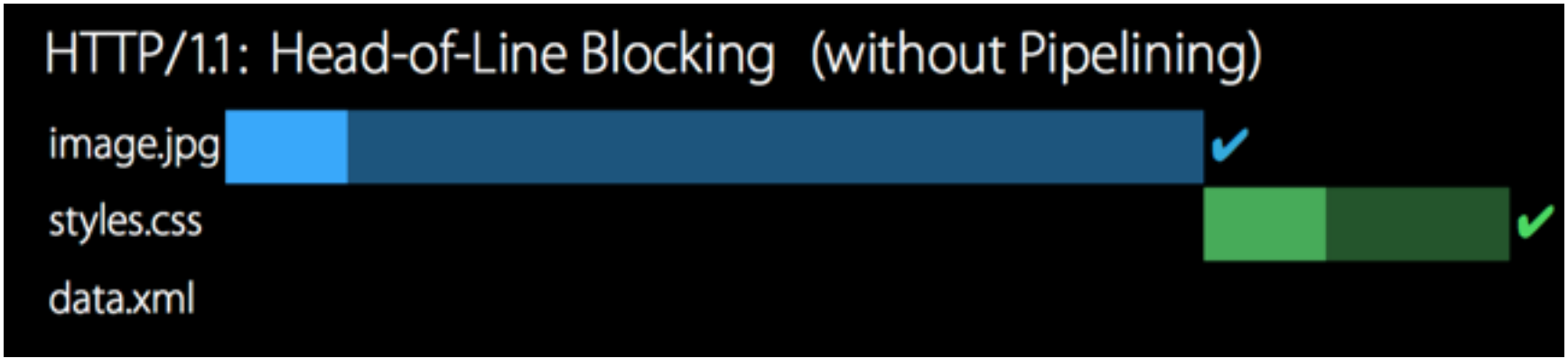
A client that supports persistent connections MAY "pipeline" its requests (i.e., send multiple requests without waiting for each response). A server MUST send its responses to those requests in the same order that the requests were received.

Clients which assume persistent connections and pipeline immediately after connection establishment SHOULD be prepared to retry their connection if the first pipelined attempt fails. If a client does such a retry, it MUST NOT pipeline before it knows the connection is persistent. Clients MUST also be prepared to resend their requests if the server closes the connection before sending all of the corresponding responses.

Clients SHOULD NOT pipeline requests using non-idempotent methods or non-idempotent sequences of methods (see section 9.1.2). Otherwise, a premature termination of the transport connection could lead to indeterminate results. A client wishing to send a non-idempotent request SHOULD wait to send that request until it has received the response status for the previous request.

Head of line blocking

HOL blocking



Luke Case - “Networking with NSURL Session” - WWDC 2015

Pipelining improvements

Network Working Group
Internet-Draft
Expires: 1 November 2001

Jeffrey Mogul, Compaq W
6 April 200

Support for out-of-order responses in HTTP

draft-mogul-http-ooo-00.txt

STATUS OF THIS MEMO

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

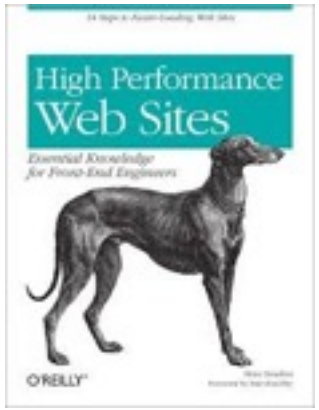
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

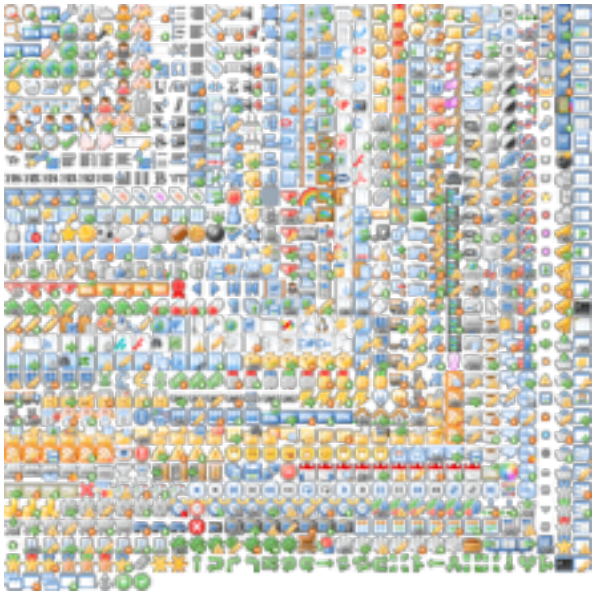
The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Clever techniques

- Domain sharding
- Image sprites
- JS/CSS concatenation
- Inlining
- FEO





HTTP/1.1 core problems

- Poor concurrency and multiplexing
- No interleaving
- Bad interaction with TCP
- Headers too large and repetitive

HTTP/2

RFC 7540

Internet Engineering Task Force (IETF)
Request for Comments: 7540
Category: Standards Track
ISSN: 2070-1721

M. Belshe
BitGo
R. Peon
Google, Inc
M. Thomson, Ed.
Mozilla
May 2015

Hypertext Transfer Protocol Version 2 (HTTP/2)

Abstract

This specification describes an optimized expression of the semantics of the Hypertext Transfer Protocol (HTTP), referred to as HTTP version 2 (HTTP/2). HTTP/2 enables a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection. It also introduces unsolicited push of representations from servers to clients.

This specification is an alternative to, but does not obsolete, the HTTP/1.1 message syntax. HTTP's existing semantics remain unchanged.

Status of This Memo

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/rfc7540.

A little history

- Started with Google's SPDY
- Call for proposals for HTTP/2, SPDY was used as a foundation
- Results: RFC 7540 (and RFC 7541)

HTTP/2 design goals

- Solve multiplexing / interleaving
- Solve header bloat
- More efficient implementations
- Retain the semantics of HTTP/1.1: headers, caching …

What is HTTP/2?

An analogy

Sand transport

Sand transport

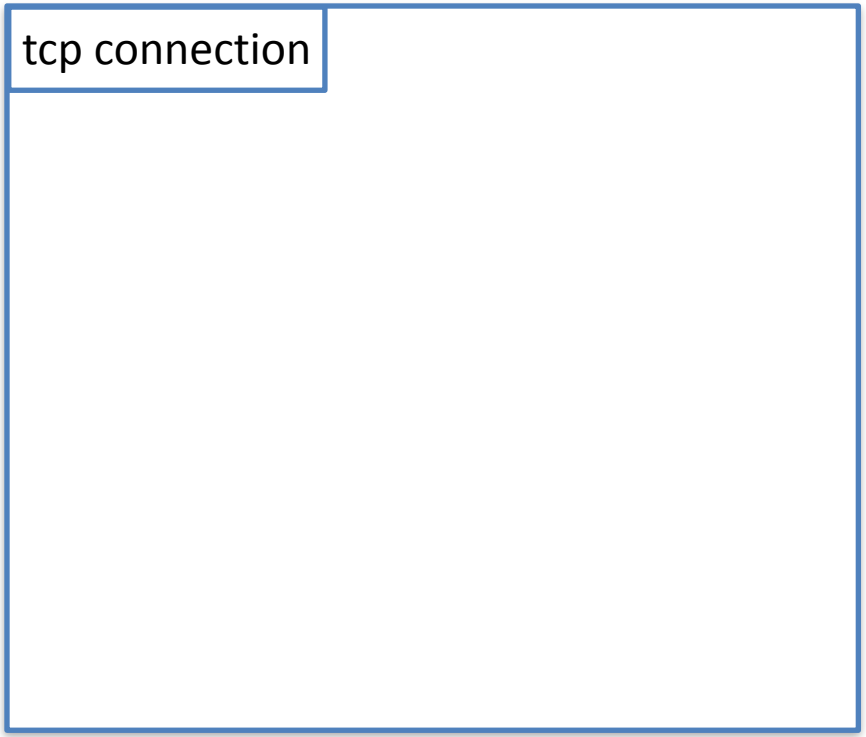
Sand transport

Sand transport

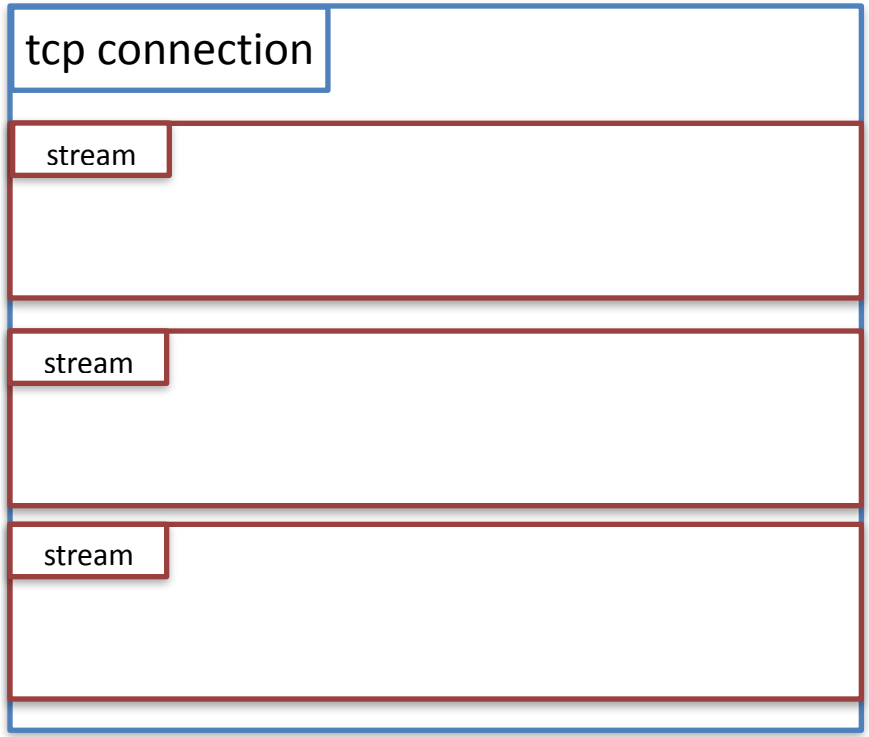
Sand transport

How does it work?
HTTP/2's components

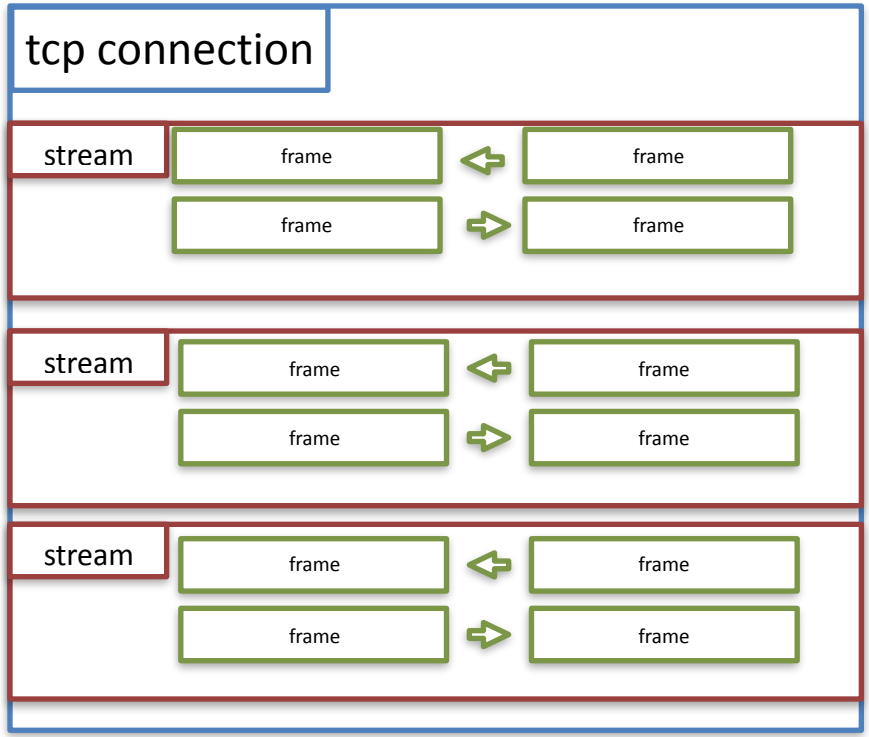
H2 components - TCP connection



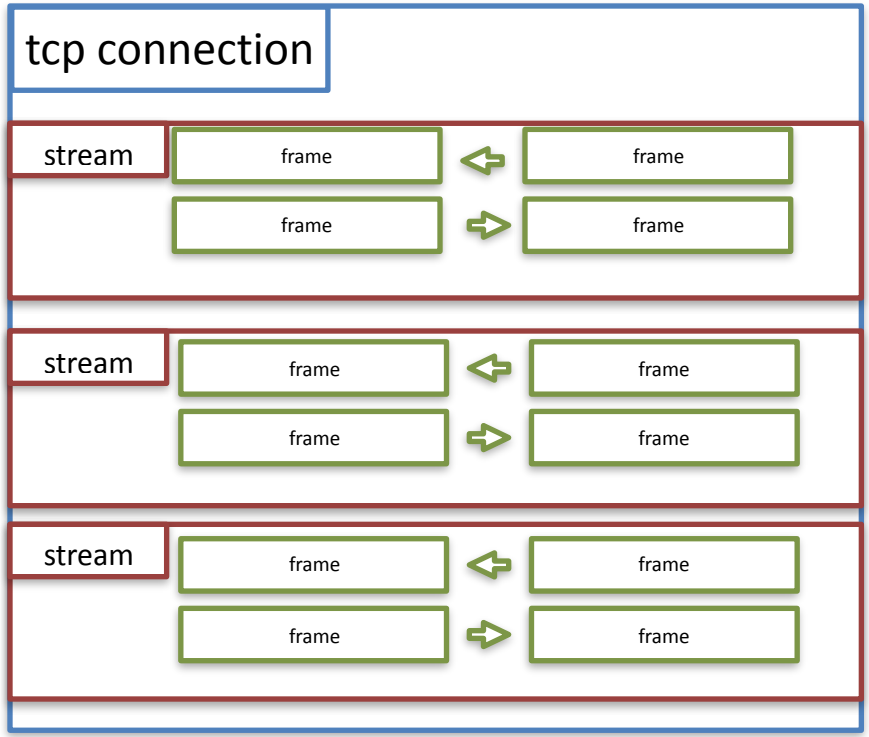
H2 components - Streams



H2 components - Frame



H2 components



Binary protocol

HTTP/1.1 - user-agent: human

```
$ telnet www.fastly.com 80
Trying 151.101.25.57...
Connected to prod.www-fastly-com.map.fastlylb.net.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.fastly.com

HTTP/1.1 301 Moved Permanently
Server: Varnish
Retry-After: 0
Location: https://www.fastly.com/
Content-Length: 0
Accept-Ranges: bytes
```

```
h2get> connect https://www.google.com
h2get> prefix
================================================================================
000: 50 52 49 20 2a 20 48 54 54 50 2f 32 2e 30 0d 0a PRI * HTTP/2.0..
010: 0d 0a 53 4d 0d 0a 0d 0a                         ..SM....
================================================================================
h2get> settings
================================================================================
000: 00 00 00 04 00 00 00 00 00                      .........
================================================================================
h2get> get
================================================================================
000: 00 00 23 01 05 00 00 00 01                      ..#......
================================================================================
================================================================================
000: 82 84 87 41 8b f1 e3 c2 f3 1c f3 50 55 c8 7a 7f ...A.......PU.z.
010: 53 03 2a 2f 2a 90 7a 8b aa 69 d2 9a c4 c0 57 08 S.*/*.z..i....W.
020: 57 7c 1f                                        W|.
================================================================================
```

Binary protocol

- The good:
  - Well defined format, strict framing
  - Easier to implement
- The bad:
  - Much harder to diagnose and troubleshoot
  - Need decoding tools like wireshark
  - Translation issues when proxying

The connection

A single connection

- Everything in HTTP/2 happens over a single, long lived connection
- Theoretically, this means better congestion management between peers
  - Standard TCP: each connection operates independently
  - Multiple connections are not competing for the same network resources

h2 & h2c

- The protocol supports both secure and clear-text versions:
  - h2 (secure)
  - h2c (clear)
- Browser implementers have opted only to support the secure version

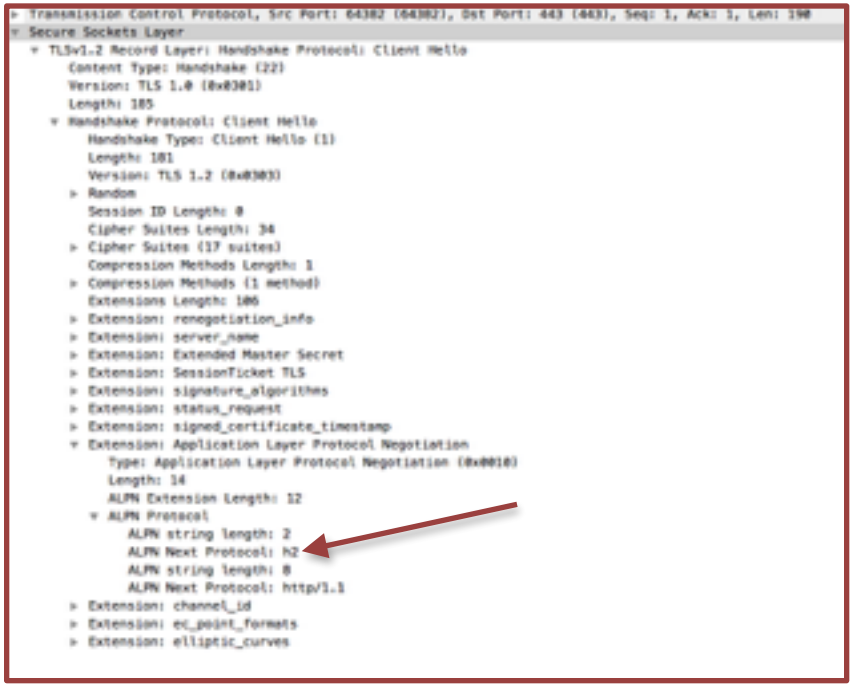
ALPN

- Since we connect to port 443, we need a way to discover HTTP/2 support
- Application-Layer Protocol Negotiation — RFC 7301
- Used to be NPN, but Chrome dropped support

ALPN


▸ Transmission Control Protocol, Src Port: 64382 (64382), Dst Port: 443 (443), Seq: 1, Ack: 1, Len: 190
▾ Secure Sockets Layer
▾ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 185
▾ Handshake Protocol: Client Hello
Handshake Type: Client Hello (1)
Length: 181
Version: TLS 1.2 (0x0303)
▸ Random
Session ID Length: 0
Cipher Suites Length: 34
▸ Cipher Suites (17 suites)
Compression Methods Length: 1
▸ Compression Methods (1 method)
Extensions Length: 106
▸ Extension: renegotiation_info
▸ Extension: server_name
▸ Extension: Extended Master Secret
▸ Extension: SessionTicket TLS
▸ Extension: signature_algorithms
▸ Extension: status_request
▸ Extension: signed_certificate_timestamp
▾ Extension: Application Layer Protocol Negotiation
Type: Application Layer Protocol Negotiation (0x0010)
Length: 14
ALPN Extension Length: 12
▾ ALPN Protocol
ALPN string length: 2
ALPN Next Protocol: h2
ALPN string length: 8
ALPN Next Protocol: http/1.1
▸ Extension: channel_id
▸ Extension: ec_point_formats
▸ Extension: elliptic_curves
Hey!
I support h2
and http/1.1

ALPN


▸ Transmission Control Protocol, Src Port: 64382 (64382), Dst Port: 443 (443), Seq: 1, Ack: 1, Len: 190
▾ Secure Sockets Layer
▾ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 185
▾ Handshake Protocol: Client Hello
Handshake Type: Client Hello (1)
Length: 181
Version: TLS 1.2 (0x0303)
▸ Random
Session ID Length: 0
Cipher Suites Length: 34
▸ Cipher Suites (17 suites)
Compression Methods Length: 1
▸ Compression Methods (1 method)
Extensions Length: 106
▸ Extension: renegotiation_info
▸ Extension: server_name
▸ Extension: Extended Master Secret
▸ Extension: SessionTicket TLS
▸ Extension: signature_algorithms
▸ Extension: status_request
▸ Extension: signed_certificate_timestamp
▾ Extension: Application Layer Protocol Negotiation
Type: Application Layer Protocol Negotiation (0x0010)
Length: 14
ALPN Extension Length: 12
▾ ALPN Protocol
ALPN string length: 2
ALPN Next Protocol: h2
ALPN string length: 8
ALPN Next Protocol: http/1.1
▸ Extension: channel_id
▸ Extension: ec_point_formats
▸ Extension: elliptic_curves


▸ Transmission Control Protocol, Src Port: 443 (443), Dst Port: 64382 (64382), Seq: 1, Ack: 191, Len: 1448
▾ Secure Sockets Layer
▾ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
Content Type: Handshake (22)
Version: TLS 1.2 (0x0303)
Length: 78
▾ Handshake Protocol: Server Hello
Handshake Type: Server Hello (2)
Length: 74
Version: TLS 1.2 (0x0303)
▸ Random
Session ID Length: 0
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
Compression Method: null (0)
Extensions Length: 34
▸ Extension: server_name
▸ Extension: renegotiation_info
▸ Extension: ec_point_formats
▸ Extension: SessionTicket TLS
▸ Extension: status_request
▾ Extension: Application Layer Protocol Negotiation
Type: Application Layer Protocol Negotiation (0x0010)
Length: 5
ALPN Extension Length: 3
▾ ALPN Protocol
ALPN string length: 2
ALPN Next Protocol: h2
Ok, let's do h2

Connection reuse

- Connection can be reused between this client and another hostname if:
  - Both hostnames resolve to the same IP
  - SAN cert has both hostnames used in TLS handshake
- New status code: 421 (Misdirected request)

Connection reuse

- Browsers implement this differently
  - Firefox will re-use the connection if the domains were seen using the same IP at least once
  - Chrome is more conservative and will re-use if the IPs happen to be the same

Connection reuse

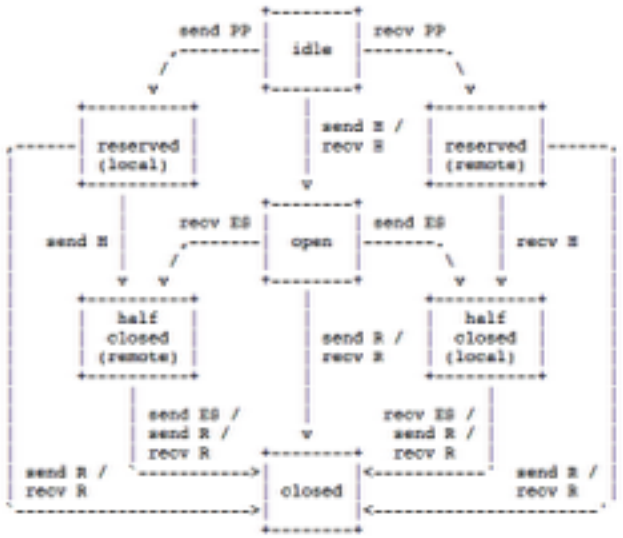
- Matters for testing, it's not enough to have another CNAME
- Domain sharding will behave differently
- Better for memory usage, on both sides

Streams

Streams

- Virtual channels of communication
  - Translate roughly to a request/response exchange
  - Client or server can initiate or terminate
- Frame order within a stream is significant
  - Frames are processed in the order they are received
- Stream IDs:
  - Client: odd, server: even, 0: connection-wide/reserved
  - Each new ID has to be larger than the previous ones
  - Stream IDs can’t be reused (max is 2^31)

Streams



send: endpoint sends this frame
recv: endpoint receives this frame

H: HEADERS frame (with implied CONTINUATIONs)
PP: PUSH_PROMISE frame (with implied CONTINUATIONs)
ES: END_STREAM flag
R: RST_STREAM frame

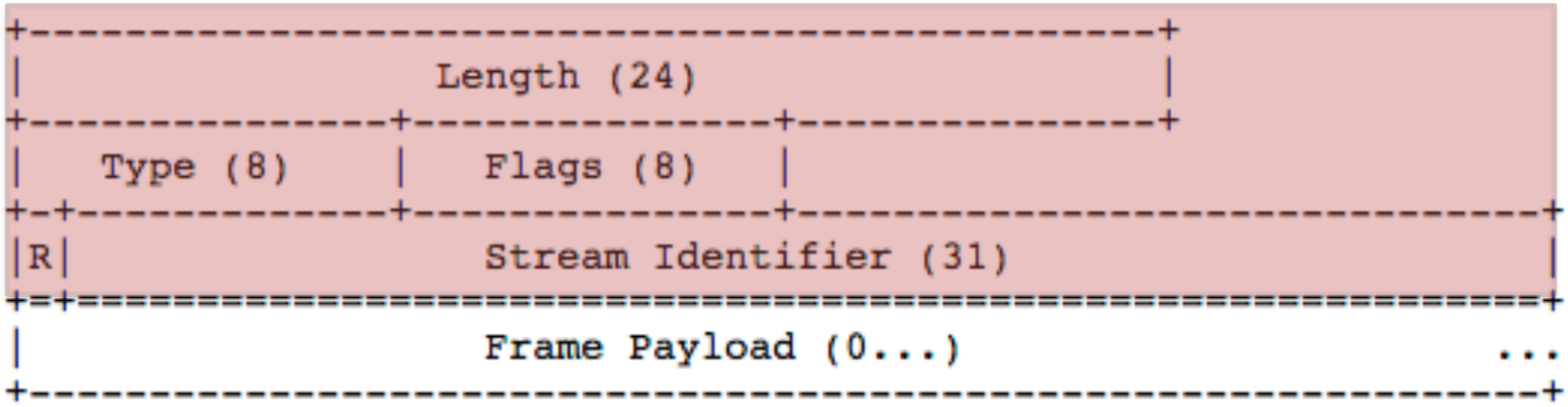
Streams - https://tools.ietf.org/html/draft-benfield-http2-debug-state-01

```
0 $ curl -k --http2 https://127.0.0.1:8081/.well-known/h2/state
{
  "version": "draft-01",
  "settings": {
    "SETTINGS_HEADER_TABLE_SIZE": 4096,
    "SETTINGS_ENABLE_PUSH": 0,
    "SETTINGS_MAX_CONCURRENT_STREAMS": 100,
    "SETTINGS_INITIAL_WINDOW_SIZE": 16777216,
    "SETTINGS_MAX_FRAME_SIZE": 16384
  },
  "peerSettings": {
    "SETTINGS_HEADER_TABLE_SIZE": 4096,
    "SETTINGS_ENABLE_PUSH": 1,
    "SETTINGS_MAX_CONCURRENT_STREAMS": 4294967295,
    "SETTINGS_INITIAL_WINDOW_SIZE": 65535,
    "SETTINGS_MAX_FRAME_SIZE": 16384
  },
  "connFlowIn": 65535,
  "connFlowOut": 65535,
  "streams": {
    "1": {
      "state": "HALF_CLOSED_REMOTE",
      "flowIn": 16777216,
      "flowOut": 65535,
      "dataIn": 0,
      "dataOut": 0,
      "created": 1471459379
    }
  }
}
```

Frames

H2 Frame

Fixed 9 bytes header

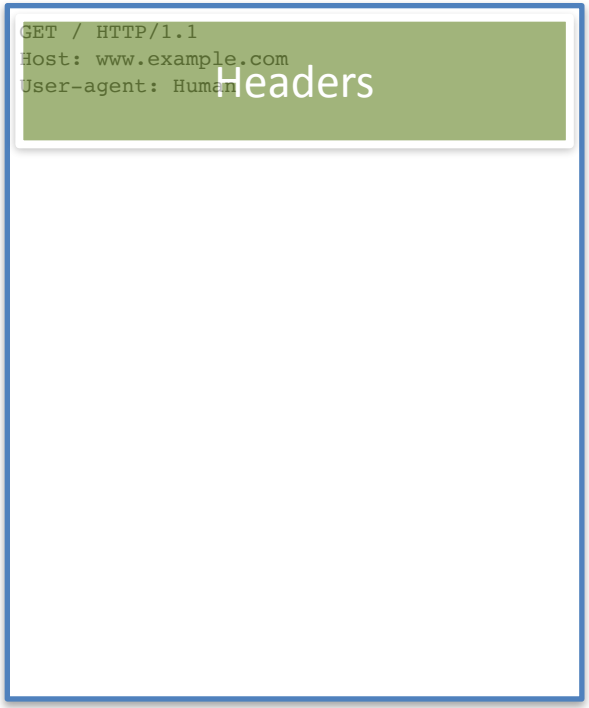


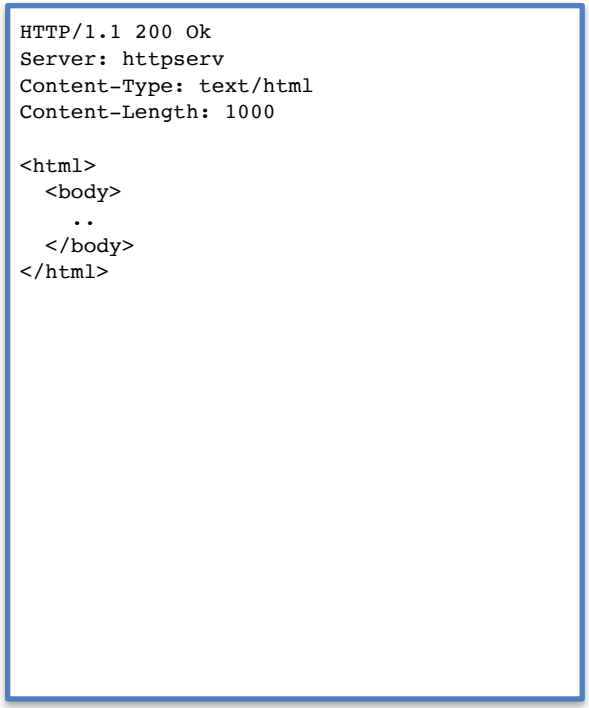
H2 Frames

```
GET / HTTP/1.1
Host: www.example.com
User-agent: Human
```

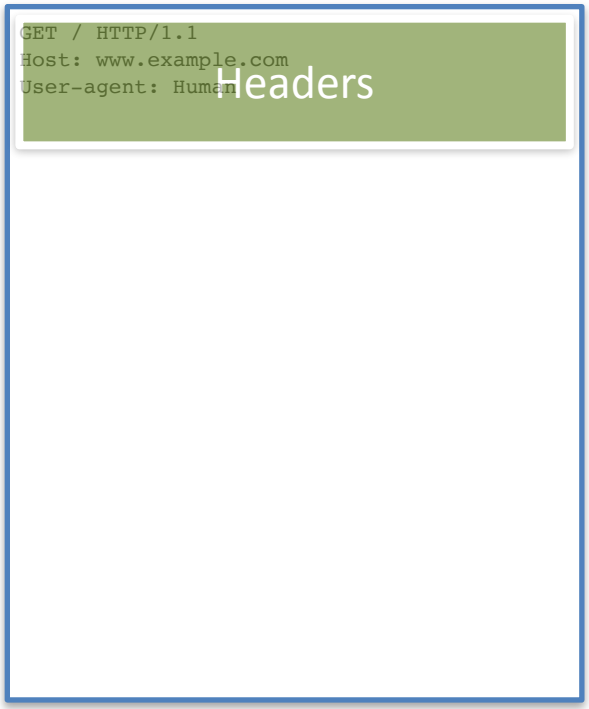
```
HTTP/1.1 200 Ok
Server: httpserv
Content-Type: text/html
Content-Length: 1000

<html>
  <body>
    ..
  </body>
</html>
```

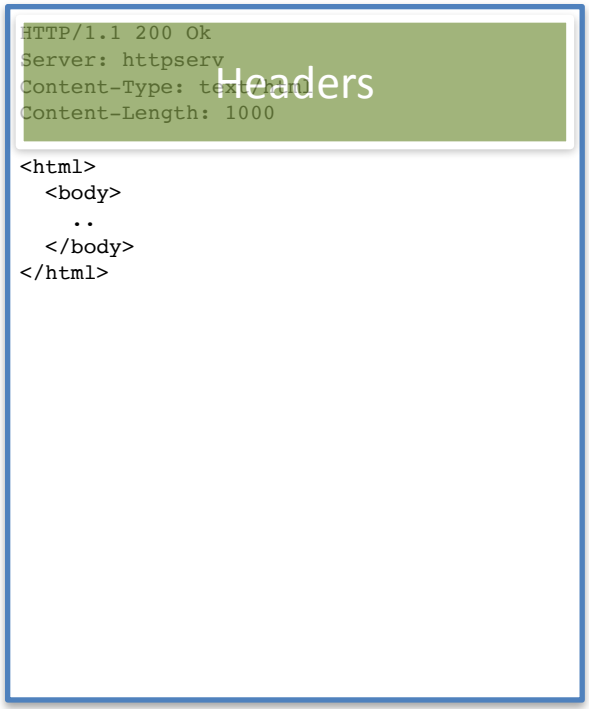
H2 Frames


GET / HTTP/1.1
Host: www.example.com
User-agent: Human
Headers

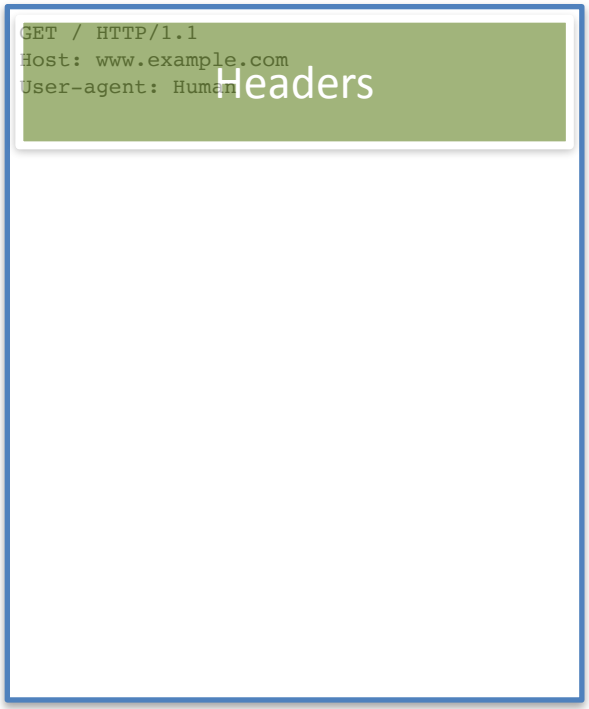

HTTP/1.1 200 Ok
Server: httpserv
Content-Type: text/html
Content-Length: 1000
<html>
<body>
..
</body>
</html>

H2 Frames


GET / HTTP/1.1
Host: www.example.com
User-agent: Human
Headers

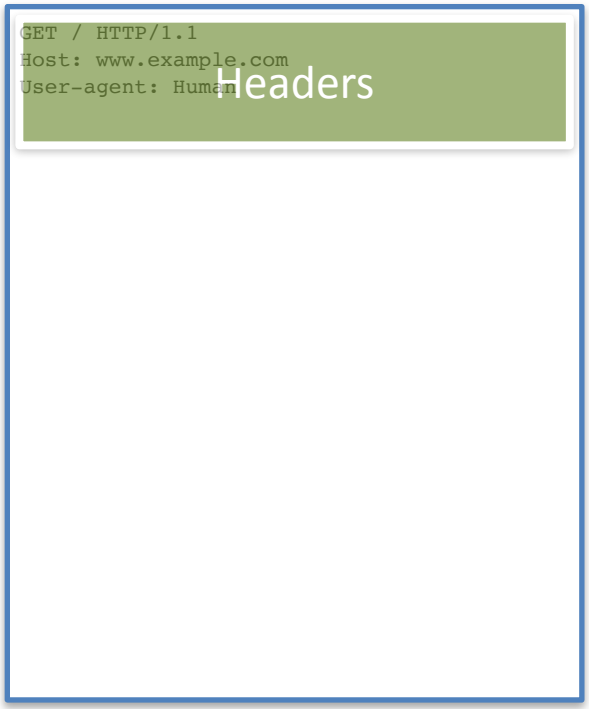

HTTP/1.1 200 Ok
Server: httpserv
Content-Type: text/html
Content-Length: 1000
Headers
<html>
<body>
..
</body>
</html>

H2 Frames


GET / HTTP/1.1
Host: www.example.com
User-agent: Human
Headers


HTTP/1.1 200 Ok
Server: httpserv
Content-Type: text/html
Content-Length: 1000
Headers
<html>
<body>
..
</body>
</html>
Data
Data
Data
Data

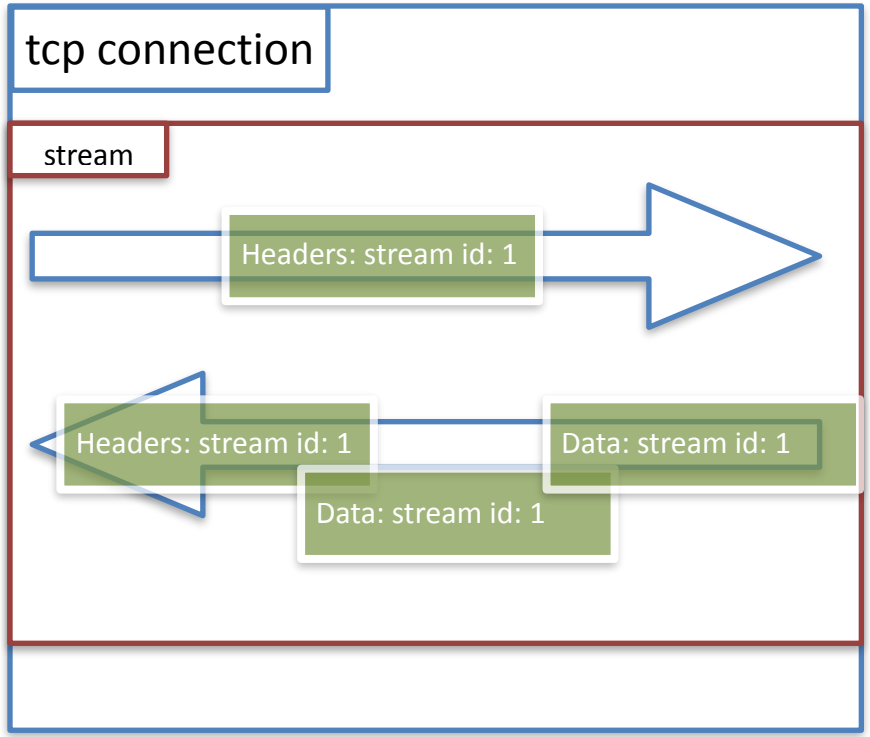
H2 Frames


GET / HTTP/1.1
Host: www.example.com
User-agent: Human
Headers

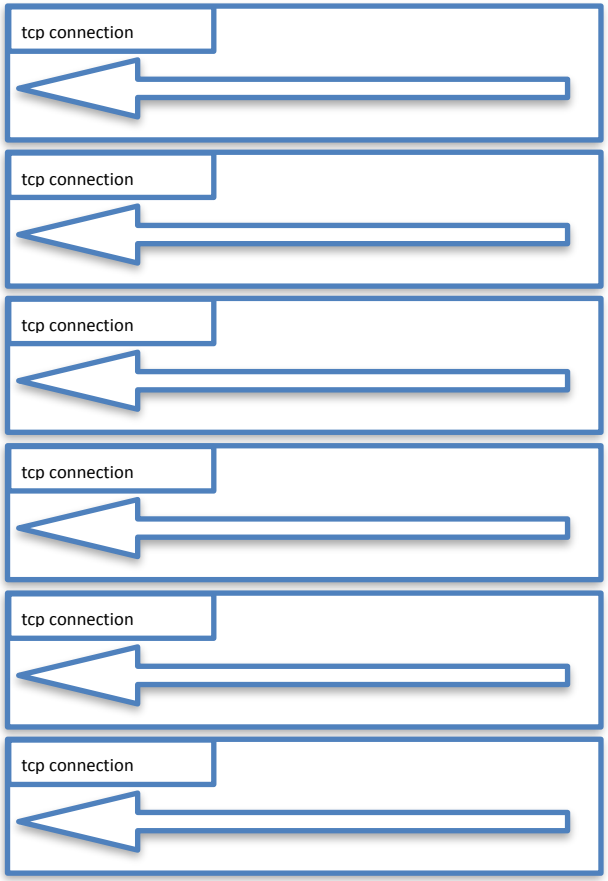

HTTP/1.1 200 Ok
Server: httpserv
Content-Type: text/html
Content-Length: 1000
Headers
Cont
Cont
<html>
<body>
..
</body>
</html>
Data
Data
Data
Data

Protocol flow

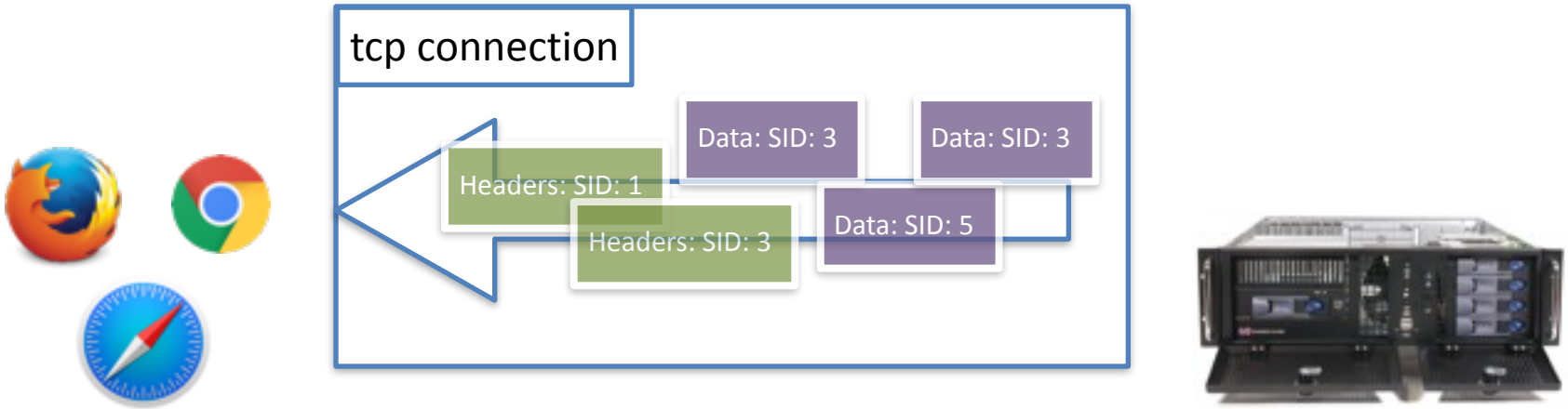
H2 Frames


tcp connection
stream
Headers: stream id: 1
Headers: stream id: 1
Data: stream id: 1
Data: stream id: 1

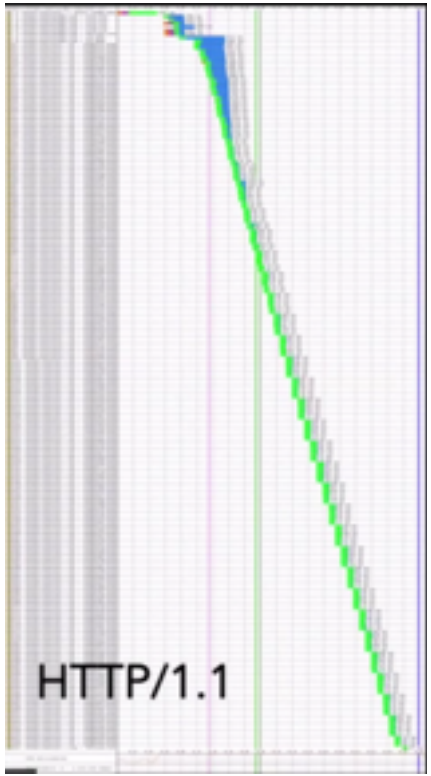
HTTP/1.1 used to be



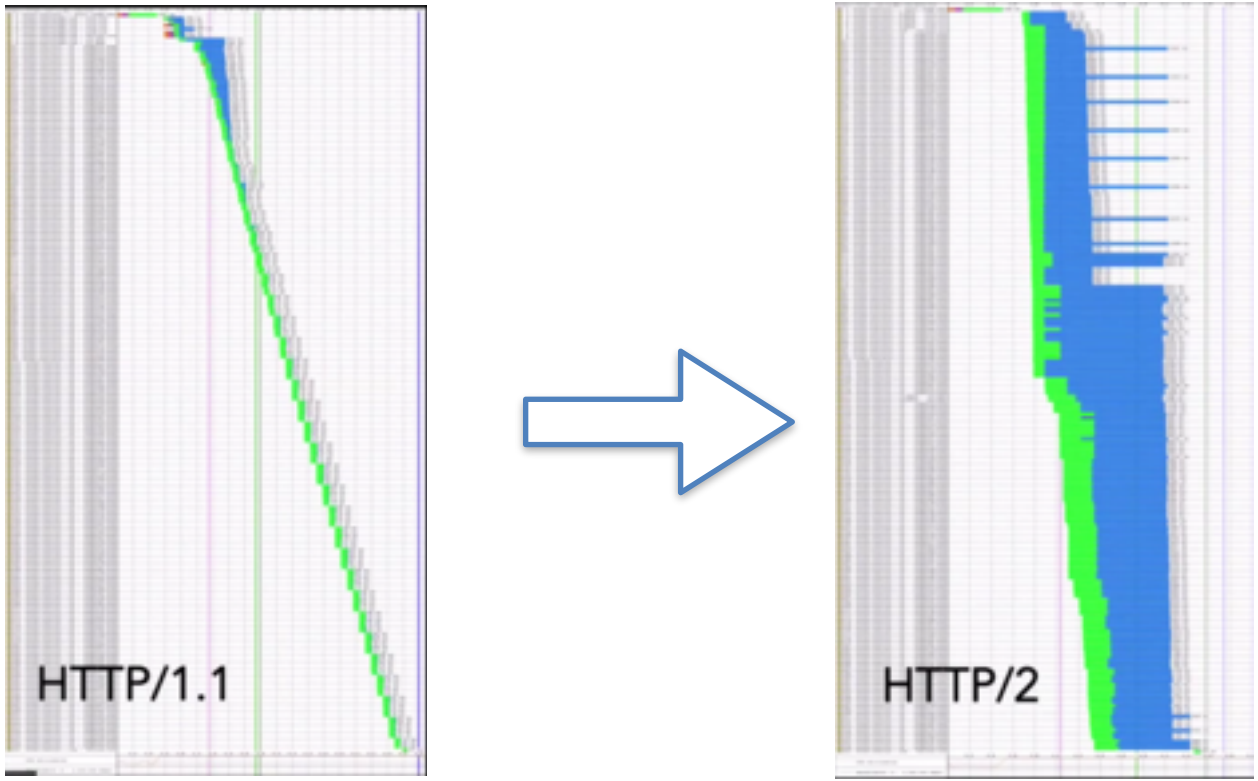
H2 Frames


tcp connection
Data: SID: 3
Data: SID: 3
Headers: SID: 1
Data: SID: 5
Headers: SID: 3

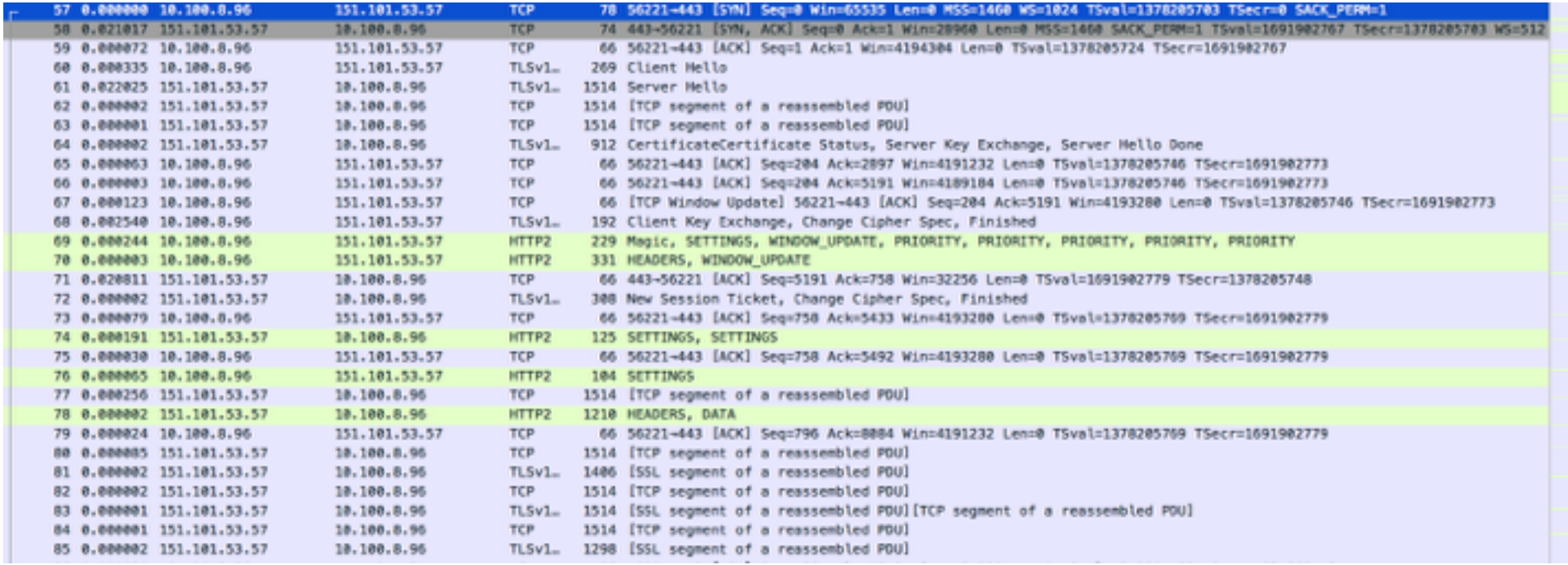
H2 Frames



H2 Frames

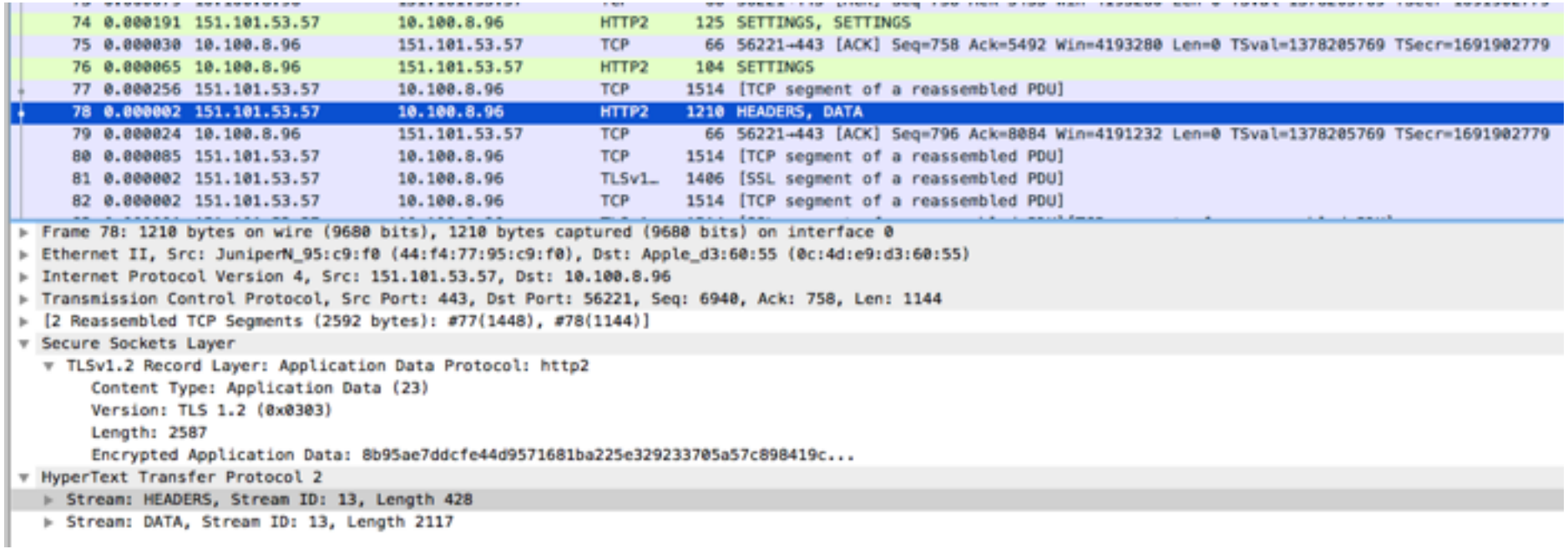


H2 Frames - Wireshark

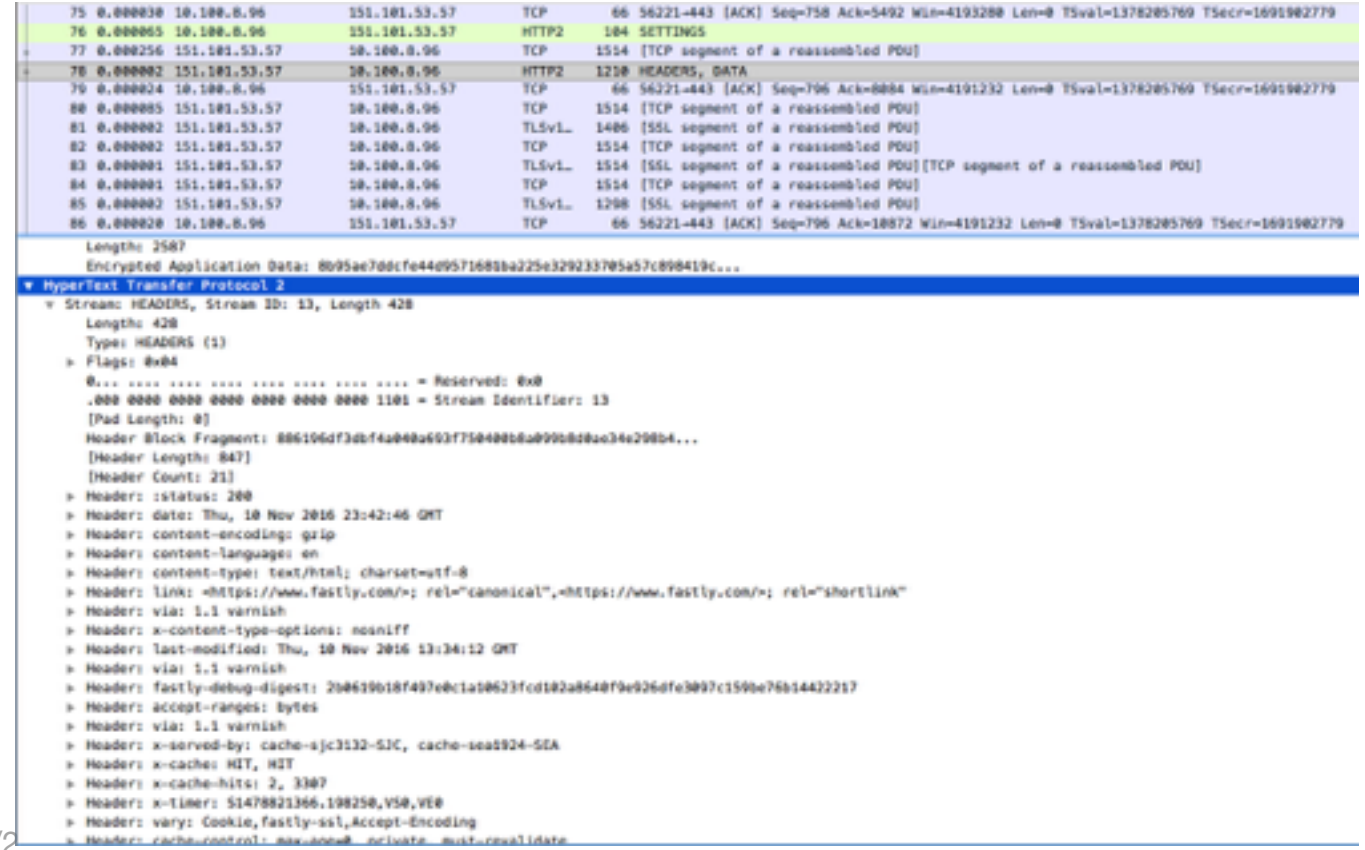


57	0.000000	10.100.8.96	151.101.53.57	TCP	78	56221→443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=1024 TSval=1378205703 TSecr=0 SACK_PERM=1
58	0.021017	151.101.53.57	10.100.8.96	TCP	74	443→56221 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1691902767 TSecr=1378205703 WS=512
59	0.000072	10.100.8.96	151.101.53.57	TCP	66	56221→443 [ACK] Seq=1 Ack=1 Win=4194304 Len=0 TSval=1378205724 TSecr=1691902767
60	0.000335	10.100.8.96	151.101.53.57	TLSv1…	269	Client Hello
61	0.022025	151.101.53.57	10.100.8.96	TLSv1…	1514	Server Hello
62	0.000002	151.101.53.57	10.100.8.96	TCP	1514	[TCP segment of a reassembled PDU]
63	0.000001	151.101.53.57	10.100.8.96	TCP	1514	[TCP segment of a reassembled PDU]
64	0.000002	151.101.53.57	10.100.8.96	TLSv1…	912	CertificateCertificate Status, Server Key Exchange, Server Hello Done
65	0.000063	10.100.8.96	151.101.53.57	TCP	66	56221→443 [ACK] Seq=204 Ack=2897 Win=4191232 Len=0 TSval=1378205746 TSecr=1691902773
66	0.000003	10.100.8.96	151.101.53.57	TCP	66	56221→443 [ACK] Seq=204 Ack=5191 Win=4189184 Len=0 TSval=1378205746 TSecr=1691902773
67	0.000123	10.100.8.96	151.101.53.57	TCP	66	[TCP Window Update] 56221→443 [ACK] Seq=204 Ack=5191 Win=4193280 Len=0 TSval=1378205746 TSecr=1691902773
68	0.002540	10.100.8.96	151.101.53.57	TLSv1…	192	Client Key Exchange, Change Cipher Spec, Finished
69	0.000244	10.100.8.96	151.101.53.57	HTTP2	229	Magic, SETTINGS, WINDOW_UPDATE, PRIORITY, PRIORITY, PRIORITY, PRIORITY, PRIORITY
70	0.000003	10.100.8.96	151.101.53.57	HTTP2	331	HEADERS, WINDOW_UPDATE
71	0.020811	151.101.53.57	10.100.8.96	TCP	66	443→56221 [ACK] Seq=5191 Ack=758 Win=32256 Len=0 TSval=1691902779 TSecr=1378205748
72	0.000002	151.101.53.57	10.100.8.96	TLSv1…	308	New Session Ticket, Change Cipher Spec, Finished
73	0.000079	10.100.8.96	151.101.53.57	TCP	66	56221→443 [ACK] Seq=758 Ack=5433 Win=4193280 Len=0 TSval=1378205769 TSecr=1691902779
74	0.000191	151.101.53.57	10.100.8.96	HTTP2	125	SETTINGS, SETTINGS
75	0.000030	10.100.8.96	151.101.53.57	TCP	66	56221→443 [ACK] Seq=758 Ack=5492 Win=4193280 Len=0 TSval=1378205769 TSecr=1691902779
76	0.000065	10.100.8.96	151.101.53.57	HTTP2	104	SETTINGS
77	0.000256	151.101.53.57	10.100.8.96	TCP	1514	[TCP segment of a reassembled PDU]
78	0.000002	151.101.53.57	10.100.8.96	HTTP2	1210	HEADERS, DATA
79	0.000024	10.100.8.96	151.101.53.57	TCP	66	56221→443 [ACK] Seq=796 Ack=8004 Win=4191232 Len=0 TSval=1378205769 TSecr=1691902779
80	0.000085	151.101.53.57	10.100.8.96	TCP	1514	[TCP segment of a reassembled PDU]
81	0.000002	151.101.53.57	10.100.8.96	TLSv1…	1406	[SSL segment of a reassembled PDU]
82	0.000002	151.101.53.57	10.100.8.96	TCP	1514	[TCP segment of a reassembled PDU]
83	0.000001	151.101.53.57	10.100.8.96	TLSv1…	1514	[SSL segment of a reassembled PDU][TCP segment of a reassembled PDU]
84	0.000001	151.101.53.57	10.100.8.96	TCP	1514	[TCP segment of a reassembled PDU]
85	0.000002	151.101.53.57	10.100.8.96	TLSv1…	1298	[SSL segment of a reassembled PDU]

H2 Frames - Wireshark



H2 Frames - Wireshark



Frame types

Frame types

Frame Type	Code	Section
DATA	0x0	Section 6.1
HEADERS	0x1	Section 6.2
PRIORITY	0x2	Section 6.3
RST_STREAM	0x3	Section 6.4
SETTINGS	0x4	Section 6.5
PUSH_PROMISE	0x5	Section 6.6
PING	0x6	Section 6.7
GOAWAY	0x7	Section 6.8
WINDOW_UPDATE	0x8	Section 6.9
CONTINUATION	0x9	Section 6.10

Frame types

Frame Type	Code	Section
DATA	0x0	Section 6.1
HEADERS	0x1	Section 6.2
PRIORITY	0x2	Section 6.3
RST_STREAM	0x3	Section 6.4
SETTINGS	0x4	Section 6.5
PUSH_PROMISE	0x5	Section 6.6
PING	0x6	Section 6.7
GOAWAY	0x7	Section 6.8
WINDOW_UPDATE	0x8	Section 6.9
CONTINUATION	0x9	Section 6.10

Frame types

Frame Type	Code	Section
DATA	0x0	Section 6.1
HEADERS	0x1	Section 6.2
PRIORITY	0x2	Section 6.3
RST_STREAM	0x3	Section 6.4
SETTINGS	0x4	Section 6.5
PUSH_PROMISE	0x5	Section 6.6
PING	0x6	Section 6.7
GOAWAY	0x7	Section 6.8
WINDOW_UPDATE	0x8	Section 6.9
CONTINUATION	0x9	Section 6.10

Frame types

Frame Type	Code	Section
DATA	0x0	Section 6.1
HEADERS	0x1	Section 6.2
PRIORITY	0x2	Section 6.3
RST_STREAM	0x3	Section 6.4
SETTINGS	0x4	Section 6.5
PUSH_PROMISE	0x5	Section 6.6
PING	0x6	Section 6.7
GOAWAY	0x7	Section 6.8
WINDOW_UPDATE	0x8	Section 6.9
CONTINUATION	0x9	Section 6.10

Frame types

Frame Type	Code	Section
DATA	0x0	Section 6.1
HEADERS	0x1	Section 6.2
PRIORITY	0x2	Section 6.3
RST_STREAM	0x3	Section 6.4
SETTINGS	0x4	Section 6.5
PUSH_PROMISE	0x5	Section 6.6
PING	0x6	Section 6.7
GOAWAY	0x7	Section 6.8
WINDOW_UPDATE	0x8	Section 6.9
CONTINUATION	0x9	Section 6.10

Frame types

Frame Type	Code	Section
DATA	0x0	Section 6.1
HEADERS	0x1	Section 6.2
PRIORITY	0x2	Section 6.3
RST_STREAM	0x3	Section 6.4
SETTINGS	0x4	Section 6.5
PUSH_PROMISE	0x5	Section 6.6
PING	0x6	Section 6.7
GOAWAY	0x7	Section 6.8
WINDOW_UPDATE	0x8	Section 6.9
CONTINUATION	0x9	Section 6.10

Frame types

- Extensible
  - Currently discussed new frames:
    - ORIGIN HTTP/2 Frame
    - CACHE_DIGEST Frame
- Implementations should ignore unknown frames

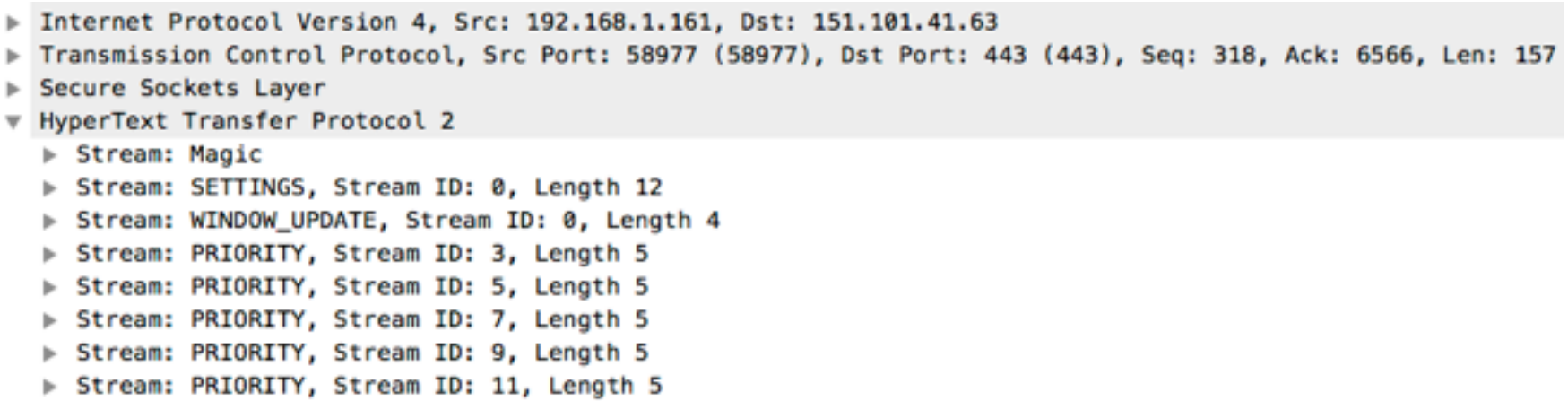
SETTINGS Frame

- Defines parameters for the connection
  - Max concurrent streams
  - Max window size
  - …
- Are sent and acknowledged by both sides
- Can be sent at any time

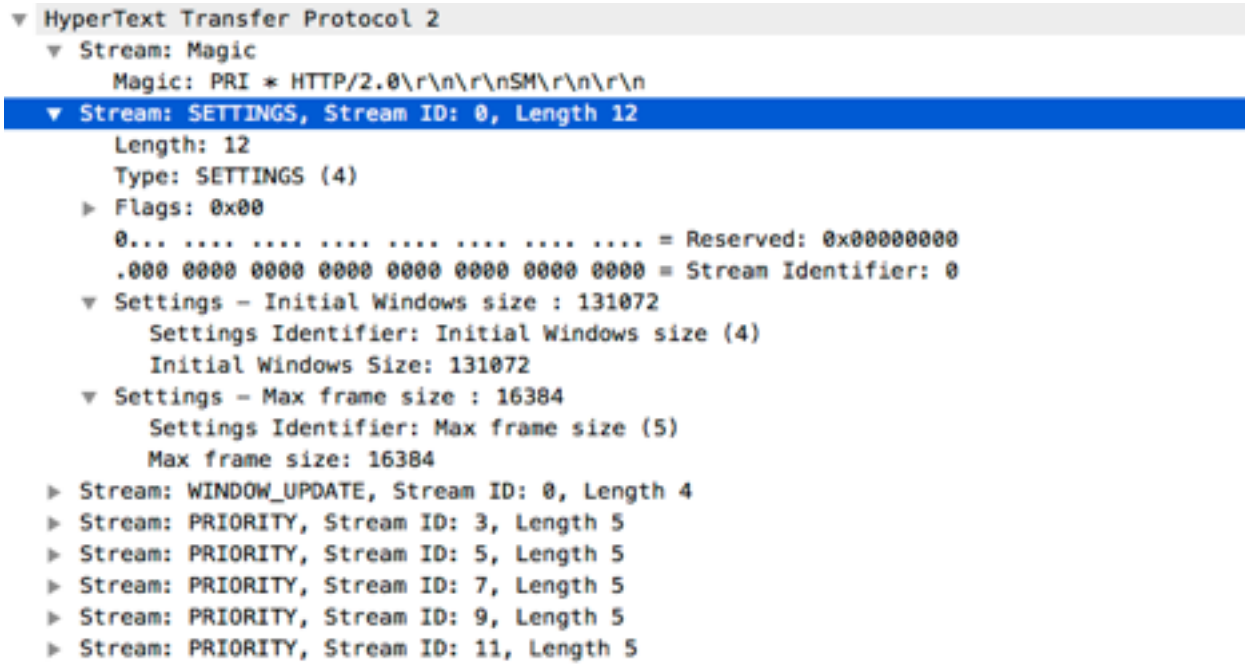
SETTINGS frame

No.	Time	Source	Destination	Protocol	Length	Info
6036	26.871477	192.168.1.161	151.101.41.63	HTTP2	223	Magic, SETTINGS, WINDOW_UPDATE, PRIORITY, PRIORITY, PRIORITY, PRIORITY, PRIORITY
6037	26.871507	192.168.1.161	151.101.41.63	HTTP2	277	HEADERS, WINDOW_UPDATE
6038	26.884571	151.101.41.63	192.168.1.161	HTTP2	125	SETTINGS, SETTINGS
6040	26.884906	192.168.1.161	151.101.41.63	HTTP2	104	SETTINGS
6056	27.176189	151.101.41.63	192.168.1.161	HTTP2	1015	HEADERS, DATA
6086	27.465413	192.168.1.161	94.46.159.28	HTTP2	211	Magic, SETTINGS, WINDOW_UPDATE, PRIORITY, PRIORITY, PRIORITY, PRIORITY, PRIORITY
6087	27.465508	192.168.1.161	94.46.159.28	HTTP2	254	HEADERS, WINDOW_UPDATE
6088	27.470270	192.168.1.161	94.46.159.28	HTTP2	92	SETTINGS
6089	27.479843	94.46.159.28	192.168.1.161	HTTP2	92	SETTINGS
6119	27.507205	94.46.159.28	192.168.1.161	HTTP2	1514	HEADERS, DATA
6136	27.531261	94.46.159.28	192.168.1.161	HTTP2	1514	DATA
6139	27.532658	94.46.159.28	192.168.1.161	HTTP2	496	DATA
6148	27.714003	192.168.1.161	151.101.41.63	HTTP2	145	HEADERS, WINDOW_UPDATE
6154	28.022941	151.101.41.63	192.168.1.161	HTTP2	941	HEADERS, DATA
6156	28.023766	192.168.1.161	151.101.41.63	HTTP2	140	HEADERS, WINDOW_UPDATE
6164	28.350460	151.101.41.63	192.168.1.161	HTTP2	917	HEADERS, DATA

SETTINGS frame



SETTINGS Frame



SETTINGS Frame

- SETTINGS_HEADER_TABLE_SIZE 4K
- SETTINGS_ENABLE_PUSH true
- SETTINGS_MAX_CONCURRENT_STREAMS unlimited
- SETTINGS_INITIAL_WINDOW_SIZE 64K
- SETTINGS_MAX_FRAME_SIZE 16K
- SETTINGS_MAX_HEADER_LIST_SIZE unlimited

HPACK (RFC 7541)

HPACK

55 pages

HPACK

- Addresses the issue of header bloat

```
$ cat req_bytes1 | awk '{print $4}' | grep -oE "[0-9]+"
| awk '{ print int(($0-1)/1000)*1000 + 1000; }' |  head
-n 1000 | sort -n | uniq -c | sort -n
      6 5000
     18 4000
     89 3000
    192 2000
    695 1000
```

HPACK

- More requests will fit in the initial window
- Smaller headers can help save a roundtrip

HPACK

- Addresses the issue of header bloat
- Two ways:
  - Huffman encoding
  - Indexed tables

HPACK - Huffman encoding

'U' (85)	|1110000	70	[7]
'V' (86)	|1110001	71	[7]
'W' (87)	|1110010	72	[7]
'X' (88)	|11111100	fc	[8]
'Y' (89)	|1110011	73	[7]
'Z' (90)	|11111101	fd	[8]
'[' (91)	|11111111|11011	1ffb	[13]
'\' (92)	|11111111|11111110|000	7fff0	[19]
']' (93)	|11111111|11100	1ffc	[13]
'^' (94)	|11111111|111100	3ffc	[14]
'_' (95)	|100010	22	[6]
'`' (96)	|11111111|1111101	7ffd	[15]
'a' (97)	|00011	3	[5]
'b' (98)	|100011	23	[6]
'c' (99)	|00100	4	[5]
'd' (100)	|100100	24	[6]
'e' (101)	|00101	5	[5]
'f' (102)	|100101	25	[6]
'g' (103)	|100110	26	[6]
'h' (104)	|100111	27	[6]
'i' (105)	|00110	6	[5]
'j' (106)	|1110100	74	[7]
'k' (107)	|1110101	75	[7]
'l' (108)	|101000	28	[6]
'm' (109)	|101001	29	[6]

HPACK - static table

Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206
11	:status	304
12	:status	400
13	:status	404
14	:status	500
15	accept-charset	
16	accept-encoding	gzip, deflate
17	accept-language	
18	accept-ranges	
19	accept	
20	access-control-allow-origin	
21	age	
22	allow	
23	authorization	
24	cache-control	
25	content-disposition	
26	content-encoding	
27	content-language	
28	content-length	
29	content-location	
30	content-range	

HPACK - dynamic tables

- Dynamic tables size is dictated via the SETTINGS frame
- The compression context is the connection, longer lived connections are favoured

HPACK - dynamic table

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com

Hex dump of encoded data:

8286 8441 8cf1 e3c2 e5f2 3a6b a0ab 90f4 | ...A......:k....
ff                                      | .

Decoding process:

82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :scheme: http
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :path: /
41                                      | == Literal indexed ==
                                        |   Indexed name (idx = 1)
                                        |     :authority
8c                                      |   Literal value (len = 12)
                                        |     Huffman encoded:
f1e3 c2e5 f23a 6ba0 ab90 f4ff           | .....:k.....
                                        |     Decoded:
                                        | www.example.com
                                        | -> :authority:
                                        |   www.example.com

Dynamic Table (after decoding):

[  1] (s =  57) :authority: www.example.com
      Table size:  57
```

HPACK - dynamic table

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
cache-control: no-cache

Hex dump of encoded data:

8286 84be 5885 a8eb 1064 9cbf           | ....X....d..

Decoding process:

82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :scheme: http
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :path: /
be                                      | == Indexed - Add ==
                                        |   idx = 62
                                        | -> :authority:
                                        |   www.example.com
58                                      | == Literal indexed ==
                                        |   Indexed name (idx = 24)
                                        |     cache-control
86                                      |   Literal value (len = 6)
                                        |     Huffman encoded:
a8eb 1064 9cbf                          | ...d..
                                        |     Decoded:
                                        | no-cache
                                        | -> cache-control: no-cache
Dynamic Table (after decoding):

[  1] (s =  53) cache-control: no-cache
[  2] (s =  57) :authority: www.example.com
Table size: 110
```

HPACK - security aspects

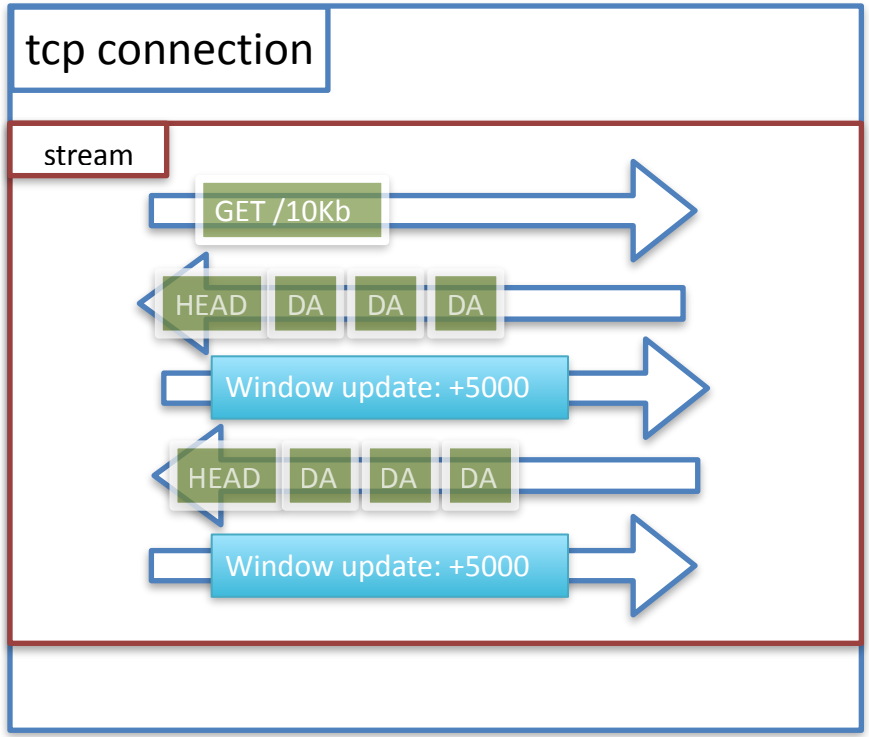
- Because compression can reveal information (https://en.wikipedia.org/wiki/CRIME), you can opt out compression.
- It’s possible to never compress information that’s sent, but you can’t control what the peer will do

Flow control

Flow control basics

- Goal: provide a mechanism for each side to control the flow of data - on top of TCP
- Hop by hop mechanism
  - client -> proxy -> server: 2 flow controls
- Credit system controlled by the receiver

Flow control



Flow control

- Per connection and per stream
- Connection always defaults to 64k
- Each stream defaults to settings
  - but that can be overridden by SETTINGS, INITIAL_WINDOW_SIZE
- WINDOW_UPDATE is used to make it bigger
- Max is 2^31 bytes

Flow control

- Cannot be disabled
- Only applies to DATA frames
- Negative windows are legal, they just mean the peer must not send any more DATA frames

Priorities

Prioritization basics

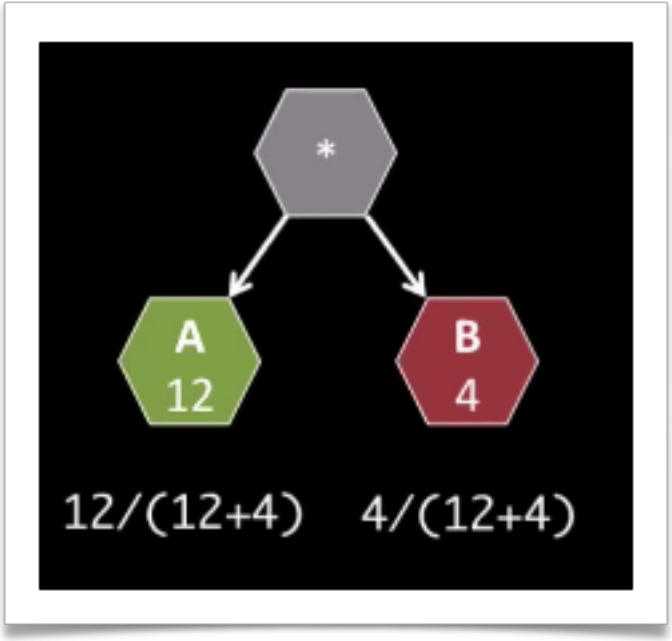
- Tools help manage HTTP/2's concurrency
- RFC doesn't specify the implementation details
- Can be set using `HEADERS` or `PRIORITY` frames
- `PRIORITY` can be sent for streams in any state

Prioritization basics

- Weight
  - 0-255 prioritizes siblings
- Parent stream id
  - Allows to build a dependency tree
- Exclusive flag

Flow control - Example 1

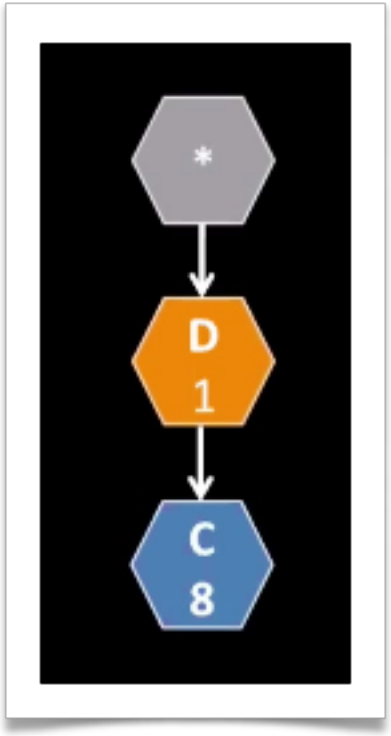
- A gets 3/4 of the resources
- B gets 1/4 of the resources



Example by Ilya Grigorik

Flow control - Example 2

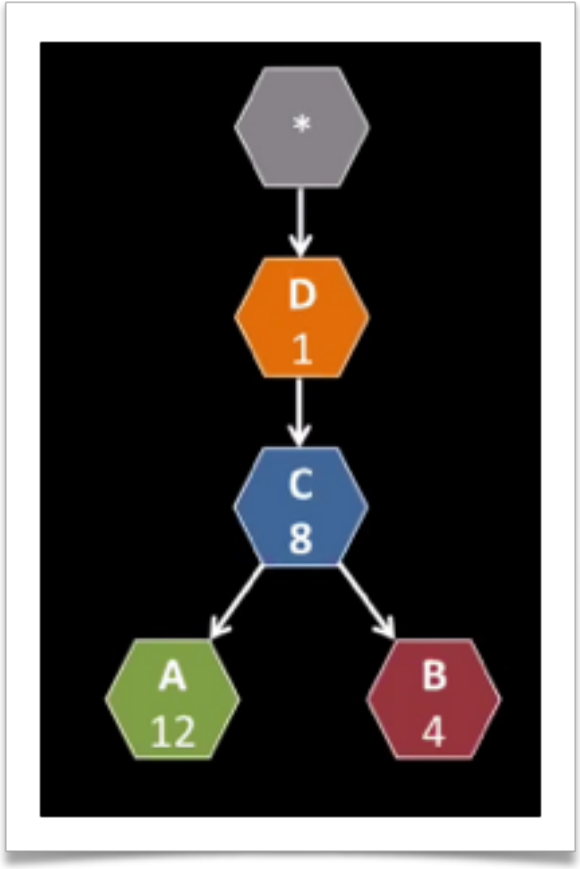
- D gets all resources
- After D is done, C gets all resources
- Weights have no meanings since there a re no siblings



Example by Ilya Grigorik

Flow control - Example 3

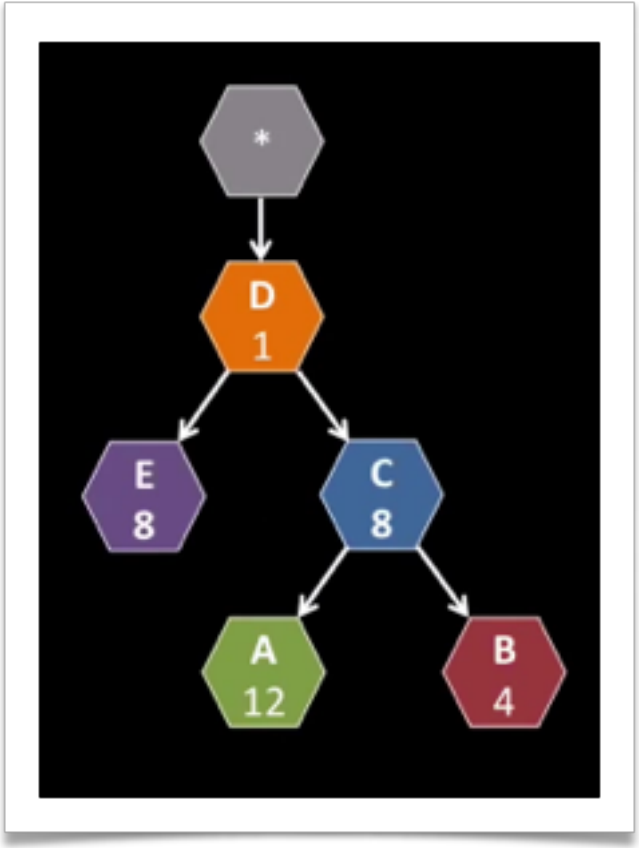
- D gets all resources
- After D is done, C gets all resources
- After C is done A gets 3/4 and B gets 1/4



Example by Ilya Grigorik

Flow control - Example 4

- D gets all resources
- After D is done, E and C get half the resources
- After C is done A gets 3/4 and B gets 1/4

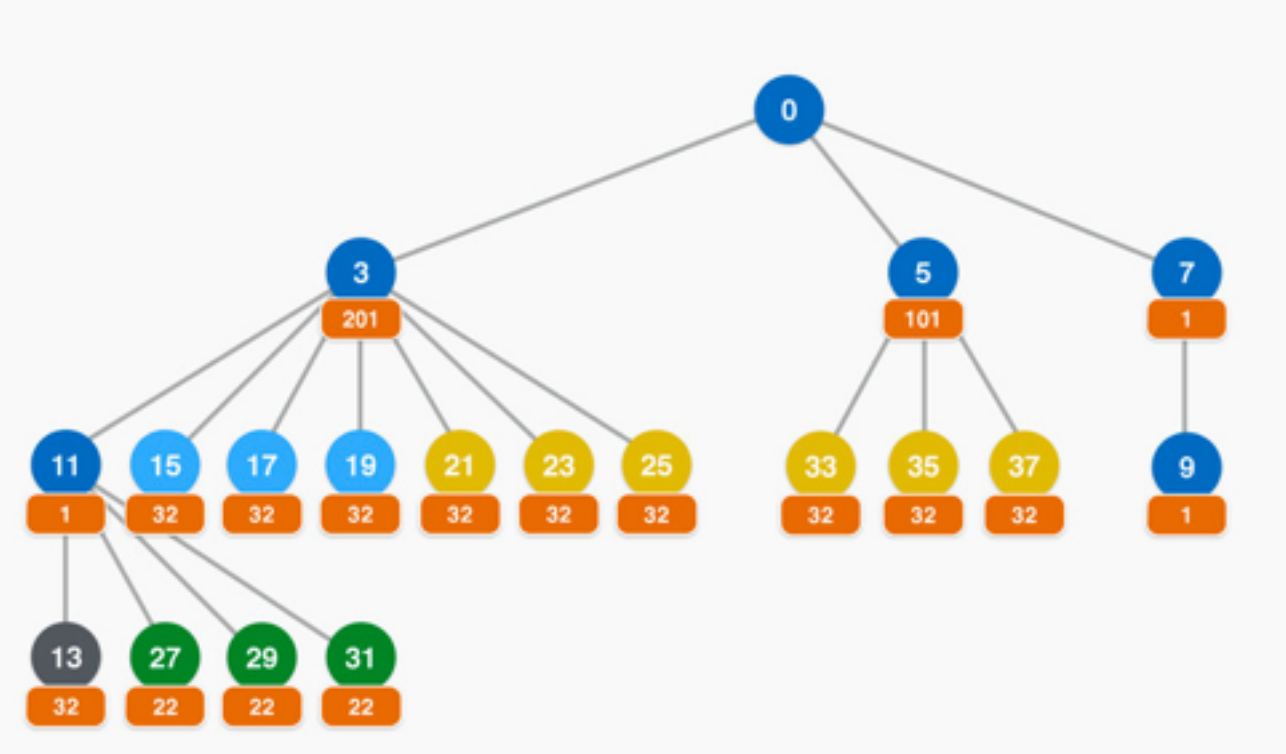


Example by Ilya Grigorik

Browser trees

Moto Ishizawa from Yahoo! Japan

Firefox priority tree

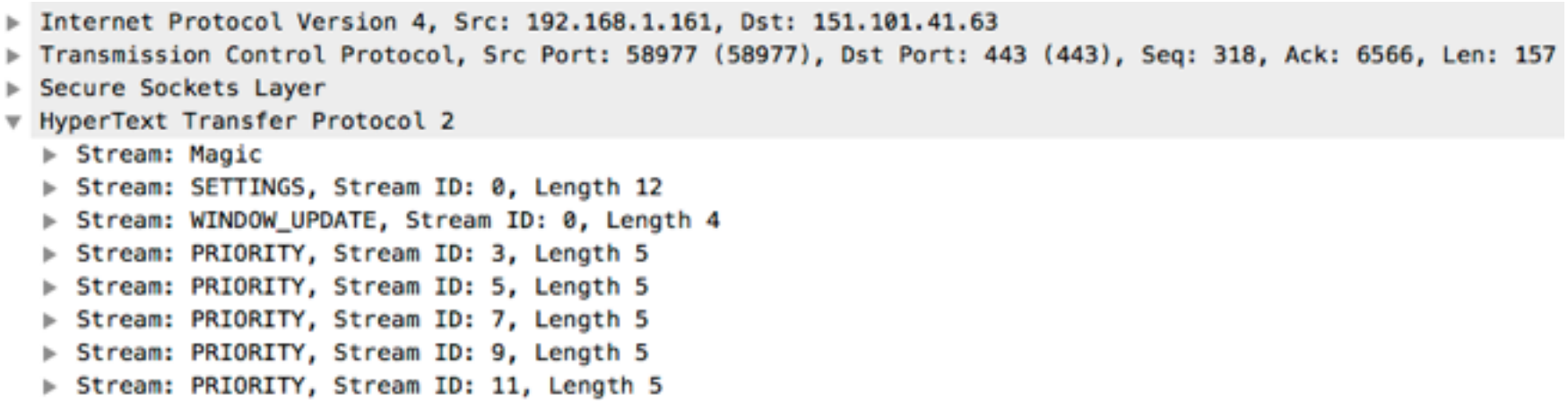


Understanding HTTP/2 prioritization by Moto Ishizawa

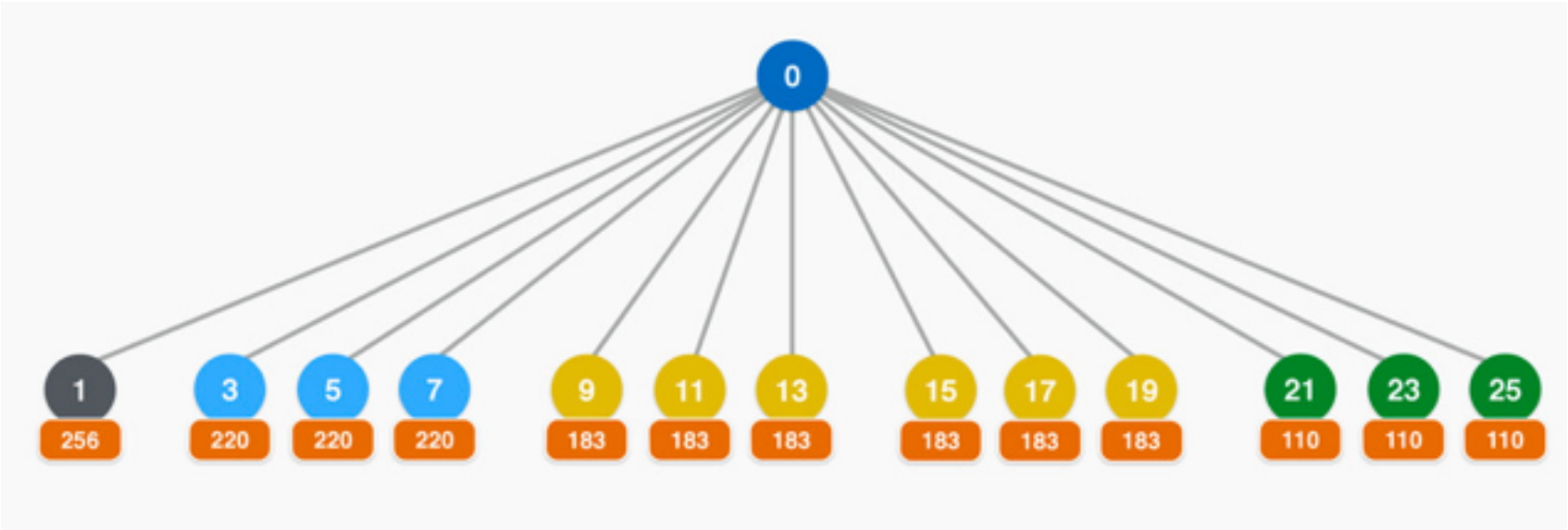
SETTINGS frame

No.	Time	Source	Destination	Protocol	Length	Info
6036	26.871477	192.168.1.161	151.101.41.63	HTTP2	223	Magic, SETTINGS, WINDOW_UPDATE, PRIORITY, PRIORITY, PRIORITY, PRIORITY, PRIORITY
6037	26.871507	192.168.1.161	151.101.41.63	HTTP2	277	HEADERS, WINDOW_UPDATE
6038	26.884571	151.101.41.63	192.168.1.161	HTTP2	125	SETTINGS, SETTINGS
6040	26.884906	192.168.1.161	151.101.41.63	HTTP2	104	SETTINGS
6056	27.176189	151.101.41.63	192.168.1.161	HTTP2	1015	HEADERS, DATA
6086	27.465413	192.168.1.161	94.46.159.28	HTTP2	211	Magic, SETTINGS, WINDOW_UPDATE, PRIORITY, PRIORITY, PRIORITY, PRIORITY, PRIORITY
6087	27.465508	192.168.1.161	94.46.159.28	HTTP2	254	HEADERS, WINDOW_UPDATE
6088	27.470270	192.168.1.161	94.46.159.28	HTTP2	92	SETTINGS
6089	27.479843	94.46.159.28	192.168.1.161	HTTP2	92	SETTINGS
6119	27.507205	94.46.159.28	192.168.1.161	HTTP2	1514	HEADERS, DATA
6136	27.531261	94.46.159.28	192.168.1.161	HTTP2	1514	DATA
6139	27.532658	94.46.159.28	192.168.1.161	HTTP2	496	DATA
6148	27.714003	192.168.1.161	151.101.41.63	HTTP2	145	HEADERS, WINDOW_UPDATE
6154	28.022941	151.101.41.63	192.168.1.161	HTTP2	941	HEADERS, DATA
6156	28.023766	192.168.1.161	151.101.41.63	HTTP2	140	HEADERS, WINDOW_UPDATE
6164	28.350460	151.101.41.63	192.168.1.161	HTTP2	917	HEADERS, DATA

SETTINGS frame

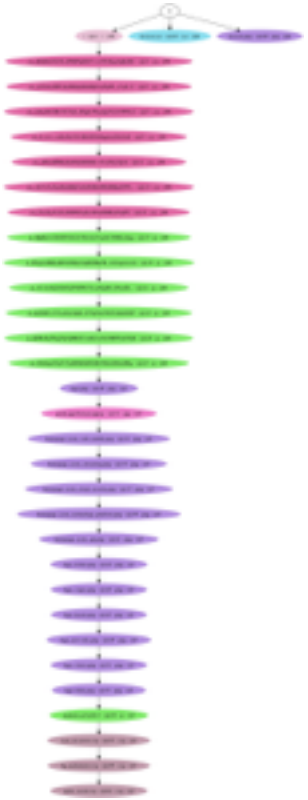


Chrome priority tree



Understanding HTTP/2 prioritization by Moto Ishizawa

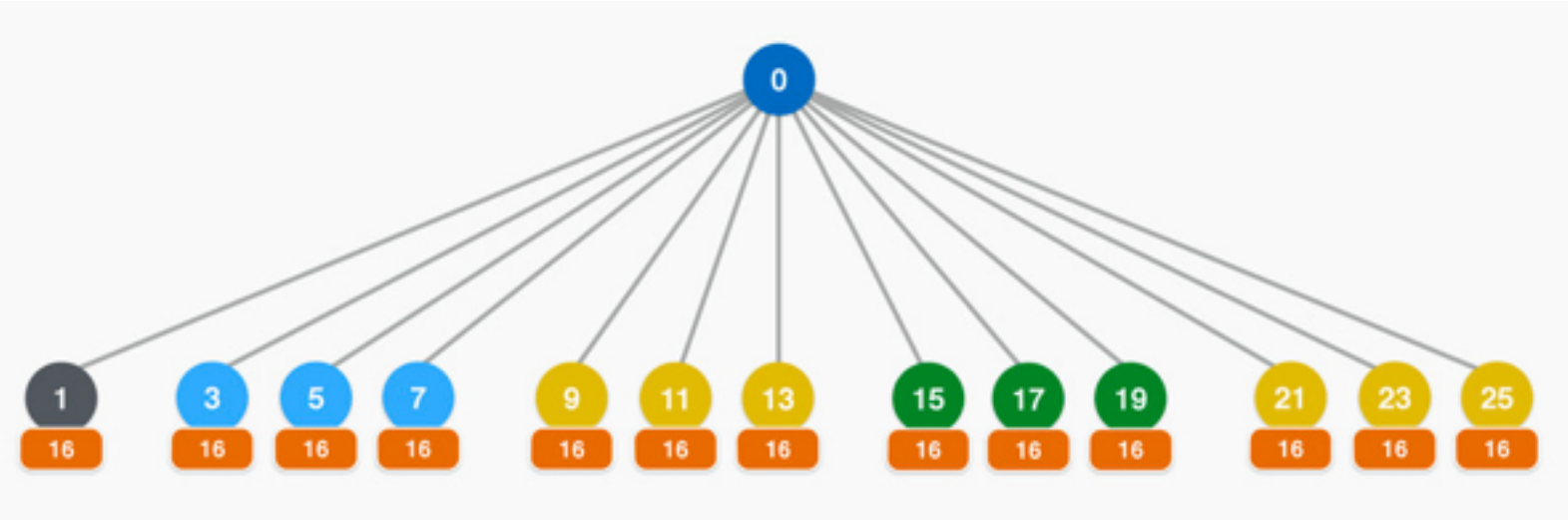
Chrome priority tree - new



Chrome priority tree - new

- All requests use exclusive dependencies
- Waterfalls might appear similar to http/1
- Will have HOL blocking for big objects

MS Edge priority tree



Understanding HTTP/2 prioritization by Moto Ishizawa

Server push

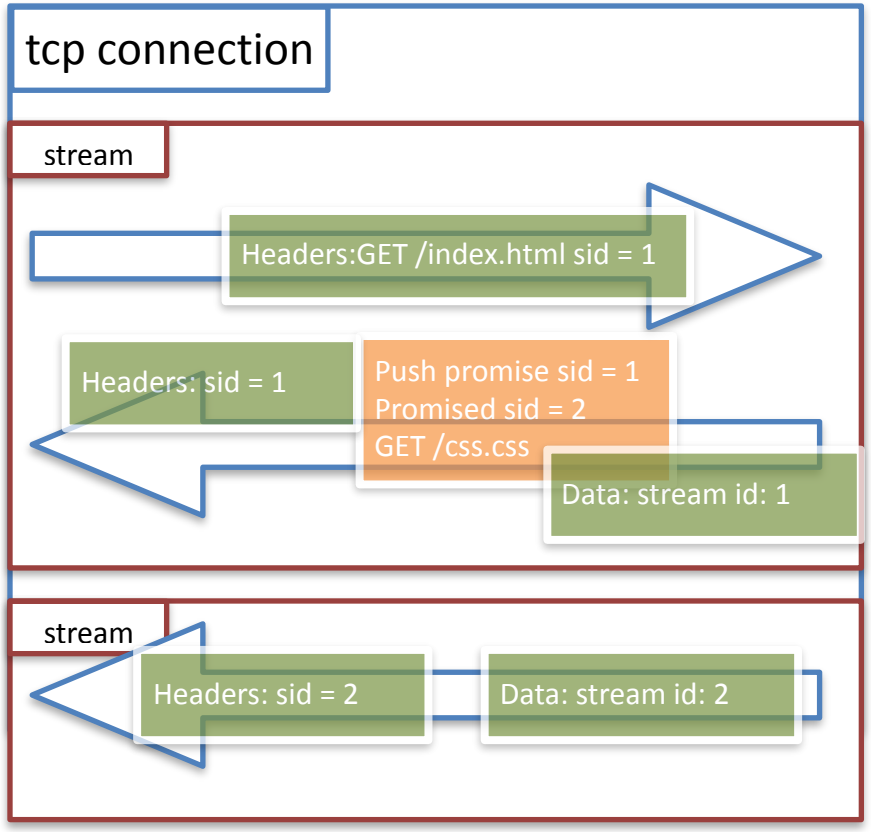
Server push basics

- Ability to push a resource before the client requests it
  - Possibly before the client even knows it needs it
- Operates hop by hop

Server push basics

- Only a server can push
- The capability to accept pushes is advertised through the SETTINGS frame

Push promise



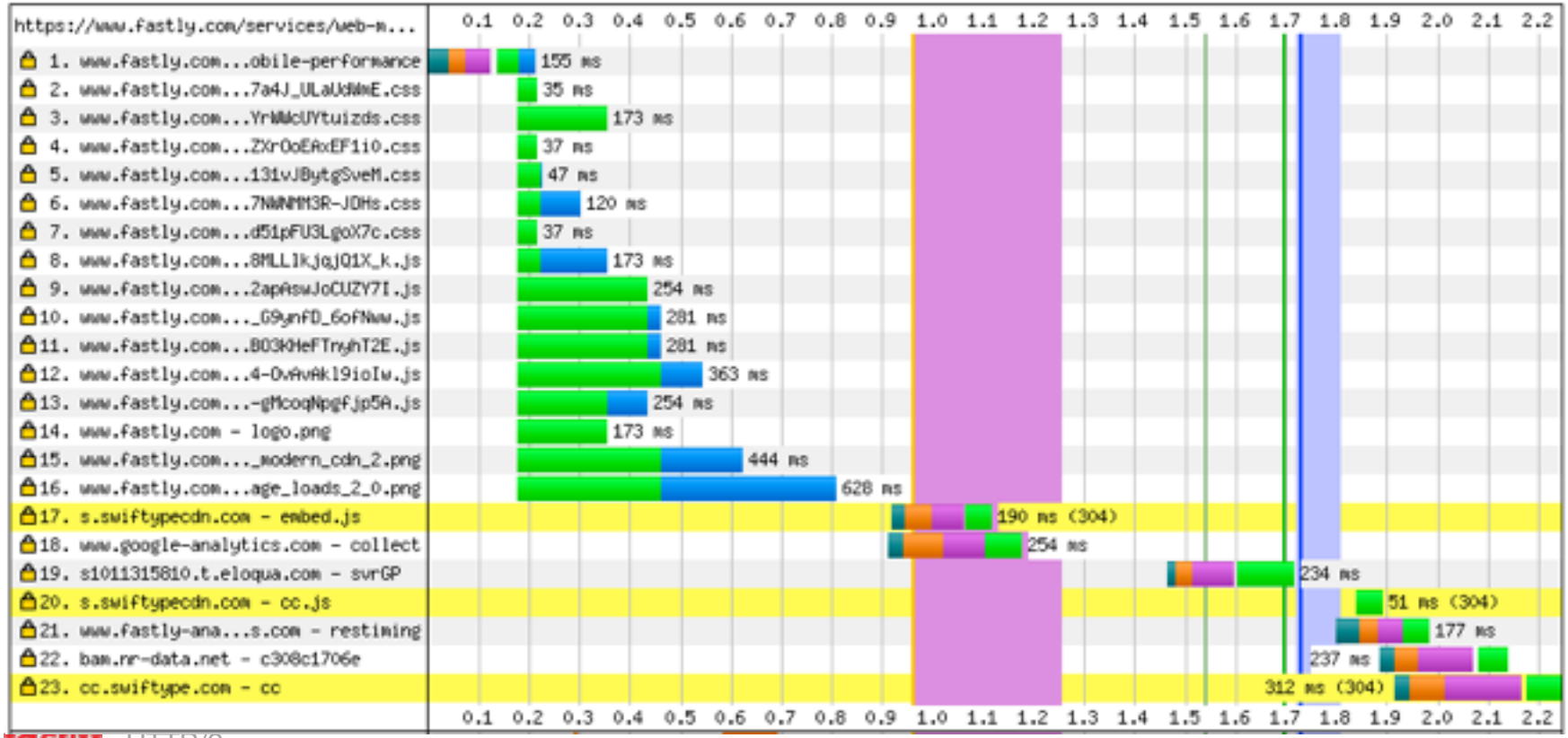
PUSH_PROMISE frame

- Key elements:
  - Would-be request headers for the resource (can be followed by CONTINUATION frames)
  - Promised stream id (even)
- Sent over a client initiated stream
- Client can reject with RST_STREAM

Server push

- What can be pushed:
  - Safe method (GET/HEAD/TRACE/OPTIONS)
  - No request body
- Client can control how many pushed streams it gets through the MAX_CONCURRENT_STREAMS entry in SETTINGS

Server push in action



Getting to know the client's cache

- HTTP/2 extension:
  - http://httpwg.org/http-extensions/cache-digest.txt
- Cookie based: H2O's CASPer (cache-aware server push)
  - https://h2o.examp1e.net/configure/http2_directives.html#http2-casper

Google's Rules of Thumb for HTTP/2 Push

- Push just enough resources to fill idle network time, and no more
- Push resources in evaluation-dependence order
- Consider using special strategies to track the client-side cache - see previous slide
- Use the right cookies when pushing resources that vary by cookie
- Use server push to fill the initial cwnd and consider using preload links to reveal the remaining critical or hidden resources
- https://docs.google.com/document/d/1K0NykTXBbbbTlv60t5MyJvXjqKGsCVNYHyLEXIxYMv0/edit

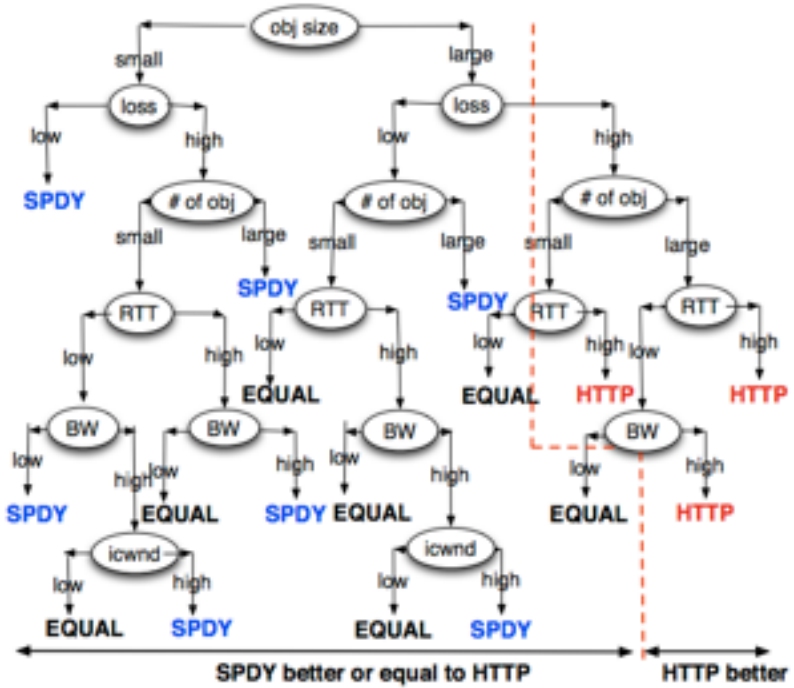
Two clear uses cases

- Use server thinking time
  - 1xx early hints
  - 307 trick with H2O
- Push from the edge

Performance

(it’s complicated)

The parameters



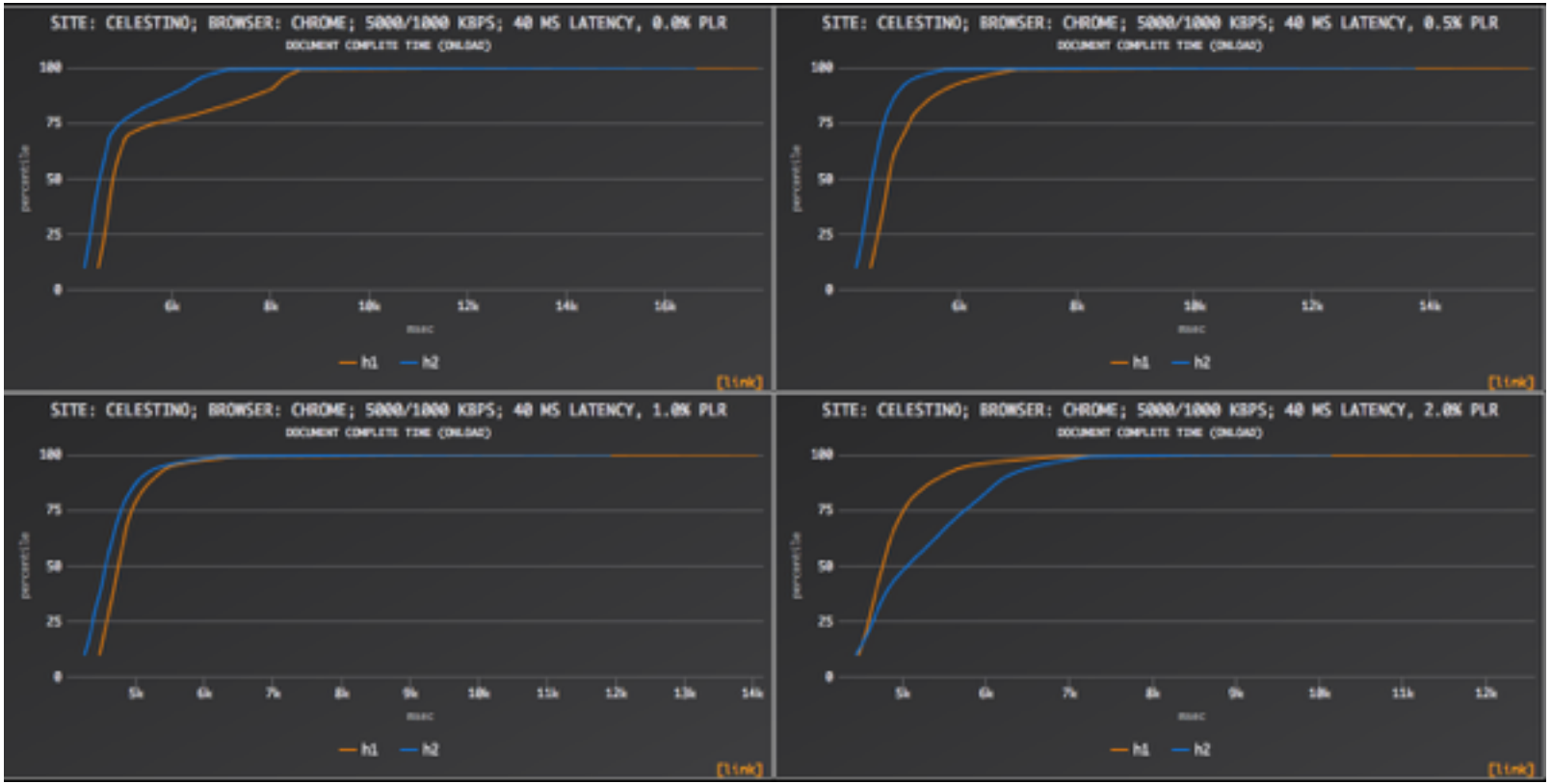
How Speedy is SPDY? - Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall, University of Washington

webpagetest.org

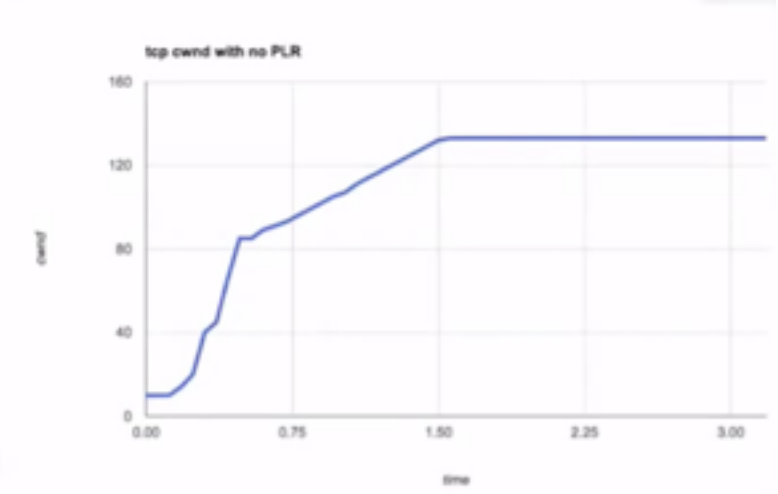
- Scriptable
- Performance tests for varying latencies, bandwidths, browsers
- Records videos, load times for all requests, tcpdump captures
- Can be self hosted: https://github.com/WPO-Foundation/webpagetest

“Classic” web page

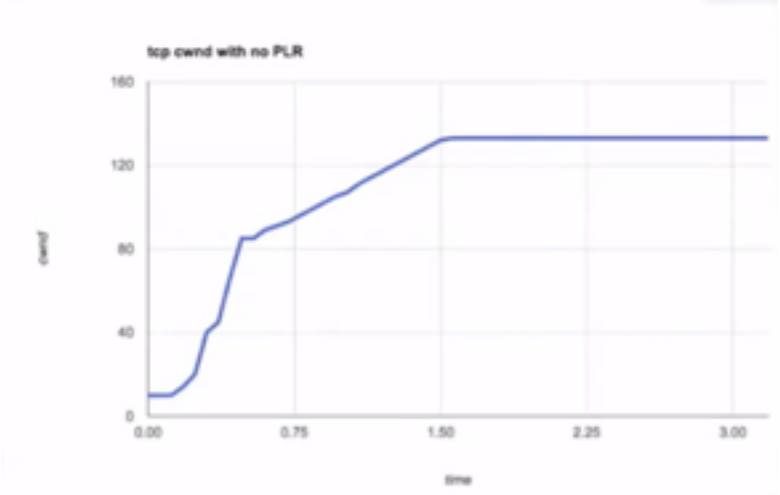
“Classic” web page

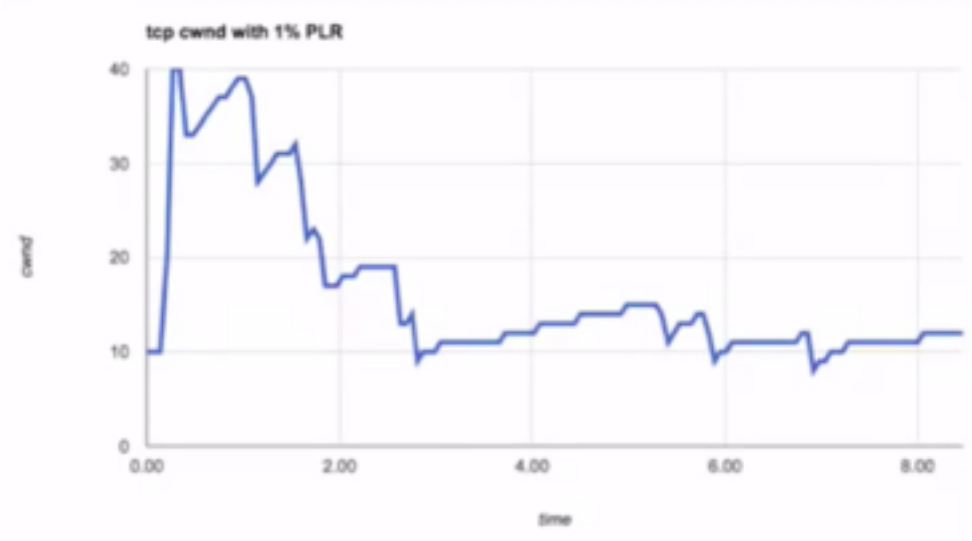


A single TCP connection



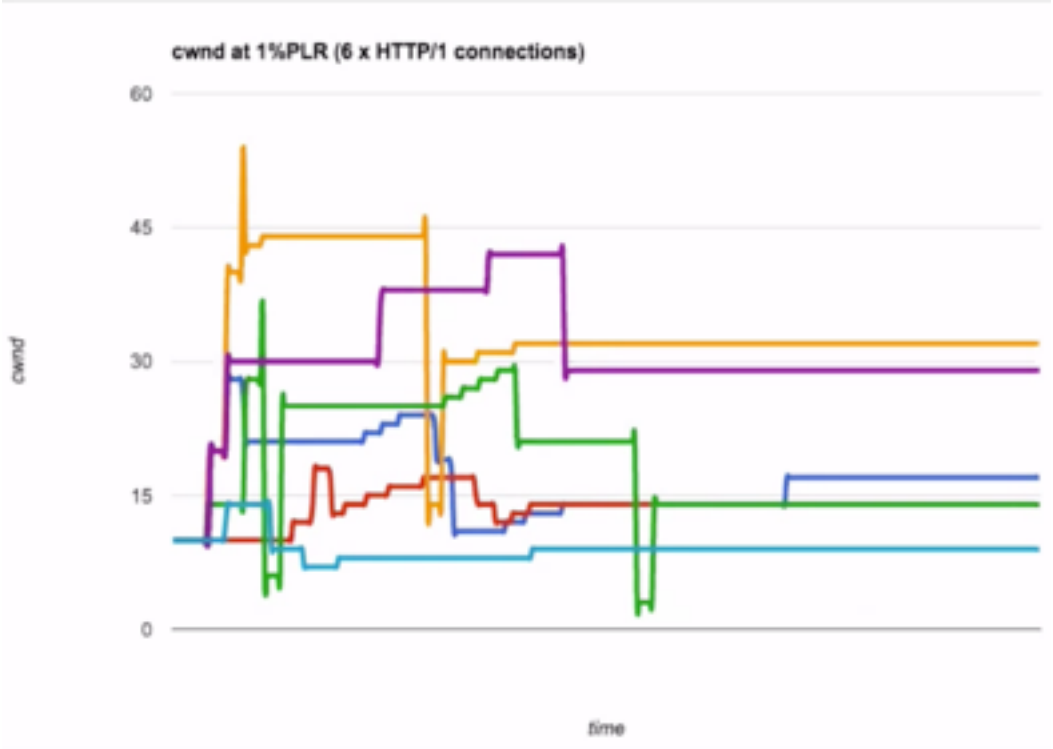
A single TCP connection



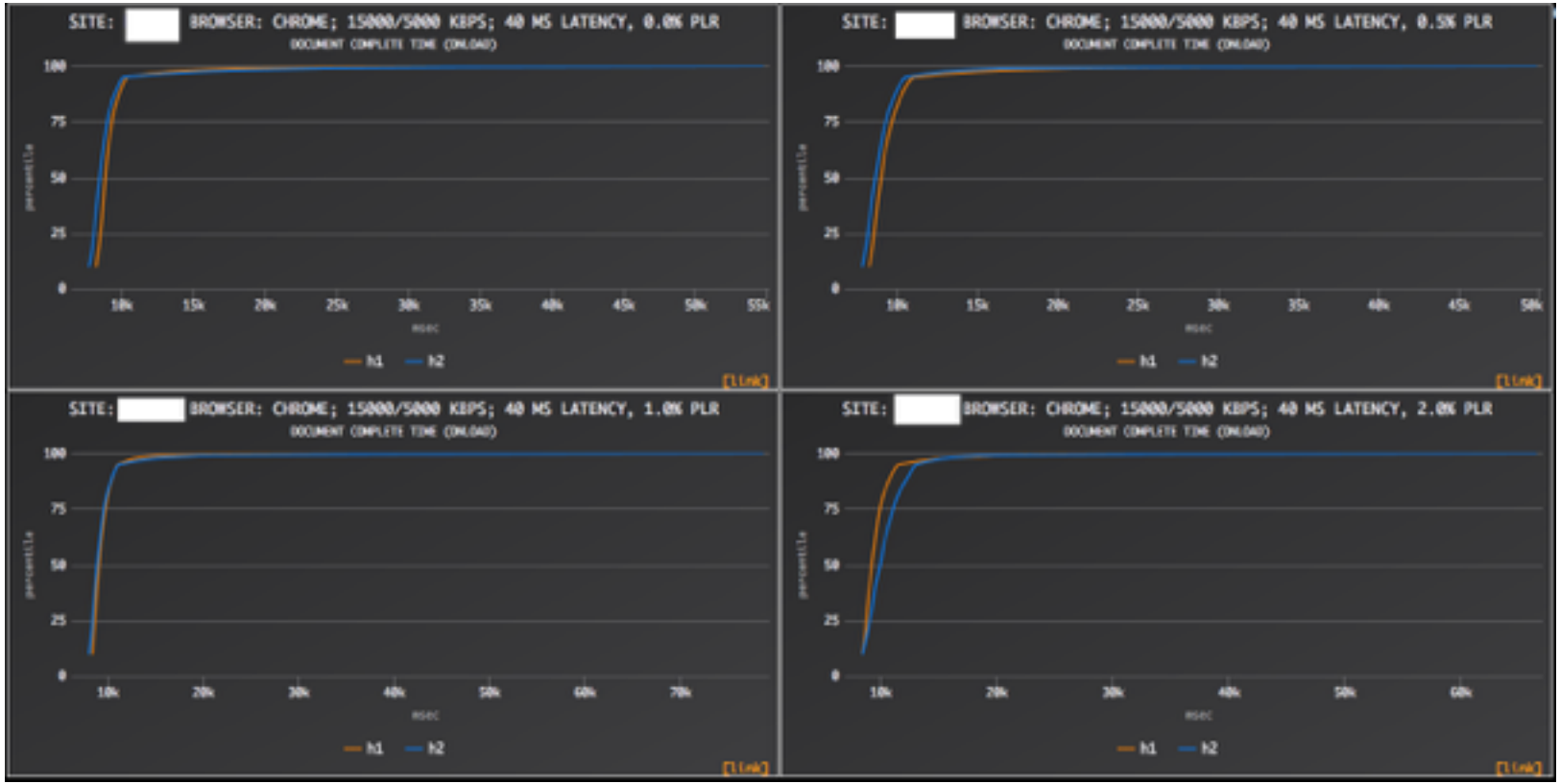


HTTP/1.1 with packet loss

Could sharding actually help even in HTTP/2



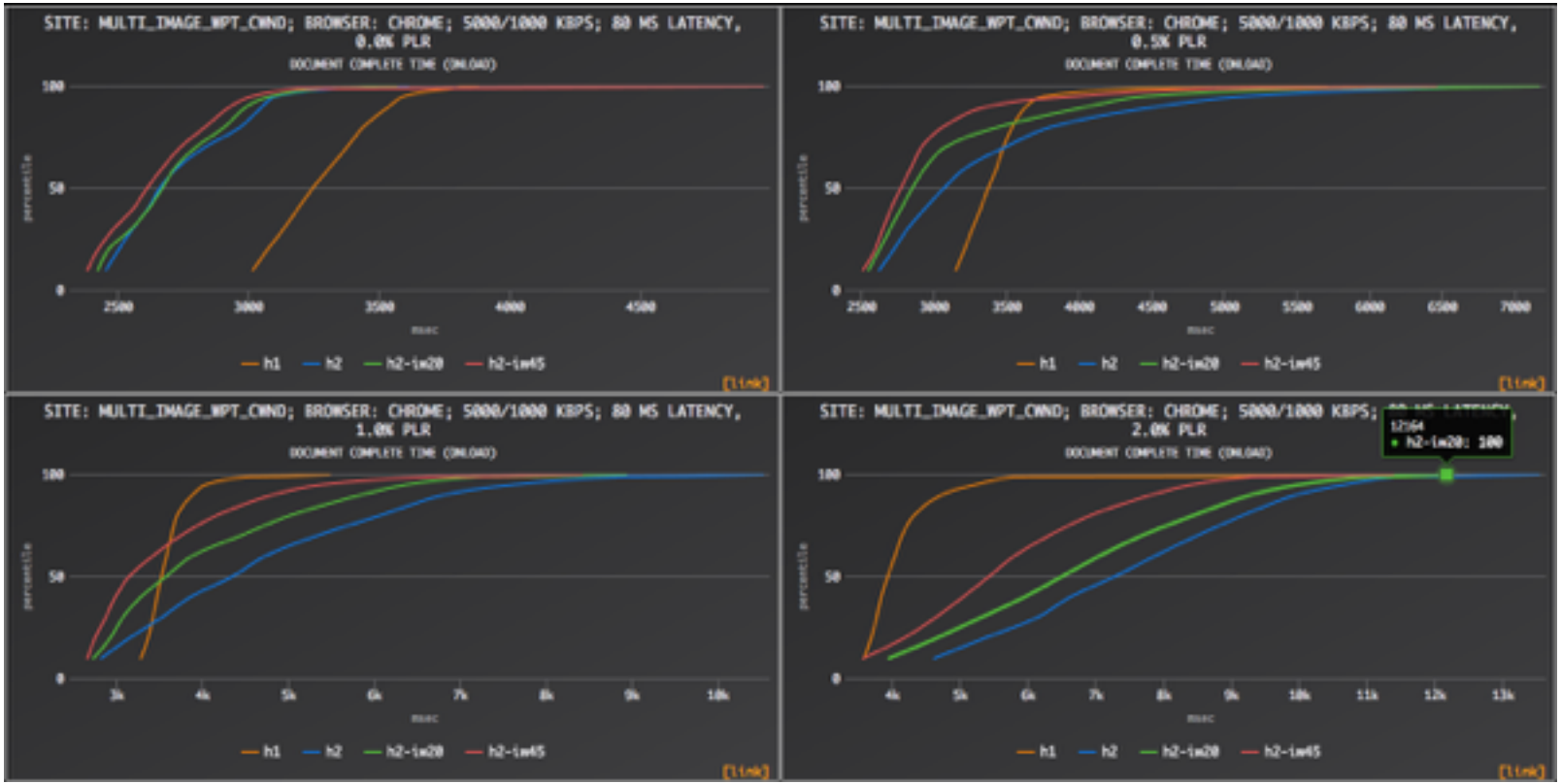
Real web page



ICWND experiment

- HTTP/1.1 has 6 times ICWND of 10
- What if we gave the H2 connection a ICWND of 60?

ICWND experiment



HTTP/2 implementations

Browsers


HTTP/2 protocol - OTHER
Global
70.13% + 5.95% = 76.08%
Networking protocol for low-latency transport of content over the web. Originally started out from the SPDY protocol, now standardized as HTTP version 2.
Current aligned
Usage relative
Show all
IE
Edge
Firefox
Chrome
Safari
Opera
iOS Safari
Opera Mini
Android Browser
Chrome for Android
8
11
13
14
47
48
49
50
51
29
49
50
51
52
53
54
55
9.1
10
TP
39
40
41
9.2
9.3
all
4.3
4.4
4.4.4
51
51
Notes
Known issues (0)
Resources (6)
Feedback
See also support for the SPDY protocol, precursor of HTTP2.
1 Partial support in IE11 refers to being limited to Windows 10.
2 Only supports HTTP2 over TLS (https)
3 Partial support in Safari refers to being limited to OSX 10.11+
4 Only supports HTTP2 if servers support protocol negotiation via ALPN

Servers

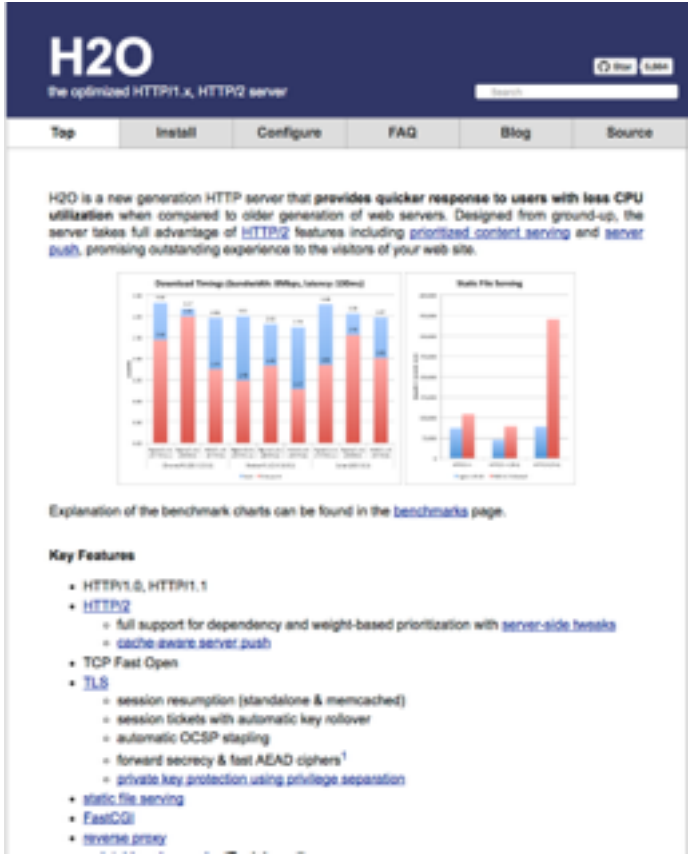
Server software

- Apache 2.4.12 supports HTTP/2 via the module mod_h2;[53] although appropriate patches must be applied to the source code of the server in order for it to support that module. As of Apache 2.4.17 all patches are included in the main Apache source tree, although the module itself was renamed mod_http2.[54] Old versions of SPDY were supported via the module mod_spdy,[55] however the development of the module has stopped.[56]
- Apache Tomcat supports HTTP/2 with version 8.5 and newer with a configuration change. [57]
- Apache Traffic Server supports HTTP/2.[58]
- Caddy supports HTTP/2.[59]
- Citrix NetScaler 11.x supports HTTP/2.[60]
- Sucuri Supports HTTP/2.[61]
- F5 BIG-IP Local Traffic Manager 11.6 supports HTTP/2.[62]
- h2o was built from the ground up for HTTP/2 support.[63]
- Jetty 9.3 supports HTTP/2.[64]
- LiteSpeed Web Server 5.0 supports HTTP/2.[65]
- Microsoft IIS supports HTTP/2 in Windows 10[66] and Windows Server 2016.
- Netty 4.1 supports HTTP/2.[67]
- nginx 1.9.5 supports HTTP/2.[68]
- node.js 5.0 supports HTTP/2.[69]
- OpenLiteSpeed 1.3.11 and 1.4.8 supports HTTP/2.[70]
- Proxygen supports HTTP/2.
- Radware Alteon NG supports HTTP/2.[71]
- ShimmerCat was built from the ground up for HTTP/2 support.[72]
- Vert.x 3.3 supports HTTP/2
- Warp (Haskell web server, used by default in Yesod) supports HTTP/2.
- Wildfly 9 supports HTTP/2.

Content delivery networks

- Akamai is the first major CDN to support HTTP/2 and HTTP/2 Server Push. http2.akamai.com showcases Akamai's HTTP/2 implementation, including Server Push.
- CDN77 supports HTTP/2 using nginx (August 20, 2015). http2demo.io is a demonstration of CDN77's HTTP/2 implementation.
- CloudFlare supports HTTP/2 using nginx with SPDY as a fallback for browsers without support, whilst maintaining all security and performance services.[73] CloudFlare are the first major CDN to support HTTP/2 Server Push.[74]
- Fastly supports HTTP/2 including Server Push.[75]
- Imperva Incapsula CDN supports HTTP/2.[76] http2.incapsula.com showcases Incapsula's HTTP/2 implementation. The implementation includes support for WAF and DDoS mitigation features as well.
- KeyCDN supports HTTP/2 using nginx (October 6, 2015). HTTP/2 Test is a test page to verify if your server supports HTTP/2.

Servers



H2O

the optimized HTTP/1.x, HTTP/2 server

Top | Install | Configure | FAQ | Blog | Source

H2O is a new generation HTTP server that **provides quicker response to users with less CPU utilization** when compared to older generation of web servers. Designed from ground-up, the server takes full advantage of HTTP/2 features including prioritized content serving and server push, promising outstanding experience to the visitors of your web site.

Explanation of the benchmark charts can be found in the benchmarks page.

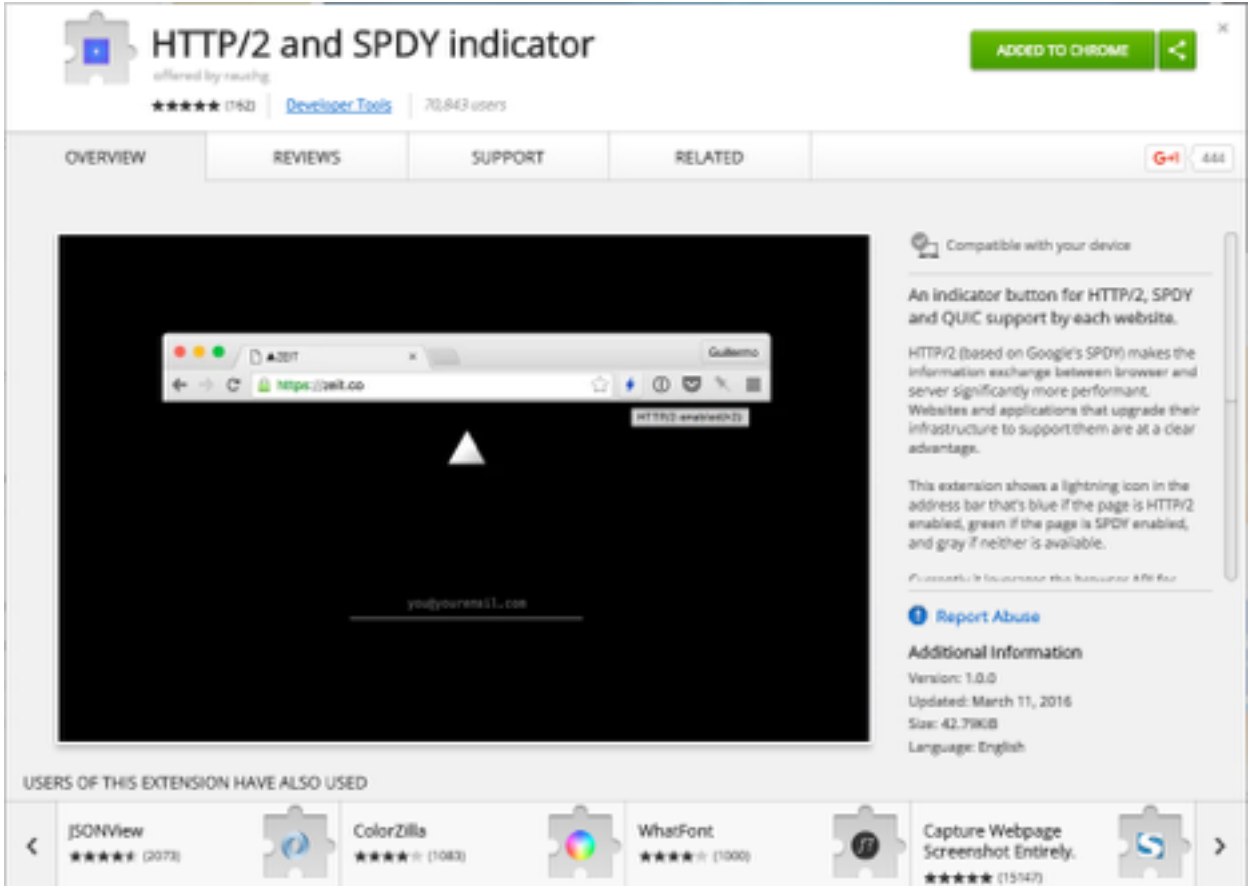
Key Features

- HTTP/1.0, HTTP/1.1
- HTTP/2
  - full support for dependency and weight-based prioritization with server-side tweaks
  - cache aware server push
- TCP Fast Open
- TLS
  - session resumption (standalone & memcached)
  - session tickets with automatic key rollover
  - automatic OCSP stapling
  - forward secrecy & fast AEAD ciphers[1]
  - private key protection using privilege separation
- static file serving
- FastCGI
- reverse proxy

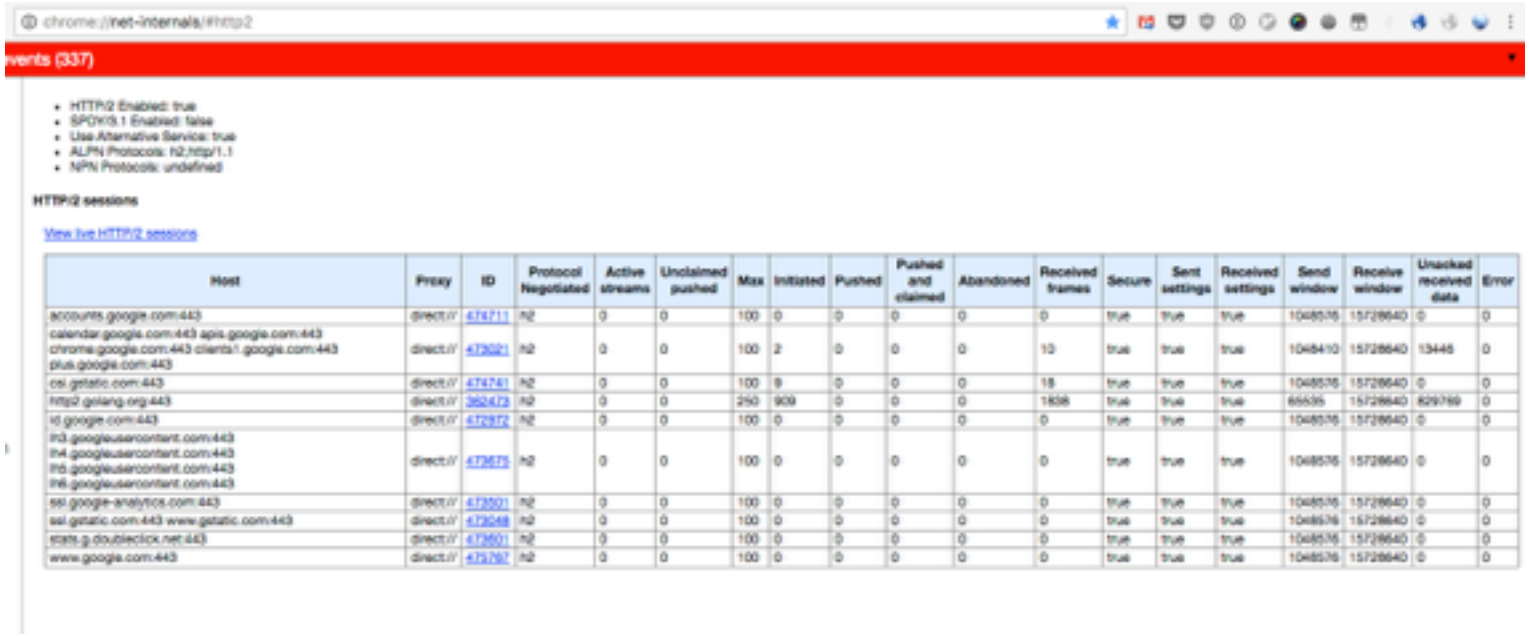
Servers - H2O

- In production at Fastly since July — opt-in
- 150K req/s
- We contribute code back as much as possible: 26 PRs

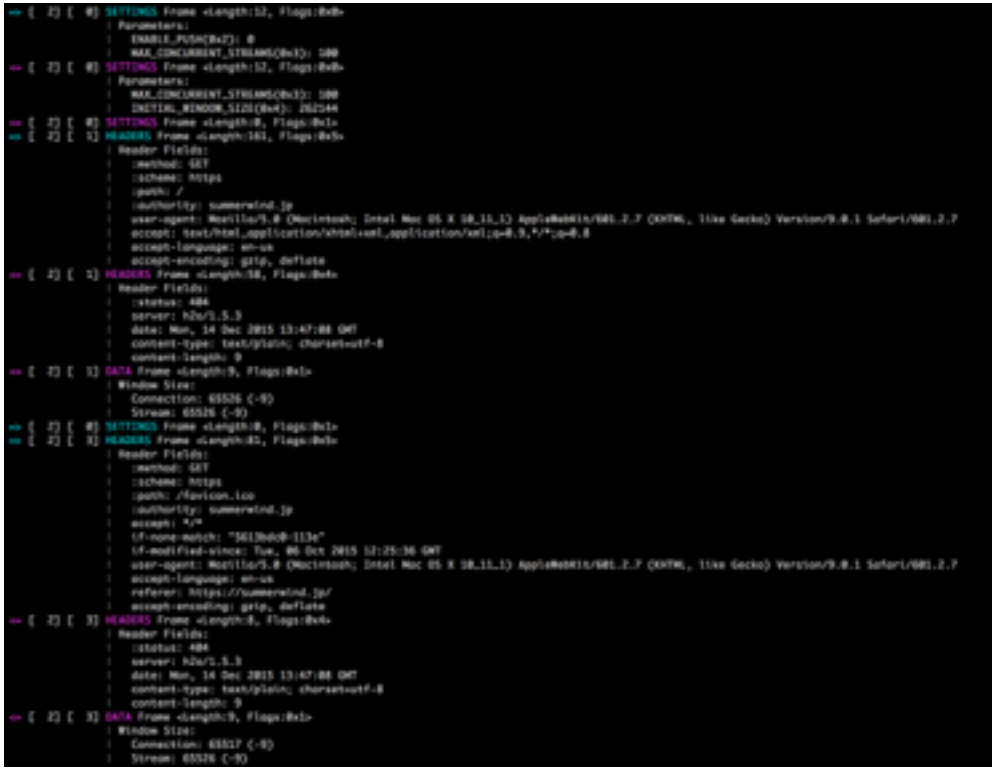
HTTP/2 tooling


HTTP/2 and SPDY indicator
offered by rauchg
★★★★★ (162)
Developer Tools
70,843 users
ADDED TO CHROME
OVERVIEW
REVIEWS
SUPPORT
RELATED
Compatible with your device
An indicator button for HTTP/2, SPDY and QUIC support by each website.
HTTP/2 (based on Google's SPDY) makes the information exchange between browser and server significantly more performant. Websites and applications that upgrade their infrastructure to support them are at a clear advantage.
This extension shows a lightning icon in the address bar that's blue if the page is HTTP/2 enabled, green if the page is SPDY enabled, and gray if neither is available.
Report Abuse
Additional Information
Version: 1.0.0
Updated: March 11, 2016
Size: 42.79KiB
Language: English
USERS OF THIS EXTENSION HAVE ALSO USED
JSONView
ColorZilla
WhatFont
Capture Webpage Screenshot Entirely.

Chrome’s net internals



h2a - a mitmproxy



curl - since 7.43.0

```
$  curl --http2 -I  https://www.fastly.com
HTTP/2.0 200
date:Thu, 18 Aug 2016 22:54:47 GMT
content-language:en
content-type:text/html; charset=utf-8
link:<https://www.fastly.com/>; rel="canonical",<https://www.fastly.com/>; rel="shortlink"
via:1.1 varnish
x-content-type-options:nosniff
x-generator:Drupal 7 (http://drupal.org)
last-modified:Thu, 18 Aug 2016 01:51:46 GMT
via:1.1 varnish
```

nghttp2

```
$ nghttp -nv https://www.fastly.com
[  0.017] Connected
The negotiated protocol: h2
[  0.045] send SETTINGS frame <length=12, flags=0x00, stream_id=0>
          (niv=2)
          [SETTINGS_MAX_CONCURRENT_STREAMS(0x03):100]
          [SETTINGS_INITIAL_WINDOW_SIZE(0x04):65535]
[  0.045] send PRIORITY frame <length=5, flags=0x00, stream_id=3>
          (dep_stream_id=0, weight=201, exclusive=0)
[  0.045] send PRIORITY frame <length=5, flags=0x00, stream_id=5>
          (dep_stream_id=0, weight=101, exclusive=0)
[  0.045] send PRIORITY frame <length=5, flags=0x00, stream_id=7>
          (dep_stream_id=0, weight=1, exclusive=0)
[  0.045] send PRIORITY frame <length=5, flags=0x00, stream_id=9>
          (dep_stream_id=7, weight=1, exclusive=0)
[  0.045] send PRIORITY frame <length=5, flags=0x00, stream_id=11>
          (dep_stream_id=3, weight=1, exclusive=0)
[  0.045] send HEADERS frame <length=39, flags=0x25, stream_id=13>
          ; END_STREAM | END_HEADERS | PRIORITY
          (padlen=0, dep_stream_id=11, weight=16, exclusive=0)
          ; Open new stream
          :method: GET
          :path: /
          :scheme: https
          :authority: www.fastly.com
          accept: */*
          accept-encoding: gzip, deflate
          user-agent: nghttp2/1.13.0
[  0.058] recv SETTINGS frame <length=12, flags=0x00, stream_id=0>
          (niv=2)
          [SETTINGS_MAX_CONCURRENT_STREAMS(0x03):100]
          [SETTINGS_INITIAL_WINDOW_SIZE(0x04):16777216]
[  0.058] recv SETTINGS frame <length=0, flags=0x01, stream_id=0>
          ; ACK
          (niv=0)
[  0.058] send SETTINGS frame <length=0, flags=0x01, stream_id=0>
          ; ACK
          (niv=0)
```

Scripting languages

- Python
  - https://github.com/python-hyper/hyper-h2
- Ruby
  - https://github.com/igrigorik/http-2

Future

TCP HOL blocking

- Servers could be smarter by not overcommitting data at the beginning of the connection

TLS 1.3

- 0-RTT handshake
- Preliminary work for H2O:
  - github.com/h2o/picotls

QUIC

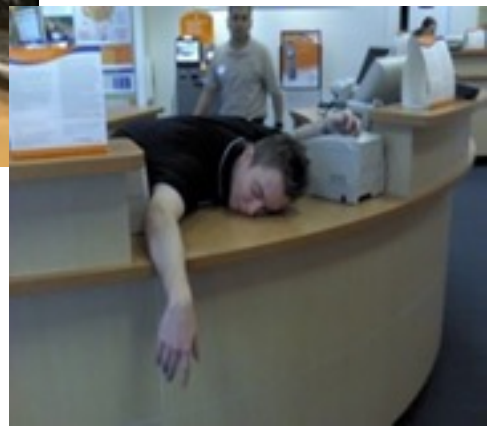
- Now has a working group, since 25th Oct
  - https://datatracker.ietf.org/wg/quic/charter/
- UDP
  - eliminates TCP HOL
- Full control of congestion control algorithms
  - Lesson from slowly evolving TCP stacks

QUIC

- Chrome
- Server implementations:
  - https://github.com/google/proto-quic
  - https://devsisters.github.io/goquic

To recap

- HTTP/2 is really a browser transport protocol
- We've moved blocking to TCP
- We need more real world data to understand HTTP/2 performance



Thank you