http2

Berkeley #wt294

bradfitz@golang.org
2015-04-15

Me

- Brad Fitzpatrick
- bradfitz {.com, twitter, @golang.org}
- Go standard library
  - net/http, ...
- Camlistore
- Previous life:
  - LiveJournal, memcached, OpenID, Gearman, MogileFS, djabberd
  - HTTP/1.x servers/proxies

Talk overview

- HTTP/2: what, why, history
- Writing an HTTP/2 server (library) in Go!

History

Gopher,

HTTP/0.9,

HTTP/1.0,

HTTP/1.1,

HTTP/2

Gopher (1991)

/Reference\r\n

1CIA World Factbook /Archives/mirrors/textfiles.com/politics/CIA gopher.quux.org 70

0Jargon 4.2.0 /Reference/Jargon 4.2.0 gopher.quux.org 70 +

1Online Libraries /Reference/Online Libraries gopher.quux.org 70 +

(disconnect)

HTTP/0.9 (1991)

```
GET /\r\n
<HTML>
<HEAD><TITLE>...</TITLE></HEAD>
<BODY>
...
</BODY></HTML>
```

(disconnect)

HTTP/1.0 (1996)

```
GET / HTTP/1.0\r\n
User-Agent: Mozilla/3.0 (X11; blah)\r\n
Cookie: foo=val\r\n
\r\n
HTTP/1.0 200 OK
Content-Type: text/html

<HTML>...
(disconnect)
```

HTTP/1.0, evolved

```
GET / HTTP/1.0\r\n
User-Agent: Mozilla/3.0 (X11; blah)\r\n
Connection: keep-alive\r\n
\r\n
HTTP/1.0 200 OK
Content-Length: 1203
Content-Type: text/html
Connection: keep-alive

<HTML>...
GET /underconstruction.gif HTTP/1.1\r\n  ...
```

HTTP/1.1 (~1999)

```
GET / HTTP/1.1\r\n
User-Agent: Mozilla/3.0 (X11; blah)\r\n
Host: example.com\r\n
\r\n
HTTP/1.1 200 OK
Content-Type: text/html
Transfer-Encoding: chunked

3FF
<HTML>...
GET /underconstruction.gif HTTP/1.1\r\n ...
```

(**demo**)

$ telnet ip.appspot.com 80

HTTP/1.1 (~1999)

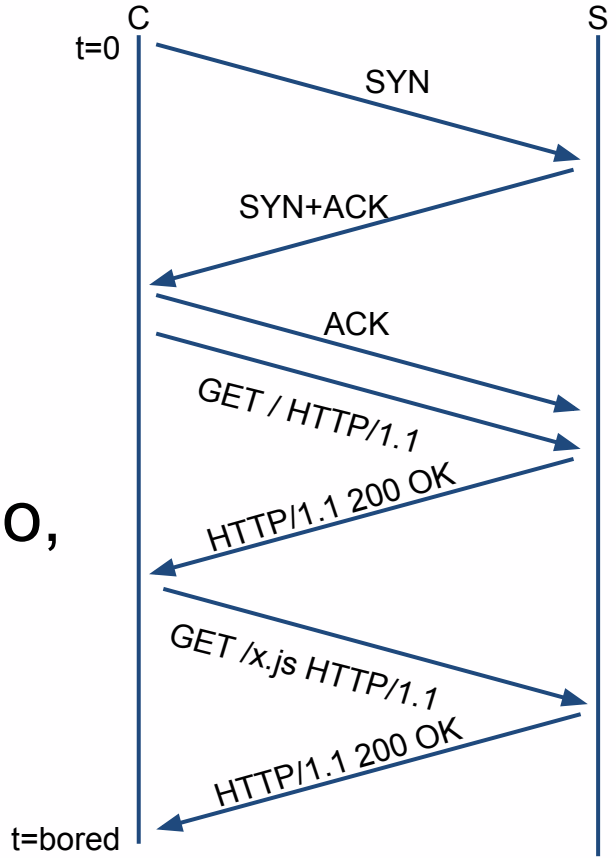
```
GET / HTTP/1.1\r\n
User-Agent: Mozilla/3.0 (X11; blah)\r\n
Host: example.com\r\n
\r\n
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1023

<HTML>...
GET /underconstruction.gif HTTP/1.1\r\n
...
```

HTTP/2 (2012-)

```
0x00000c0105000000018284668 82f91d35d055c87a7
```

Problems with HTTP/1.x

- One thing at a time
- Lots of waiting
- TCP setup, slow start
- TLS handshakes too: ClientHello, ServerHello
- requests are ~800B+cookies

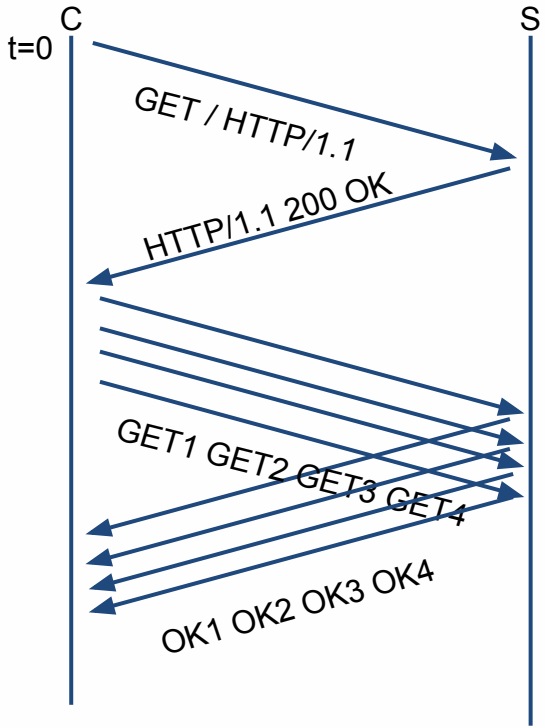


Problems with HTTP/1.x

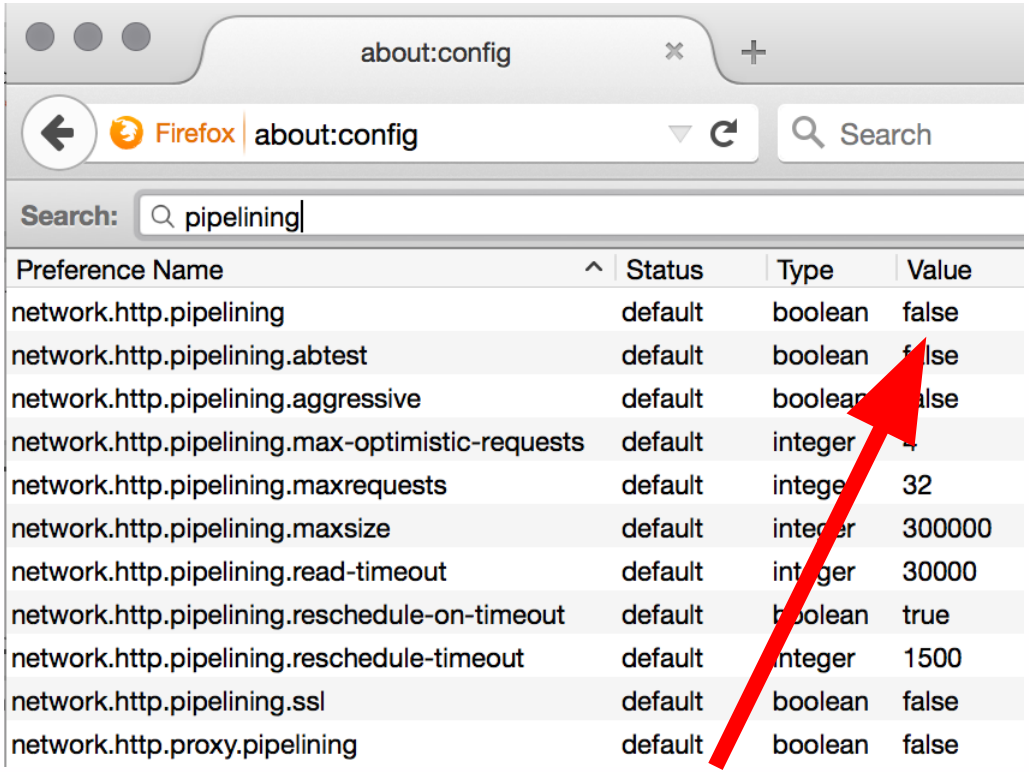
- No way for servers to gracefully shut down
  - Server TCP FIN & client request both in flight
- No way for client to abort a request without killing TCP connection
- Workaround, special cases

HTTP/1.1 pipelining

IN THEORY



IN PRACTICE



Proxies

- proxies (transparent and otherwise) try to “add value”
- don’t usually speak full, compliant HTTP

Rule of the the Internet: speak UDP, TCP:80, TCP:443 and don’t try anything fancy, including full HTTP/1.1

HTTP/1.1 from 1999-2013

- No real improvements
- Pipelining? Fail.
- 2 connections at once? Sure.
- 6 connections at once? Sure.

HTTP/1.1: More hacks

- Domain sharding: 6 * N parallel TCP connections! :(

  tiles{1,2,N}.googlemaps.com

- Spriting, concat, inlining: fighting cacheability, dev costs

Fix HTTP instead!

SPDY (~2009)

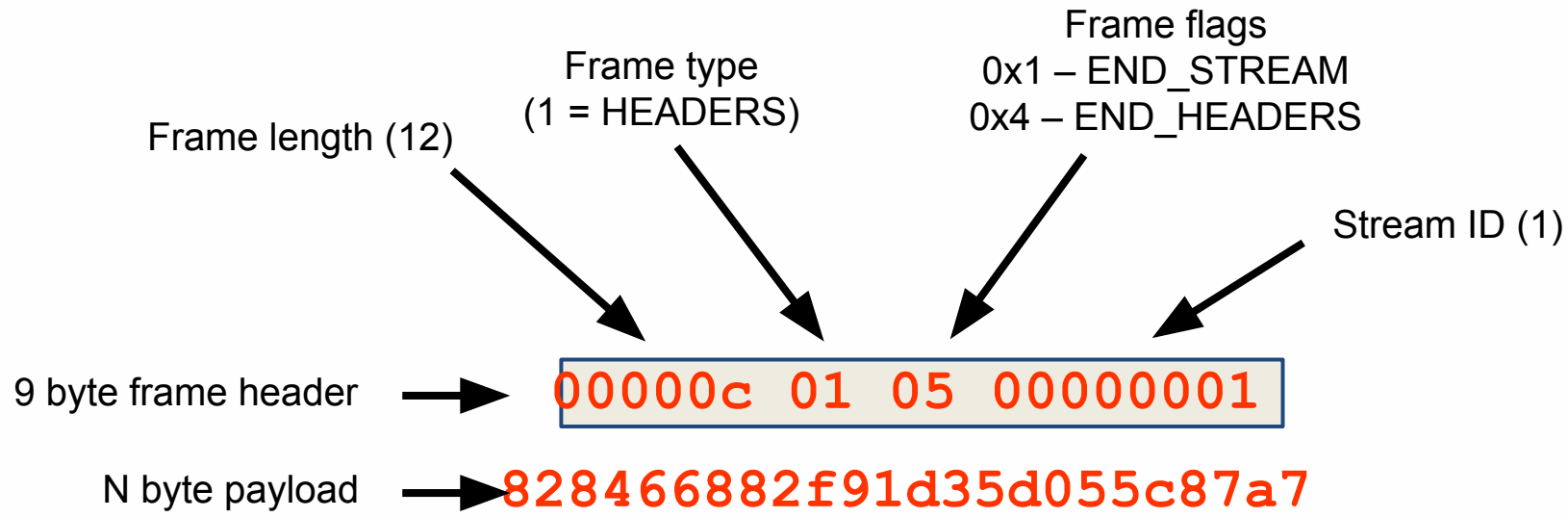
- Google
- Chrome + GFE experiments
- became starting point of HTTP/2

Single request frame

```
0x00000c0105000000001828466882f91d35d055c87a7
```

00000c 01 05 00000001

828466882f91d35d055c87a7


Frame length (12)
Frame type
(1 = HEADERS)
Frame flags
0x1 – END_STREAM
0x4 – END_HEADERS
Stream ID (1)
9 byte frame header
00000c 01 05 00000001
N byte payload
828466882f91d35d055c87a7

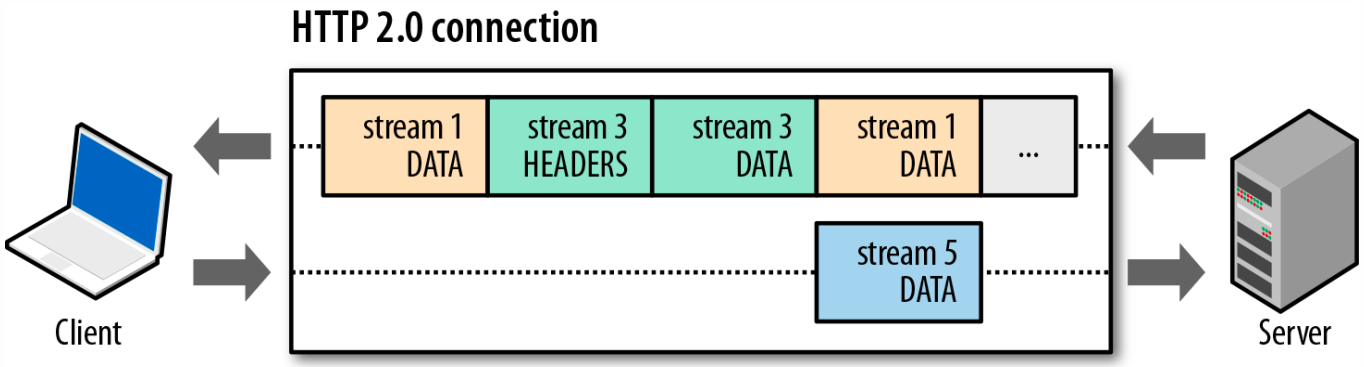


image by @igrigorik

(**demo**)

http://http2.golang.org/gophertiles

Frame types

- SETTINGS
- PING
- HEADERS, CONTINUATION
- DATA
- GOAWAY
- RST_STREAM
- WINDOW_UPDATE
- PRIORITY
- PUSH_PROMISE

… extensible

Frame: SETTINGS

- Negotiate peer limits
- Max frame size (16K default, 16M max)
- Max concurrent requests
- Must be first frame exchanged

Frame: PING

- Still there?
- Prevent idle timeouts.

Frame: HEADERS & CONTINUATION

- Clients: start a new “stream” (HTTP request)
- Servers: response headers
- Each stream has a unique, ever-incrementing “Stream ID”
  - Clients must use odd numbers
- Clients may specify priority
- “GET / HTTP/1.1” replaced by “pseudo headers”:
  “:method”=“GET”, “:path”=“/”
- If headers too large for one frame, add CONTINUATION(s)

Frame: DATA

- Request or response body
- Contains END_STREAM flag

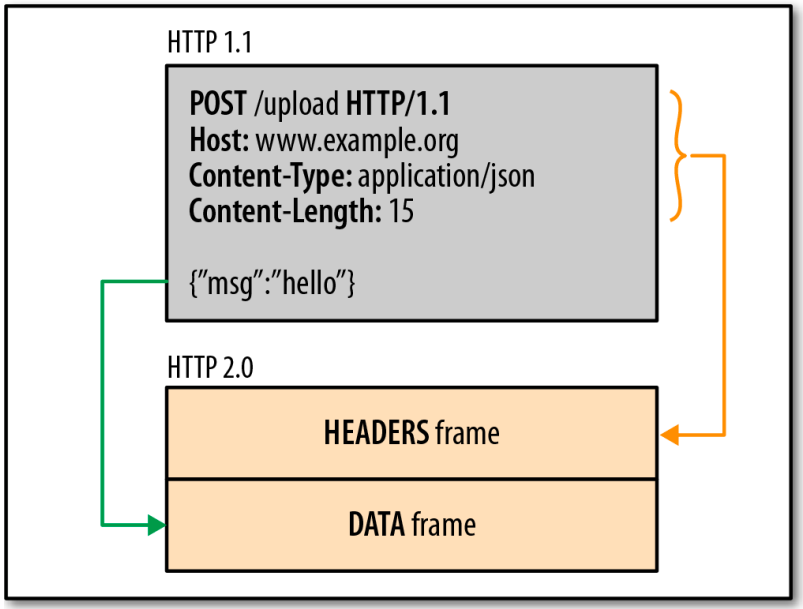


image by @igrigorik

Frame: GOAWAY

- For servers to shut down gracefully

Server: *“I’ve seen Stream ID 17.”*

Client: (Oh phew, that POST with my credit card details to purchase that plane ticket with Stream ID 19 was ignored, even if it made it to the server!)

…. starts new TCP connection, resends 19 (as 1, probably)

Frame: RST_STREAM

- abort a stream (HTTP request / response)
- without killing the whole TCP connection

e.g. canceling a download

Frame: WINDOW_UPDATE

- Flow control
- “Stream ID 43 can send 4096 more bytes”
- Can be used for prioritization:

T(0): I am willing to receive **64 KB** of kittens.jpg.
T(0): I am willing to receive **500KB** of critical.js
...
T(n): Ok, now send the **remainder** of kittens.jpg.

Client controls how and when the stream and connection window is incremented!

example by @igrigorik

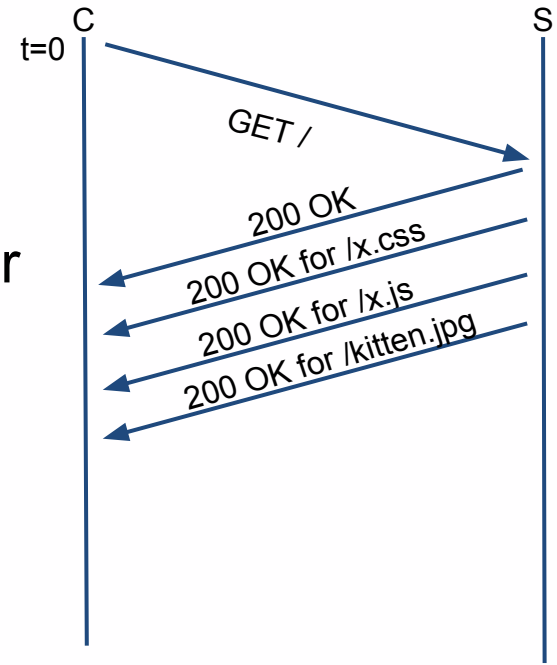
Frame: PRIORITY

- Client: “Changed my mind: kittens.jpg before JS”

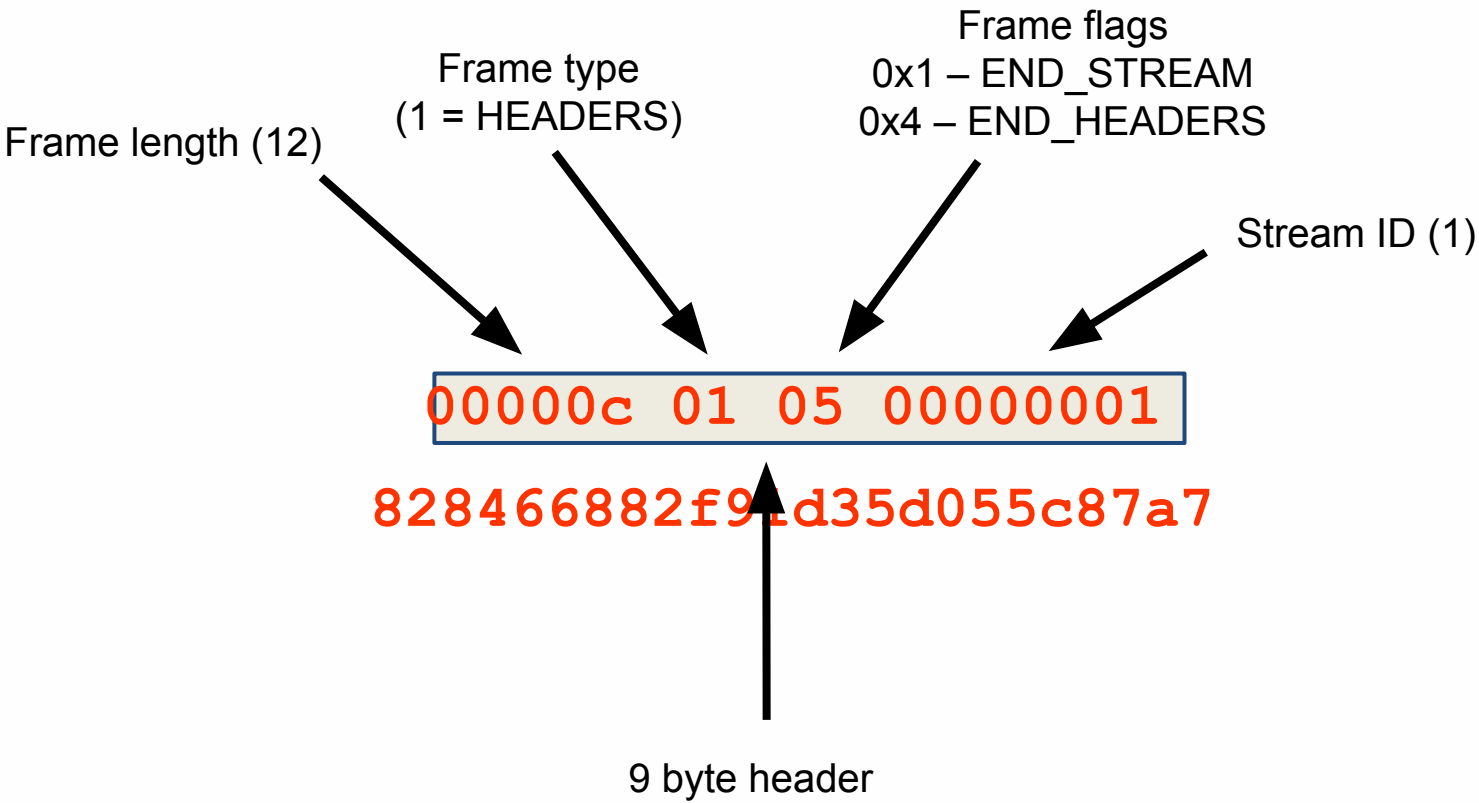
Frame: PUSH_PROMISE

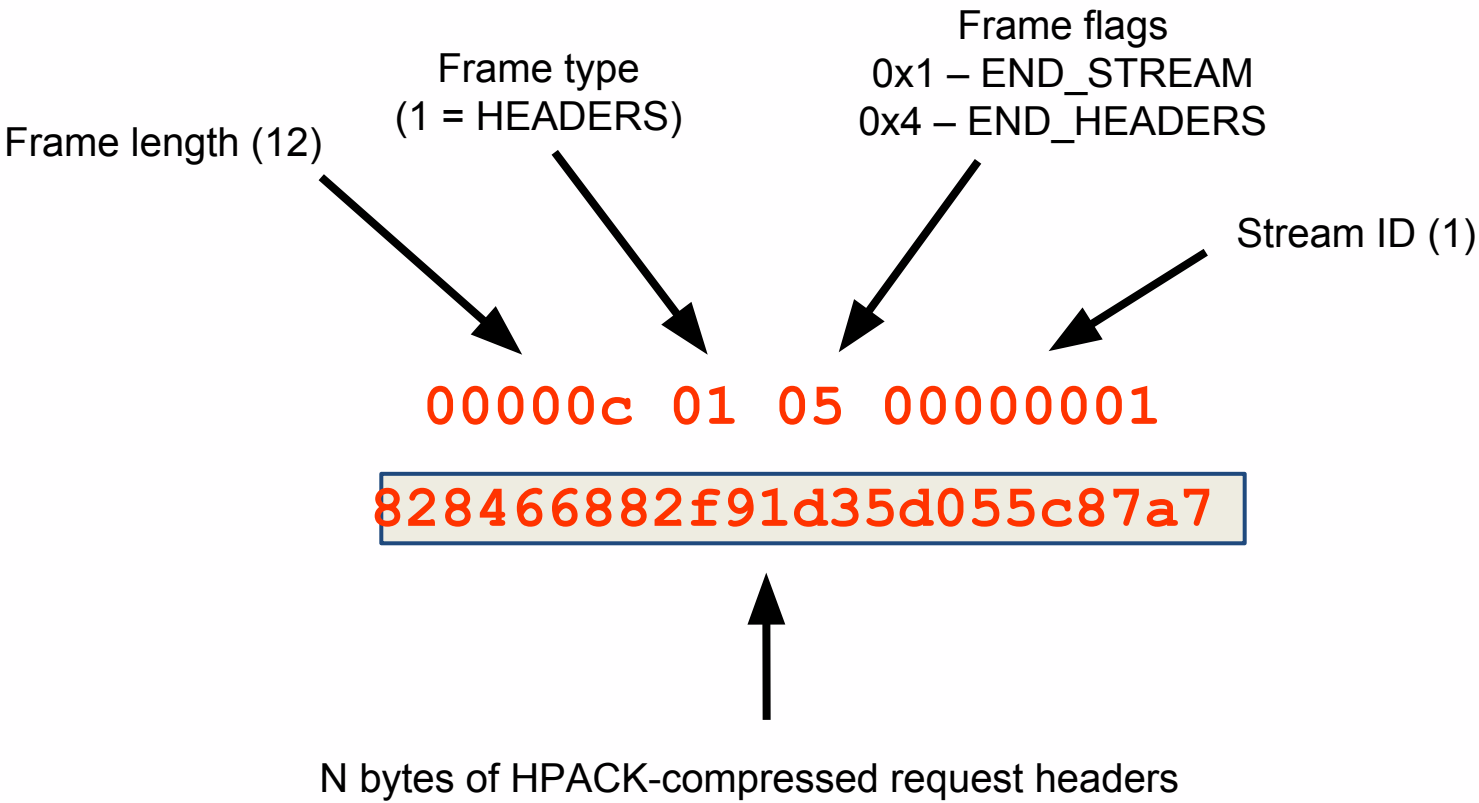
- Server gives client a response for a hypothetical request
- Server: even Stream IDs
- Push cache invalidation HEAD response

“If you had said HEAD /foo.jpg, the answer would be Last-Modified: <data>.”



Back to HEADERS ...


Frame length (12)
Frame type
(1 = HEADERS)
Frame flags
0x1 – END_STREAM
0x4 – END_HEADERS
Stream ID (1)
00000c 01 05 00000001
828466882f91d35d055c87a7
9 byte header


Frame length (12)
Frame type
(1 = HEADERS)
Frame flags
0x1 – END_STREAM
0x4 – END_HEADERS
Stream ID (1)
00000c 01 05 00000001
828466882f91d35d055c87a7
N bytes of HPACK-compressed request headers

HPACK

- Encodes list of (header, value) pairs
- New since after SPDY for HTTP/2
- Resists compression attacks to steal cookies
- Tables of common values

HPACK: Static Table

- 61 common values
- Packed integer encoding:
  small integers take fewer bytes
- Single byte 0x82 means add
  ":method" == "GET"

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	

HPACK: Dynamic Table

C.3.1 First Request

Header list to encode:

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

Hex dump of encoded data:

```
8286 8441 0f77 7777 2e65 7861 6d70 6c65 | ...A.www.example
2e63 6f6d                               | .com
```

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
0f                                      |   Literal value (len = 15)
7777 772e 6578 616d 706c 652e 636f 6d   | www.example.com
                                        | -> :authority: www.example\
                                        |   .com
```

Dynamic Table (after decoding):

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

HPACK: Dynamic Table

C.3.2 Second Request

Header list to encode:

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

Hex dump of encoded data:

```
8286 84be 5808 6e6f 2d63 6163 6865       | ....X.no-cache
```

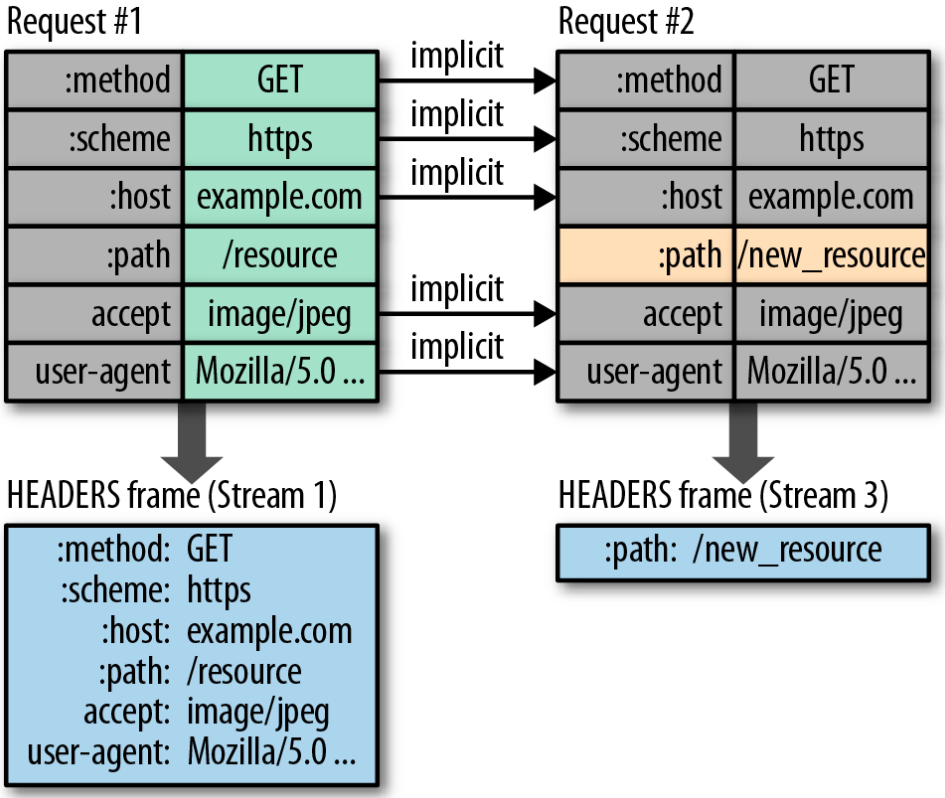
Decoding process:

```
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
08                                       |   Literal value (len = 8)
6e6f 2d63 6163 6865                      | no-cache
                                         | -> cache-control: no-cache
```

HPACK: Huffman Encoding

```
    ( 30)  |11111111|11111111|11111111|1010          fffffa  [28]
    ( 31)  |11111111|11111111|11111111|1011          fffffb  [28]
' ' ( 32)  |010100                                       14  [ 6]
'!' ( 33)  |11111110|00                                  3f8  [10]
'"' ( 34)  |11111110|01                                  3f9  [10]
'#' ( 35)  |11111111|1010                                ffa  [12]
'$' ( 36)  |11111111|11001                              1ff9  [13]
'%' ( 37)  |010101                                       15  [ 6]
'&' ( 38)  |11111000                                     f8  [ 8]
''' ( 39)  |11111111|010                                 7fa  [11]
'(' ( 40)  |11111110|10                                  3fa  [10]
')' ( 41)  |11111110|11                                  3fb  [10]
'*' ( 42)  |11111001                                     f9  [ 8]
'+' ( 43)  |11111111|011                                 7fb  [11]
',' ( 44)  |11111010                                     fa  [ 8]
'-' ( 45)  |010110                                       16  [ 6]
'.' ( 46)  |010111                                       17  [ 6]
'/' ( 47)  |011000                                       18  [ 6]
'0' ( 48)  |00000                                         0  [ 5]
'1' ( 49)  |00001                                         1  [ 5]
'2' ( 50)  |00010                                         2  [ 5]
'3' ( 51)  |011001                                       19  [ 6]
'4' ( 52)  |011010                                       1a  [ 6]
'5' ( 53)  |011011                                       1b  [ 6]
'6' ( 54)  |011100                                       1c  [ 6]
'7' ( 55)  |011101                                       1d  [ 6]
'8' ( 56)  |011110                                       1e  [ 6]
'9' ( 57)  |011111                                       1f  [ 6]
':' ( 58)  |1011100                                      5c  [ 7]
';' ( 59)  |11111011                                     fb  [ 8]
'<' ( 60)  |11111111|1111100                           7ffc  [15]
'=' ( 61)  |100000                                       20  [ 6]
'>' ( 62)  |11111111|1011                                ffb  [12]
'?' ( 63)  |11111111|00                                  3fc  [10]
'@' ( 64)  |11111111|11010                             1ffa  [13]
'A' ( 65)  |100001                                       21  [ 6]
'B' ( 66)  |1011101                                      5d  [ 7]
'C' ( 67)  |1011110                                      5e  [ 7]
'D' ( 68)  |1011111                                      5f  [ 7]
'E' ( 69)  |1100000                                      60  [ 7]
```

HTTP 2.0 provides header compression!



- Both sides maintain “header tables”
- New requests “toggle” or “insert” new values into the table

- New header set is a “diff” of the previous set of headers
- Repeat request (polling) with exact same headers incurs **no overhead**

*Byte cost of new stream: **9 bytes! ****

** as low as 9 bytes for an identical request.*

Upgrading to HTTP/2

- **https://**
  - ALPN TLS extension
  - Client: “I prefer h2, h2-14”
  - Server: “I prefer h2-14”
  - They pick “h2-14” (If no agreement: http/1.1)
  - Client: “PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n” + SETTINGS frame
  - Server: SETTING frame.
- **http://**
  - spec defines a way (~politics)
  - Arguments against:
    - doesn't work in the wild (yay proxies!)
    - it's time to encrypt
  - Google, Firefox, Go et al not implementing

HTTP/1.[01] vs HTTP/2

- text protocol
- one thing at a time*
- tons of legacy special cases and hacks
- GET, POST, cookies, etc
- tcp:80 and tcp:443

- binary protocol
- many things at a time*
- still pretty clean and consistent
- GET, POST, cookies, etc (1:1 semantics)
- tcp:443 (encrypted only in practice)

Future

TCP Fast Open (TFO)

- skip TCP three-way handshake
- RFC 7413 (Dec 2014)
- On by default in Linux 3.13
- In Chrome Linux, ChromeOS, Android
- In development for Go

After HTTP/2

HTTP/2 problems

- head-of-line blocking problems
- lose one packet ⇒ single TCP stream blocked
- TLS round trips for setup
- {TCP, TLS, HTTP/2} have own frames/boundaries
  - … all likely misaligned

Guess IP MTU size - TLS record size - 9 byte HTTP/2 frame header.

Guess wrong? 1 big IP packet, 1 small. More packets, more problems.

QUIC

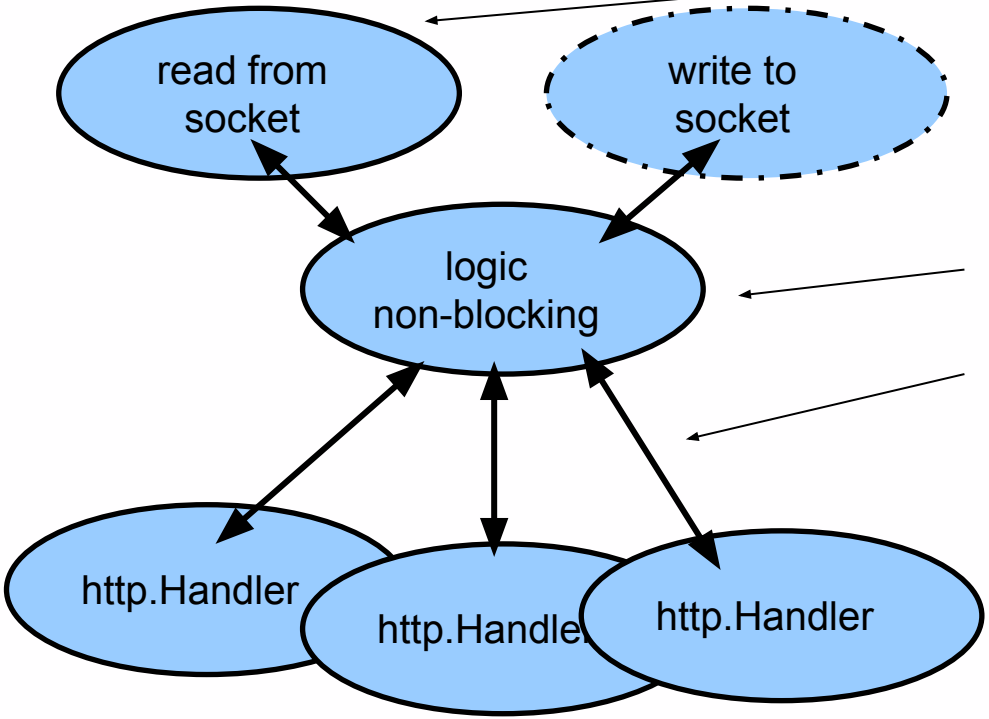
- merges HTTP/2 + TLS + TCP into one protocol
- aligns boundaries, tolerates packet loss better
- 0-RTT
- over UDP
- UDP is only non-TCP protocol that will work on net
- research for fixing TCP
- multi-path (e.g. cell network & wifi) without drops
- ongoing development in Chrome, GFE

Let's write a Go HTTP/2 server implementation!

(if we have time)

(**switch to code**)

Goroutines per HTTP/2 connection



Waiting for a new frame to arrive, stuck in a Read call forever (readFrameHeader)

All HTTP/2 connection state, state of read/write goroutines

Pushback on Handlers