Counting HTTP 0.9...3

Oleksii Kachaiev, @kachayev

Year	Protocol
1991	HTTP/0.9
1996	HTTP/1.0
1997	HTTP/1.1
Feb, 2010 - Dec, 2011	WebSocket
Jul, 2012 - Feb, 2015	SPDY
Jun, 2014	revisioned HTTP/1.1
Aug, 2014 - May, 2015	HTTP/2.0
Jun, 2017 - Jul (?), 2019	HTTP/3.0

HTTP: The Idea

```
$ telnet 127.0.0.1 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/0.9

HTTP/0.9 200 OK
Date: Sun, 08 Jul 2018 20:56:34 GMT
content-length: 18

hello, world 😌
```

HTTP/0.9 ↠ HTTP/1.0

- New methods
- Content negotiation
- Optional "Host" header
- "Connection: keep-alive"
  - TCP handshake is slow
  - Even 20+ years ago it was a problem

HTTP/1.0 ↠ HTTP/1.1

- Mandatory "Host" header
- "100 Continue"
- Chunked & byte-range transfers
- Compression
- "Upgrade" header (TLS, websockets)

HTTP/1.1: Reduce Latency

- Defaults to "Connection: keep-alive"
- Pipelining
  - Didn't work out

```
$ curl -v http://localhost:8080
* Rebuilt URL to: http://localhost:8080/
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8080 (#0)
> GET / HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Server: Aleph/0.4.7
< Connection: Keep-Alive
< Date: Sun, 08 Jul 2018 20:46:32 GMT
< content-length: 18
<
hello, world 😌
* Connection #0 to host localhost left intact
```

WebSocket

WebSocket

- Full-duplex communication
- RFC 6455 "The WebSocket Protocol"
- Handshaking using HTTP `Upgrade` header (compatibility)
- Framing (text, binary, ping/pong, close, continuation)

WebSocket Upgrade

```
$ curl -v -H "Upgrade: websocket" \
          -H "Connection: upgrade" \
          -H "Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==" \
          -H "Sec-WebSocket-Protocol: chat" \
          -H "Sec-WebSocket-Version: 13" \
          http://echo.websocket.org
* Rebuilt URL to: http://echo.websocket.org/
*   Trying 174.129.224.73...
* TCP_NODELAY set
* Connected to echo.websocket.org (174.129.224.73) port 80 (#0)
> GET / HTTP/1.1
> Host: echo.websocket.org
> Upgrade: websocket
> Connection: upgrade
> Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
> Sec-WebSocket-Protocol: chat
> Sec-WebSocket-Version: 13
```

WebSocket Upgrade

```
<
< HTTP/1.1 101 Web Socket Protocol Handshake
< Connection: Upgrade
< Date: Tue, 10 Jul 2018 09:11:33 GMT
< Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
< Server: Kaazing Gateway
< Upgrade: websocket
<
```

TLS, ALPN

ALPN

- Application-Layer Protocol Negotiation Extention, RFC 7301
- allows the application layer to negotiate which protocol should be performed
- replaced NPN (Next Protocol Negotiation Extension)
- emerged from SPDY development

ALPN

```
$ curl -v https://github.com/
*   Trying 192.30.253.112...
* TCP_NODELAY set
* Connected to github.com (192.30.253.112) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* Cipher selection: ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRENGTH
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
....
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server accepted to use http/1.1
* Server certificate:
....
*  SSL certificate verify ok.
> GET / HTTP/1.1
> Host: github.com
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
```

HTTP/2

HTTP/2: WHY?

- TCP handshake is still slow
- Head-of-line blocking
- Server to initiate the communication
- TCP congestion control with tons of connections

HTTP/2: HOW?

- Binary, compression of HTTP headers (HPACK)
- Pipelining of requests
- Multiplexing over a single TCP connection
- HTTP/2 Server Push
- Settings management flow, priorities
- ALPN, "Upgrade"

HTTP/3

HTTP/3 =

HTTP/2 over QUIC*

W U no TCP?

- Head of line blocking (still 😑)
  - Streams are not independent
- 3 different handshakes: TCP, TLS, HTTP
  - Better with TLS1.3, but still 😑
- TCP relies on IP address
- TCP is "hardcoded" into infrastructure (links, routers, kernel)

QUIC: HOW?

- UDP-based transport
- Handshakes: 0-RTT, 1-RTT, early data
- TLS1.3 encryption
- `ConnectionID` instead of IP
- Streams

QUIC: UDP???

- User-space reliability:
  - re-transmissions, congesion, pacing
  - faster improvements/experiments
  - error prone?
- Supported everywhere (heh DNS)
- Kernels are mot optimized

QUIC: Streams

- Bi- or uni- directional
- Initiated by any party
- Independent not only logically!

HTTP/3: HOW?

- Application level protocols
- H3
- QPACK instead of HPACK
- Upgrade: Alt-Svc

Thanks!

q&a please