HTTP2

Towards a faster & scalable web

Vinayak Hegde

VP Engineering, Zoomcar

A Brief history of HTTP

- **Early 1990s** -- Tim Berners-Lee starts implementation of simple stateless protocol in CERN
- **May 1996** -- RFC 1945 – HTTP/1.0 specification
- **Jan 1997** -- RFC 2068 – HTTP/1.1 first specified
- **Jun 1999** -- RFC 2616 – HTTP/1.1 long lived draft (improved performance and tightens specification)
- **2000s craze** -- The web explodes and the (and subsequent bust)
- **June 2014** -- Comprehensive overhaul based on real-world usage and issues - RFC 723[0-5]
- **May 2015** -- Starts with Google SPDY and ends with HTTP/2

Background IETF Info

- Worked on by HTTPBis working group
- Multiple contributors
- HTTP2 does not obsolete HTTP/1.1 (co-exists with it)
- **The HTTP version *only* indicates wire compatibility, not feature sets or "marketing."**
- Can be extended by other protocols like WEBDAV, HTTPS
- Other protocols can use Mappings (CoAP/QUIC?)
- Goto https://datatracker.ietf.org/wg/httpbis/documents/
- Defined in RFC 7540

HTTP/1.1

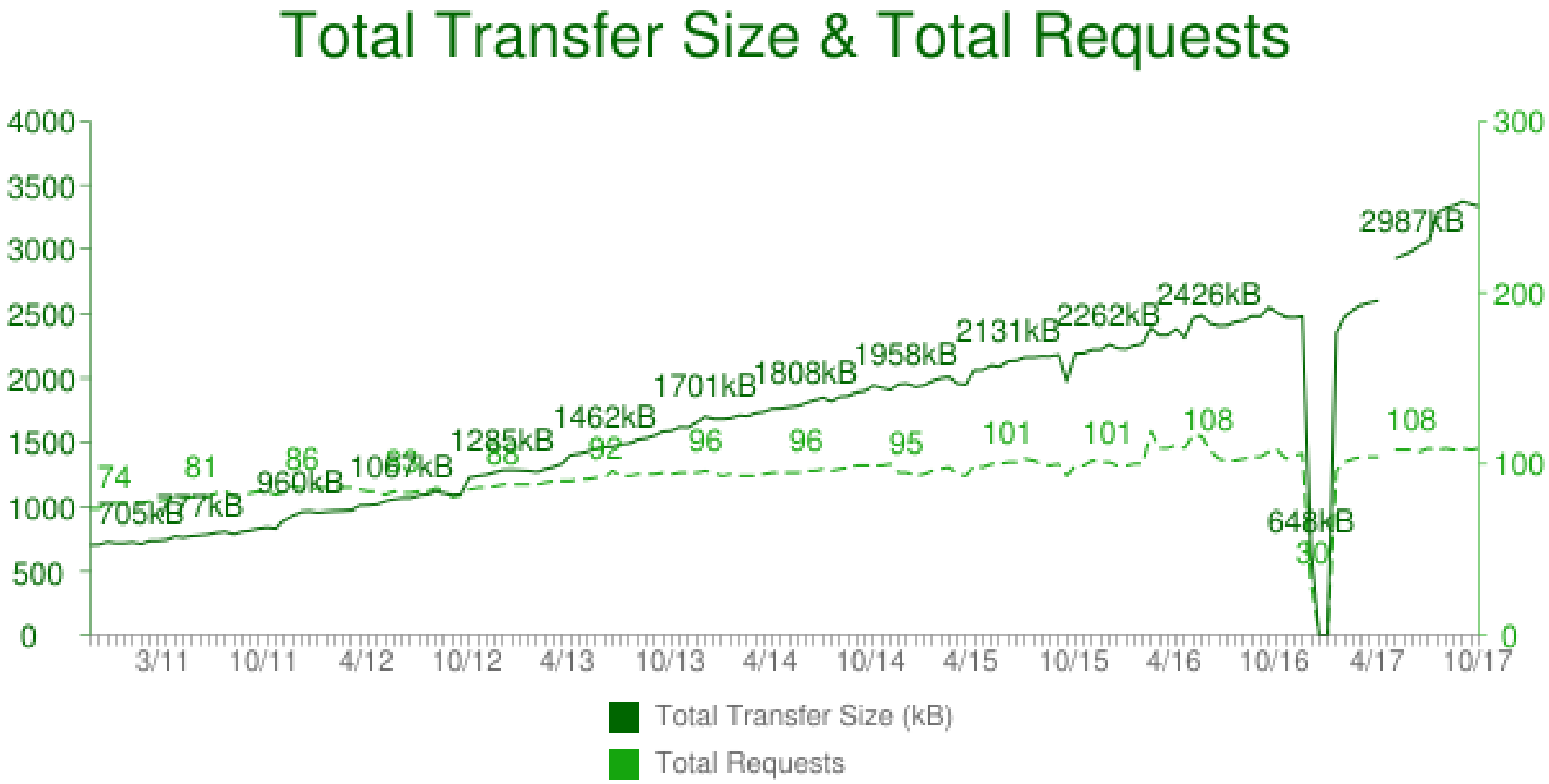
HTTP/1.1 is a group of RFCs (obsoletes RFC 2616)

- RFC 7230 - HTTP/1.1: Message Syntax and Routing
- RFC 7231 - HTTP/1.1: Semantics and Content
- RFC 7232 - HTTP/1.1: Conditional Requests
- RFC 7233 - HTTP/1.1: Range Requests
- RFC 7234 - HTTP/1.1: Caching
- RFC7235 - HTTP/1.1: Authentication

The World changed

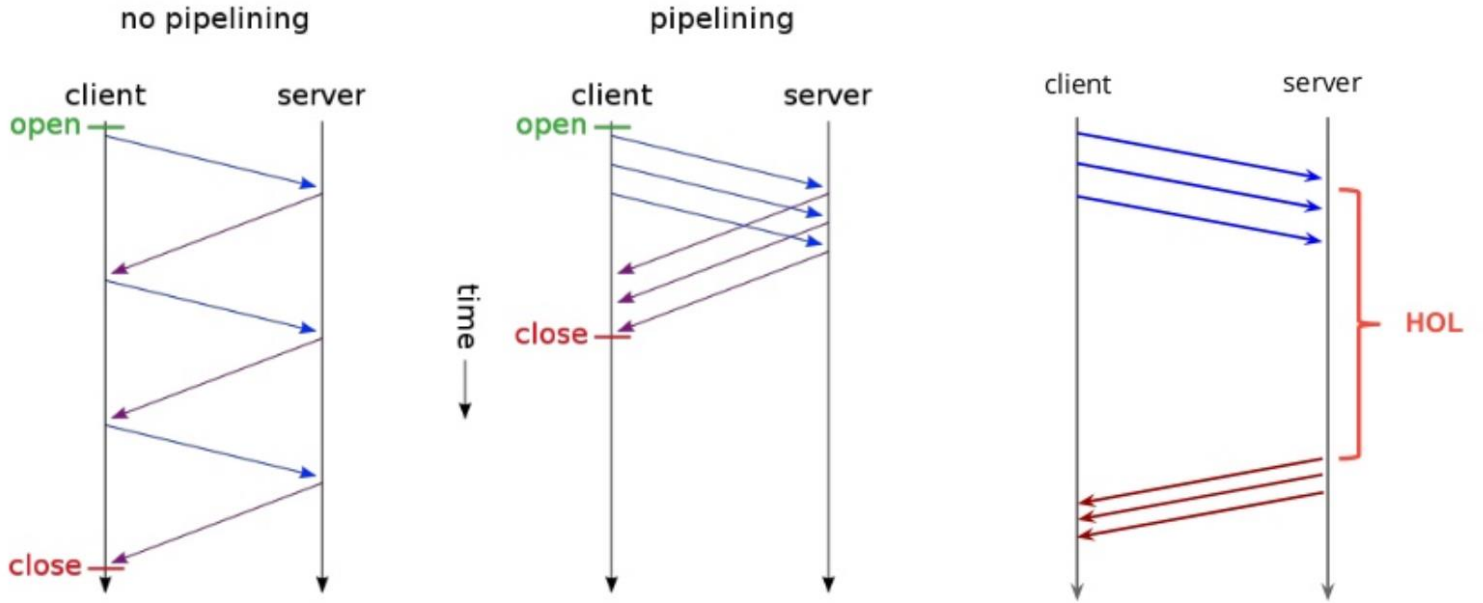
- HTTP/1.1 is great but the world changed
  - The pipes became bigger (more bandwidth)
  - Better connectivity (lower latency)
  - CDN coverage increased (lower latency)
- Webpages became
  - More complex with more resources
  - Became heavier (in several MBs sometimes)

Web page size (Nov 2010 - Nov 2017)



HTTP/1.1 pipelining detour

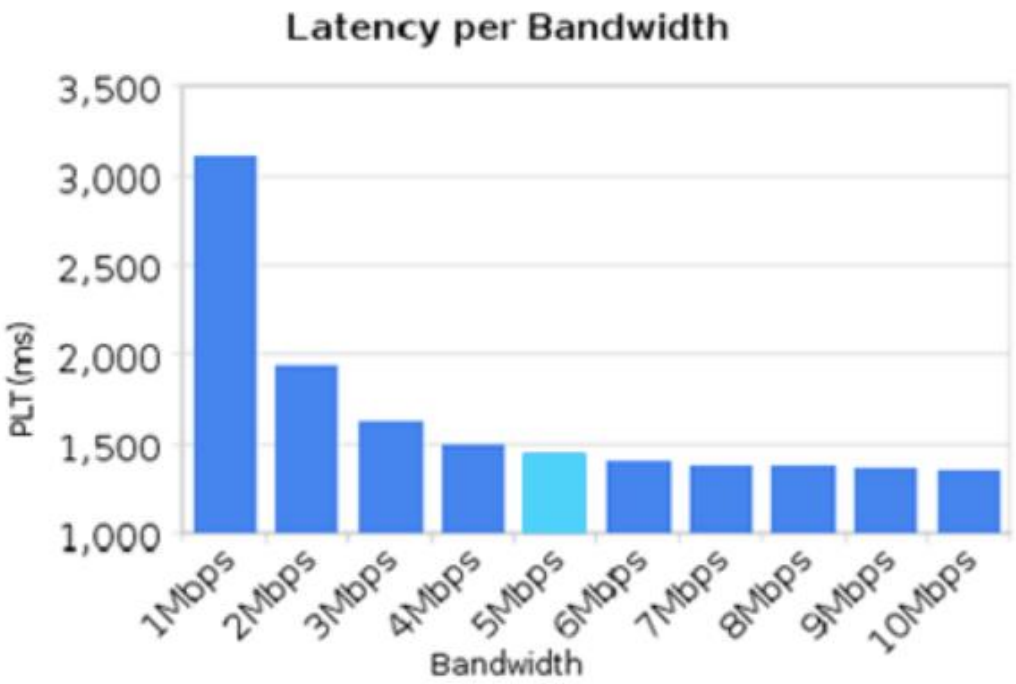
HTTP does not support multiplexing!

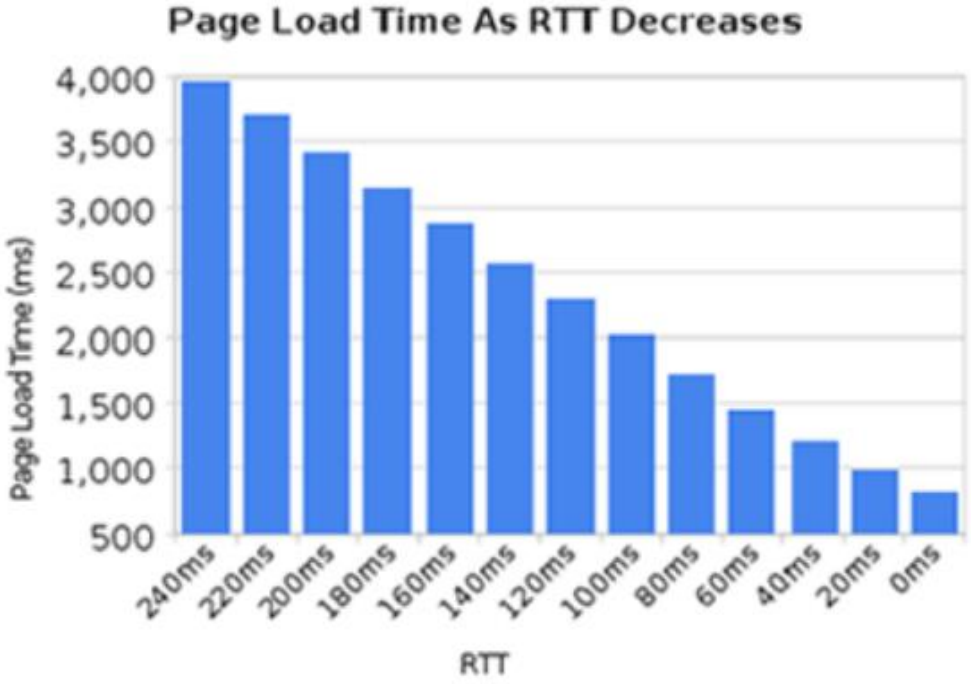


- **No pipelining:** request queuing
- **Pipelining*:** response queuing

- **Head of Line blocking**
  - It's a guessing game...
  - Should I wait, or should I pipeline?

Bandwidth Vs Latency





So how do we make HTTP better ?

- What if we could redesign http for the modern age (fast networks, low latency)
  - It would be less sensitive to network delay
  - fixed pipelining and HOL blocking
  - performed well regardless of number of tcp connections used
  - had long lived connections to take better advantage of TCP's congestion control to kick in
  - used the same semantics as http/1.1

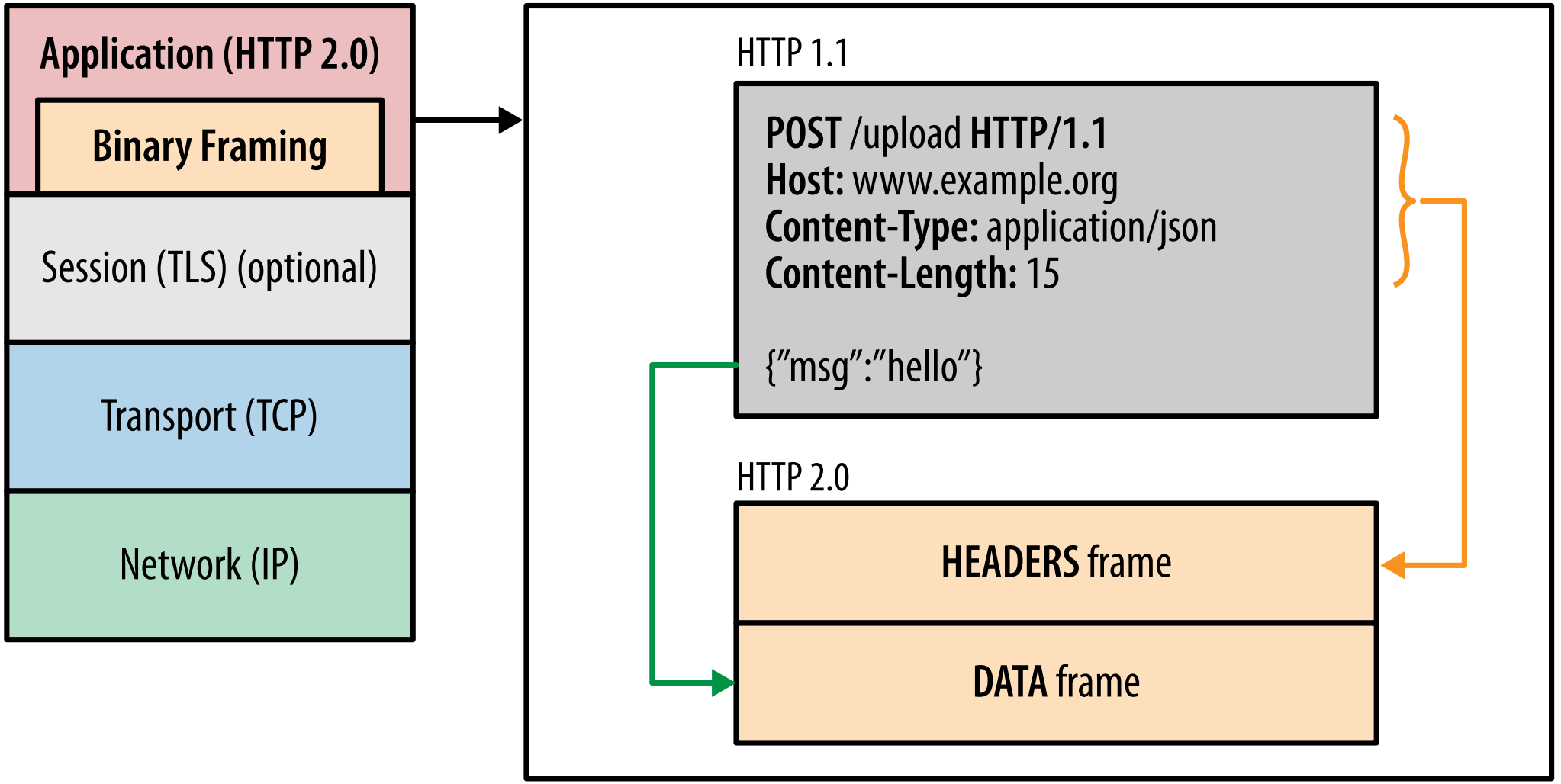
A Brief History of HTTP/2

- **November 2009**: Mike Belshe and Roberto Peon announce SPDY
- **March 2011**: Mike talks about SPDY to the HTTPbis WG atIETF80
- **~April 2011**: Chrome, Google start using SPDY
- **March 2012**: HTTPbis solicits proposals for new protocol work
- **March 2012**: Firefox 11 ships with SPDY (off by default)
- **May 2012**: Netcraft finds 339 servers that support SPDY
- **June 2012**: Nginx announces SPDY implementation
- **July 2012**: Akamai announces SPDY implementation
- **July 2012**: HTTPbis re-chartered to work on HTTP/2.0, based on SPDY
- **May 2015**: RFC 7540 specifies HTTP2

HTTP/2 features

- **Multiplexing and concurrency**: Several requests can be sent in rapid succession on the same TCP connection, and responses can be received out of order - eliminating the need for multiple connections between the client and the server
- **Stream dependencies**: the client can indicate to the server which of the resources are more important than the others
- **Header compression**: HTTP header size is drastically reduced
- **Server push**: The server can send resources the client has not yet requested
- **“Upgrade:”** header or TLS ALPN negotiation

HTTP2 Concepts – Binary Framing Layer



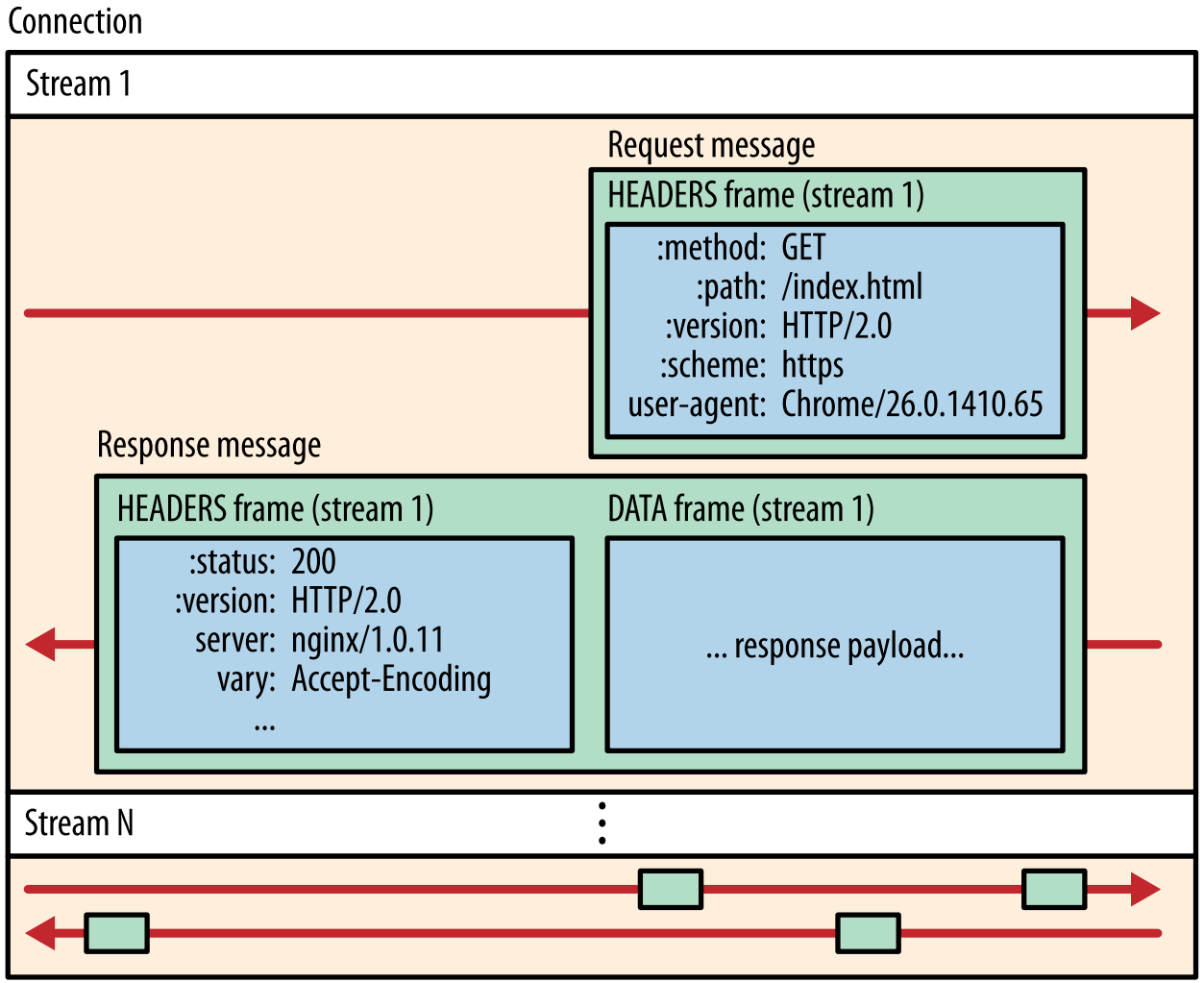
HTTP2 Concepts

- **Stream**
  - A bidirectional flow of bytes within an established connection, which may carry one or more messages.
- **Messages**
  - A complete sequence of frames that map to a logical request or response message.
- **Frames**
  - The smallest unit of communication in HTTP/2, each containing a frame header, which at a minimum identifies the stream to which the frame belongs.

HTTP2 Concepts

- All *communication* is performed over a single TCP connection that can carry any number of bidirectional streams.
- Each *stream* has a unique identifier and optional priority information that is used to carry bidirectional messages.
- Each *message* is a logical HTTP message, such as a request, or response, which consists of one or more frames.
- The *frame* is the smallest unit of communication that carries a specific type of data—e.g., HTTP headers, message payload, and so on. Frames from different streams may be interleaved and then reassembled via the embedded stream identifier in the header of each frame.

How does this look on the wire ?



Types of Streams

- DATA - Convey Arbitrary data associated with stream
- HEADERS - Used to open a stream and carries name:value pairs
- PRIORITY - Specifies sender-advised priority of streams
- RST_STREAM - Allows abnormal termination of stream
- SETTINGS - Conveys configuration parameters that affect how end-points communicate

Types of Streams

- PUSH-PROMISE - Used to notify the peer endpoint in advance of streams the sender intends to initiate
- PING - Measuring a minimal round-trip time from the sender; checks if a connection is still alive
- GOAWAY - Informs the remote peer to stop creating streams on this connection
- WINDOW_UPDATE - Used to implement flow control on each individual stream or on the entire connection.
- CONTINUATION - Used to continue a sequence of headerblock fragments

A note about Header Compression

- Defined in RFC 7451
- HPACK - Header compression algorithm
- HPACK has been invented because of attacks like CRIME and BREACH attacks

- HPACK Stats
  - Cloudflare saw an 76% compression for ingress headers and 53% drop in ingress traffic due to HPACK (Request traffic)
  - Cloudflare saw a 69% compression for egress headers and 1.4% drop in egress traffic (Response traffic)

How HPACK Works - Dictionaries

- 3 types of compression
  - **Static Dictionary**: A predefined dictionary of 61 commonly used header fields, some with predefined values.
  - **Dynamic Dictionary**: A list of actual headers that were encountered during the connection. This dictionary has limited size, and when new entries are added, old entries might be evicted.
  - **Huffman Encoding**: A static Huffman code can be used to encode any string: name or value. This code was computed specifically for HTTP Response/Request headers – ASCII digits and lowercase letters are given shorter encodings. The shortest encoding possible is 5 bits long, therefore the highest compression ratio achievable is 8:5 (or 37.5%smaller)

How HPACK works - Algorithm

- When HPACK needs to encode a header in the format name:value, it will first look in the static and dynamic dictionaries.
- If the full name:value is present, it will simply reference the entry in the dictionary. This will usually take one byte, and in most cases two bytes will suffice. A whole header encoded in a single byte.
- Since many headers are repetitive, this strategy has a very high success rate. (like long cookie headers)

How HPACK works - Algorithm

- When HPACK can't match a whole header in a dictionary, it will attempt to find a header with the same name.
  - Most of the popular header names are present in the static table, for example: content-encoding, cookie, etag.
  - The rest are likely to be repetitive and therefore present in the dynamic table.
  - Cloudflare assigns a unique cf-ray header to each response, and while the value of this field is always different, the name can be reused.

A note about ALPN

- A TLS extension that permits the application layer to negotiate protocol selection within the TLS handshake.
- Defined in RFC 7301
- With ALPN, the client sends the list of supported application protocols as part of the TLS ClientHello message. The server chooses a protocol and sends the selected protocol as part of the TLS ServerHello message. The application protocol negotiation can thus be accomplished within the TLS handshake, without adding network round-trips, and allows the server to associate a different certificate with each application protocol, if desired.

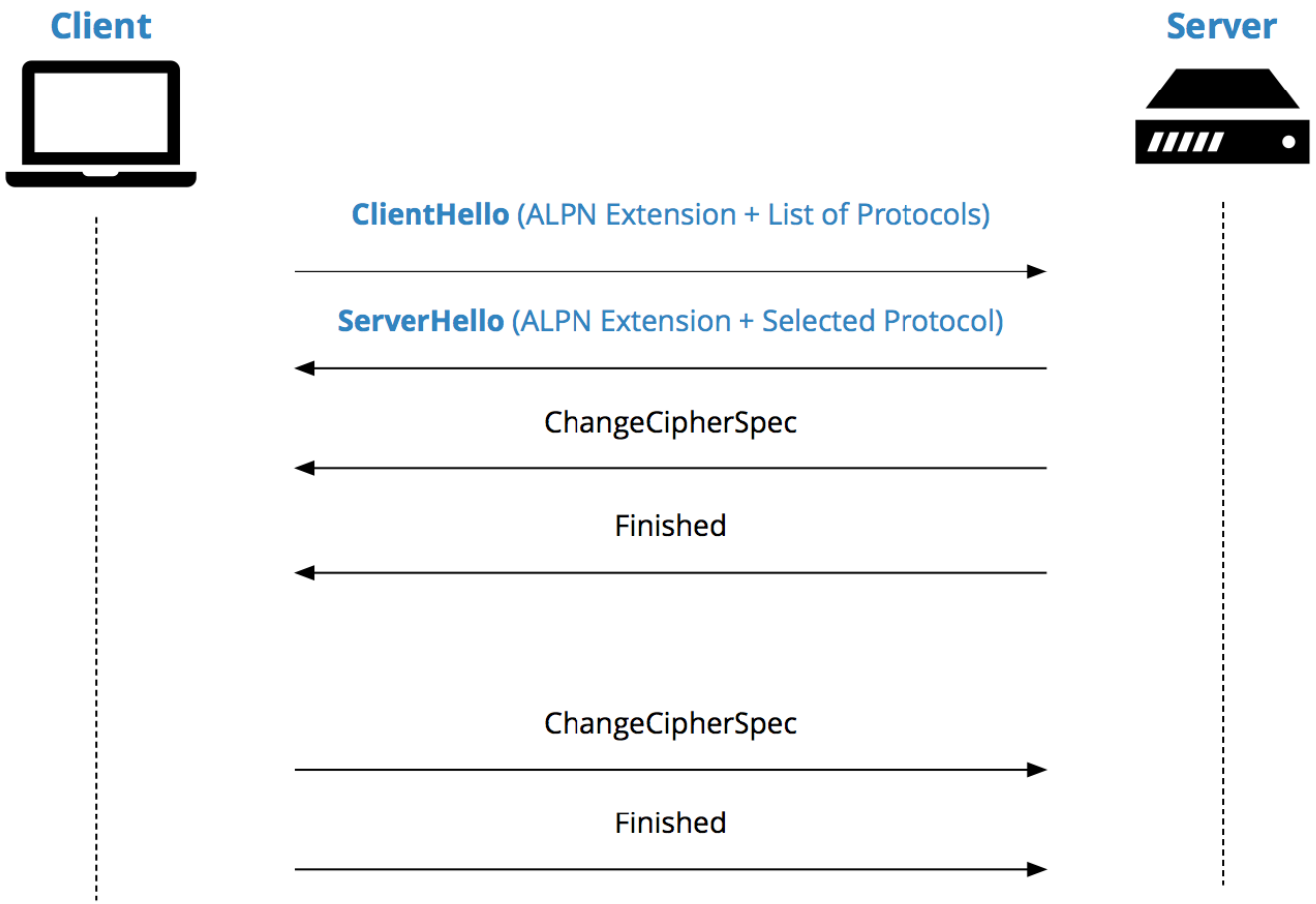
Why ALPN ?

- When multiple application protocols are supported on a single server-side port number, such as port 443, the client and the server need to negotiate an application protocol for use with each connection. It is desirable to accomplish this negotiation without adding network roundtrips between the client and the server, as each round-trip will degrade an end-user’s experience. Further, it would be advantageous to allow certificate selection based on thenegotiated application protocol. (RFC 7301)

How TLS works (RFC 5246)

- The Client and Server exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.
- An exchange of the necessary cryptographic parameters allow the client and server to agree on a premaster secret.
- A master secret is generated from the premaster secret and exchanged random values.
- Security parameters are provided to the record layer.
- The Client and server verify that their peer has calculated the same security parameters and that the handshake took place without tampering by an attacker.

How ALPN works (RFC 7301)



HTTP2 Tradeoffs and Drawbacks

- Security is hard to get right
- TCP is awkward
  - In-order delivery = head-of-line blocking
  - Initial congestion window is small
  - Packet loss isn’t handled well
- Binary protocol so debugging is hard(er)

HTTP2 Support

- Now supported by all major browsers – Firefox, Chrome, Safari, Opera and MS Edge
- Also supported by major HTTP servers - Nginx, Apache, Lightspeed, Microsoft IIS and HAProxy
- Supported by Major CDNs – Akamai, Cloudflare, Fastly and AWS Cloudfront

Further Work – IETF 100 Singapore

- HTTP Representation Variants
- Cache Digest for HTTP/2
- Secondary Certificates
- Bootstrapping Websockets with HTTP2
- 451 Protocol Elements
- Header Common Structure

Resources

- HTTP2 FAQ - https://http2.github.io/faq/
- HTTP2 by Ilya Grigorik - https://www.slideshare.net/heavybit/heavybit-presents-ilya-grigorik-on
- HTTP Archive - http://httparchive.org/
- High Performance Browser networking - https://hpbn.co/http2/
- HPACK on Cloudflare - https://blog.cloudflare.com/hpack-the-silent-killer-feature-of-http-2/

Contacts

- Twitter @vinayakh
- vinayakh@gmail.com