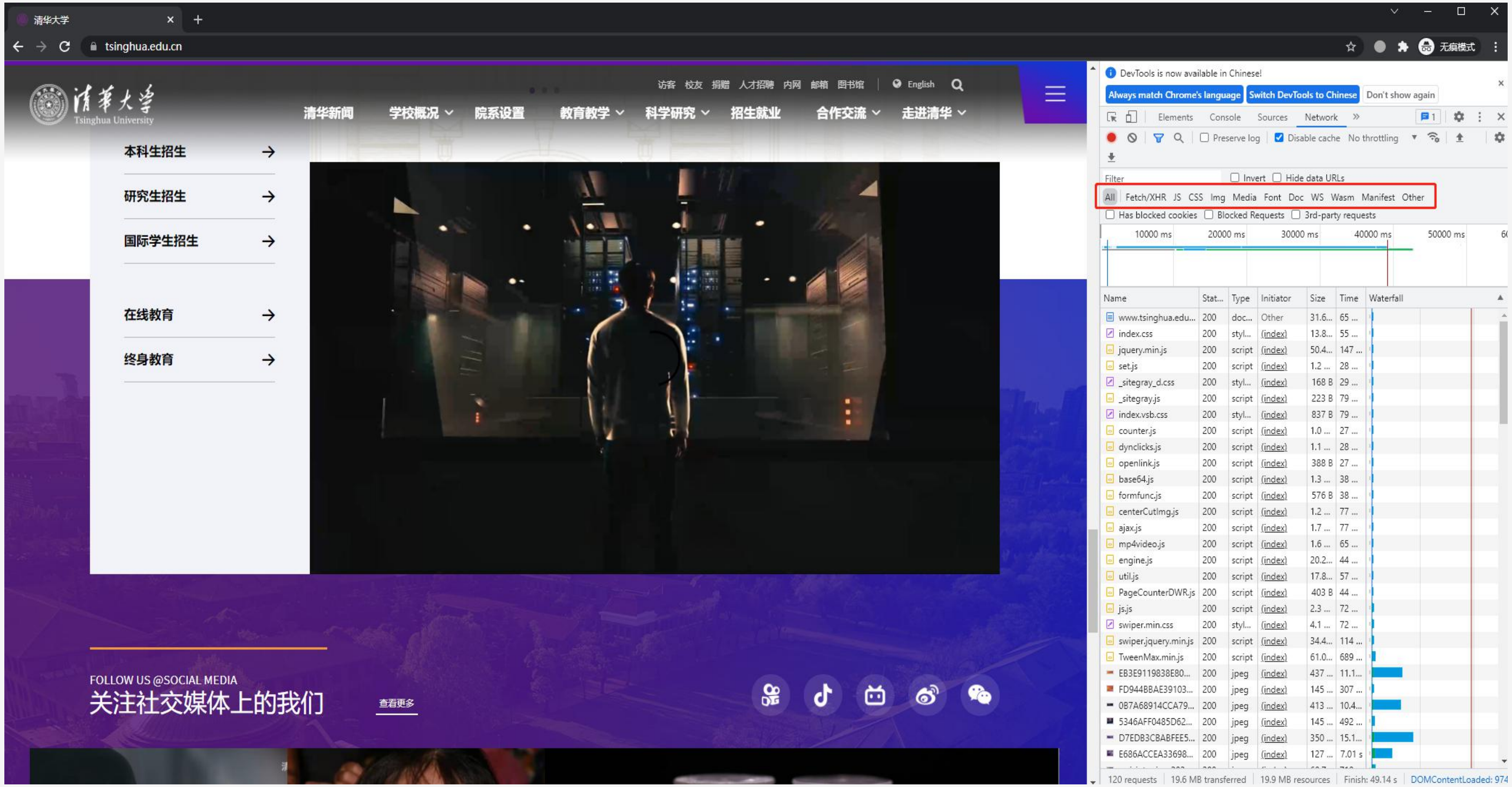
QUIC PROTOCOL

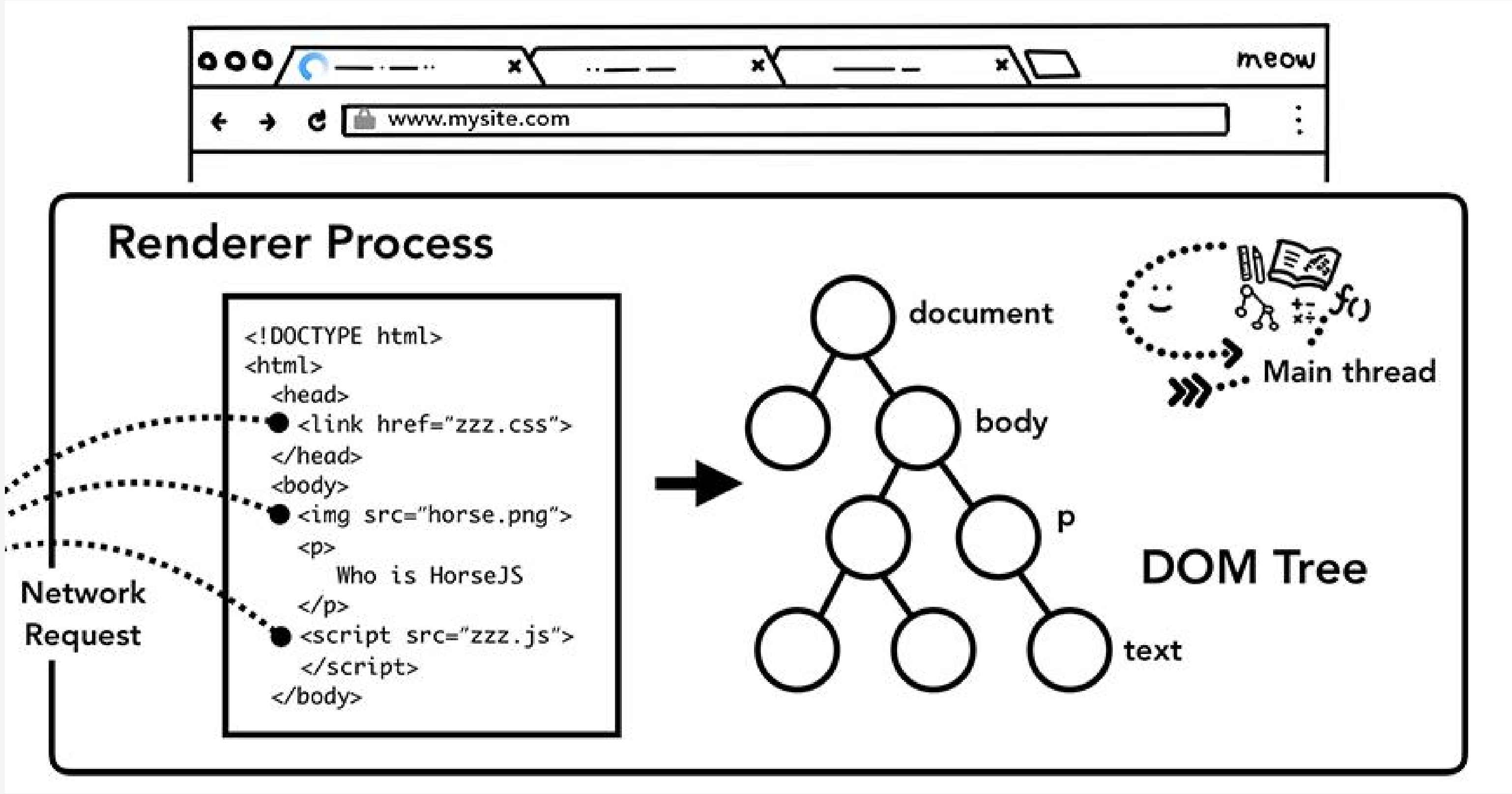
HTTP/3

罗龙君(luolongjuna@gmail.com)

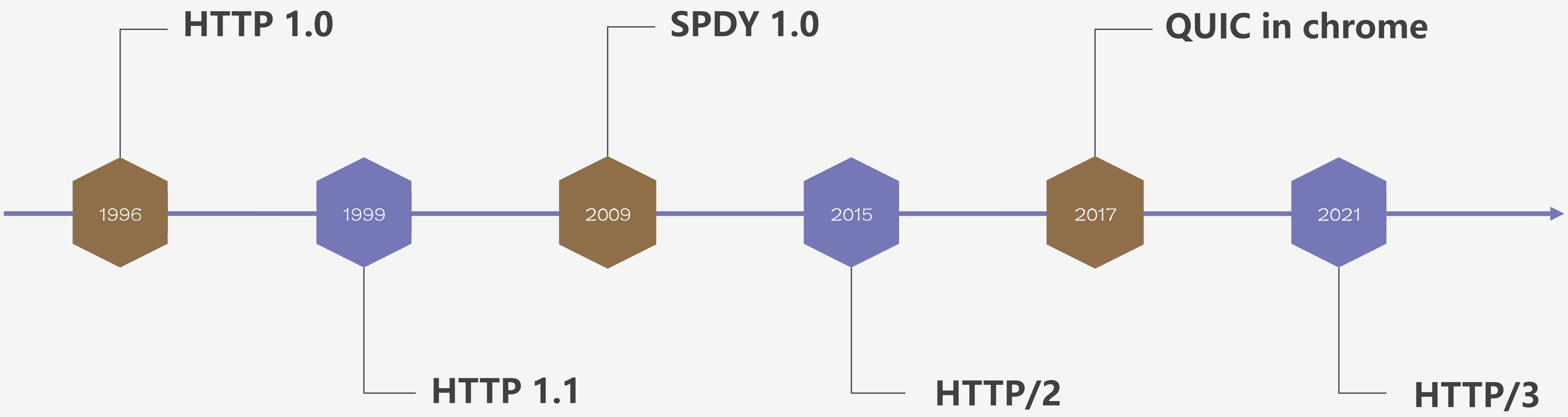
Contents of a website



Contents of a website



Timeline of HTTP protocol



HTTP 1.0 VS HTTP 1.1

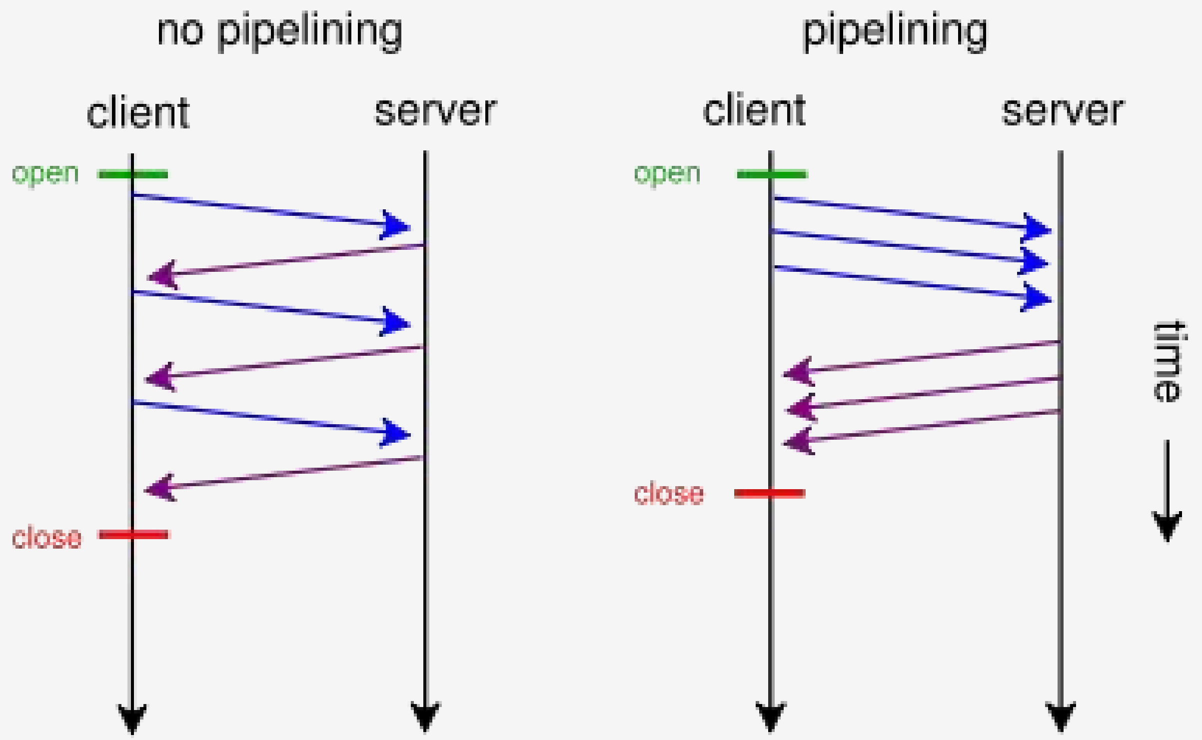
HTTP 1.0

- Single request/reponse per connection
- Host header optional
- Limited support for caching
- Simple status code

HTTP 1.1

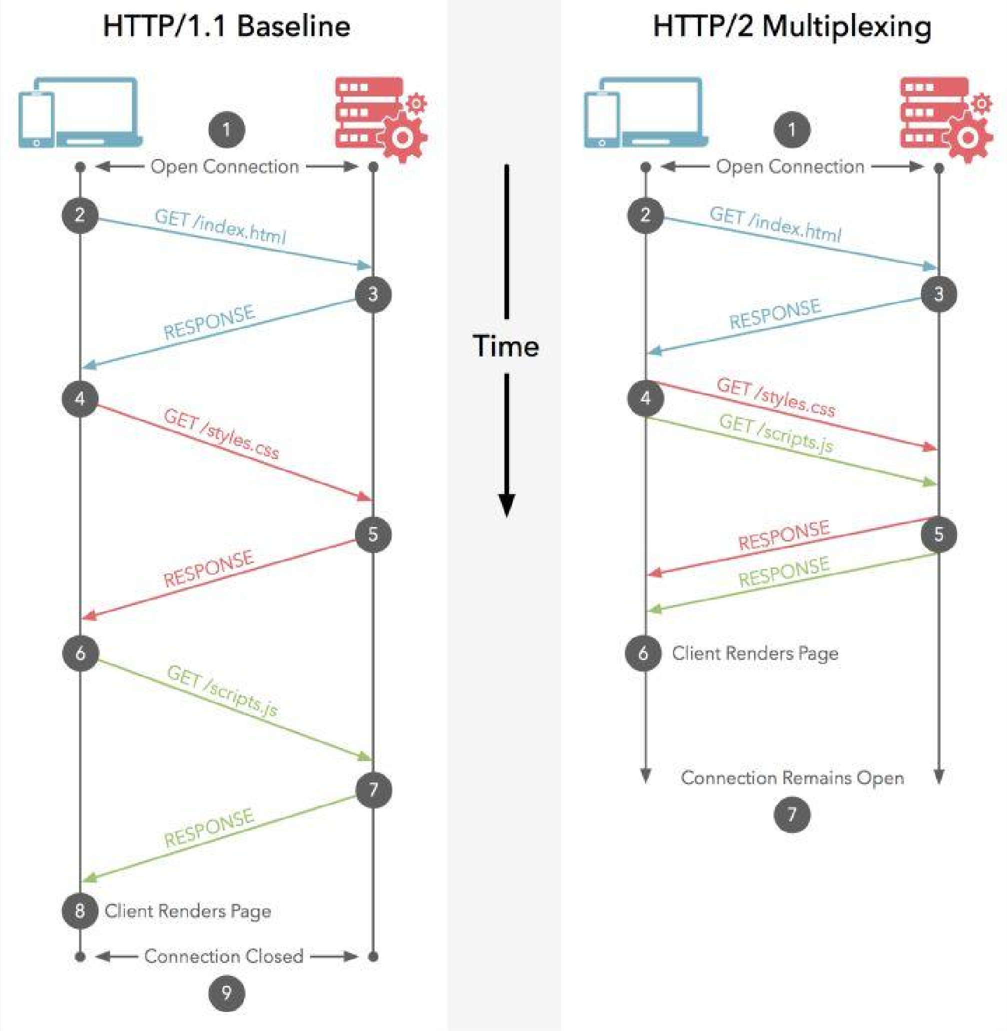
- Persistent and pipelined connection
- Need host filed
- Caching support
- HTTP status code
- Chunked transfers / byte-range transfers
- Compression/decompression
- More ……

HTTP 1.1 pipeline



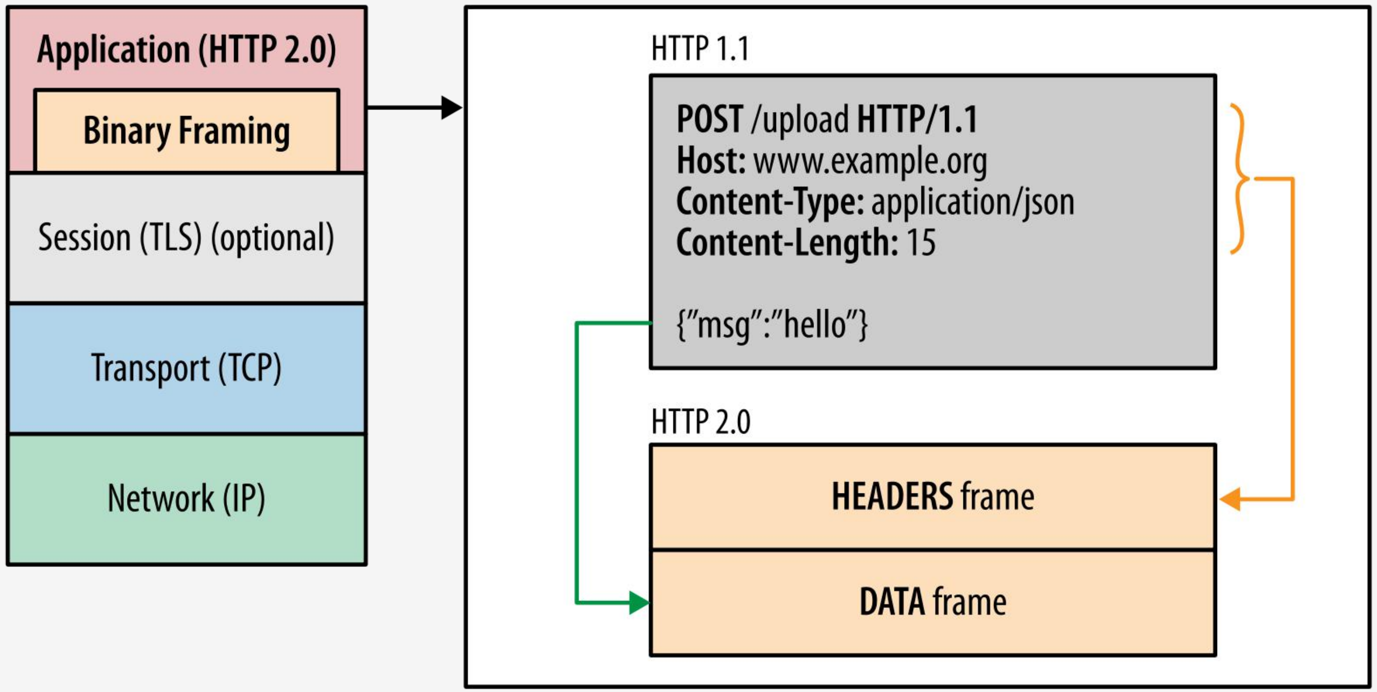
- Solve HOL problem partly (FIFO)
- Hard to use (Idempotent request)
- Not be used in default

SPDY and HTTP/2

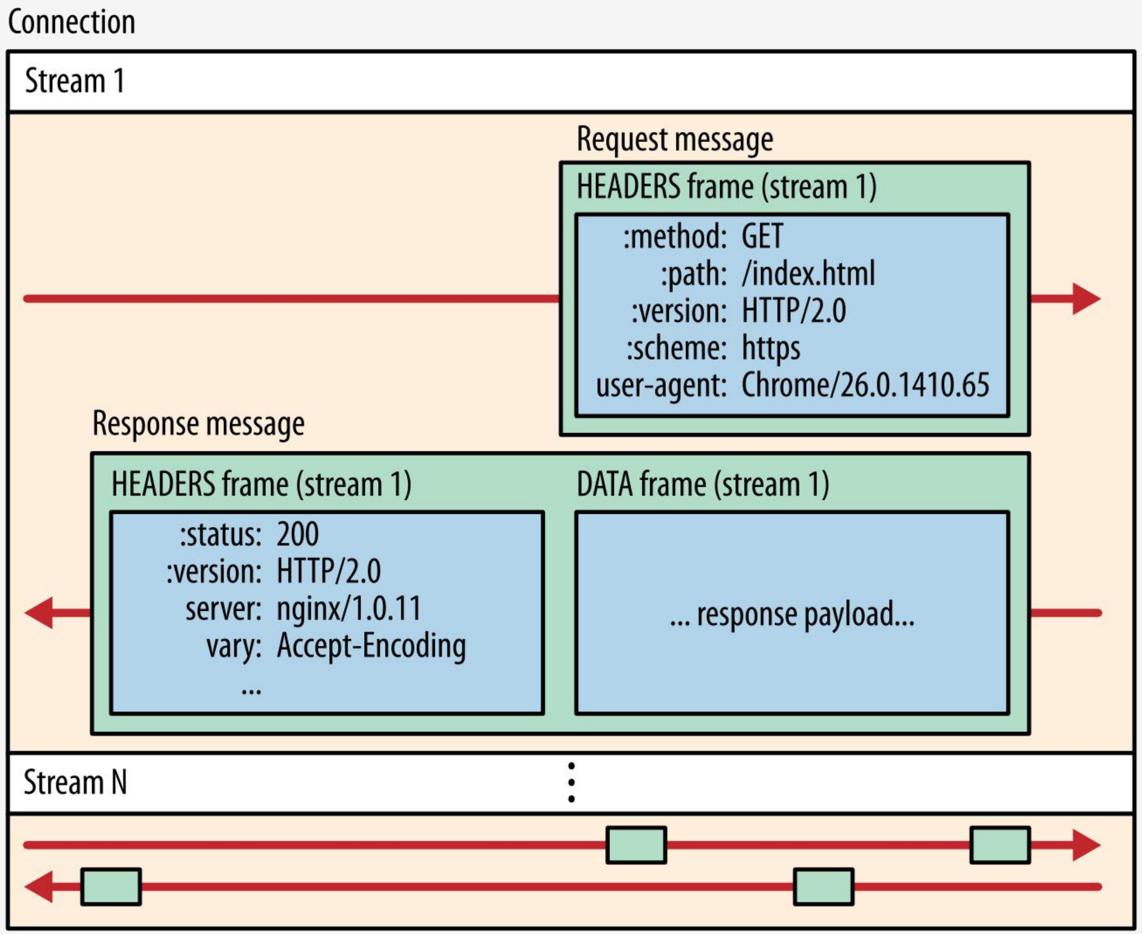


- Multiplexing
- Introduce priority
- flow control
- Server push
- Hpack

A frames protocol



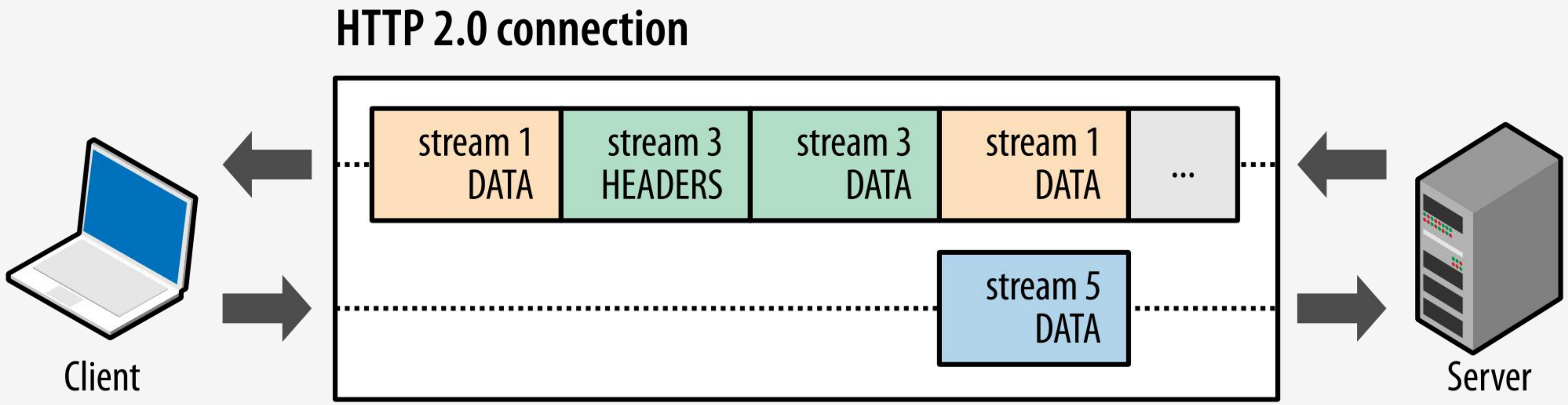
A frames protocol



- Stream
  - 已建立的连接内的双向字节流，可以承载一条或多条消息。
- Message
  - 每个数据流都有一个唯一的标识符和可选的优先级信息，用于承载双向消息。
- Frame
  - 帧是最小的通信单位，承载着特定类型的数据，例如 HTTP 标头、消息负载等等。 来自不同数据流的帧可以交错发送，然后再根据每个帧头的数据流标识符重新组装

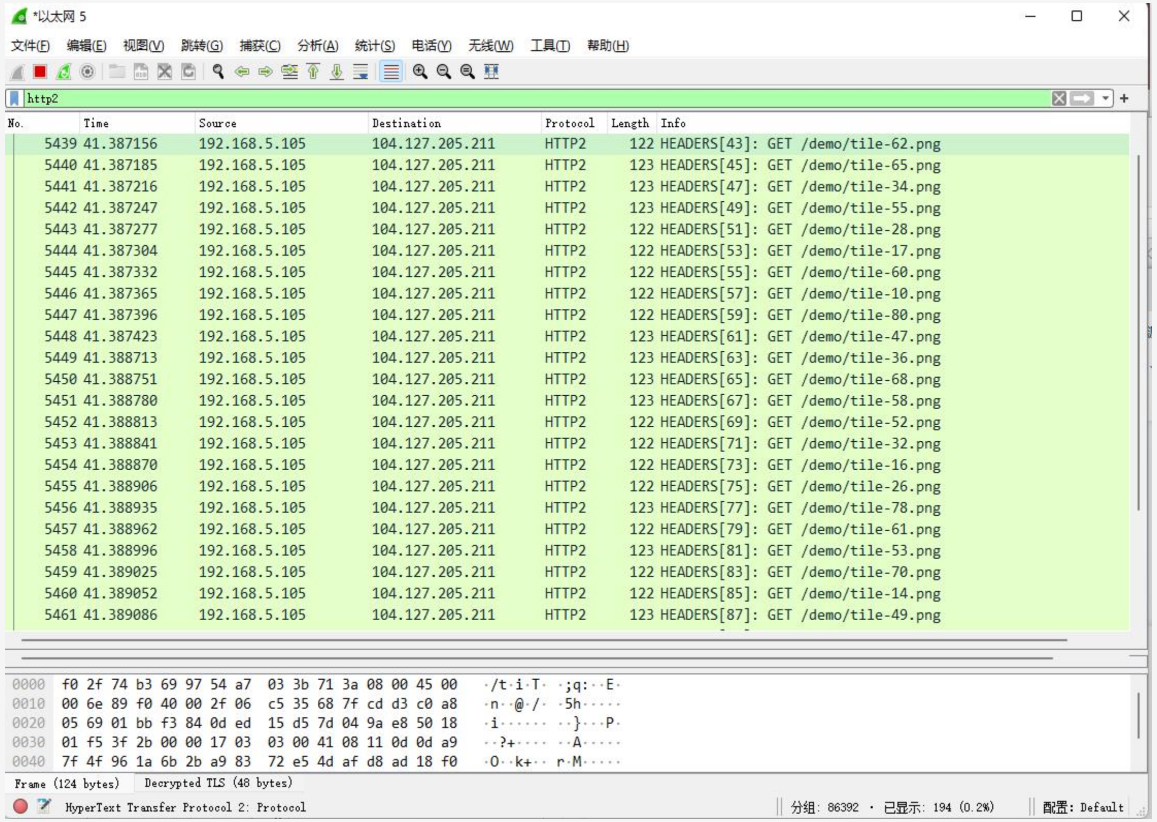
Multiplexing

- 并行交错地发送多个请求，请求之间互不影响。
- 并行交错地发送多个响应，响应之间互不干扰。
- 使用一个连接并行发送多个请求和响应。
- 消除不必要的延迟和提高现有网络容量的利用率，从而减少页面加载时间。



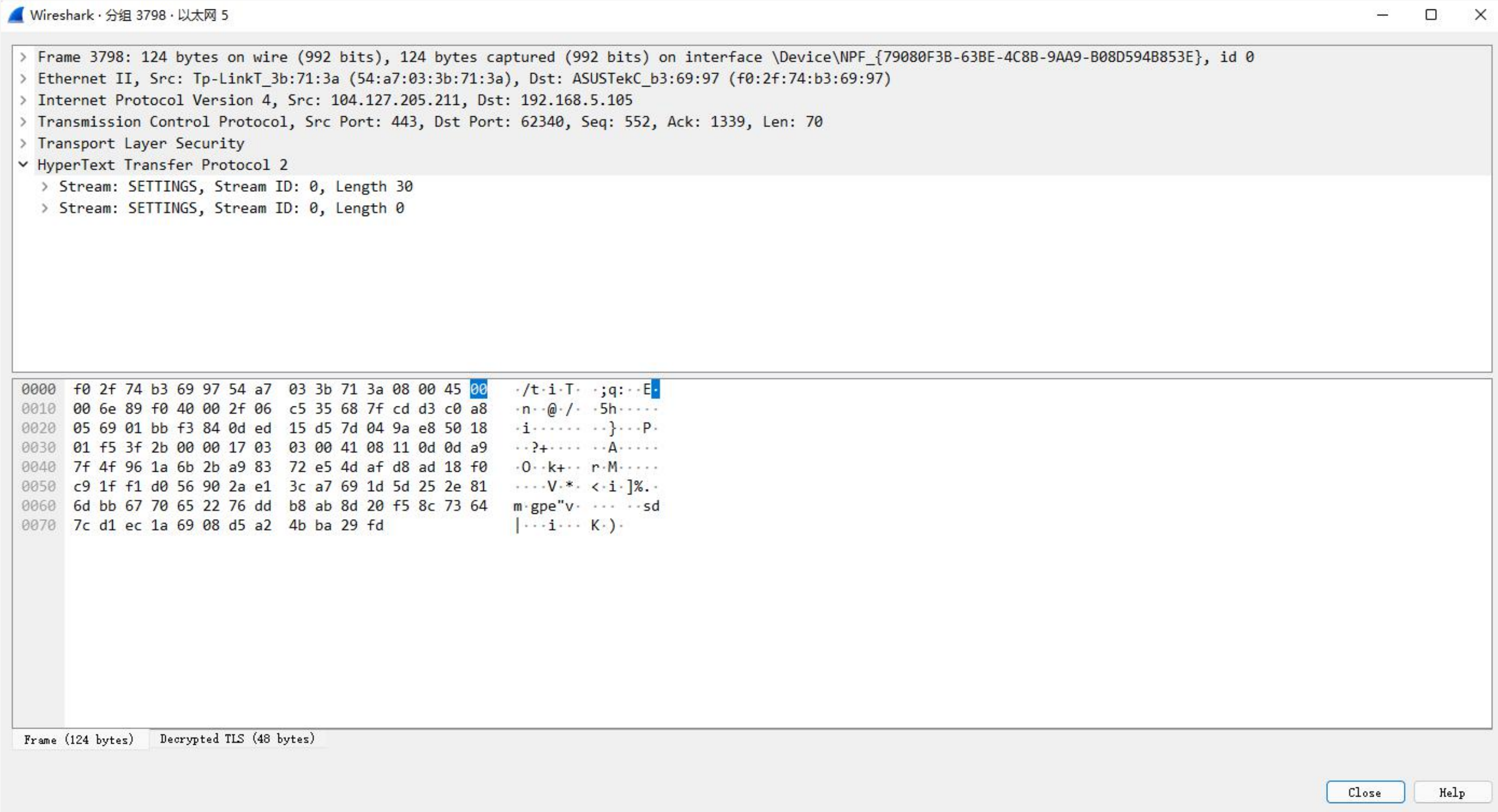
Multiplexing

https://http2.akamai.com/demo

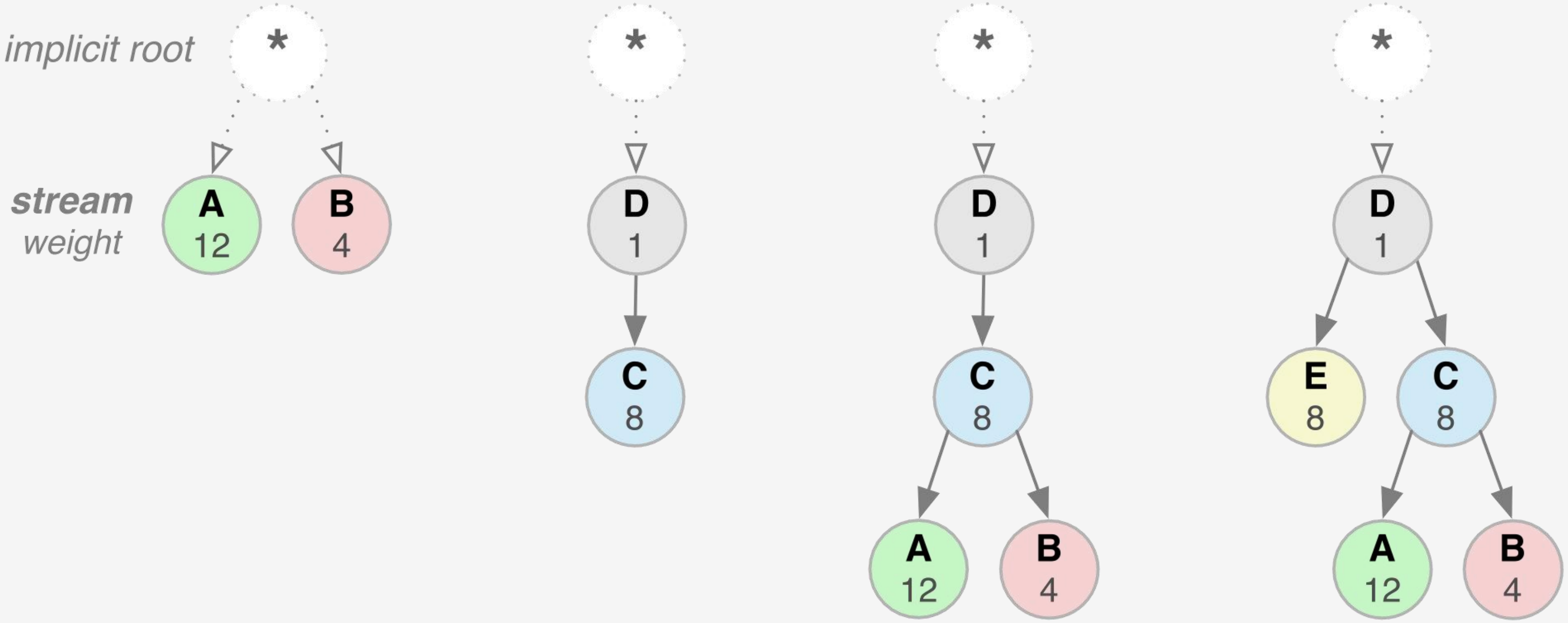


Multiplexing

https://http2.akamai.com/demo

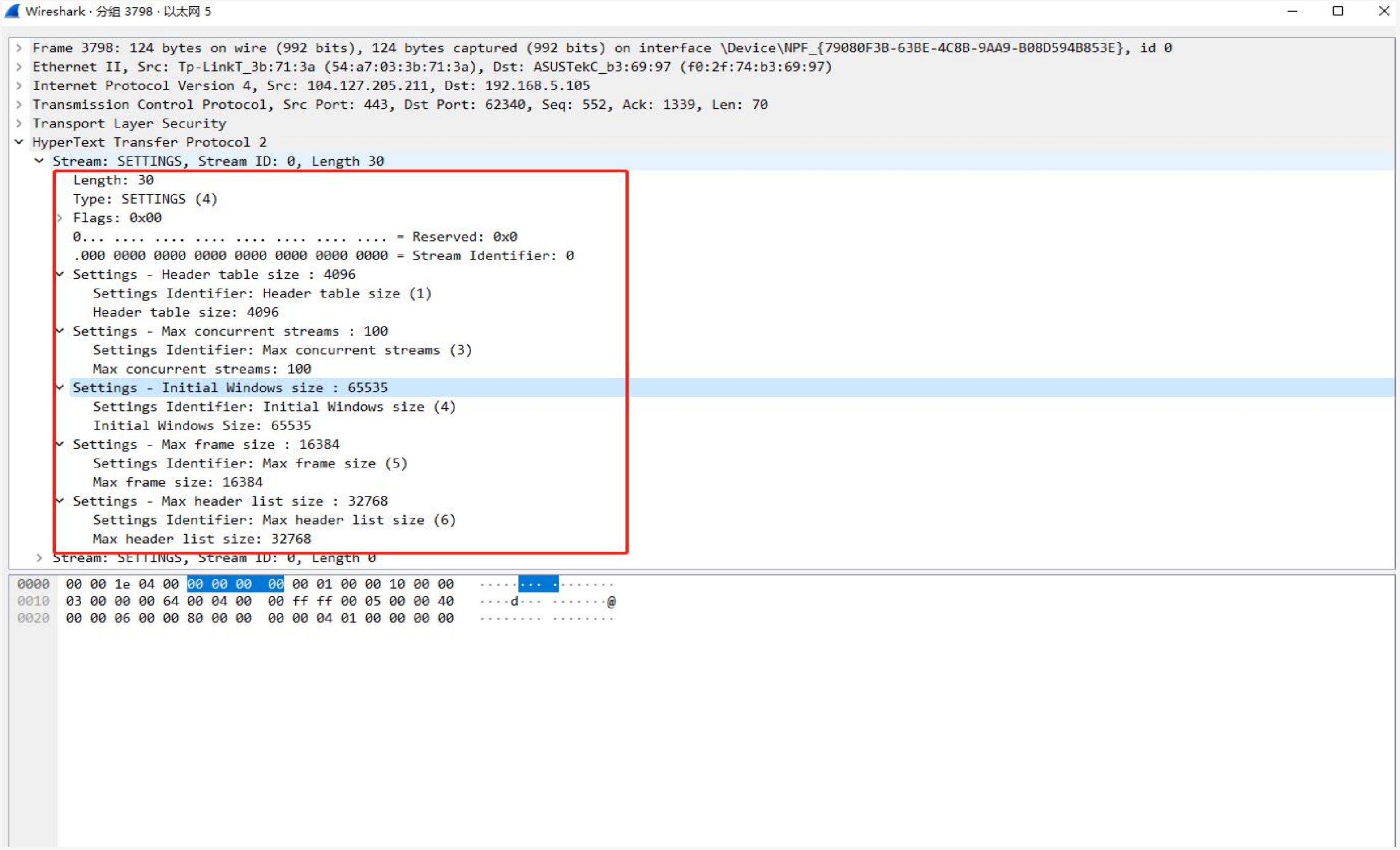


Introduce priority



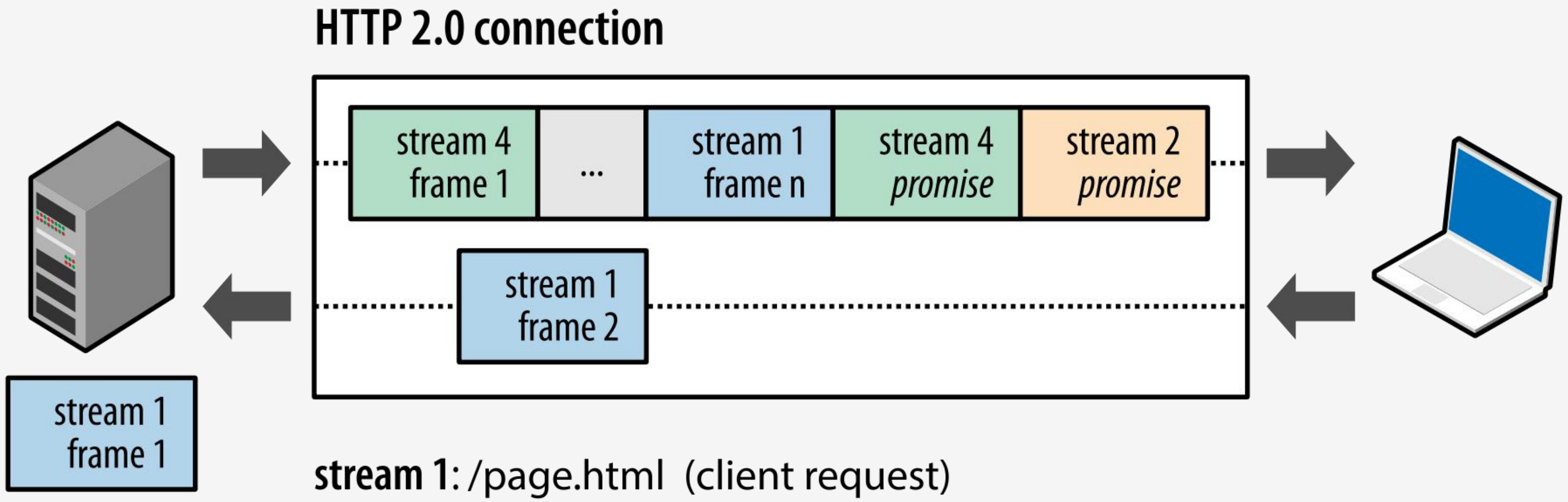
- 每个流都分配一个1~256优先级
- 流之间存在依赖关系

flow control



- 流控制具有方向性
- 仅针对data frame
- 可针对连接限制(stream id = 0)
- 可针对流限制(stream id > 0)

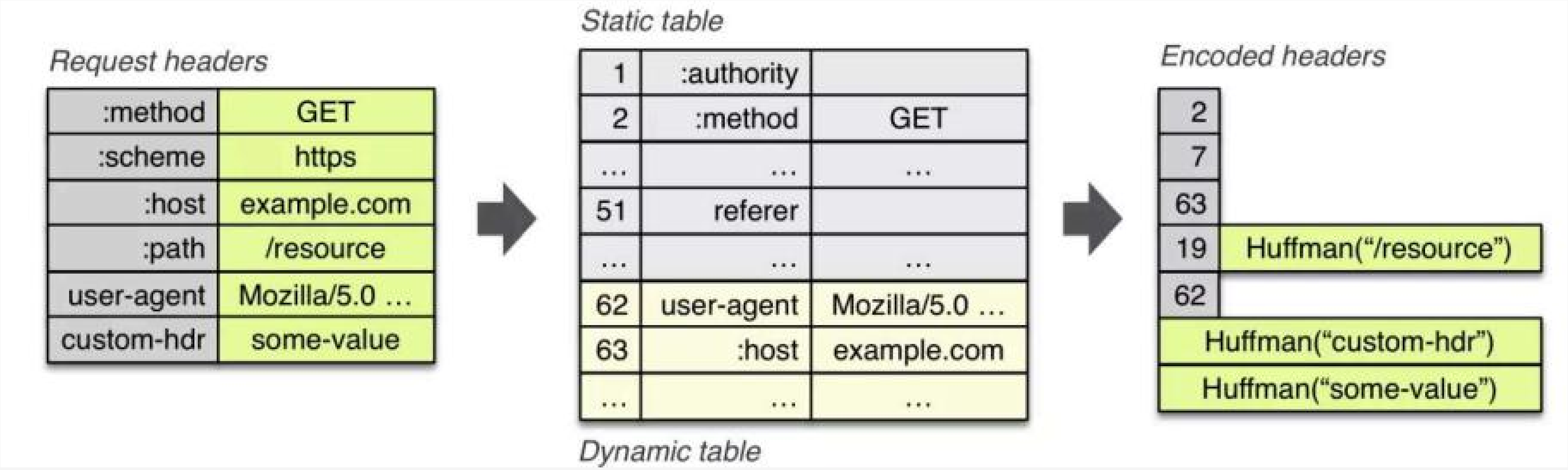
server push



stream 1: /page.html (client request)
stream 2: /script.js (push promise)
stream 4: /style.css (push promise)

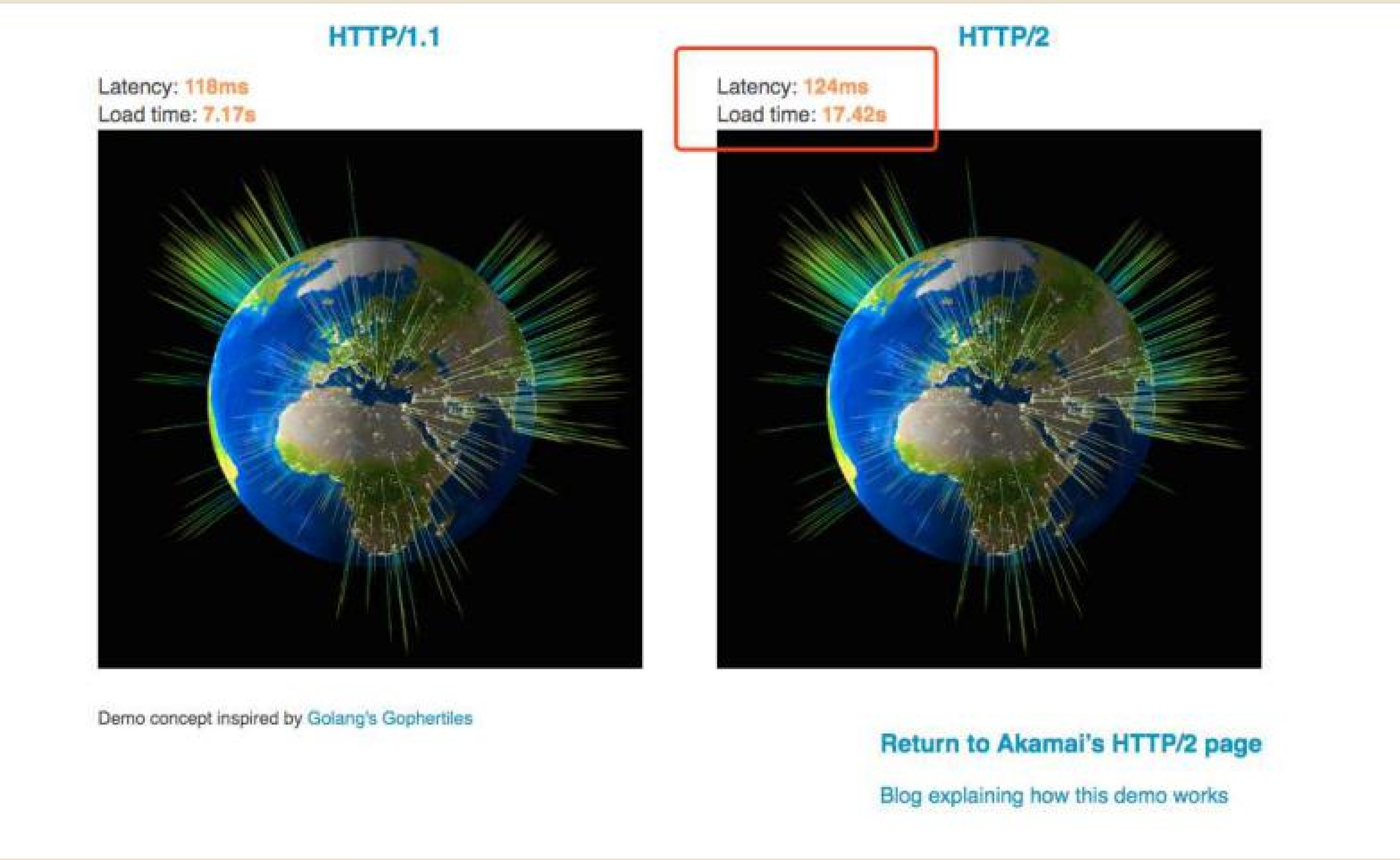
- 预知client的资源需求
- 由客户端缓存
- 不同页面间进行复用
- 由服务器设定优先级
- 客户端可以拒绝
- 帧类型为PUSH_PROMISE，新建stream

Hpack



- RFC中规定了静态的表(最常用的)
- 传输时，维护了一个动态表
- 如果命中，仅需要传递索引号
- 使用HPACK算法压缩（RFC7541）

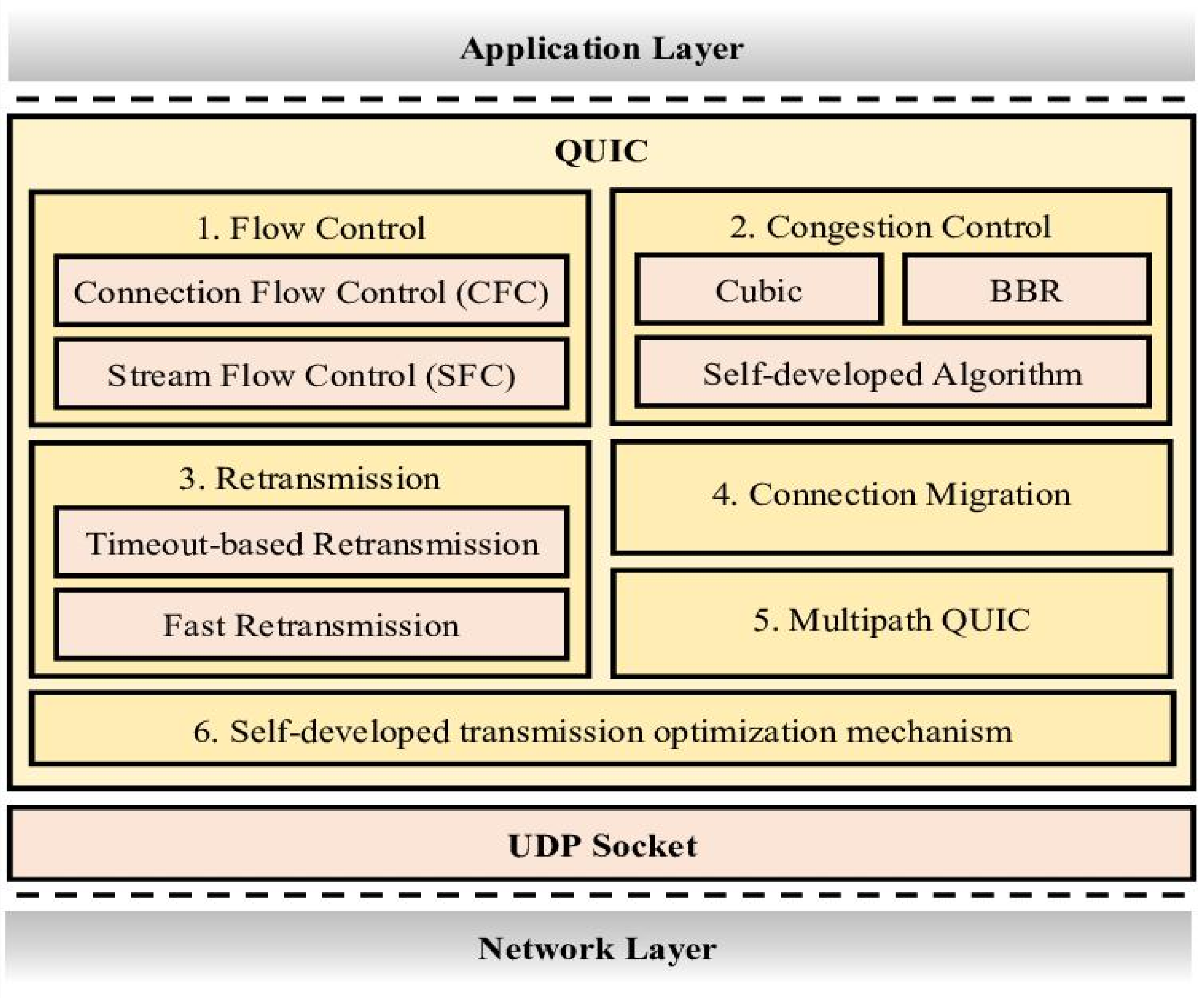
HTTP2 问题 (TCP ?)



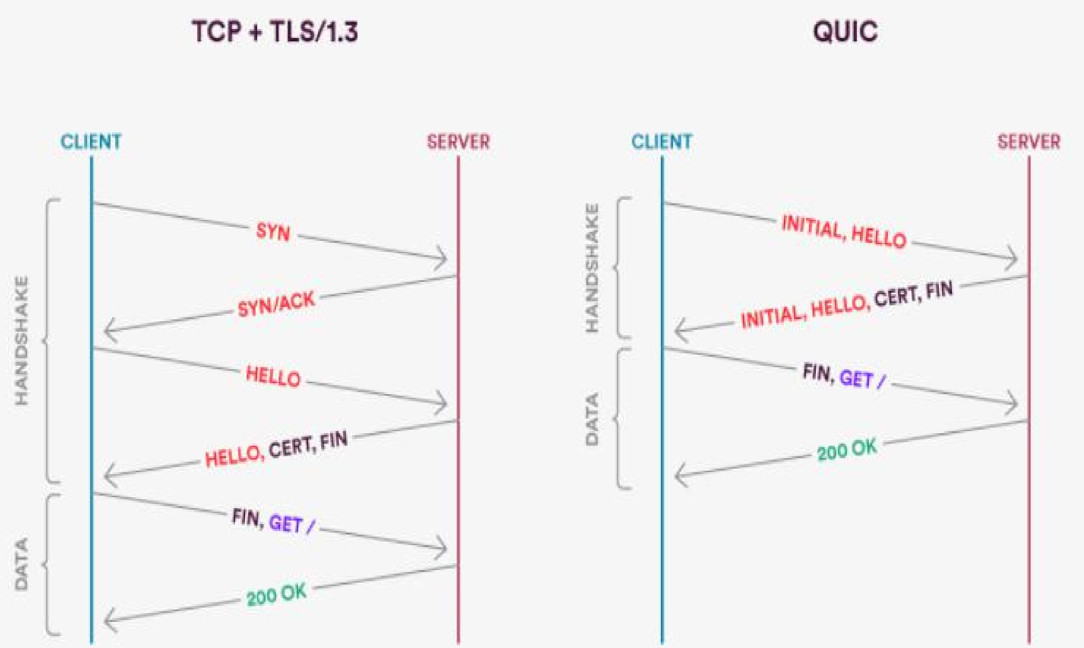
- 频繁丢包的慢网环境下，http2差于http1.1
- TCP的丢包重传带来了HOL问题
- TCP的拥塞控制导致整个链接传输窗口变小
- TCP需要三次握手，TLS也需要握手
- 移动网络变化IP地址
- SCTP/RTP ....

QUIC

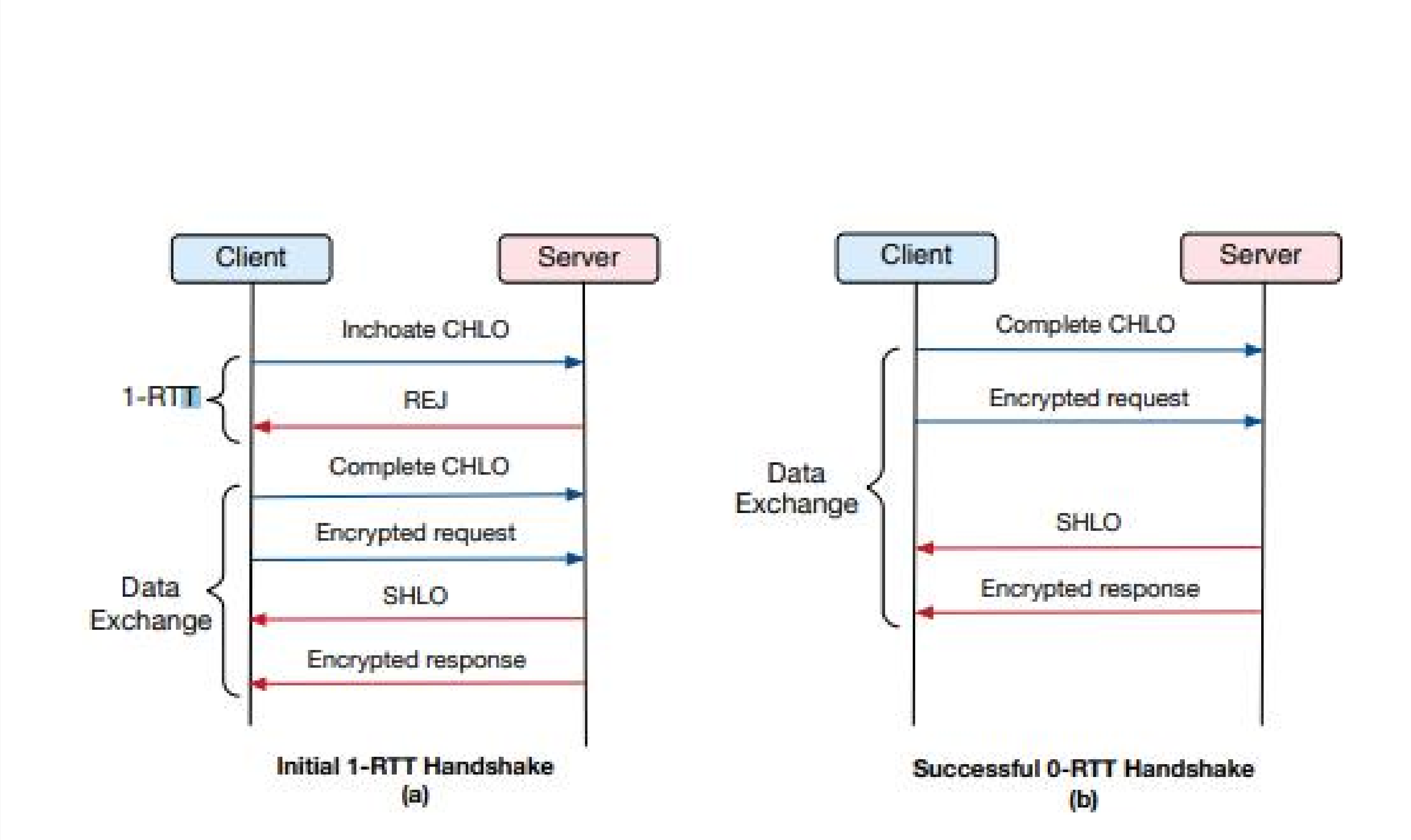
loss detection / congestion control (RFC9002)



1 RTT (RFC 9001)

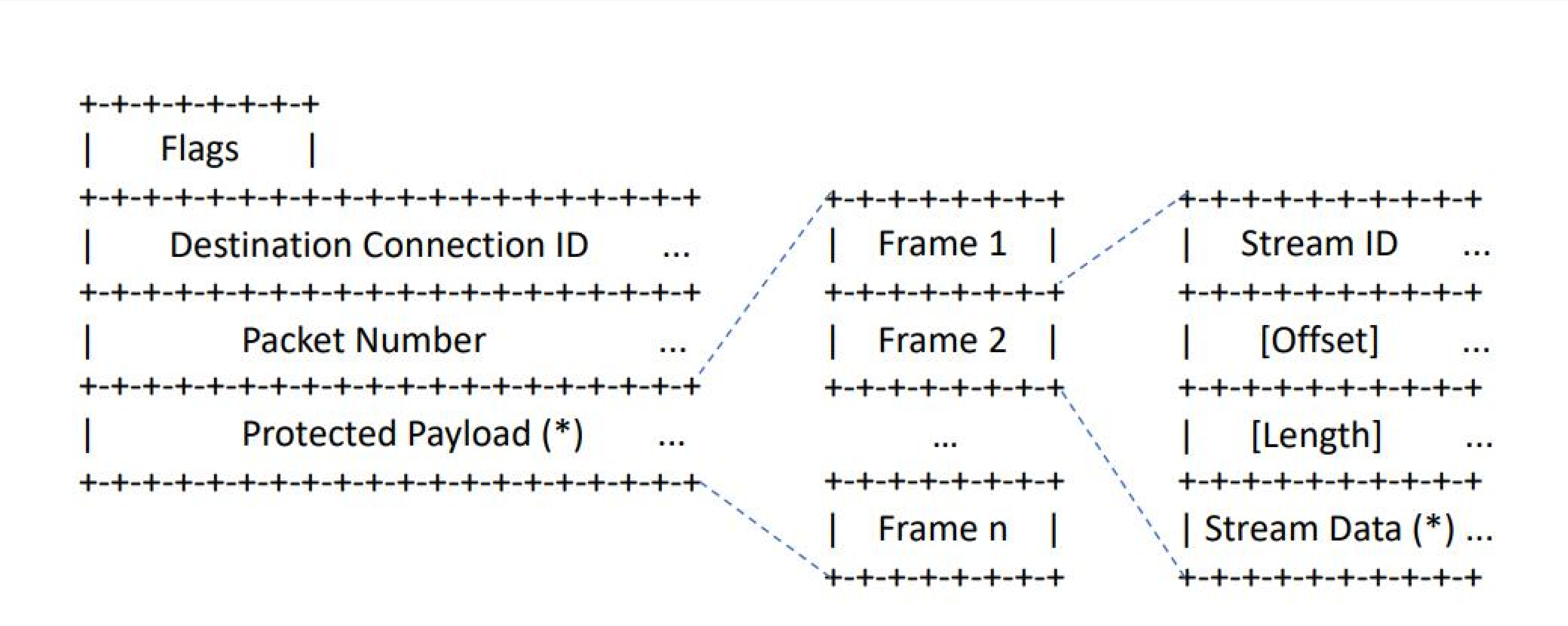


0 RTT (RFC 9001)

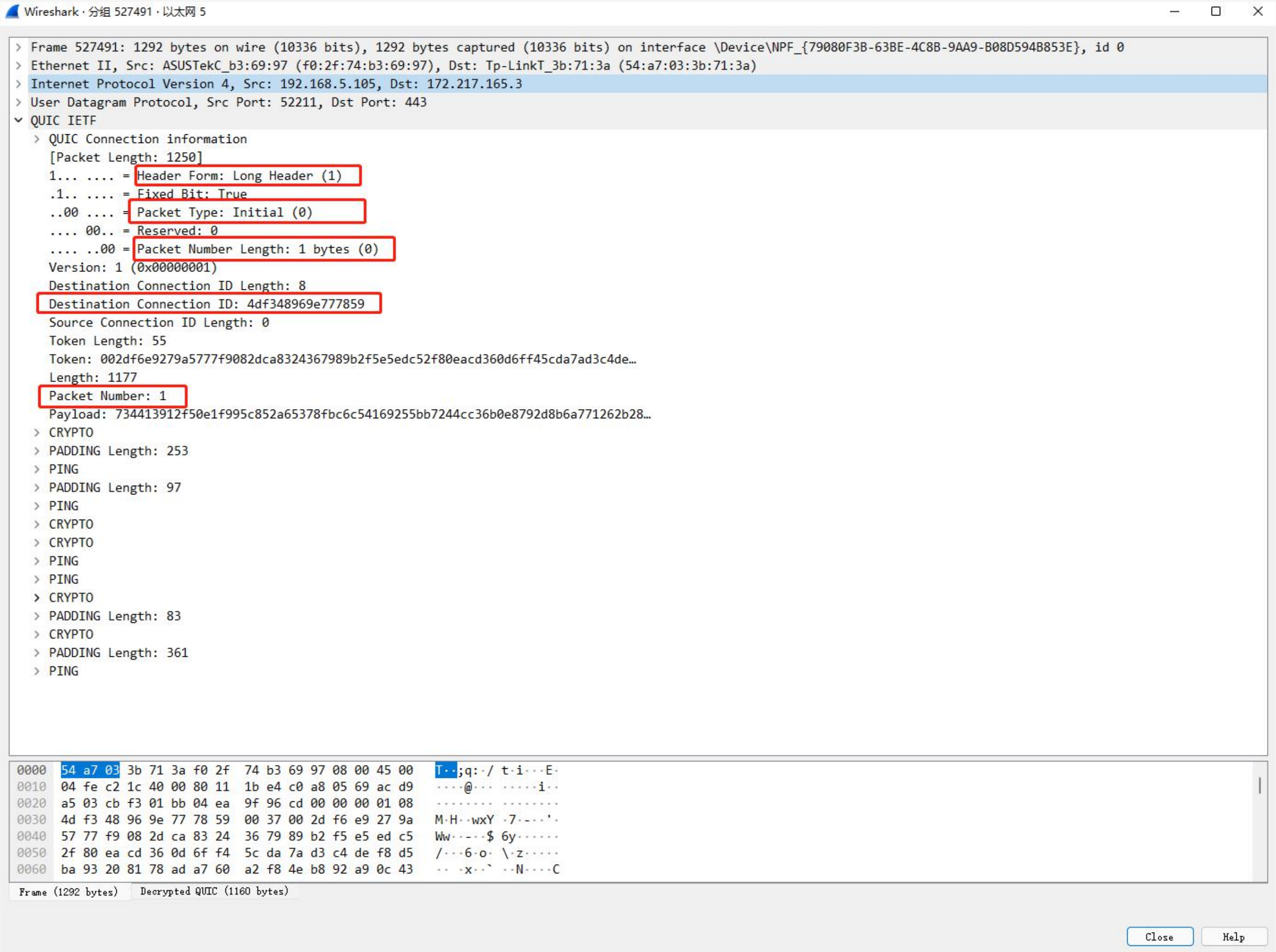


QUIC connection/stream/frame (RFC 9000)

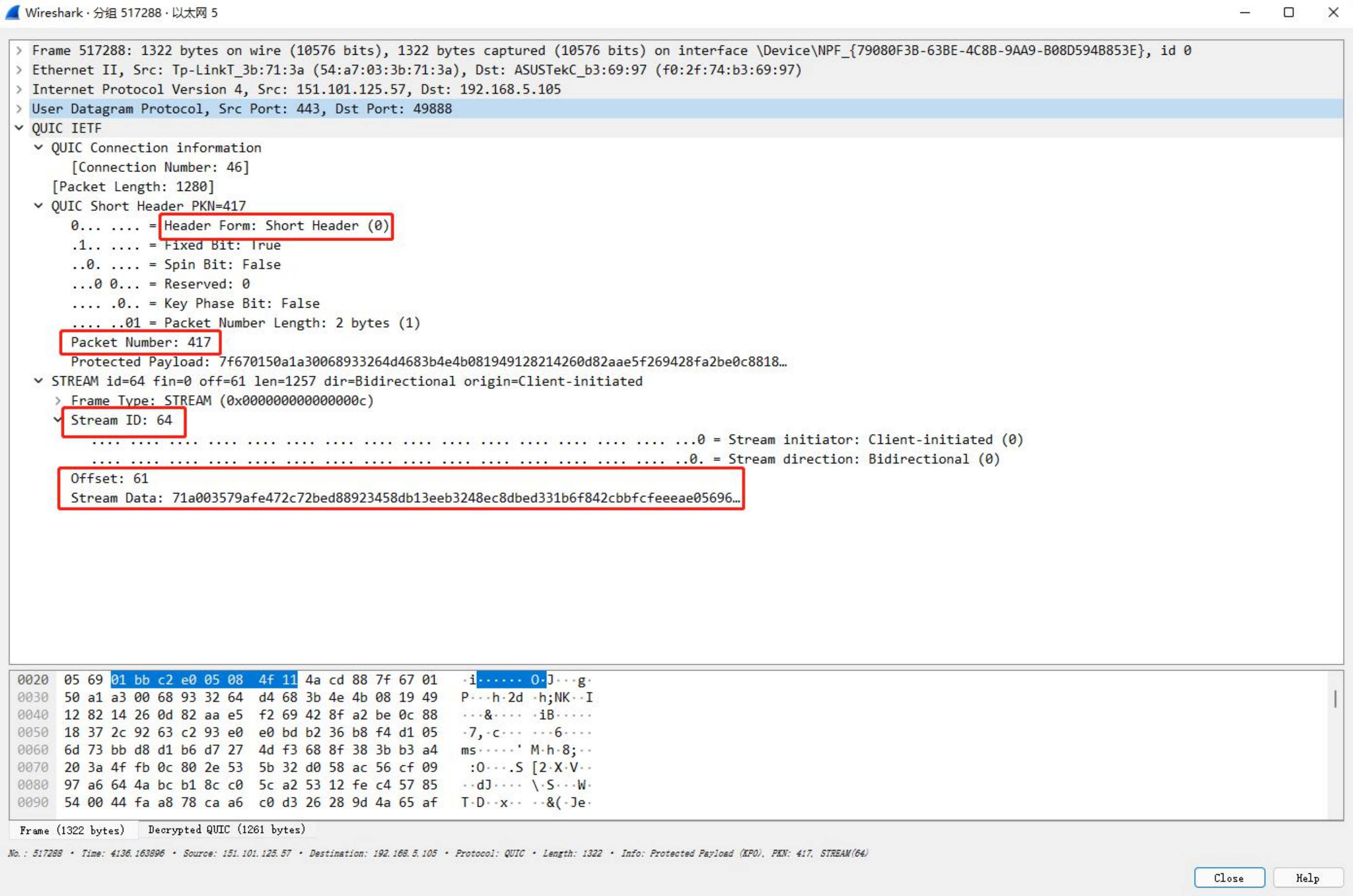
P12.3



QUIC connection/stream/frame (RFC 9000)



QUIC connection/stream/frame (RFC 9000)



QUIC encryption level (RFC 9001)

12.3. Packet Numbers

The packet number is an integer in the range 0 to 2^62-1. This number is used in determining the cryptographic nonce for packet protection. Each endpoint maintains a separate packet number for sending and receiving.

Packet numbers are limited to this range because they need to be representable in whole in the Largest Acknowledged field of an ACK frame (Section 19.3). When present in a long or short header, however, packet numbers are reduced and encoded in 1 to 4 bytes; see Section 17.1.

Version Negotiation (Section 17.2.1) and Retry (Section 17.2.5) packets do not include a packet number.

Packet numbers are divided into three spaces in QUIC:

Initial space: All Initial packets (Section 17.2.2) are in this space.

Handshake space: All Handshake packets (Section 17.2.4) are in this space.

Application data space: All 0-RTT (Section 17.2.3) and 1-RTT (Section 17.3.1) packets are in this space.

As described in [QUIC-TLS], each packet type uses different protection keys.

Conceptually, a packet number space is the context in which a packet can be processed and acknowledged. Initial packets can only be sent with Initial packet protection keys and acknowledged in packets that are also Initial packets. Similarly, Handshake packets are sent at the Handshake encryption level and can only be acknowledged in Handshake packets.

This enforces cryptographic separation between the data sent in the different packet number spaces. Packet numbers in each space start at packet number 0. Subsequent packets sent in the same packet number space MUST increase the packet number by at least one.

5.1. Packet Protection Keys

QUIC derives packet protection keys in the same way that TLS derives record protection keys.

Each encryption level has separate secret values for protection of packets sent in each direction. These traffic secrets are derived by TLS (see Section 7.1 of [TLS13]) and are used by QUIC for all encryption levels except the Initial encryption level. The secrets for the Initial encryption level are computed based on the client's initial Destination Connection ID, as described in Section 5.2.

The keys used for packet protection are computed from the TLS secrets using the KDF provided by TLS. In TLS 1.3, the HKDF-Expand-Label function described in Section 7.1 of [TLS13] is used, using the hash function from the negotiated cipher suite. All uses of HKDF-Expand-Label in QUIC use a zero-length Context.

Note that labels, which are described using strings, are encoded as bytes using ASCII [ASCII] without quotes or any trailing NUL byte.

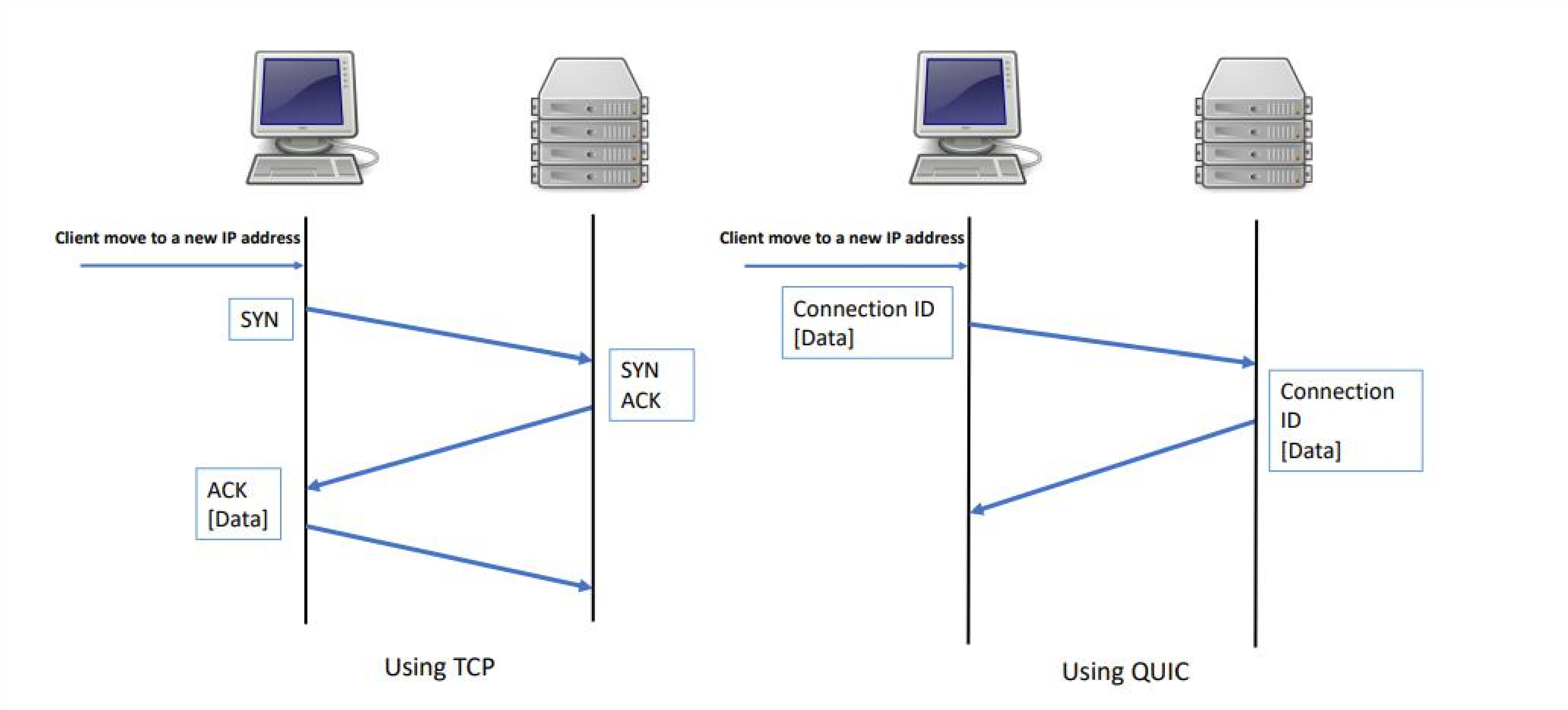
Other versions of TLS **MUST** provide a similar function in order to be used with QUIC.

The current encryption level secret and the label "quic key" are input to the KDF to produce the AEAD key; the label "quic iv" is used to derive the Initialization Vector (IV); see Section 5.3. The header protection key uses the "quic hp" label; see Section 5.4. Using these labels provides key separation between QUIC and TLS; see Section 9.6.

Both "quic key" and "quic hp" are used to produce keys, so the Length provided to HKDF-Expand-Label along with these labels is determined by the size of keys in the AEAD or header protection algorithm. The Length provided with "quic iv" is the minimum length of the AEAD nonce or 8 bytes if that is larger; see [AEAD].

The KDF used for initial secrets is always the HKDF-Expand-Label function from TLS 1.3; see Section 5.2.

connection migration



HTTP 2 VS HTTP 3 performence

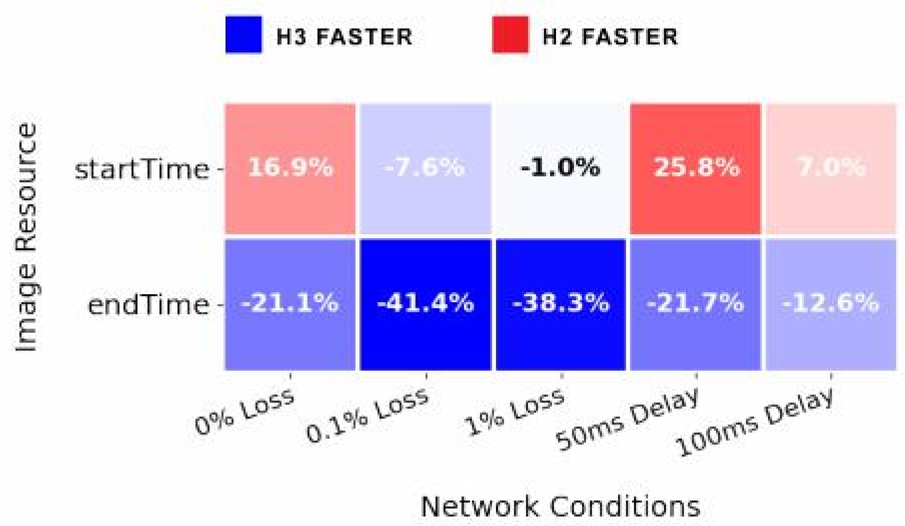


Figure 15: H2 vs H3 performance for the start and end times of the 'main' image resource of our large-sized Cloudflare web-page.

HTTP 2 VS HTTP 3 performence

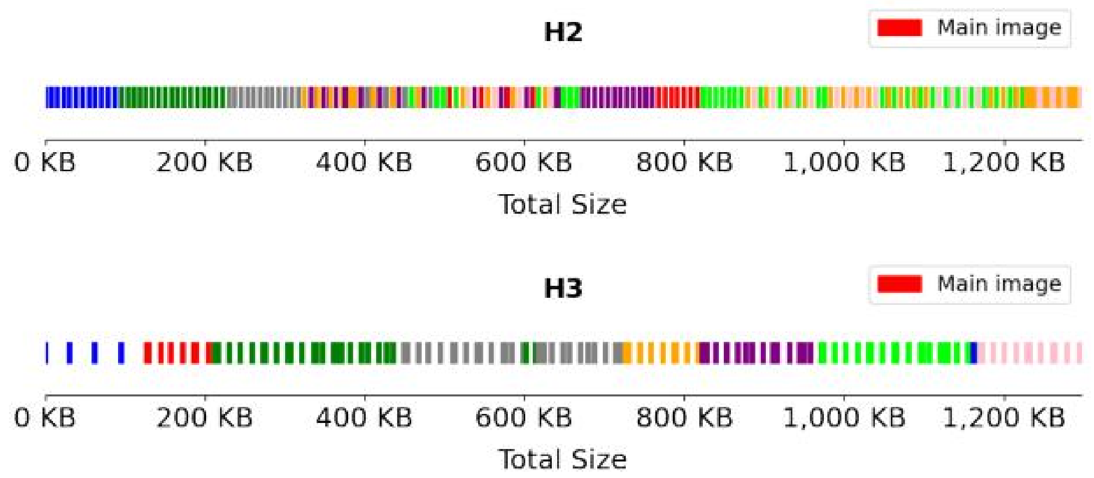
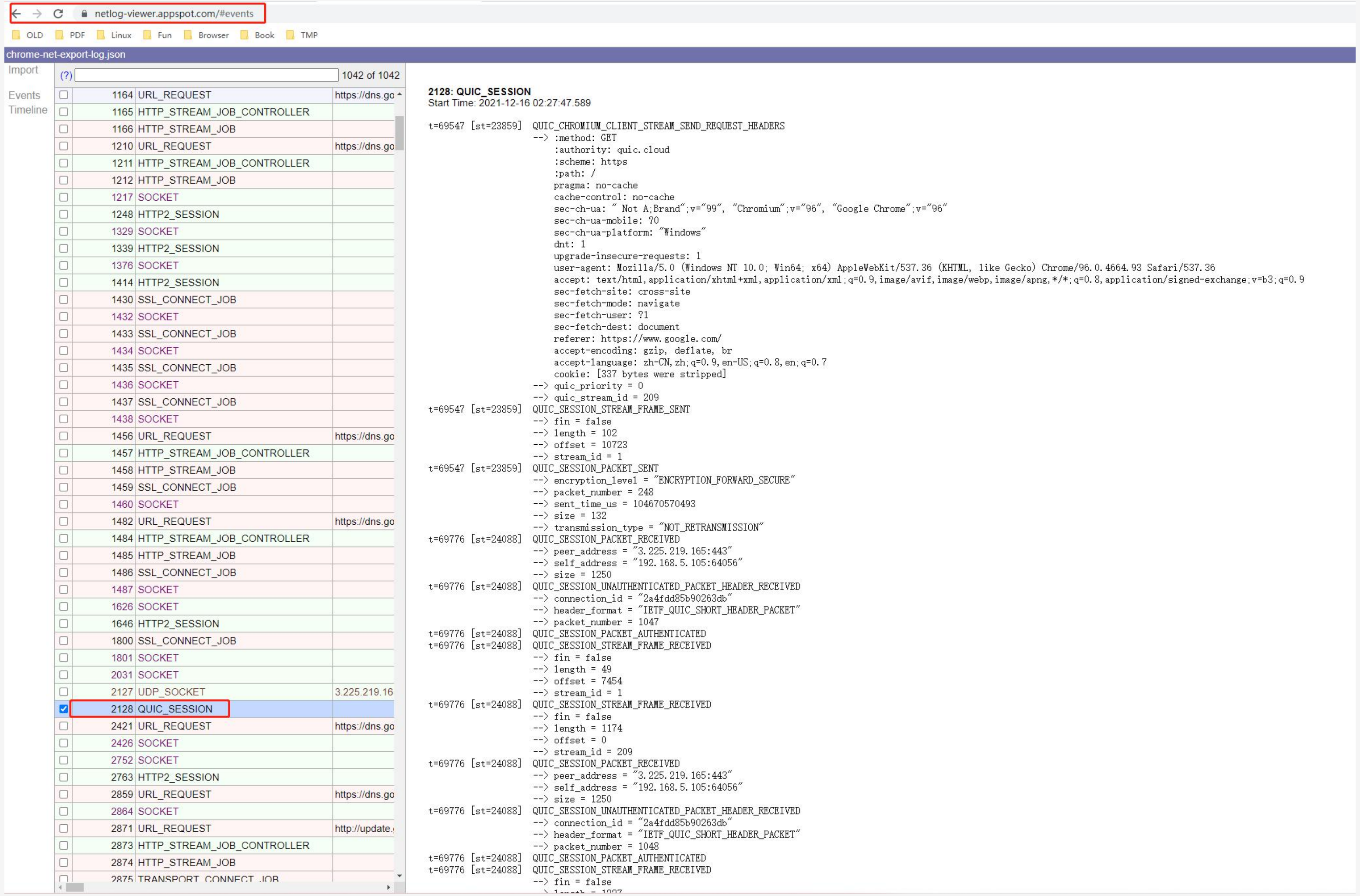


Figure 16: HTTP frames received by Chrome for 8 simultaneously-requested image/gif resources from our large-sized Cloudflare web-page. Each color represents a unique HTTP stream and each bar represents a chunk of data read by the application layer. Notice that H3 not only received the main image's[5] frames earlier, but also received them exclusively (H2 started receiving main image frames around the 500KB mark).

Problems

- UDP rate limiting and blocking
- More cpu usage
- DDOS
- Replay attack

debug time: QUIC complete process (chrome://net-internals/#events)(https://quic.cloud/)



Reference

- https://web.cs.ucla.edu/~lixia/papers/UnderstandQUIC.pdf
- https://datatracker.ietf.org/doc/html/draft-ietf-quic-recovery-34
- https://medium.com/jspoint/how-the-browser-renders-a-web-page-dom-cssom-and-rendering-df10531c9969
- https://developers.google.com/web/fundamentals/performance/http2
- https://www.slideshare.net/lmacvittie/http2-changes-everything
- http://xiaorui.cc/static/http2quic.pdf
- https://arxiv.org/pdf/1810.07730.pdf
- https://cs.brown.edu/~tab/papers/QUIC_WWW21.pdf