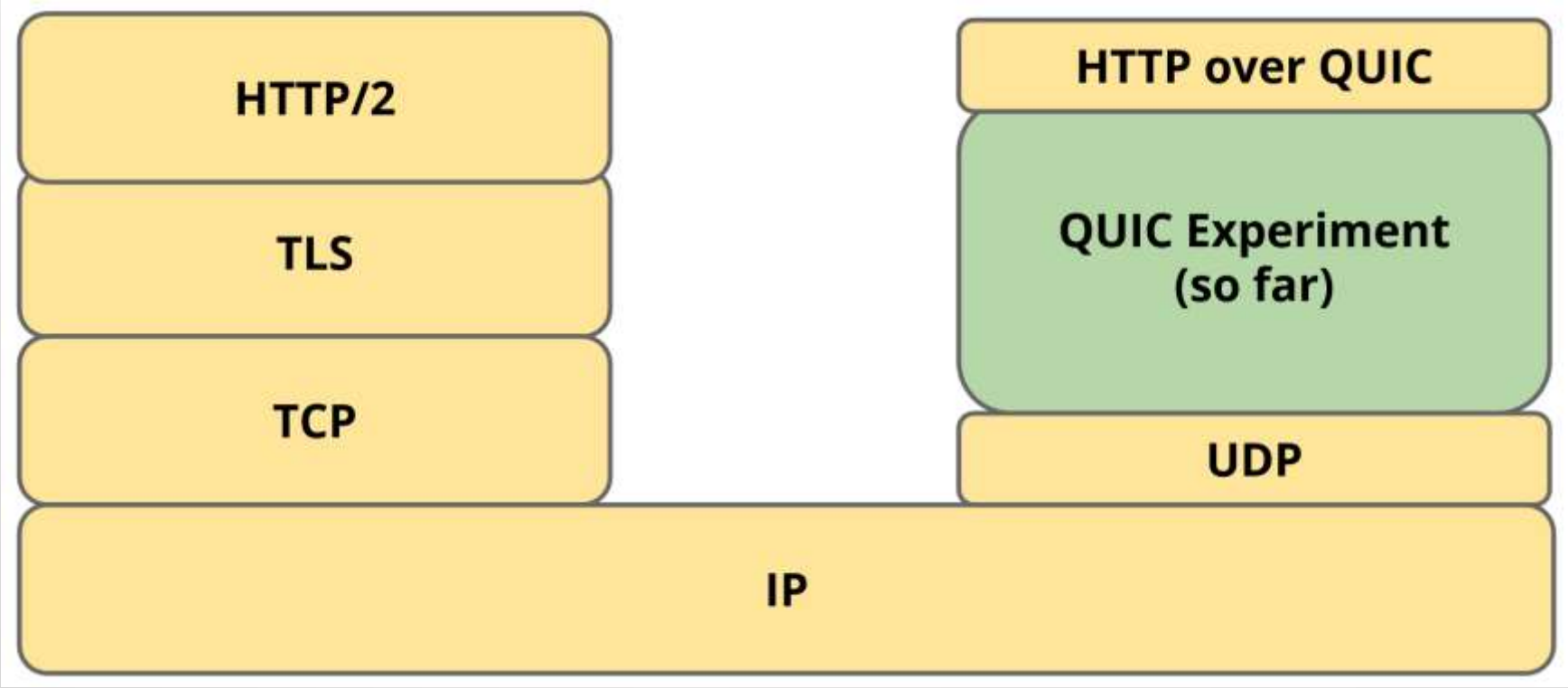
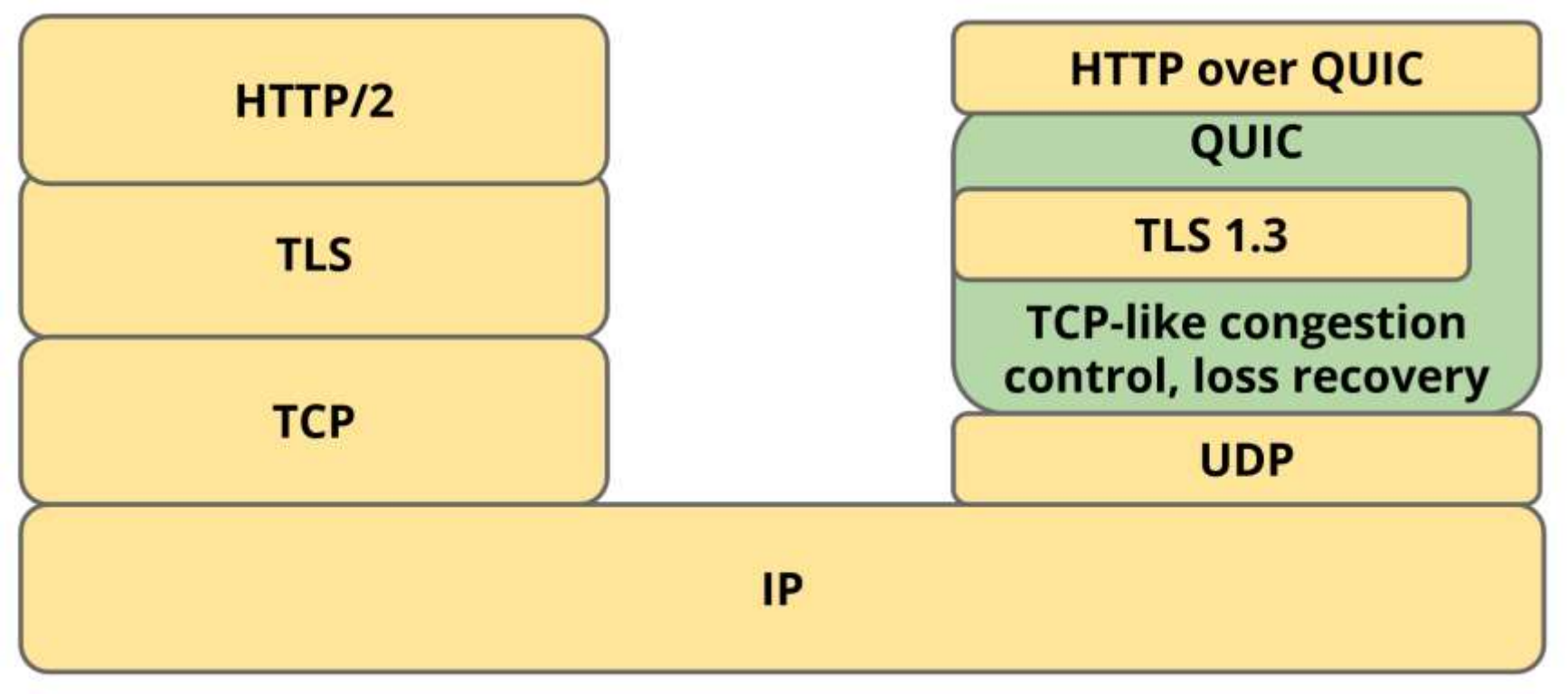
HTTP/QUIC

Update for HTTPbis WG

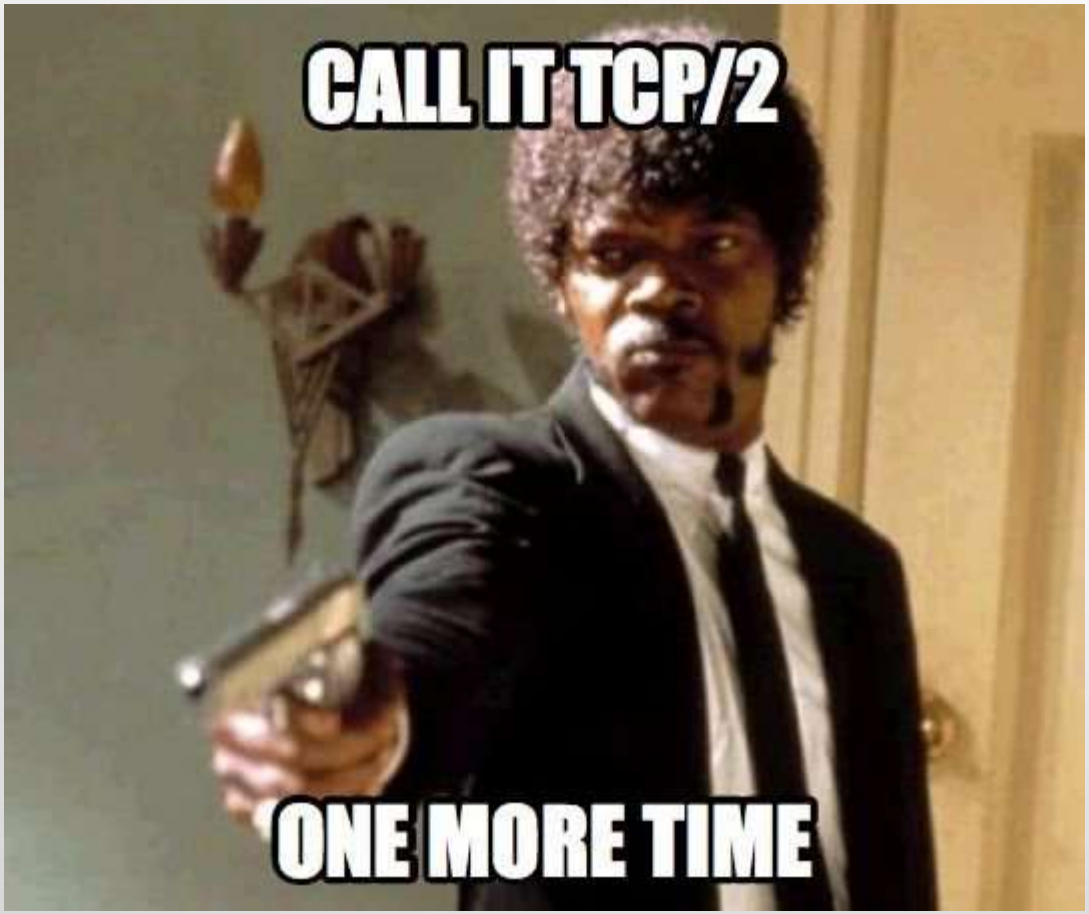
Google's QUIC Experiment



The IETF Version



A.k.a.



QUIC in a Nutshell

- Handshake establishes QUIC version, parameters, crypto, and app protocol in 0-2 RTTs
  - 0-RTT if you get the version right and can do TLS 1.3 resumption
- QUIC packets are encrypted containers of frames
- Loss detection identifies lost packets
  - …but lost frames get retransmitted
- Most frames are control-oriented; STREAM frames contain data from a particular stream
  - Odd-numbered streams are client-initiated
  - Even-numbered streams are server-initiated

Why? Agility!

- A UDP-based protocol *can* be implemented at the app layer
  - Ships with apps, so updates at the app's cadence, not the OS vendor's or device owner's
  - Ability to "reach inside" and pass more information if appropriate
  - But doesn't have to be!
- An authenticated/encrypted protocol blocks middlebox tampering
  - Apparently protocol innovation is hard to deploy because transparent intermediaries change bits or choke! Who knew?
  - QUIC incorporates many proposed TCP (or SCTP) improvements which haven't been successfully deployed

QUIC as Transport

TCP

- Headers protected against accidental corruption
- Payload in the clear (app can encrypt)
- Single-bytestream abstraction
- Congestion control
- Reliable delivery
- In-order delivery

QUIC

- Headers protected against any modification
- Payload encrypted
- Multiple-bytestream abstraction
- Congestion and flow control
- Reliable delivery
- In-order delivery *on each stream*
  - Order between streams not guaranteed

QUIC as HTTP/2 Substrate

H2 over TLS over TCP

- Headers protected against any modification
- Payload encrypted
- Multiple message-sequence abstraction with message types
- Congestion and flow control
- Reliable delivery
- In-order delivery *across all streams*
  - Relies on ordering between frames on different streams

QUIC

- Headers protected against any modification
- Payload encrypted
- Multiple-bytestream abstraction
- Congestion and flow control
- Reliable delivery
- In-order delivery *on each stream*
  - Order between streams not guaranteed

QUIC as HTTP/2 Substrate

H2 over TLS over TCP

- Headers protected against any modification
- Payload encrypted
- Multiple message-sequence abstraction with message types
- Congestion and flow control
- Reliable delivery
- In-order delivery *across all streams*
  - Relies on ordering between frames on different streams

QUIC

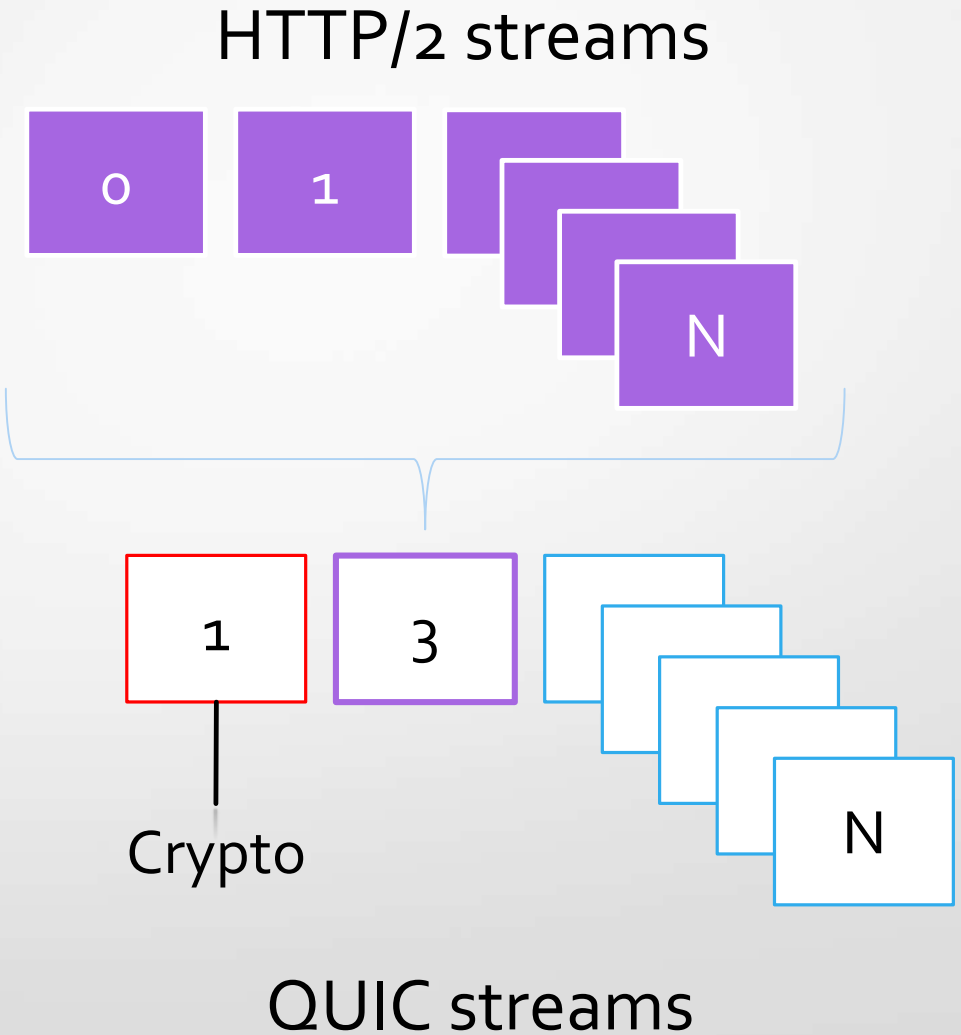
- Headers protected against any modification
- Payload encrypted
- Multiple-bytestream abstraction
- Congestion and flow control
- Reliable delivery
- In-order delivery *on each stream*
  - Order between streams not guaranteed

Connection Negotiation

- HTTP/QUIC support detected by use of Alt-Svc
- New Alt-Svc "quic" parameter as version negotiation hint
  - QUIC uses optimistic version negotiation
    - Client proposes a version
    - Server either accepts or responds with a list of versions
    - Client retries with a mutually supported version
  - Discovering supported version(s) via Alt-Svc saves 1 RTT
- Currently no way to declare an HTTP/QUIC URL directly
  - "httpq" proposed…?
- ALPN token is hq
  - hq-xx for drafts (e.g. hq-02)

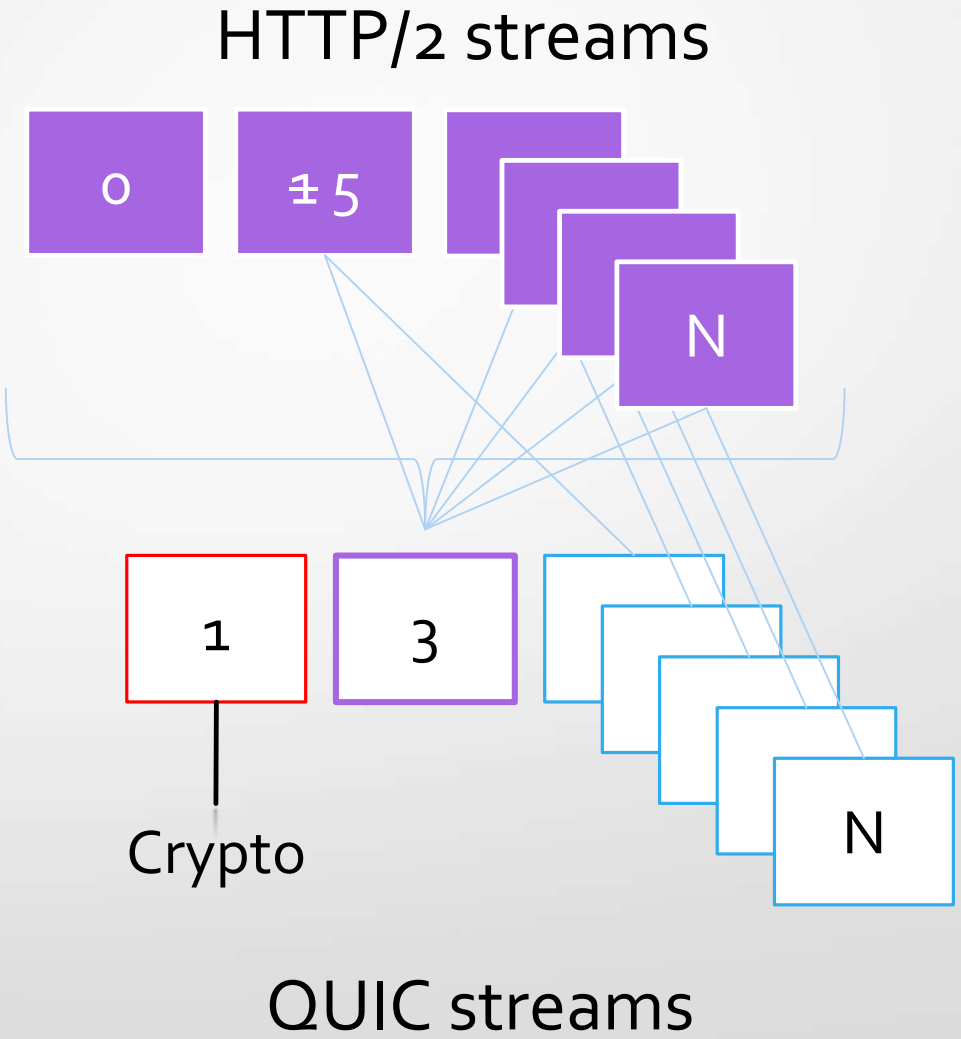
Google QUIC Stream Usage

- Stream 1 reserved for crypto
- Stream 3 reserved for abridged HTTP/2 session
  - Reflects migration path from TCP to QUIC
  - Functionality added to QUIC is removed from HTTP/2
    - PING
    - GOAWAY
    - Flow Control



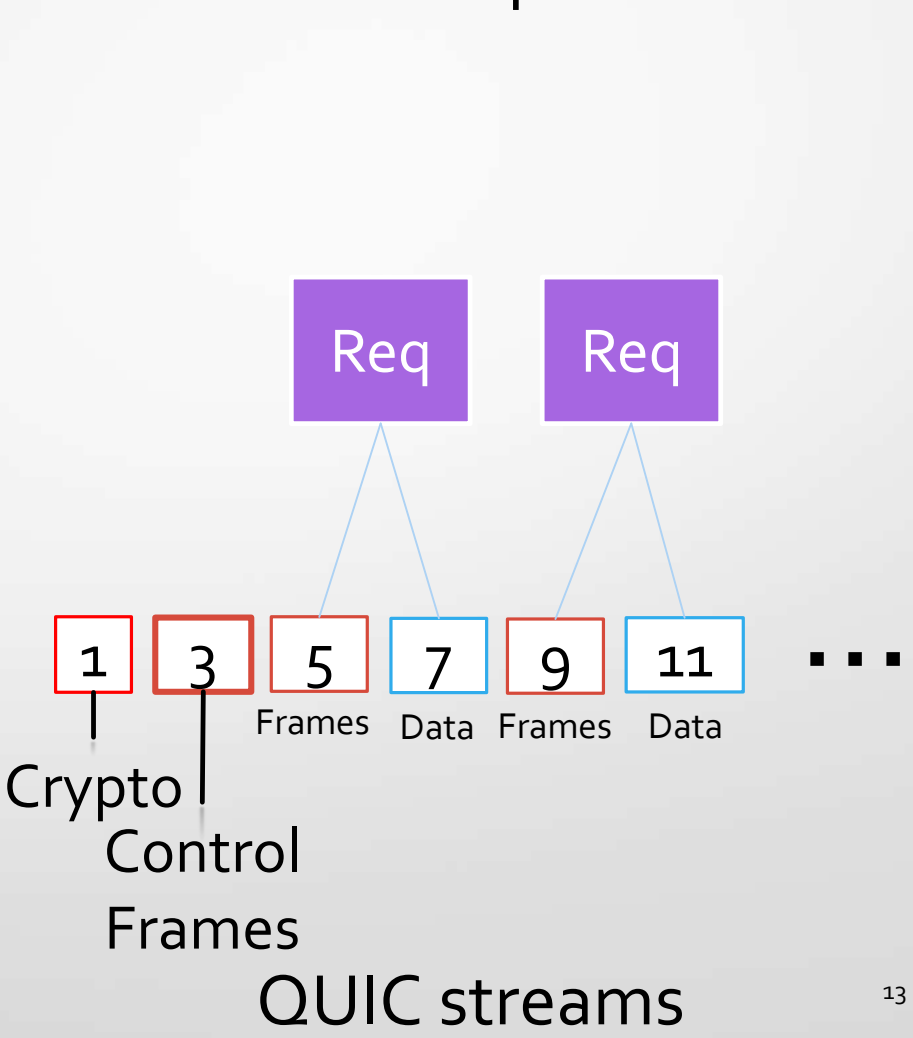
Google QUIC Stream Usage

- Stream 1 reserved for crypto
- Stream 3 reserved for abridged HTTP/2 session
- HTTP/2 streams straddle QUIC Stream 3 *and* another QUIC stream
  - H2 Stream 0 is only on QUIC Stream 3
  - Other QUIC streams replace DATA frames
  - All other frames (HPACK) on QUIC Stream 3



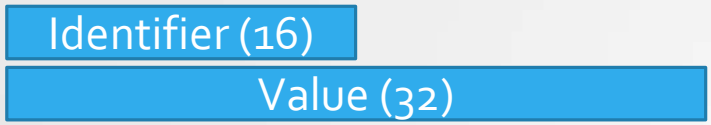
Current HTTP/QUIC Stream Usage

- Stream 3 – Connection Control Stream
  - Carries session-wide info (SETTINGS, PRIORITY)
- Each request occupies two streams
  - Message control stream – HEADERS, etc.
  - Unframed data stream carries message payload
- No muxing in HTTP-layer framing, but still uses frames

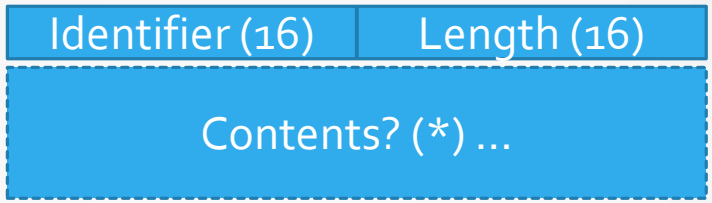


Adopted EXTENDED_SETTINGS

HTTP/2 SETTINGS



HTTP/QUIC SETTINGS

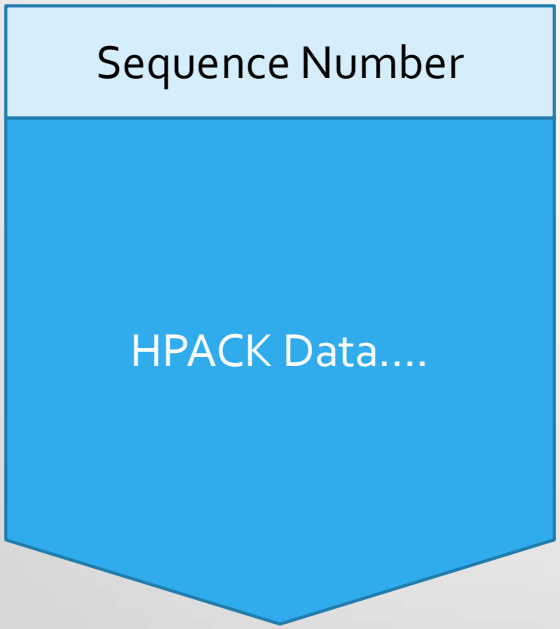


- Based on draft-bishop-httpbis-extended-settings
  - HttpBis feedback: Save for protocol rev, not an HTTP/2 extension with separate identifier space
- Borrows heavily from RFC7540 SETTINGS text
- Values are length-prefixed blobs
- Optimization for Boolean values
  - If length=0, true; not sent is false

Where there is no order….

- Changes to priority tree aren't commutative
  - PRIORITY frames on Stream 3 (== Stream 0 in HTTP/2) to preserve ordering
- SETTINGS ACK gets *really* hard
  - Need to ACK on every open stream, plus on Stream 3 identify which streams were open when the SETTINGS frame was processed
  - Simpler: Just don't allow mid-session changes; new connections are cheap
- And then there's HPACK….

Shoehorning HPACK



- HTTP/QUIC -02 still uses HPACK
- Adds a counter on HPACK frames
  - Requires decoder process frames in encode-order
- No *more* HOLB than before, but no less
- Can't reset message control streams
- Alternatively, QPACK proposals:
  - draft-bishop-quic-http-and-qpack
  - draft-krasic-quic-hpack

HTTP/2 Extensions in HTTP/QUIC

- Separate error registry
  - Because QUIC has a unified error space for use in RST_STREAM, CONNECTION_CLOSE
  - Need to redefine extension-based error codes
- Shared frame registry with HTTP/2
  - But many HTTP/2 frames don't exist and none are identical!
  - Need to define how extension's frames work in different context; some changes could be required
- Shared SETTINGS registry with HTTP/2
  - But half the HTTP/2 settings don't exist and one has opposite semantics!
  - Need to define what extension settings mean in different context; some changes could be required
- Discussion on splitting from HTTP/2 IANA registries

Summary

- QUIC is a new alternative to TCP
  - UDP is just an design detail
- QUIC includes many features HTTP/2 constructed on top of TCP
  - …which means we no longer need them at our layer
- HTTP/QUIC isn't *quite* HTTP/2, but it's related enough it should look quite familiar
- HTTP/QUIC, like HTTP/2, attempts to carry HTTP semantics unchanged
  - For semantically-different work, QUIC may be an excellent transport, but a poor WG home