HTTP/QUIC

draft-ietf-quic-http-04

Recap: Why are we here?

- 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

Recap: Why is QUIC this way?

- 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 limits 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

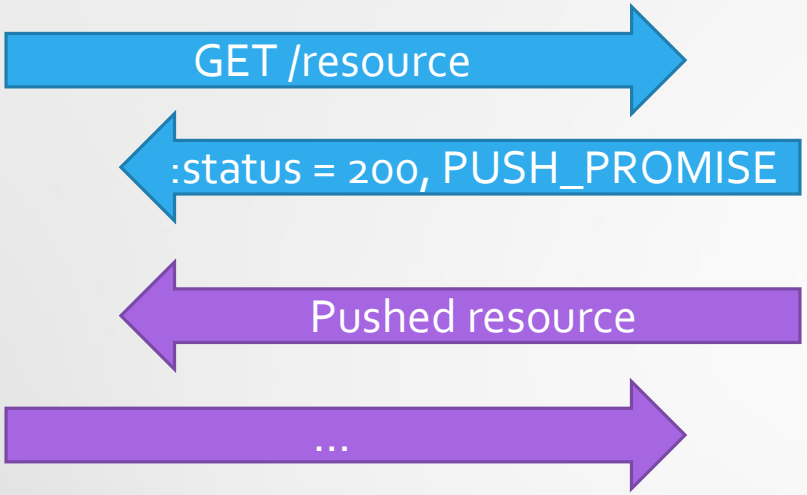
Changes since Chicago

- HTTP draft pretty quiet – current focus is on transport
- Minor changes in response to transport changes
  - Crypto moved to stream 0, so control moved to stream 1
- Clarified that since Alt-Svc always specifies port, there's no designated port for HTTP/QUIC

Key Issues for HTTP/QUIC

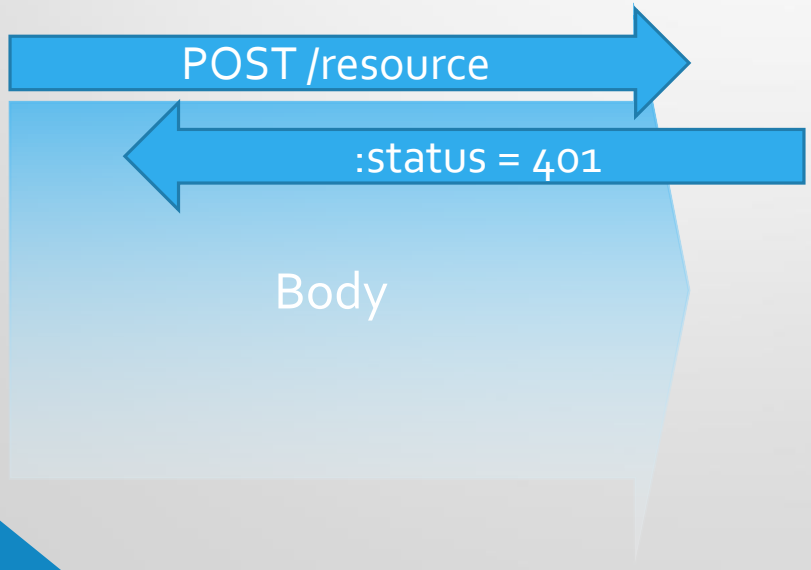
- Stream Issues
- Header compression
- Settings
- Priorities
- Coupling with HTTP/2
- Authority

Stream Awkwardness in HTTP



Push streams only carry data from server to client…

…but client is required to respond (close the stream)



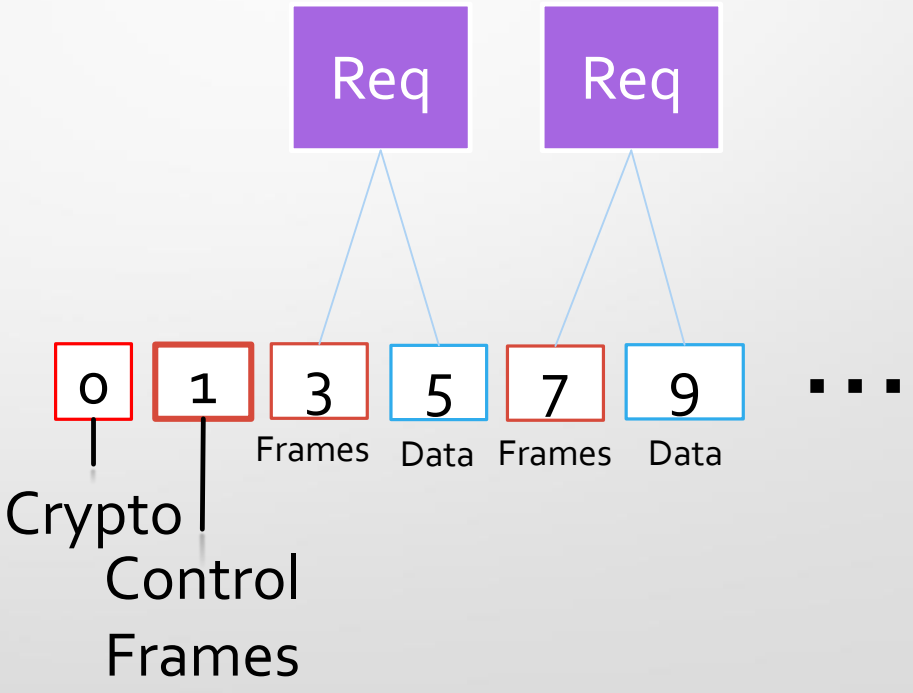
When the server is done answering, but the client isn't done talking yet….

Unidirectional Proposals

- Status quo – Streams are bidirectional
  - Idle -> open -> half-closed (local/remote) -> closed
- Pairs of unidirectional streams
  - Each direction is independently idle -> open -> closed
  - "Half-closed" is a shorthand for describing a pair of streams
- Streams can be opened unidirectionally
  - Either of the previous, but adds an idle -> half-closed transition
- Fully unidirectional
  - Separate stream ID space in each direction
  - Each stream is idle -> open -> closed
  - Someone has to handle correlation

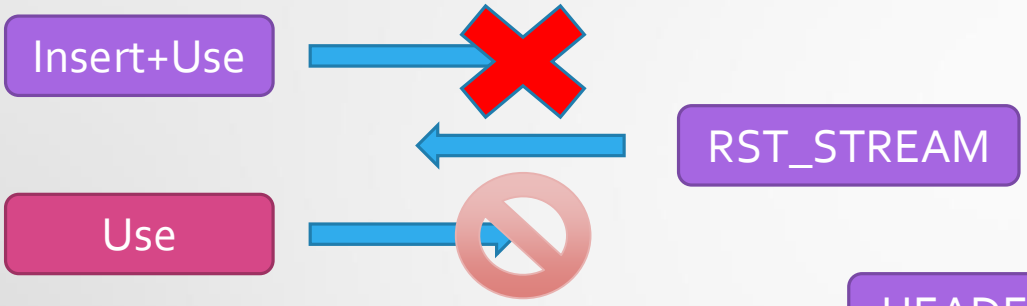
Current Stream Usage

- Stream 1 – 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



Why two streams?

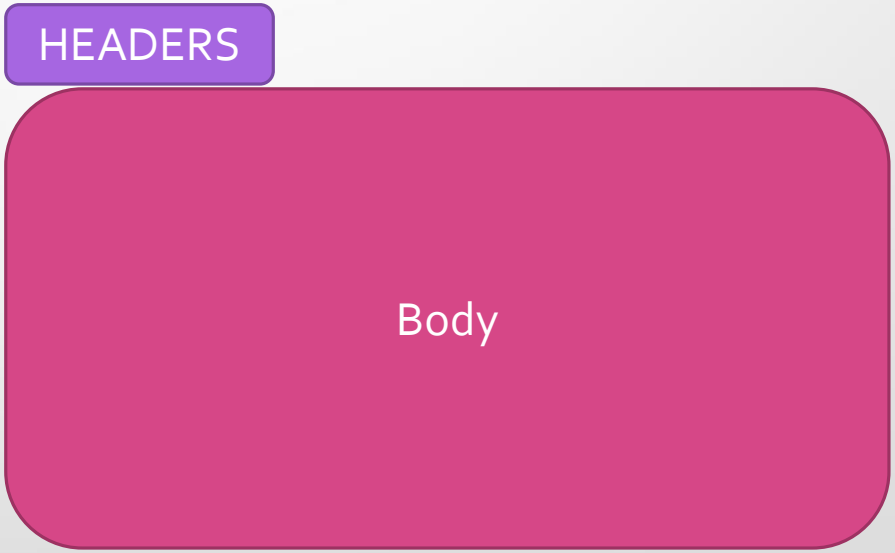
HPACK



HPACK data can never be reset safely!

Separate body stream means no extra framing at HTTP layer

Ability to separately flow-control headers versus body



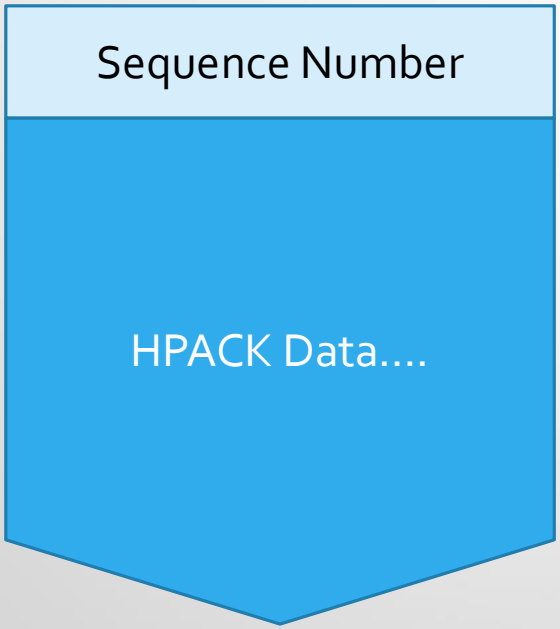
Why one stream?

No way to support PUSH_PROMISE ordering on different streams

And anyway….

- We need to fix the vulnerability to loss in the header compression
- The extra overhead of DATA frames is minimal – 4 bytes per up to 64KB
- DATA frames may still be required on two streams to solve ordering problems

Shoehorning HPACK



- HTTP/QUIC -04 still uses HPACK
- Adds a counter on HPACK frames
  - Requires decoder process frames in encode-order
- No *more* HOLB than before, but no less
- Alternative proposals:
  - draft-bishop-quic-http-and-qpack
  - draft-krasic-quic-qcram
  - HPACK with zero dynamic table size (temporary)

HPACK Alternatives

QPACK

- HPACK-derived, new wire format
- Allows trade-off between HOLB risk and compression efficiency
- Avoids lost-data risk by using dedicated stream for table changes
- Does not require access to transport-level knowledge (but implementation might leverage)

QCRAM

- Uses HPACK wire format, with additions to HEADERS frame
- Entirely prevents HOLB
- Will incorporate a similar technique via two passes over headers
- Assumes some access to packetization logic / rollback

SETTINGS and Handshake

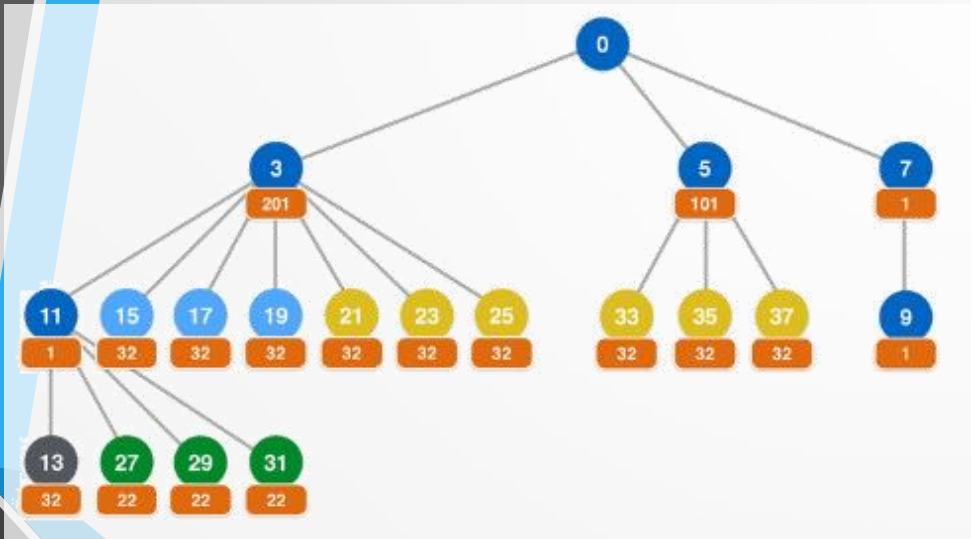
- HTTP/QUIC uses an HTTP/2-like SETTINGS frame
  - Only at the beginning – no changes!
- Proposal to simplify further and combine HTTP settings with QUIC settings in handshake
  - Strawman:
    - Each application gets a QUIC transport setting with a blob value
    - Pack the contents of a SETTINGS frame into this blob
    - Include the application settings for any application(s) you're offering
  - Potential drawback: Client's settings are in the clear

Integrated Errors

- QUIC currently divides error space into four regions
- Connection or stream can close for any error in any region

- Discuss:
  - Transport should never close streams
    - Streams close only for application-defined reasons
    - Transport errors are fatal
  - Can application terminate connection with error? What does this look like?

Priorities and Placeholders



- Some UAs implement priorities using idle streams which are never used
- QUIC has a strong preference for contiguous stream use in order
- How do we want to deal with this?

HTTP/2 Divergence

- Separate error registry
  - Because QUIC has a unified error space for use in RST_STREAM, CONNECTION_CLOSE
- Shared frame registry with HTTP/2
  - Some HTTP/2 frames don't exist in HTTP/QUIC
  - Of those that do, zero frames are identical between HTTP/2 and HTTP/QUIC
  - Extensions don't automatically work
- Shared SETTINGS registry with HTTP/2
  - Half the HTTP/2 settings don't exist
  - Extensions don't automatically work
- Added a "Transitioning from HTTP/2" section describing differences

Authority

- Authority over an origin in HTTP is defined by the URI's scheme, hostname, and port
  - 'http' and 'https' schemes are defined to use TCP
- Alt-Svc allows delegation from an authoritative origin to another protocol/host/port tuple
  - …which need not be UDP 443
- When either client or server doesn't have/want TCP, how do we do this?