QMux?

~~QUIC on Streams~~

from scope to requirements

QUIC WG Interim, June 2025
Kazuho Oku, Lucas Pardue

Why are we doing this?

Quoting Alan's slide from IETF 122:

- QUIC's stream multiplexing abstraction is powerful and widely used.
- It's the basis of new protocols (e.g., MoQ) that want to be able to run over both UDP and not UDP. Having a standard prevents bespoke fallback.
- It's generally useful for applications and protocols that might never use QUIC (e.g., datacenter RPC).
- There's already a way to use QUIC's Multiplexing with TCP+TLS - it's called WebTransport over H2.
- The dependency on HTTP is overkill outside of browsers.
- WT/H2 would likely have used QMux if it existed.
- At least 3 implementations of Kazuho's draft already exist.

Why are we doing this?

Quoting Alan's slide from IETF 122:

- QUIC's stream multiplexing abstraction is powerful and widely used.
- **It's the basis of new protocols (e.g., MoQ) that want to be able to run over both UDP and not UDP**. Having a standard prevents bespoke fallback.
- **It's generally useful for applications and protocols that might never use QUIC (e.g., datacenter RPC).**
- There's already a way to use QUIC's Multiplexing with TCP+TLS - it's called WebTransport over H2.
- The dependency on HTTP is overkill outside of browsers.
- WT/H2 would likely have used QMux if it existed.
- At least 3 implementations of Kazuho's draft already exist.

Reliable stream transports are ubiquitous

- Many systems already expose an in-order, reliable byte stream abstraction:
  - TCP
  - TLS
  - UNIX domain sockets
  - InfiniBand RC, FiberChannel*
- Not all these transports rely on loss recovery and congestion control; some have reliable delivery, use flow control to control congestion.
- Some transports are secure; otherwise TLS provides authentication and encryption.

*note: these systems can also transmit where message boundaries are in addition to sending bytes

Features provided by QUIC version 1

- Stream multiplexing
- Framework for extensions
  - that deal with data: e.g., datagrams, reliable resets
  - that do not: ack-frequency, load balancers, multipath
- Packetization
- Loss recovery
- Congestion control
- Authentication and Encryption

Features needed by reliable stream transports

- Stream multiplexing
- Framework for extensions
  - that deal with data: e.g., datagrams, reliable resets
  - ~~that do not: ack-frequency, load balancers, multipath~~
- ~~Packetization~~
- ~~Loss recovery~~
- ~~Congestion control~~
- ~~Authentication and Encryption~~

costly features are better to be avoided, if unnecessary

Hence the requirements are

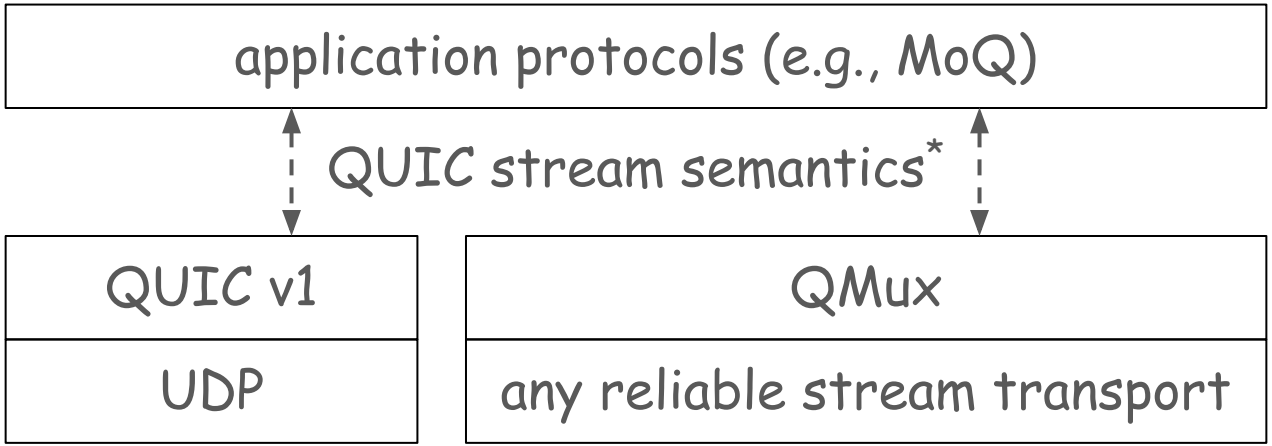
- Provide multiplexing compatible with QUIC.
  - maintain the same semantics for streams
    - e.g., reset, stop-sending
  - support for extensions like DATAGRAMs, reliable resets
- Operate on **any** transport that:
  - provides in-order delivery of bytes in both directions
  - provides guaranteed delivery
  - uses congestion control if necessary
  - provides confidentiality and integrity if necessary

see Section 3.1 of the QoS draft

Out of scope

- Optimize **for** a specific underlying transport e.g.:
  - TCP modifications to avoid head-of-line blocking.
  - Address migration.
  - Use TLS extensions to exchange Transport Parameters.
  - Rely on the transport features to communicate frame boundary.

Layers



*note: same stream model (e.g., establishment, termination, flow control) and support for certain extensions

Mapping QMux to applications

- QMux, like QUIC version 1, provides multiplexed streams etc. to application protocols, therefore…
- application protocols must define their own mapping to QMux:
  - e.g., TLS ALPN "fooU" indicates foo-over-QUICv1, "fooT" indicates foo-over-QMux-over-TLS
- QMux should have applicability statements
  - pointing out the differences from QUIC version 1
    - e.g., head-of-line blocking, lack of migration support

Backup Slides

Design of quic-on-streams-00

- send QUIC v1 frames directly on reliable stream transports
- send Transport Parameters as a special initial frame
  - so that the protocol works on any reliable stream transport
- eliminate ACKs
  - as they are pure cost
- prohibit use of frames and Transport Parameters unrelated to stream state machinery