

QMux

draft-ietf-quic-qmux-00

IETF 125

Kazuho Oku, Lucas Pardue, Jana Iyengar, Eric Kinnear

Current Status

- 13 open issues
 - 6 pull requests to close some of the issues
- Let's start by looking at issues that have pull requests

Two-layer encoding
issue [#21](#) [#24](#) / PR [#26](#)

Issue #21: Two layer encoding

- QUIC uses the packet boundary to determine the end of STREAM and DATAGRAM frames that omit the length field.
- QMux-00 uses a different approach: it uses max_frame_size TP to communicate the length of these frames that omit the length.
- Proposal:
 - Introduce a record layer that carries one or more QUIC frames.
 - Frame encoding / decoding logic becomes identical.

Issue #24: QMux over WebSockets

- When implementing a variant of QMux over WebSockets over HTTPS (i.e., a message-oriented, ordered and bidirectional stream), it helps if QMux has a record layer, because then multiple QUIC frames can be sent using one WebSocket frame.

PR #26: Two layer encoding

- Add a record layer that carries one or more QUIC frames:

```
QMux Record {  
    Size (i),  
    Frames (...)  
}
```

- Benefit:
 - Frame encoding / decoding becomes identical to that of QUIC v1.
 - Becomes easier to port QMux onto message-oriented transports.
- Downside:
 - Slight increase of overhead due to the Size field of QMux Records (2 byte per 16KB record)

Risks of deadlock due to flow control
issue [#9](#) / PR [#27](#)

Issue #9: Risks of deadlock due to flow control

- Underlying transports like TCP have their own flow control. If QMux stops reading from the underlying transport, QMux might enter a deadlock.
- HTTP/2 requires stacks to never stop reading, QMux should follow the same path.

PR #27: Add forward-progress requirements

- Adds text stating that endpoints:
 - “MUST continue reading from the underlying transport even when delivery of STREAM data to the application is temporarily blocked”
 - “MUST NOT couple reads from the underlying transport to application reads on any single QUIC stream”
 - “MAY drop received datagrams when they cannot be promptly delivered to the application”
- Also add a note that:
 - “Continuing to read does not imply unbounded buffering of STREAM data, as the amount of stream data a peer can send is limited by QUIC flow control.”

TLS Profile

issue [#12](#) [#25](#) / PR [#33](#)

Issue #12: TLS Profile

- Current text describes how 0-RTT (early-data) can be used, but does not describe how ALPN should be used.
- Can TLS records be used for representing QMux Records?
 - Pushback: No, because most TLS stacks do not expose record boundary to applications, and RFC 8446 allows TLS stacks to split records as they fit.

Issue #25: QMux Version Negotiation

- How do we negotiate the QMux version to be used?
- What is the ALPN to be used before publication?
- E.g.,
 - “moqt-14-qmux-00” - specifies draft version of MoQ and QMux
 - “moqt-14” - people specifying (or using) moqt-14 would agree on a particular draft version of QMux to be used

PR #33: Must use ALPN

- Follows the lead of RFC 9001 and states:
 - MUST use ALPN unless another mechanism is used for agreeing on an application protocol.
 - Application protocols that use QMux over TLS MUST designate their ALPN identifier and specify that they use QMux.
 - QMux is not itself an application protocol and does not have an ALPN identifier.
- Adds a note to be removed before publication, stating that:
 - For interop using the draft versions of QMux, suggest use of “moqt-14”-style ALPN that implicitly binds to a specific version of QMux

TLS Profile
issue [#18](#) / PR [#28](#)

Issue #18: Put TPs in TLS handshake

- QUIC puts Transport Parameters, including ones carrying initial flow control credits, in the TLS handshake messages, so that endpoints can start sending stream data immediately. QMux as it stands lacks such capability.
- Counter arguments:
 - Some TLS stacks might not provide hooks to send / receive arbitrary TLS extensions.
 - QMux becomes harder to use on non-TLS substrates.
 - The only case that gets blocked is the server when 0-RTT is not used. In all other cases, flow control credits become available at (almost) the same moment as the traffic secrets.

PR #28: Clarify TP timing and 0-RTT

- States “endpoints MUST NOT send frames whose use depends on peer TP until the peer’s QX_TRANSPORT_PARAMETERS frame has been arrived and processed.”
- Then goes on to clarify that, in practice, this restriction affects only the server not using 0-RTT; i.e.,
 - When 0-RTT is not used, the server sends QX_TRANSPORT_PARAMETERS in 0.5 RTT and the client receives them immediately after it obtains the traffic secrets.
 - When 0-RTT is used, remembered TPs are used.

Implicit acks & ping
issue [#22](#) / PR [#23](#)

Issue #22: Do we need some form of acks?

- State machinery of QUIC is driven using ACKs. Clarification is needed how QMux drives the stream state machinery.
- How does idle timeout work without ACKs?
- Solution:
 - Explicitly state that, for the state machinery, frames are considered acknowledged once they are passed to the underlying transport.
 - Regarding idle timeout, QMux does not rely on ACKs, but resets the idle timer when sending or receiving QMux frames.

PR #23: Implicit acks

- “QMux does not use ACK frames; in terms of the QUIC protocol machinery, frames are considered acknowledged once they are passed to the underlying transport for sending.”
- “For the stream state machinery defined in Section 3 of QUIC, references to acknowledgment are interpreted accordingly.”
- “Applications cannot assume that the peer application has consumed data based solely on transport events. ACKs in QUIC version 1 do not provide that guarantee either.”
- “Endpoints reset the idle timer when sending or receiving QMux frames.”

Multipath TCP
issue [#5](#) / PR [#29](#)

Issue #5: migration can be provided by MPTCP

- As to the draft stating: “Use of frames that communicate Connection IDs and those related to path migration is forbidden.” - maybe you can add here: TCPMP provides path migration.

PR #29: refer to MPTCP as one possible substrate

- In the intro, explicitly refer to MPTCP as one possible substrate, stating that QMux ...
 - “allows application protocols built on QUIC to run on transport protocols that provide a single bi-directional, byte-oriented stream, such as TCP or Multipath TCP, combined with TLS providing security.

Issues without pull requests

Issue #4: remove the offset from STREAM frames

- QMux guarantees that STREAM frames belonging to the same stream are never dropped and received in order. Then, the offset field is unneeded; parsing and checking the value is a pure overhead. Should we remove the field?
- Proposal:
 - Close with no action. QMux is a fallback of QUIC, and therefore it makes sense to keep the divergence from QUIC minimal.

Issue #6: Clearly document the downsides

- Downsides of this proposal compared to native QUIC could be more clearly documented.
- TODO: Write Text.

Issue #17: optionally leverage QUIC TLS negotiation

- Use case: QMux over RDMA.
 - Reuse the handshake mechanism and packet (AEAD) encryption provided by QUIC v1, but leverage from the delivery guarantees provided by the underlying transport.
- Questions raised:
 - If the intent is to use RDMA as a transport with ordering guarantee, but have 1:1 mapping between RDMA messages AEAD encryption units, nobody forbids one from developing TLS stacks optimized for that traffic pattern, and running QMux on top of that.
 - If the intent is different, then the design might interfere with QMux assuming the use of in-order and reliable transport.

Issue #30: do we need flow control TPs?

- Do we need flow control Transport Parameters when they are sent as frames? Could we just rely on MAX_* frames?
- Proposal: Close with no action, as the TPs are occasionally used.
 - initial_max_stream_data_* are needed so that the endpoint can start sending data on *new* streams
 - Initial_max_data, initial_streams_(bi|uni) are needed for 0-RTT to work

Issue #31: Generate large codepoints for interop test

- We should.

Issue #34: Maximum size of DATAGRAM frames

- RFC 9221 RECOMMENDS endpoints “to send a value of 65535 in the `max_datagram_frame_size` transport parameter to indicate that this endpoint will accept any DATAGRAM frame that fits inside a QUIC packet.”.
 - This means that, QMux endpoints that follow this would consider 16,381 bytes as the MTU by default.
- Proposal: Explicitly state (at least) that the DATAGRAM frame size is capped by the `max_frame_size` TP.