QUIC
Internet-Draft
Intended status: Standards Track
Expires: 18 May 2026

K. Oku
Fastly
L. Pardue
Cloudflare
J. Iyengar
Netflix
E. Kinnear
Apple

14 November 2025

QMux
draft-opik-quic-qmux-01

Abstract

This document specifies QMux version 1. QMux version 1 provides, over bi-directional streams such as TLS, the same set of stream and datagram operations that applications rely upon in QUIC version 1.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the QUIC Working Group mailing list (quic@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/quic/.

Source for this draft and an issue tracker can be found at https://github.com/kazuho/draft-opik-quic-qmux.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 18 May 2026.

Table of Contents

1. Introduction

QUIC version 1 [QUIC] is a bi-directional, authenticated transport-layer protocol built on top of UDP [UDP]. The protocol provides multiplexed flow-controlled streams without head-of-line blocking as a core service. It also offers low-latency connection establishment and efficient loss recovery.

However, there are downsides to QUIC.

One downside is that QUIC, being based on UDP, is not as universally accessible as TCP [TCP], due to occasionally being blocked by middleboxes.

Another downside is that QUIC is computationally more expensive compared to TLS [TLS13] over TCP. This increased cost is partly because QUIC encrypts each packet, which is smaller than the encryption unit of TLS, leading to more overhead, and partly because UDP is less optimized within computing infrastructures.

Due to these limitations, applications are often served using both QUIC and TCP. QUIC is employed to provide the optimal user experience, while TCP acts as a fallback for ensuring network reachability and computational efficiency as needed.

One such example is HTTP, which has different bindings for QUIC (HTTP/3 [HTTP3]) and TCP (HTTP/2 [HTTP2]). Recently, security concerns have prompted proposals to revise HTTP/2 ([h2-stream-limits]), which has sparked discussions about the costs of maintaining multiple HTTP versions.

Another example is WebTransport, a superset of HTTP. Because HTTP has different bindings for QUIC and TCP, WebTransport defines its own extensions for the two HTTP variants ([webtrans-h3], [webtrans-h2]).

To reduce or eliminate the costs associated with duplicated efforts in providing services on top of both transport protocols, this document specifies a polyfill that allows application protocols built on QUIC to run on transport protocols that provide single bi-directional, byte-oriented stream such as TCP or TLS.

The specified polyfill provides a compatibility layer for the set of operations (i.e., API) required by QUIC, as specified in Section 2.4 and Section 5.3 of [QUIC].

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. The Protocol

QMux can be used on any transport that provides a bi-directional, byte-oriented stream that is ordered and reliable; for details, see Section 3.1.

QUIC frames are sent directly on top of the transport.

The frames are not encrypted. It is the task of the transport (e.g., TLS) to provide confidentially and integrity.

QUIC packet headers are not used.

For exchanging the Transport Parameters, a new frame called QX_TRANSPORT_PARAMETERS frame is defined.

3.1. Properties of Underlying Transport

QMux is designed to work on top of transport layer protocols that provide the following capabilities:

In-order delivery of bytes in both direction: Underlying transport provides a byte-oriented and bi-directional stream that deliver the bytes in order; i.e., bytes that were sent in one order become available to the receiving side in the same order.

Guaranteed delivery: If the transport runs on top of a lossy network, that transport recovers the bytes lost; e.g., by retransmitting them. This requires buffering and reassembly, in order to achieve the first bullet point (in-order delivery).

Congestion control: When used on a shared network, the transport is congestion controlled. Implementations of QMux simply write outgoing frames to the transport when that transport permits.

Confidentially and Integrity: Unless used upon endpoints between which tampering or monitoring is a non-concern, the transport provides confidentially and integrity protection.

TLS over TCP provides all these capabilities.

UNIX sockets are an example that provides only the first two. Congestion control is not employed, as UNIX sockets do not face a shared bottleneck. Confidentiality and integrity protection are deemed unnecessary in environments where the operating system is trusted.

4. QUIC Frames

In QMux, the following QUIC frames can be used, as if they were sent or received in the application packet number space:

* PADDING
* RESET_STREAM
* STOP_SENDING
* STREAM
* MAX_DATA
* MAX_STREAM_DATA
* MAX_STREAMS
* DATA_BLOCKED
* STREAM_DATA_BLOCKED
* STREAMS_BLOCKED
* CONNECTION_CLOSE

The frame formats are identical to those in QUIC version 1. Likewise, the meaning and requirements for the use of these frames are consistent with QUIC version 1, with the exception to the specific changes made to the STREAM frames, as detailed in Section 4.1.

Use of other frames defined in QUIC version 1 is prohibited for various reasons. ACK frames are not used because the underlying transport guarantees delivery. Frames related to the cryptographic handshake are not used because an underlying security layer can provide equivalent features. Use of frames that communicate Connection IDs and those related to path migration is forbidden.

The full list of prohibited frames is:

* PING
* ACK
* CRYPTO
* NEW_TOKEN
* NEW_CONNECTION_ID
* RETIRE_CONNECTION_ID
* PATH_CHALLENGE
* PATH_REPONSE
* HANDSHAKE_DONE

Endpoints MUST NOT send prohibited frames. If an endpoint receives one it MUST close the connection with an error of type FRAME_ENCODING_ERROR.

4.1. STREAM Frames

While the frame format remains unchanged, there are two differences in the handling of STREAM frames between QUIC version 1 and QMux.

4.1.1. STREAM Frames without the Length Field

In QMux, when a STREAM frame that omits the Length field is used, the size of that STREAM frame is determined by the maximum frame size, as regulated by the max_frame_size Transport Parameter (Section 5.2).

This behavior contrasts with that of QUIC version 1, where the absence of the Length field implies that the STREAM frame extends to the end of the QUIC packet payload.

This variation arises due to the characteristics of the underlying transports of QMux, which may not have, or provide visibility into, the packet boundaries.

4.1.2. Ordering of STREAM frames

For each stream being sent, senders MUST send stream payload in order.

When receiving a STREAM frame that carries a payload not immediately following the payload of the previous STREAM frame for the same Stream ID, receivers MUST close connection with an error of type PROTOCOL_VIOLATION_ERROR.

This change from QUIC version 1 eliminates the need for implementations to buffer and reassemble the stream payload. As a result, the payload being received can be directly passed to the application as it is read from the transport. This efficiency is due to the underlying transport's guarantee of in-order delivery.

These changes do not impact the senders' capability to interleave STREAM frames from multiple streams.

4.2. QX_TRANSPORT_PARAMETERS Frames

In QMux, Transport Parameters are exchanged as frames.

QX_TRANSPORT_PARAMETERS frames are formatted as shown in Figure 1.

```
QX_TRANSPORT_PARAMETERS Frame {
  Type (i) = 0x3f5153300d0a0d0a,
  Length (i),
  Transport Parameters (..),
}
```

Figure 1: QX_TRANSPORT_PARAMETERS Frame Format

QX_TRANSPORT_PARAMETERS frames contain the following fields:

Length: A variable-length integer specifying the length of the Transport Parameters field in this QX_TRANSPORT_PARAMETERS frame.

Transport Parameters: The Transport Parameters. The encoding of the payload is as defined in Section 18 of [QUIC].

The QX_TRANSPORT_PARAMETERS frame is the first frame sent by endpoints. Endpoints MUST send the QX_TRANSPORT_PARAMETERS frame as soon as the underlying transport becomes available. Note neither endpoint needs to wait for the peer's Transport Parameters before sending its own, as Transport Parameters are a unilateral declaration of an endpoint's capabilities (Section 7.4 of [QUIC]).

If the first frame being received by an endpoint is not a QX_TRANSPORT_PARAMETERS frame, the endpoint MUST close the connection with an error of type TRANSPORT_PARAMETER_ERROR.

The frame type (0x3f5153300d0a0d0a; "\xffQMX\r\n\r\n" on wire) has been chosen so that it can be used to disambiguate QMux from HTTP/1.1 [HTTP1] and HTTP/2.

4.3. QX_PING Frames

In QMux, QX_PING frames allow endpoints to test peer reachability above the underlying transport.

QX_PING frames are formatted as shown in Figure 2.

```
QX_PING Frame {
  Type (i) = 0xTBD..0xTBD+1,
  Sequence Number (i),
}
```

Figure 2: QX_PING Frame Format

Type 0xTBD is used for sending a ping (i.e., request the peer to respond). Type 0xTBD+1 is used in response.

QX_PING frames contain the following fields:

Sequence Number: A variable-length integer used to identify the ping.

When sending QX_PING frames of type 0xTBD, endpoints MUST send monotonically increasing values in the Sequence Number field, since that allows the endpoints to identify to which ping the peer has responded.

When sending QX_PING frames of type 0xTBD+1 in response, endpoints MUST echo the Sequence Number that they received.

When receiving multiple QX_PING frames of type 0xTBD before having the chance to respond, an endpoint MAY only respond with one QX_PING frame of type 0xTBD+1 carrying the largest Sequence Number that the endpoint has received.

5. Transport Parameters

QMux uses a subset of Transport Parameters defined in QUIC version 1. Also, one new Transport Parameter specific to QMux is defined.

5.1. Permitted and Forbidden Transport Parameters

In QMux, use of the following Transport Parameters is allowed.

* max_idle_timeout
* initial_max_data
* initial_max_stream_data_bidi_local
* initial_max_stream_data_bidi_remote
* initial_max_stream_data_uni
* initial_max_streams_bidi
* initial_max_streams_uni

The definition of these Transport Parameters are unchanged.

Use of other Transport Parameters defined in QUIC version 1 is prohibited. When an endpoint receives one of the prohibited Transport Parameters, the endpoint MUST close the connection with an error of type TRANSPORT_PARAMETER_ERROR.

Endpoints MUST NOT send Transport Parameters that extend QUIC version 1, unless they are specified to be compatible with QMux.

When receiving Transport Parameters not defined in QUIC version 1, receivers MUST ignore them unless they are specified to be usable on QMux.

5.2. max_frame_size Transport Parameter

The max_frame_size Transport Parameter (0xTBD) is a variable-length integer specifying the maximum size of the QUIC frame that the peer can send, in the unit of bytes.

The initial value of the max_frame_size Transport Parameter is 16384.

By sending the Transport Parameter, the maximum frame size can only be increased. When receiving a value below the initial value, receivers MUST close the connection with an error of type TRANSPORT_PARAMETER_ERROR.

Endpoints MUST NOT send QUIC frames that exceed the maximum declared by the peer.

When receiving QUIC frames that exceed the declared maximum, receivers MUST close the connection with an error of type FRAME_ENCODING_ERROR.

6. Closing the Connection

As is with QUIC version 1, a connection can be closed either by a CONNECTION_CLOSE frame or by an idle timeout.

Unlike QUIC version 1, there is no draining period; once an endpoint sends or receives the CONNECTION_CLOSE frame or reaches the idle timeout, all the resources allocated for the Service are freed and the underlying transport is closed immediately.

7. Using 0-RTT

TLS 1.3 introduced the concept of early data (also knows as 0-RTT data).

When using QMux on top of TLS that supports early data, clients MAY use early data when resuming a connection, by reusing certain Transport Parameters as defined in Section 7.4.1 of [QUIC].

Similarly, when accepting early data, the servers MUST send Transport Parameters that obey to the restrictions defined in Section 7.4.1 of [QUIC].

8. Extensions

Not all the extensions of QUIC version 1 can be used. Each extension have to define its mapping for QMux, or explicitly allow the use; see Section 5.1.

As is the case with QUIC version 1, use of extension frames have to be negotiated before use; see Section 19.21 of [QUIC].

This specification defines the mapping of the Unreliable Datagram Extension.

8.1. Unreliable Datagram Extension

The use of the Unreliable Datagram Extension [QUIC_DATAGRAM] is permitted, with one modification:

Similar to STREAM frames, when employing DATAGRAM frames of type 0x30 (i.e., DATAGRAM frames without the Length field), their size is determined by the max_frame_size Transport Parameter (Section 5.2).

Apart from this, the encoding and semantics of the Unreliable Datagram Extension remain unchanged. The use of the extension is negotiated via the Transport Parameters.

As discussed in Section 5 of [QUIC_DATAGRAM], senders can drop DATAGRAM frames if the transport is blocked by flow or congestion control.

9. Version Agility

As new versions of QUIC are specified, there may be a desire to define their reliable-byte-stream counterparts as different versions of QMux, and to provide ways of negotiating the version to be used.

QUIC establishes connections using packets carrying an explicit version number. Using that field, Version-Independent Properties of QUIC [QUIC-INVARIANTS] defines a version negotiation mechanism that involves a retry. Compatible Version Negotiation for QUIC [QUIC-CVN] defines another negotiation mechanism for switching between compatible versions during connection establishment without retrying.

By contrast, QMux does not establish connections by itself; the connections are set up by the underlying substrate, and QMux exchanges only the Transport Parameters after they are established.

Due to these differences, the negotiation mechanisms used by QUIC and QMux will differ.

This section explores some options that future versions of QMux might employ for version negotiation and upgrade.

9.1. Negotiation Using ALPN

When a new QUIC version that provides a different interface to applications is specified, application protocols developed for that version might be assigned a new identifier for the TLS Application-Layer Protocol Negotiation (ALPN) extension [ALPN].

Similarly, when TLS is the underlying transport, application protocols built on top of the QMux counterparts of such QUIC versions can rely on ALPN to negotiate both the application protocol and the underlying QMux version.

When TLS is not the underlying transport, endpoints can use the first 8 bytes exchanged on the transport (i.e., the type field of the QX_TRANSPORT_PARAMETERS frame in the encoded form) to identify whether QMux is in use.

[TODO: discuss how endpoints should behave when the first 8 bytes received are not QX_TRANSPORT_PARAMETERS.]

9.2. In-band Upgrade

A new version of QMux might first start communication using QMux version 1 and then switch versions in-band during the session. The advantage of this approach is that, even when TLS is not in use, no additional round-trip is incurred for version negotiation.

While QMux version 1 does not specify a concrete method, new versions might use the version_information Transport Parameter (Section 3 of [QUIC-CVN]) to discover supported versions and coordinate the switch.

10. Implementation Considerations

Similar to HTTP/3 with Extensible Priorities [HTTP_PRIORITY], application protocols using QUIC may employ stream multiplexing along with a system to tune the delivery sequence of QUIC streams.

To alternate between QUIC streams of varying priorities in a timely manner, it is advisable for QMux implementations to avoid creating deep buffers holding QUIC frames. Instead, endpoints should wait for the transport layer to be ready for writing. Upon becoming writable, they should write QUIC frames according to the latest prioritization signals.

Additionally, implementations may consider monitoring or adjusting the flow and congestion control parameters of the underlying transport. This approach aims to minimize data buffering within the transport layer before transmission. However, improper adjustment of these parameters could potentially lead to lower throughput.

11. Security Considerations

TODO Security

12. IANA Considerations

TODO

13. References

13.1. Normative References

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[QUIC_DATAGRAM]
Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", RFC 9221, DOI 10.17487/RFC9221, March 2022, <https://www.rfc-editor.org/rfc/rfc9221>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

13.2. Informative References

[ALPN] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/rfc/rfc7301>.

[h2-stream-limits]
Thomson, M. and L. Pardue, "Using HTTP/3 Stream Limits in HTTP/2", Work in Progress, Internet-Draft, draft-thomson-httpbis-h2-stream-limits-00, 6 November 2023, <https://datatracker.ietf.org/doc/html/draft-thomson-httpbis-h2-stream-limits-00>.

[HTTP1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/rfc/rfc9112>.

[HTTP2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/rfc/rfc9113>.

[HTTP3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[HTTP_PRIORITY]
Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", RFC 9218, DOI 10.17487/RFC9218, June 2022, <https://www.rfc-editor.org/rfc/rfc9218>.

[QUIC-CVN] Schinazi, D. and E. Rescorla, "Compatible Version Negotiation for QUIC", RFC 9368, DOI 10.17487/RFC9368, May 2023, <https://www.rfc-editor.org/rfc/rfc9368>.

[QUIC-INVARIANTS] Thomson, M., "Version-Independent Properties of QUIC", RFC 8999, DOI 10.17487/RFC8999, May 2021, <https://www.rfc-editor.org/rfc/rfc8999>.

[TCP] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/rfc/rfc9293>.

[UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

[webtrans-h2] Frindell, A., Kinnear, E., Pauly, T., Thomson, M., Vasiliev, V., and G. Xie, "WebTransport over HTTP/2", Work in Progress, Internet-Draft, draft-ietf-webtrans-http2-13, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http2-13>.

[webtrans-h3] Frindell, A., Kinnear, E., and V. Vasiliev, "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-ietf-webtrans-http3-14, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http3-14>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Kazuho Oku
Fastly
Email: kazuhooku@gmail.com

Lucas Pardue
Cloudflare
Email: lucas@lucaspardue.com

Jana Iyengar
Netflix
Email: jri.ietf@gmail.com

Eric Kinnear
Apple
Email: ekinnear@apple.com