Media Over QUIC
Internet-Draft
Intended status: Standards Track
Expires: 1 September 2026

S. Nandakumar
C. Jennings
Cisco

28 February 2026

Media over QUIC Transport (MOQT) over QMux
draft-nandakumar-moq-qmux-moqt-00

Abstract

This document specifies how Media over QUIC Transport (MOQT) can operate over QMux, enabling MOQT applications to function over reliable byte-oriented streams such as TCP with TLS. By utilizing QMux as an intermediate layer, MOQT deployments gain the ability to fall back to TCP-based transport when UDP is blocked or unreliable, while maintaining API compatibility with QUIC-native implementations. This document analyzes the benefits and trade-offs of this approach and provides implementation guidance for deploying MOQT over QMux.

About This Document

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

The latest revision of this draft can be found at https://ietf-wg-moq.github.io/draft-moqt-qmux-transport/draft-moqt-qmux-transport.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-nandakumar-moq-qmux-moqt/.

Discussion of this document takes place on the Media Over QUIC Working Group mailing list (mailto:moq@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/moq/. Subscribe at https://www.ietf.org/mailman/listinfo/moq/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-moq/draft-moqt-qmux-transport.

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 1 September 2026.

Table of Contents

1. Introduction

Media over QUIC Transport (MOQT) [MoQ-TRANSPORT] is designed to leverage QUIC's [RFC9000] advanced features for efficient media delivery, including stream multiplexing, low-latency connection establishment, and built-in encryption. However, real-world deployments face challenges where UDP traffic may be blocked, rate-limited, or unreliable due to middlebox interference, enterprise firewalls, or network policies.

QMux [QMUX] addresses this challenge by providing a "polyfill" that enables QUIC-based applications to operate over reliable, bidirectional byte-oriented streams such as TLS over TCP. QMux preserves QUIC's multiplexing semantics while delegating reliability and encryption to the underlying transport.

This document specifies how MOQT can utilize QMux as an alternative transport substrate, enabling:

* Fallback capability when QUIC/UDP is unavailable

* Single codebase for both QUIC and TCP deployments

* Broader network compatibility for media streaming applications

* Graceful degradation in restrictive network environments

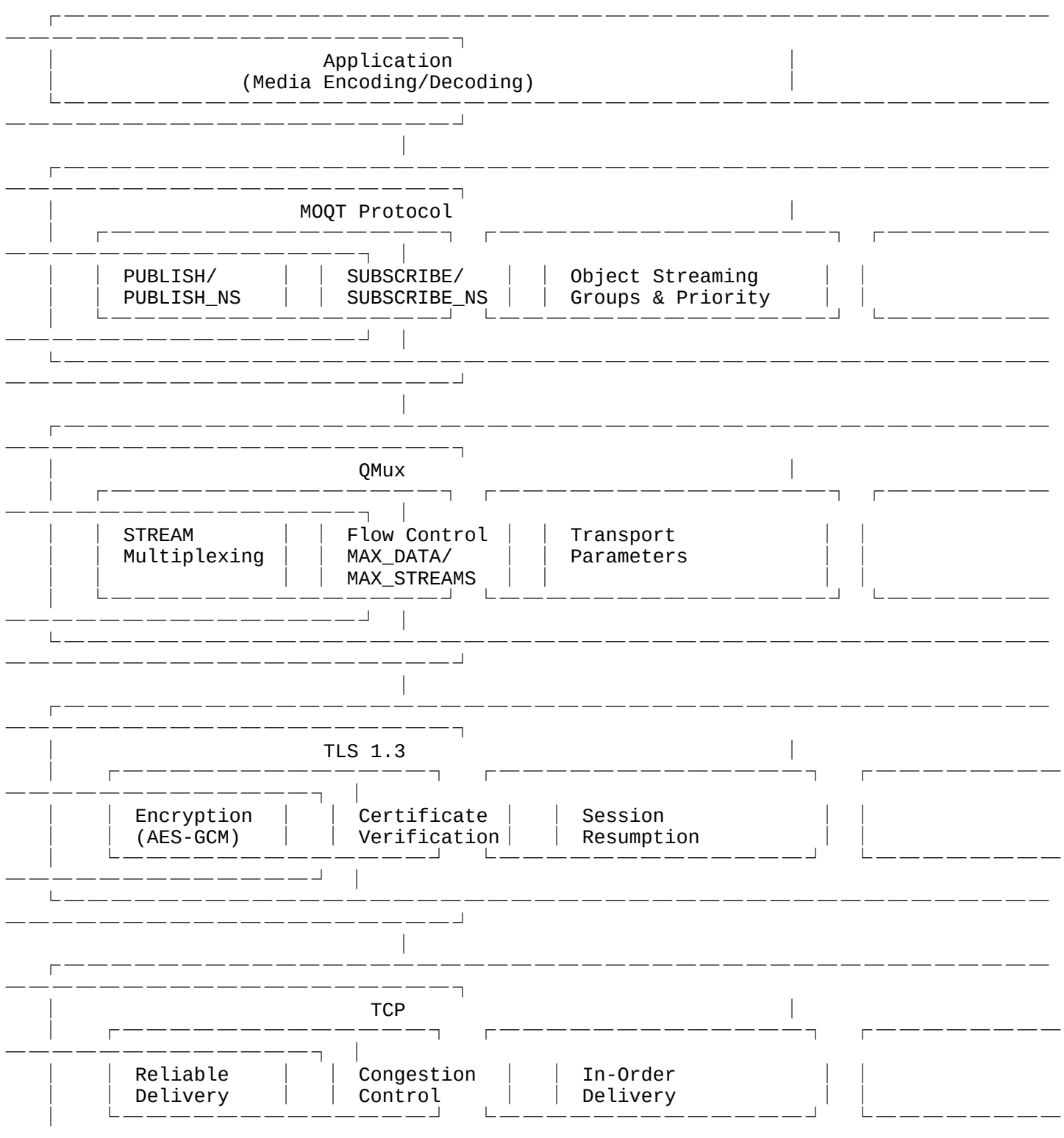
1.1. Requirements Language

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.

2. Architecture Overview

2.1. Protocol Stack

When MOQT operates over QMux, the protocol stack is organized as follows:



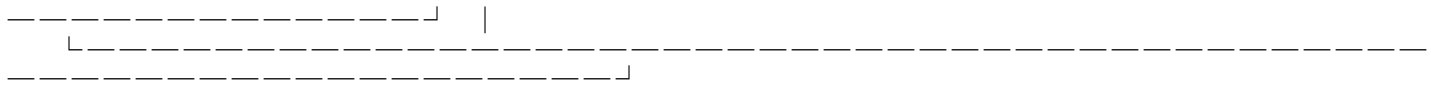


Figure 1: MOQT over QMux Protocol Stack

2.2. Transport Selection

Implementations supporting both native QUIC and QMux transports SHOULD implement transport selection logic to choose the optimal path:

Transport Selection Flow:

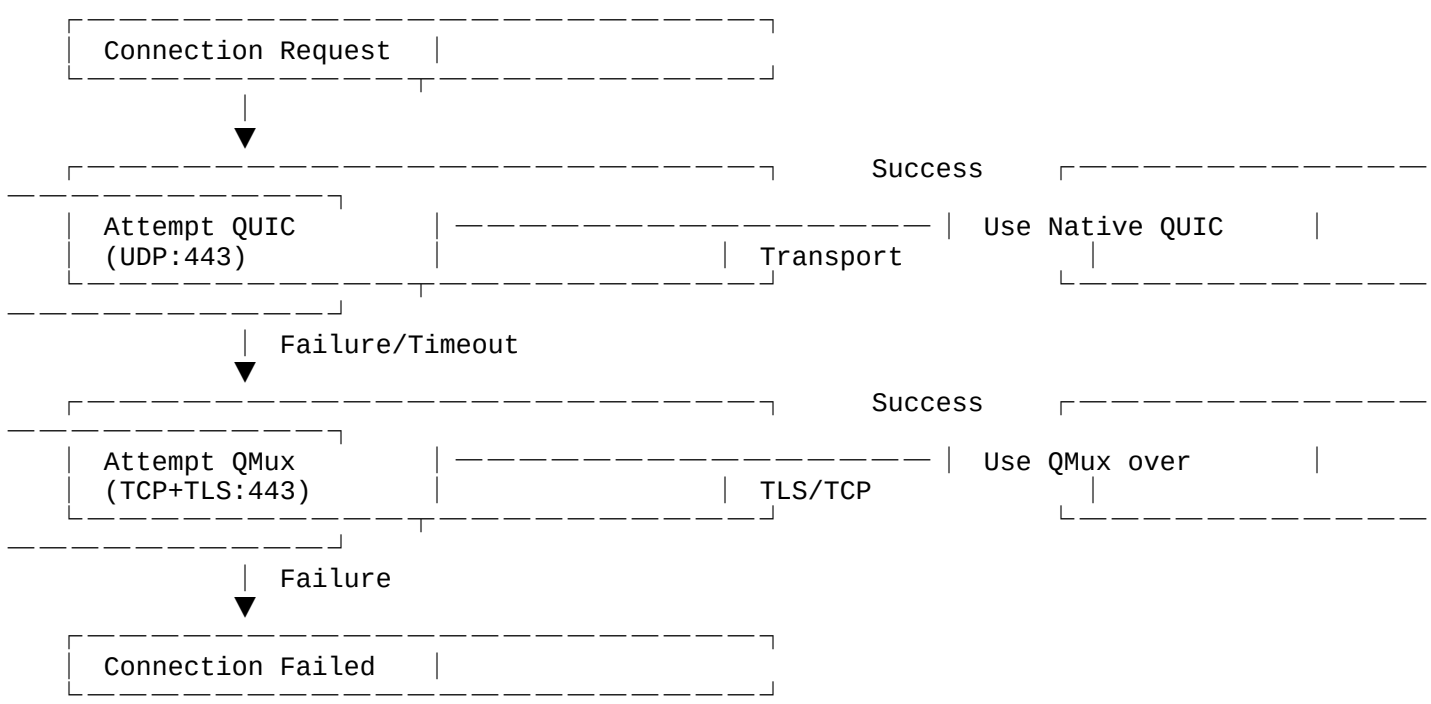


Figure 2: Transport Selection Flow

2.2.1. Happy Eyeballs for MOQT

Implementations MAY use a Happy Eyeballs-style algorithm [RFC8305] to race QUIC and QMux connections simultaneously:

1. Initiate QUIC connection attempt
2. After a short delay (e.g., 100ms), initiate QMux connection in parallel
3. Use the first connection that succeeds
4. Cancel remaining connection attempts

This approach minimizes connection latency while preferring QUIC when available.

3. Connection Establishment

3.1. QMux Handshake

When establishing MOQT over QMux, the connection sequence is:

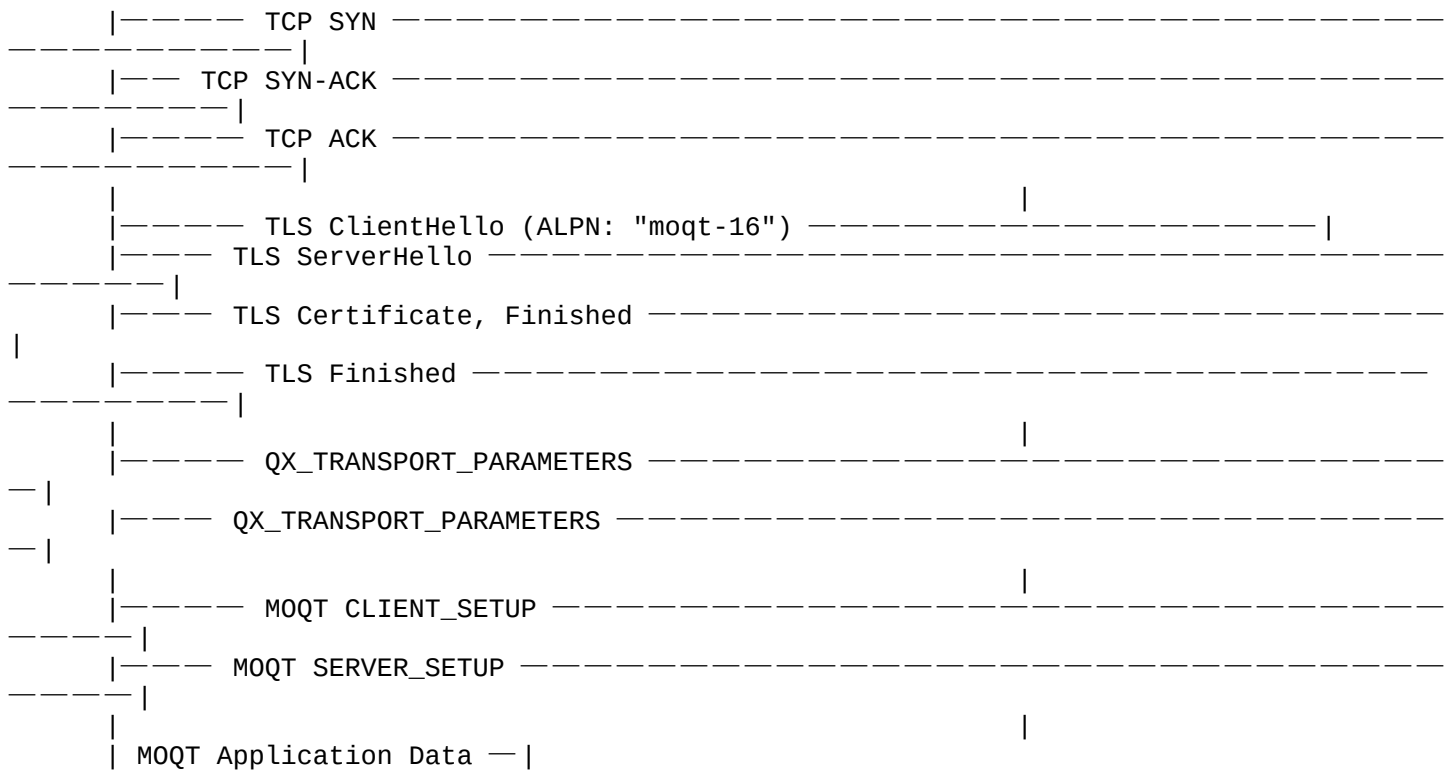


Figure 3: Connection Establishment Sequence

3.2. ALPN Identification

MOQT over QMux uses the same protocol identification as native MOQT over QUIC, as defined in [MoQ-TRANSPORT]. This document does not define a new ALPN protocol identifier.

For TLS connections carrying QMux, implementations MUST use the MOQT ALPN value corresponding to the MOQT version in use:

* For the final specification: moqt
* For draft versions: moqt-XX where XX is the draft number (e.g., moqt-16 for draft-ietf-moq-transport-16)

When MOQT operates over WebTransport [WebTransport], the WT-Available-Protocols mechanism is used instead of ALPN, following the same version identification scheme.

The underlying transport (native QUIC vs QMux over TCP) is distinguished by the connection type itself, not by the ALPN value. This allows:

* Consistent version negotiation across all transport substrates
* Simplified client implementation with a single MOQT version identifier
* Seamless fallback from QUIC to QMux without protocol identifier changes

3.3. QMux Transport Parameters

QMux [QMUX] inherits transport parameters from QUIC v1 [RFC9000], which does not specify default values for most parameters (absent parameters default to 0, disabling the corresponding feature). The only exception is max_frame_size, which QMux defines with an initial value of 16384 bytes.

The following transport parameter values are RECOMMENDED for MOQT deployments over QMux. These values are chosen to support typical media streaming workloads and are not mandated by either QMux or QUIC v1:

Parameter	Recommended Value	Rationale
max_idle_timeout	30000 ms	Common QUIC implementation default; balances resource cleanup with connection stability
initial_max_data	1048576	1 MiB; sufficient for buffering multiple media objects at connection level
initial_max_stream_data_bidi_local	262144	256 KiB; adequate per-stream buffer for control messages
initial_max_stream_data_bidi_remote	262144	256 KiB; symmetric with local for bidirectional streams
initial_max_streams_bidi	100	Supports concurrent request-response patterns
initial_max_streams_uni	100	Supports multi-track media delivery (audio, video, multiple qualities)
max_frame_size	16384	QMux initial value; matches HTTP/2 default MAX_FRAME_SIZE

Table 1

Implementations MAY adjust these values based on deployment requirements, expected media bitrates, and network conditions.

4. MOQT Mapping to QMux

4.1. Stream Mapping

MOQT streams map directly to QMux streams. The stream semantics are preserved, with QMux providing the multiplexing layer:

MOQT Stream Type	QMux Stream Type	Purpose
Control Stream	Bidirectional Stream 0	SETUP, SUBSCRIBE, PUBLISH messages
Data Streams	Unidirectional Streams	Object delivery
Request Streams	Bidirectional Streams	Request-response patterns

Table 2

4.2. Control Stream

The MOQT control stream MUST use QMux bidirectional stream ID 0. All MOQT control messages (CLIENT_SETUP, SERVER_SETUP, SUBSCRIBE, PUBLISH, etc.) are exchanged on this stream.

4.3. Object Delivery

MOQT objects are delivered over QMux unidirectional streams following the same patterns as native QUIC:

QMux Unidirectional Stream for MOQT Object:

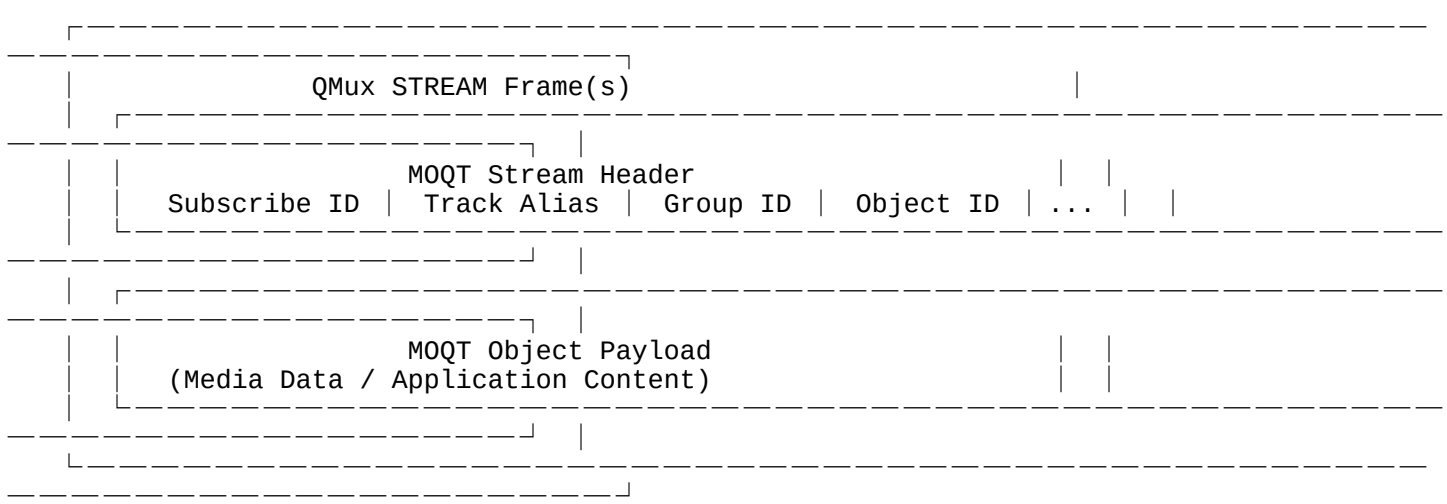


Figure 4: MOQT Object over QMux Stream

4.4. Flow Control

QMux provides flow control at both connection and stream levels, mapping to MOQT's requirements:

* Connection-level: MAX_DATA frames limit total data across all streams
* Stream-level: MAX_STREAM_DATA frames limit per-stream data
* Stream count: MAX_STREAMS frames limit concurrent streams

MOQT implementations MUST respect QMux flow control signals and implement appropriate backpressure mechanisms.

5. Benefits of MOQT over QMux

5.1. Universal Network Accessibility

The primary benefit of MOQT over QMux is network accessibility:

* *Firewall Traversal*: TCP port 443 is rarely blocked, unlike UDP
* *Middlebox Compatibility*: TCP/TLS is well-understood by NATs, proxies, and load balancers
* *Enterprise Networks*: Many corporate networks restrict UDP traffic
* *Mobile Networks*: Some carriers rate-limit or block UDP

5.2. Code Consolidation

QMux enables a single MOQT implementation to serve both transport paths:

Implementation Architecture:

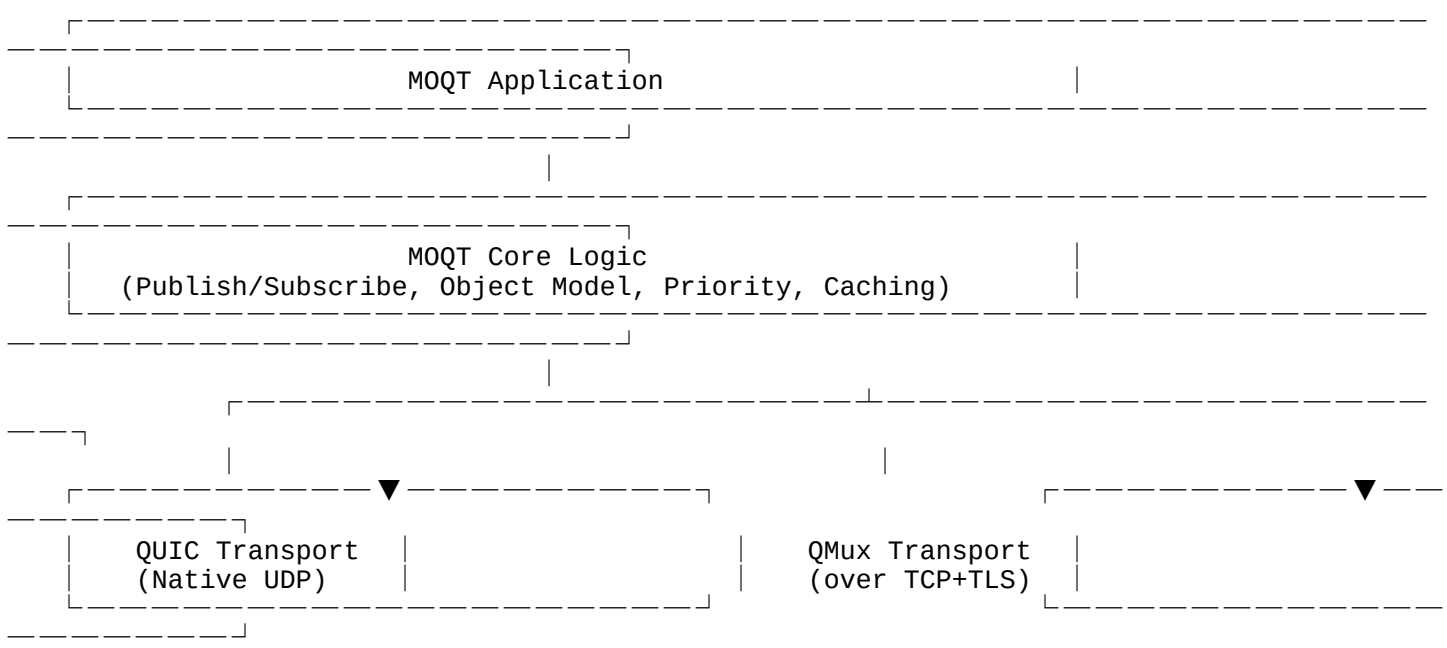


Figure 5: Unified Implementation Architecture

This architecture provides:

* Reduced development and maintenance effort
* Consistent behavior across transport paths
* Simplified testing and certification
* Single API surface for applications

5.3. Existing Infrastructure Leverage

MOQT over QMux can utilize existing TCP/TLS infrastructure:

* *Load Balancers*: Standard L4/L7 load balancers work with TCP
* *TLS Termination*: Hardware TLS accelerators can be used
* *CDN Integration*: Existing TCP-based CDNs can front MOQT traffic
* *Monitoring Tools*: TCP traffic analysis tools apply unchanged

5.4. 0-RTT Connection Resumption

When operating over TLS 1.3, MOQT/QMux supports 0-RTT session resumption:

0-RTT Connection Resumption:

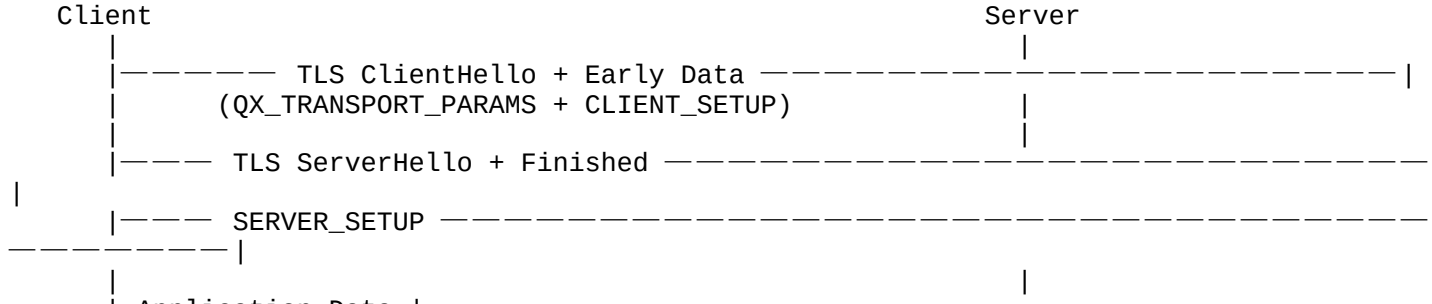


Figure 6: 0-RTT Connection Resumption

Note: 0-RTT data is subject to replay attack considerations per [RFC8446]. Implementations MUST follow TLS 1.3 guidance on 0-RTT security.

6. Trade-offs and Limitations

Operating MOQT over QMux involves several trade-offs compared to native QUIC:

* *Head-of-Line Blocking*: TCP packet loss blocks all multiplexed streams until retransmission completes, unlike QUIC where streams are independent. This impacts live streaming, multi-track media, and interactive applications. Mitigation strategies include priority-based multiplexing, reduced buffering, and application-level adaptation.

* *No Connection Migration*: QUIC's seamless IP address change and mobile handoff capabilities are unavailable. Applications requiring mobility support should prefer native QUIC or implement application-layer session resumption.

* *Additional Latency*: Initial connection adds 1-2 RTT overhead due to TCP and TLS handshakes. Packet loss recovery is also slower with connection-wide rather than per-stream congestion response.

* *Unreliable Datagram Limitations*: True unreliable delivery is impossible over TCP. Applications requiring unreliable datagrams (e.g., low-latency audio with acceptable loss) should use native QUIC.

* *Computational Overhead*: QMux adds frame encoding/decoding, stream state management, and flow control processing, though this is typically small compared to TLS and application-layer processing.

7. Implementation Considerations

Implementations SHOULD minimize buffering by keeping TCP send buffers appropriately sized and reacting promptly to flow control signals. MOQT priority semantics SHOULD be preserved by mapping Publisher Priority values to QMux stream scheduling, servicing higher-priority streams first. QMux connection errors such as CONNECTION_CLOSE terminate the MOQT session, while STREAM_RESET cancels the affected track or subscription. Implementations supporting both transports MUST ensure identical MOQT behavior, consistent object delivery

semantics, and equivalent authentication regardless of the underlying transport.

8. Security Considerations

TODO

9. IANA Considerations

TODO

10. References

10.1. Normative References

[MoQ-TRANSPORT] Curley, L., Pugin, K., Nandakumar, S., Vasiliev, V., and I. Swett, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-16, January 2026, <https://datatracker.ietf.org/doc/draft-ietf-moq-transport/>.

[QMUX] Oku, K. and J. Iyengar, "QMux: A polyfill to run QUIC applications on reliable streams", Work in Progress, Internet-Draft, draft-opik-quic-qmux-00, 2025, <https://datatracker.ietf.org/doc/draft-opik-quic-qmux/>.

[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>.

[RFC8446] 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>.

[RFC9000] 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>.

10.2. Informative References

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/rfc/rfc8305>.

[WebTransport] Vasiliev, V., "The WebTransport Protocol Framework", RFC 9297, August 2023, <https://www.rfc-editor.org/rfc/rfc9297>.

Acknowledgments

The authors would like to thank the IETF MOQT working group for their ongoing work on media transport protocols, and the authors of the QMux specification for enabling QUIC applications to operate over TCP.

Authors' Addresses

Suhas Nandakumar
Cisco
Email: snandaku@cisco.com

Cullen Jennings
Cisco
Email: fluffy@cisco.com