Media Over QUIC
Internet-Draft
Intended status: Standards Track
Expires: 18 September 2026

M. Jiang
Y. Liu
Alibaba Inc.
R. Wu
Ant Group.

17 March 2026

MoQ Multimodal Feedback
draft-jiang-moq-multimodal-feedback-00

Abstract

This document defines an extension to Media over QUIC Transport (MOQT) that enables MoQ receivers to report delivery quality information for media Objects to senders. The MoQ layer synthesizes MMF feedback and local congestion control (CC) output to compute control decisions such as bitrate, frame rate, and pacing, and inform the CC algorithm module via a cross-layer control interface. This mechanism reuses the MOQT Track/Object data model without introducing new control message types. While QUIC ACK and reception timestamp extensions continue to provide per-packet CC signals; this mechanism adds per-Object media semantic feedback when the MMF extension is negotiated and enabled.

Discussion Venues

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

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

Source for this draft and an issue tracker can be found at https://github.com/Yanmei-Liu/draft-moq-multimodal-feedback.

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/.

Copyright Notice

Table of Contents

1. Introduction

Media over QUIC Transport (MOQT, [MoQTransport]) is a QUIC-based publish/subscribe media transport framework. In low-latency interactive scenarios, senders need to obtain media delivery quality information from peer to adjust sending strategies. Adjustments occur at two layers:

* *Application layer:* Encoders adjust bitrate/frame rate, inference systems adjust generation rate, and ABR switches Tracks.

* *Transport layer:* Congestion control (CC) algorithms adjust cwnd/ pacing rate.

QUIC Transport layer feedback (QUIC-ACK, receive timestamps [quic-receive-ts]) only covers the transport layer, leaving blind spots at the MoQ semantic level (see Section 2 for details). This document defines a MoQ-layer feedback mechanism that provides per-Object media semantic feedback to the application layer. The MoQ layer also serves as the control layer for CC algorithms, synthesizing MMF signals and local transport state to issue control commands such as pacing rate and pacing gain to CC (see Section 3.3 for details).

1.1. 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.

The following terms are used throughout this document:

* *MOQT:* Media over QUIC Transport, following [MoQTransport].

* *Object:* The smallest semantic unit of data delivery in MOQT.

* *Feedback Track:* A MoQ Track that carries MMF feedback.

* *MMF (MoQ Multimodal Feedback):* A receiver report at the MoQ layer containing per-Object delivery status.

* *Cross-Layer Control Interface:* A mechanism for the MoQ layer to issue control commands to CC algorithms; the specific form is implementation- defined.

2. Motivation

QUIC layer feedback mechanisms (ACK, Receive Timestamps) operate at the packet level, leaving the following blind spots at the MoQ layer:

Blind Spot	Description
Object Semantics	QUIC ACK confirms packets but cannot perceive frame integrity, type, or deadline
Frame-Level Timing	QUIC ACK provides packet-level delay but cannot provide inter-Object arrival timing
Playback Progress	QUIC layer cannot know peer's playback buffer level or application-layer consumption state

Table 1

This document defines a MoQ-layer feedback mechanism (MMF) that supplements Object-level semantic signals which are unavailable at the QUIC layer. MMF serves both the application layer (per-Object delivery status) and CC algorithms (aggregate statistics injection). MMF enables senders to reduce sending quality before the peer's playback buffer is depleted, rather than passively responding after stuttering occurs.

3. Architecture

3.1. Three-Layer Architecture: Application / MoQ / Transport

This mechanism adopts a three-layer architecture: the application layer serves as the media source, the transport layer implements CC algorithms, and the MoQ layer resides in between, responsible for semantic understanding and signal distribution.

3.1.1. Application Layer: Media Sources

MMF does not restrict the type of application-layer media sources. Two typical media source types are:

Media Source Type	Characteristics	MMF-Driven Adjustments
Traditional Media (encoder/camera/ live streaming)	Unidirectional output, controllable frame rate/ bitrate	Unidirectional adjustment: encoding bitrate, frame rate, resolution, ABR Track switching
AI Inference Pipeline (multimodal inference engine)	Bidirectional interaction, tunable generation parameters	Bidirectional adjustment: above general adjustments + inference parameters (chunk_size, flush strategy)

Table 2

Both media source types share the same MMF format; the difference lies only in the adjustable actions that can be taken after consuming MMF.

3.1.2. MoQ Layer: Semantic Hub

The MoQ layer assumes two responsibilities in the three-layer architecture:

* *Semantic Translation:* The MoQ Track/Object/Group model provides feedback with frame-level semantics. MMF reports Object delivery status (complete, expired, lost) rather than packet-level status.

* *Control Hub:* The MoQ layer synthesizes MMF feedback and local transport state, driving application-layer adjustments upward via callbacks (encoding bitrate, frame rate) and instructing CC to adjust sending rate downward via the cross-layer control interface (pacing_rate, pacing_gain). The MoQ layer can also directly execute certain adjustments (ABR Track switching, Object transmission frequency control) without requiring application-layer media source cooperation.

The Feedback Track is a normal MoQ Track, at the same level as audio/video/text Tracks. This mechanism introduces no new QUIC frames or MOQT control messages.

3.1.3. Transport Layer: CC Algorithms

CC algorithms (BBR, GCC, etc.) are responsible for congestion detection and bandwidth estimation based on local QUIC ACK and receive-ts. In real-time media scenarios, the MoQ layer issues control commands (pacing_rate, pacing_gain) to CC via the cross-layer control interface (Section 3.3); CC algorithms SHOULD execute these commands. The MoQ layer makes decisions by synthesizing three sources of information: MMF feedback, local CC output (BWE), and frame-level statistics (Section 8.1), achieving higher information completeness than CC algorithms alone. An integration example is provided in Section 8.4.

3.2. Dual-Layer Feedback Model: QUIC receive-ts and MMF

CC algorithms rely on per-packet delay signals for bandwidth estimation and congestion detection; this signal is provided by QUIC receive-ts ([quic-receive-ts]). MMF operates at the MoQ layer, forming a dual-layer feedback which could cooperate with QUIC receive-ts. Both mechanisms can be enabled simultaneously in implementations.

Feedback Layer	Granularity	Hop	CC Role	Application Layer Role
QUIC receive-ts	per-packet (~us)	QUIC connection layer	Primary signal (delay gradient, BW est.)	None
MoQ MMF	per-Object (~ms)	Client/ Server direct	MoQ layer synthesizes and issues control commands	Primary signal (bitrate/ frame rate/ABR/ inference parameter adjustment)

Table 3

The two layers cover different granularities:

QUIC receive-ts provides per-packet inter-arrival delta, enabling CC algorithms to perform delay-based congestion detection and bandwidth estimation.

MMF supplements additional signals:

* *Per-Object Signals:* QUIC transport layer cannot determine whether an Object is complete or arrived within deadline. MMF provides per-Object status (RECEIVED_LATE, NOT_RECEIVED, etc.).

* *Application-Layer Metrics:* Playout headroom (PLAYOUT_AHEAD), receiver-side bandwidth estimation, etc.

3.2.1. QUIC receive-ts Requirements

Implementations would still need to support both QUIC receive-ts ([quic-receive-ts]) and MMF simultaneously when both extensions are negotiated.

QUIC receive-ts carries per-packet reception timestamps (Timestamp Range / Timestamp Delta) via ACK_EXTENDED frames, enabling CC algorithms to compute inter-arrival delta (delay gradient) and bandwidth estimation. Its role is equivalent to TWCC in WebRTC:

* *QUIC receive-ts:* per-packet reception timestamps --> delay-based CC (GCC/SQP, etc.)

* *MMF:* per-Object delivery status --> application-layer adaptation + MoQ-layer CC control

Compared to WebRTC GCC+TWCC, this framework adds a per-Object semantic feedback layer on top of per-packet feedback. TWCC only provides packet-level arrival times and cannot express Object completeness, expiration status, or playout headroom.

3.3. Cross-Layer Control Interface

MoQ implementations could provide a cross-layer control interface that enables the MoQ layer to issue control commands to CC algorithms. The specific form of the interface is implementation-defined.

Distinct from approaches that pass raw data to CC algorithms for independent judgment, this mechanism could help the MoQ layer to achieve better performance. The MoQ layer possesses three kinds of information resource: MMF feedback, local CC output (BWE), and frame-level statistics (Section 8.1), achieving higher information completeness than CC algorithms alone.

CC algorithms could still run their own congestion detection logic, or adapted to accept control commands issued by the MoQ layer.

Control Commands:

Command	Description
target_bitrate	Target encoding bitrate computed by MoQ layer from BWE and MMF; notifies CC of current application-layer sending budget
pacing_gain	Sending gain coefficient; MoQ layer calculates from MMF signals (Object loss rate, expiration rate, playout headroom); CC may execute as pacing_rate = BWE x pacing_gain
pacing_rate	Directly specify sending rate; MoQ layer calculates and issues in scenarios like frame-level pacing (Section 8.1.4)

Table 4

MoQ Layer Decision Inputs:

The MoQ layer synthesizes the following signals for control decisions:

* *MMF Signals (from peer):* Object loss rate, expiration rate, Avg Inter-Arrival Delta, PLAYOUT_AHEAD_MS

* *CC Output (from local):* BWE, RTT, loss rate

* *Frame-Level Statistics (from local, Section 8.1.3):* per-Object transmission/loss/delivery duration

CC Algorithm Behavior:

CC algorithms SHOULD execute control commands after receiving them:

* *Upon pacing_gain:* Update sending rate as pacing_rate = BWE x pacing_gain

* *Upon pacing_rate:* Use this value directly as sending rate

* *Upon target_bitrate:* Record current application-layer sending budget for internal decision reference

When no control commands are issued by the MoQ layer, CC algorithms operate normally according to their own logic.

The cross-layer control interface is optional. CC algorithms that do not support this interface can still operate independently but cannot benefit from MMF-driven frame-level control capabilities.

3.4. Bidirectional MMF: Output Feedback and Input Feedback

The MMF report direction described in the preceding sections is from client to server, reporting delivery quality of downstream media (audio_response, text_response). In streaming input scenarios, the server simultaneously subscribes to upstream media (audio_input, video_input) and MAY establish a reverse feedback Track to report upstream media delivery quality to the client.

Client Server | | |----- audio_input ------------->| Server subscribes |<---- input_feedback (MMF) -----| Server reports input delivery quality | | |<---- audio_response -----------| Client subscribes |-- multimodal_feedback (MMF) -->| Client reports output delivery quality | |

Output MMF (client-->server) and Input MMF (server-->client) use the same report format (Section 5). Their purposes differ:

* *Output MMF:* Server adjusts encoding bitrate, inference parameters, and CC based on it.

* *Input MMF:* Client adjusts upstream behavior based on it:

  - *Audio Input:* Adjust chunk size, encoding bitrate, transmission frequency

  - *Video Input:* Adjust frame rate, resolution, pause/resume

Input MMF is optional. Support of the input MMF is declared via bit-2 of the Setup Parameter bitmap.

4. Feedback Track

4.1. Track Definition

The Feedback Track is a normal MoQ Track where the Payload of each Object is an MMF. Each MMF is published as an independent Object with monotonically increasing Object ID within a Group.

The Group division strategy for Feedback Track is implementation-defined. It is RECOMMENDED to use a single Group (Group ID = 0) with continuously incrementing Object IDs to simplify implementation.

Each Feedback Track MUST be associated 1:1 with a media Track. The association is established during the PUBLISH phase (Section 4.3) and is not repeated in MMF reports.

When feedback is needed for multiple media Tracks, independent Feedback Tracks MUST be established separately.

4.2. Track Naming

Feedback Track naming SHOULD follow these conventions:

* *Namespace:* Same Namespace as the media Track being fed back.

* *Track Name:* multimodal-feedback/<media_track_name>.

Examples: multimodal-feedback/audio_response, multimodal-feedback/video_response.

Media Track Names MUST NOT contain the / character to avoid parsing ambiguity in Feedback Track Names.

When reverse feedback exists in a session (see Section 3.4), the Input Feedback Track Name is input-feedback/<media_track_name>.

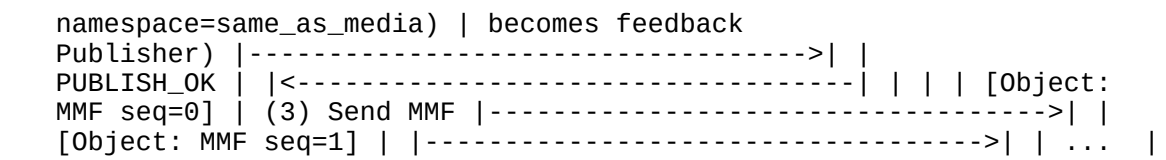
When a sender receives a PUBLISH request for a Feedback Track, it MUST identify the associated media Track via the <media_track_name> portion of the Track Name. If no matching established media Track is found, it SHOULD respond with REQUEST_ERROR.

4.3. Track Establishment and Lifecycle

The Feedback Track is established by the feedback generator (typically the media receiver) as Publisher through MOQT's PUBLISH / PUBLISH_OK negotiation.

Establishment Flow Example:

Media Receiver Media Sender (Media Subscriber / (Media Publisher / Feedback Publisher) Feedback Subscriber) | | | SUBSCRIBE(track=audio_response) | (1) Subscribe media Track |----------------------------------->| | SUBSCRIBE_OK | |<-----------------------------------| | | | PUBLISH(track=multimodal-feedback/ | (2) Publish Feedback Track | audio_response, | (Role reversal: receiver |



Lifecycle Rules:

The Feedback Track lifecycle SHOULD align with the subscribed media Track it covers. After the media Track publisher sends PUBLISH_DONE for the media Track, the Feedback Track publisher (i.e., the media receiver) SHOULD send PUBLISH_DONE for the Feedback Track after transmitting the final MMF and stop publishing Objects.

When re-establishing a Feedback Track after connection interruption, the Report Sequence SHOULD restart from 0.

4.4. Transport and Priority

The Feedback Track SHOULD be carried over QUIC Stream (consistent with ordinary MoQ Objects).

The Subscriber Priority for Feedback Track SHOULD be set lower than media Tracks ([MoQTransport], Section 7). Example priority assignment:

Track Type	Subscriber Priority	Description
audio_response	0 (highest)	Audio media
video_response	1	Video media
multimodal-feedback	3	Feedback Track

Table 5

When bandwidth contention occurs, media data SHOULD take precedence over feedback data transmission.

5. Feedback Report Format

This section defines the binary encoding format of MMF. All integer fields use QUIC Variable-Length Integer encoding (RFC 9000, Section 16) unless otherwise specified.

Fields marked as "signed encoding" use ZigZag mapping and are transmitted as unsigned QUIC varint:

* *Encoding:* unsigned = (signed << 1) ^ (signed >> 63)

* *Decoding:* signed = (unsigned >>> 1) ^ -(unsigned & 1)

Mapping examples: 0 -> 0, -1 -> 1, 1 -> 2, -2 -> 3, 2 -> 4.

5.1. Report Structure

MoQ Multimodal Feedback { Report Timestamp (i), Report Sequence (i), Object Entry Count (i), Object Entry (..) ..., Summary Stats (..), Optional Metric Count (i), Optional Metric (..) ..., }

The MMF format version is negotiated via Setup Parameter (Section 6.1) and is not repeated in each report. This document defines version 0.

Each MMF reports Object delivery status for a single media Track associated with its Feedback Track (1:1 association, see Section 4.1).

Report Timestamp (varint): The moment when the report is generated, using receiver’s local monotonic clock value in microseconds. This value is only used for Receive Timestamp Delta chain anchoring and report ordering, not for cross-end time alignment. Monotonic clock MUST be used.

Report Sequence (varint): Monotonically increasing from 0. Senders detect feedback loss via sequence gaps (Section 10.1).

Object Entry Count (varint): Number of Object Entries that follow. When the value is 0, the report contains only Summary Stats (for heartbeat purposes).

5.2. Object Entry

Object Entry { Object ID (i), Status (i), [Receive Timestamp Delta (i)], }

Object Entries within the same MMF MUST be sorted in ascending order by Object ID.

5.2.1. Object ID

The Object ID (varint) within the media Track.

5.2.2. Status

Delivery status code (varint), see Section 5.3 for values.

5.2.3. Receive Timestamp Delta

Conditional presence field (varint, signed encoding): Present only when Status is RECEIVED (0x00) or RECEIVED_LATE (0x01). Encoding rules:

* *First received Object within the same MMF* (i.e., the first entry with Status RECEIVED or RECEIVED_LATE when iterating through the list): Delta is the offset of the Object's arrival time relative to Report Timestamp in microseconds, encoded as signed varint (negative values indicate arrival time earlier than report generation time).

* *Subsequent received Objects:* Delta is the offset of the Object's arrival time relative to the arrival time of the most recent entry with Status RECEIVED or RECEIVED_LATE that precedes it in the list, encoded as signed varint.

NOT_RECEIVED and PARTIALLY_RECEIVED entries are skipped in the delta chain.

Since Object Entries are sorted by Object ID in ascending order while Objects may arrive out of order, delta values can be negative (indicating the Object arrived earlier than the previous received Object in the list). Negative delta from reordering is a valid network state signal.

This Delta chain encoding approach is consistent with QUIC Receive Timestamps (draft-ietf-quic-receive-ts-00), but at Object granularity rather than packet. For small Objects (audio frames ~200B, approximately 1 packet/Object), the Delta sequence approaches per-packet precision.

5.3. Delivery Status Codes

Value	Name	Description
0x00	RECEIVED	Completely received and within delivery deadline
0x01	RECEIVED_LATE	Completely received but exceeded delivery deadline

0x02	NOT_RECEIVED	No bytes received at report generation time
0x03	PARTIALLY_RECEIVED	Partial bytes received but Object is incomplete

Table 6

5.3.1. RECEIVED_LATE Determination

Receiver MUST determine Object expiration based on local playback deadline:

An Object is RECEIVED_LATE when its arrival time exceeds its expected playback moment.

When playback deadline is unavailable (e.g., non-real-time playback scenarios), receiver SHOULD report RECEIVED rather than RECEIVED_LATE.

5.3.2. NOT_RECEIVED Reporting Timing

Receiver SHOULD report an Object as NOT_RECEIVED when one of the following conditions is met:

* A subsequent Object with larger Object ID has arrived, but this Object has not.

* The Object’s expected arrival time has exceeded 2 x expected_interval (see Section 5.4.6) without arrival, and no subsequent Object is available for reference.

Condition 2 covers the "last Object lost" scenario (no subsequent Object to trigger Condition 1).

Receiver MUST NOT report NOT_RECEIVED for Objects that are not yet reasonably expected to arrive.

5.3.3. PARTIALLY_RECEIVED

Applicable to Objects carried over QUIC Stream: When a Stream is terminated by RESET_STREAM or closed due to timeout, Objects with partially received data SHOULD be reported as PARTIALLY_RECEIVED.

The timeout threshold used to determine whether a partially received Object should be reported as PARTIALLY_RECEIVED is application-defined. Different applications may have different latency

requirements (e.g., real-time voice vs. file transfer), and the receiver's application layer is best positioned to decide when an incomplete Object is no longer useful. Implementations SHOULD allow the application layer to configure or signal this timeout value.

For Objects carried over QUIC Datagram: Since QUIC Datagrams are not retransmitted by the transport layer, any Object whose Datagram(s) are lost results in incomplete data at the receiver. If the receiver detects that some but not all Datagrams constituting an Object have arrived, and the remaining Datagrams are not expected to arrive (e.g., a subsequent Object has arrived or the application-defined timeout has expired), the Object SHOULD be reported as PARTIALLY_RECEIVED. If no Datagrams for the Object have arrived, the Object SHOULD be reported as NOT_RECEIVED instead.

5.4. Summary Stats Block

Summary Stats { Report Interval (i), Total Objects Evaluated (i), Objects Received (i), Objects Received Late (i), Objects Lost (i), Avg Inter-Arrival Delta (i), }

Summary Stats MUST always be included in every MMF, not controlled by negotiation bitmap. It provides windowed aggregate information, enabling lightweight CC consumers that do not parse Object Entry to obtain effective signals. The MoQ layer can compute control decisions based on this block and issue them to CC algorithms via the cross-layer control interface (Section 3.3).

The Report Interval window of Summary Stats is independent from the coverage of Object Entries. Object Entries MAY include Objects outside the Report Interval window (e.g., continuously reported NOT_RECEIVED entries, see Section 7.4).

5.4.1. Report Interval

The time window length covered by this report (varint), in microseconds. This window spans from Report Timestamp - Report Interval to Report Timestamp. RECOMMENDED value is 50000-200000 (50-200ms).

5.4.2. Total Objects Evaluated

Total number of Objects (varint) evaluated within the Report Interval window. Includes Objects of all statuses. MUST equal Objects Received + Objects Received Late + Objects Lost.

5.4.3. Objects Received

Number of Objects (varint) with status RECEIVED within the window.

5.4.4. Objects Received Late

Number of Objects (varint) with status RECEIVED_LATE within the window.

5.4.5. Objects Lost

Number of Objects (varint) with status NOT_RECEIVED or PARTIALLY_RECEIVED within the window.

5.4.6. Avg Inter-Arrival Delta

Average arrival interval deviation (varint, signed encoding) of consecutive received Object pairs within the window, in microseconds. Calculation method:

For consecutive received Object pairs (i-1, i) sorted by arrival time within the window:

delta(i) = (A(i) - A(i-1)) - expected_interval Avg Inter-Arrival Delta = mean(delta(i)) for all i

Where A(i) is the arrival time of Object i, and expected_interval is the expected arrival interval.

Methods to determine expected_interval (by priority):

1. Known media frame rate (e.g., 50 obj/s --> 20000us).

2. Historical average arrival interval within current session (sliding window).

If neither method is available, receiver SHOULD use method 2 or set this field to 0.

Positive values indicate Object arrival interval greater than expected (increased queuing), negative values indicate less than expected. When fewer than 2 received Objects exist in the window, this field MUST be 0.

5.5. Optional Media Metrics

Optional Media Metrics immediately follow Summary Stats. Optional Metric Count (varint) specifies the number of subsequent Optional Metrics; a value of 0 indicates no optional metrics are included.

Each metric uses Key-Value-Pair encoding (draft-ietf-moq-transport-17, Section 1.4.2):

Optional Metric { Metric Type (i), Metric Value (i), }

Defined metric types are divided into two categories: Application-Layer Metrics and QUIC Layer Summary Metrics.

Application-Layer Metrics:

Type	Name	Unit	Description
0x02	PLAYOUT_AHEAD_MS	milliseconds	Remaining time until playback stall at receiver, i.e., buffered but not yet played media duration. Smaller values indicate closer to stall (0 = currently stalled).
0x04	ESTIMATED_BANDWIDTH_KBPS	kbps	Available bandwidth estimate observed at receiver. Can be calculated as bytes received in window / window duration. For sender to cross-reference with local bandwidth estimation.

Table 7

QUIC Layer Summary Metrics:

The following metrics expose the receiver’s local QUIC connection transport state to the sender for CC algorithm cross-validation.

Type	Name	Unit	Description
0x10	PEER_RTT_US	microseconds	Receiver's local QUIC connection smoothed RTT, corresponding to smoothed_rtt in RFC 9002. For sender to cross-validate with local RTT estimation.
0x12	PEER_LOSS_RATE	per mille	Receiver's local QUIC connection packet loss rate within Report Interval, expressed in per mille (e.g., 50 = 5.0%).

Table 8

MMF's core CC signals rely on Receive Timestamp Delta in Object Entry (Section 5.2.3, referencing QUIC receive-ts / WebRTC TWCC per-packet delta encoding approach) and Summary Stats (Section 5.4), not Optional Metrics. Optional Metrics serve only as supplements.

Type values 0x00-0x1f are reserved for this specification. Values 0x20 and above are available for application-layer custom use.

Receiver MUST ignore unrecognized Metric Types.

Optional Media Metrics MAY be included in MMF only when both parties have declared bit1=1 in Setup negotiation (Section 6). When not negotiated or bit1=0, Optional Metric Count MUST be 0.

5.6. Report Size Control

A single MMF is RECOMMENDED not to exceed 1200 bytes to avoid QUIC packet fragmentation.

When the number of Objects to report exceeds the capacity of a single MMF, receiver SHOULD:

* Prioritize including recent Object Entries (largest Object IDs).

* Trim the oldest Object Entries.

* Ensure Summary Stats covers the complete Report Interval window (Summary Stats is not affected by Object Entry trimming).

5.6.1. Encoding Example

The following is an encoding structure of a typical MMF reporting delivery status of the 5 most recent Objects on the audio_response Track. Object Entries are sorted in ascending order by Object ID. Signed fields use ZigZag-encoded unsigned values (encoding convention at the beginning of Section 5).

‘‘‘ MoQ Multimodal Feedback: Report Timestamp: 2000000 (2000000us = 2s since setup) Report Sequence: 10 (10th report) Object Entry Count: 5 (5 Objects)

Object Entry [0]: (smallest Object ID, first in list) Object ID: 96 Status: 0x00 (RECEIVED) Recv Ts Delta: 169999 (ZigZag(-85000): 85ms before Report Timestamp) (First received Object, baseline=Report Timestamp)

Object Entry [1]: Object ID: 97 Status: 0x02 (NOT_RECEIVED) (No Recv Ts Delta)

Object Entry [2]: Object ID: 98 Status: 0x01 (RECEIVED_LATE) Recv Ts Delta: 100000 (ZigZag(+50000): 50ms later than Object 96) (Skip NOT_RECEIVED 97, baseline=Object 96)

Object Entry [3]: Object ID: 99 Status: 0x00 (RECEIVED) Recv Ts Delta: 40000 (ZigZag(+20000): 20ms later than Object 98)

Object Entry [4]: Object ID: 100 Status: 0x00 (RECEIVED) Recv Ts Delta: 40000 (ZigZag(+20000): 20ms later than Object 99)

Summary Stats: Report Interval: 100000 (100000us = 100ms) Total Objects Evaluated: 5 Objects Received: 3 Objects Received Late: 1 Objects Lost: 1 Avg Inter-Arrival Delta: 6000 (ZigZag(+3000): avg arrival interval 3ms larger)

Optional Metric Count: 2 Optional Metric [0]: Metric Type: 0x02 (PLAYOUT_AHEAD_MS) Metric Value: 150 (playout headroom 150ms) Optional Metric [1]: Metric Type: 0x04 (ESTIMATED_BANDWIDTH_KBPS) Metric Value: 800 (estimated bandwidth 800kbps) ‘‘‘

In this example, Object 97 is lost and Object 98 arrived late. Object 98's delta (+50ms) is significantly larger than normal interval (~20ms). Avg Inter-Arrival Delta is positive (+3ms) indicating larger-than-expected arrival intervals. PLAYOUT_AHEAD_MS is only 150ms. These signals combined indicate deteriorating network conditions.

6. Negotiation

6.1. Setup Parameter

During MOQT Setup phase, both parties declare Multimodal Feedback capability via Setup Parameter.

MOQT_MULTIMODAL_FEEDBACK Setup Parameter { Type = TBD1 (i), Length (i), Value (i), }

The Value field is a capability bitmap (varint) with the following bit definitions:

Bit	Name	Description
0	OUTPUT_FEEDBACK	Support output direction Feedback Track (receiver-->sender)
1	OPTIONAL_METRICS	Support Optional Media Metrics (Section 5.5)
2	INPUT_FEEDBACK	Support input direction Feedback Track (sender-->receiver, Section 3.4)
3-62	Reserved	Sender MUST set to 0, receiver MUST ignore

Table 9

Negotiation Example:

Client Server | | | CLIENT_SETUP(| | version=16, | | params=[| | {type=0x00, value=0x02}, | (ROLE=Publisher) | {type=TBD1, value=0x03} | (MULTIMODAL_FEEDBACK: bit0|1=1) |]) | |----------->| | | | SERVER_SETUP(| | version=16, | | params=[| | {type=0x00, value=0x03}, | (ROLE=PubSub) | {type=TBD1, value=0x01} | (MULTIMODAL_FEEDBACK: bit0=1) |]) | |<-----------| | | | Negotiation Result: | | bit0=1 (Output Feedback) | | bit1=0 (Client declared but | | Server did not, | | Optional Metrics disabled) |

6.2. Capability Negotiation Rules

Feature enable conditions (both parties MUST declare corresponding bit as 1):

Feature	Enable Condition	Dependency
Output Feedback	Both bit0=1	None
Optional Media Metrics	Both bit1=1	Output Feedback enabled
Input Feedback	Both bit2=1	None

Table 10

When a feature is not enabled:

* Receiver MUST NOT publish Feedback Track in corresponding direction.

* When sender receives PUBLISH request for Feedback Track in un-negotiated direction, it SHOULD respond with REQUEST_ERROR (draft-ietf-moq-transport-17, Section 9.8).

* MMF MUST NOT include un-negotiated optional fields (e.g., Optional Metrics).

6.3. Behavior When Parameter Not Declared

When peer’s Setup does not include MOQT_MULTIMODAL_FEEDBACK Parameter, it is equivalent to Value=0 (no Multimodal Feedback capability supported).

This end MUST NOT proactively establish Feedback Track.

6.4. Runtime Capability Change

This version does not support runtime changes to Multimodal Feedback capability. If change is needed, MoQ session MUST be re-established.

7. Receiver Behavior

7.1. Arrival Time Recording

Receiver MUST record the arrival time of each Object. Arrival time is defined as the moment when the last byte of the Object arrives at the receiver’s MoQ layer. Implementation MUST use monotonic clock, unaffected by system time adjustments. Time precision SHOULD be no less than 1 millisecond, RECOMMENDED to be microsecond-level.

7.2. Delivery Status Determination

Receiver MUST maintain delivery status for each known Object:

* Last byte of Object arrives and within deadline: RECEIVED (0x00).

* Last byte of Object arrives but exceeds deadline: RECEIVED_LATE (0x01).

* Object has not arrived but is reasonably considered lost (see Section 5.3): NOT_RECEIVED (0x02).

* Object partially arrived and carrying Stream is closed: PARTIALLY_RECEIVED (0x03).

Object status MAY be updated in subsequent MMF. For example, from NOT_RECEIVED to RECEIVED (when a delayed Object is eventually received). Therefore, counts such as Objects Lost in Summary Stats reflect an observation snapshot at report generation time and are not guaranteed to be consistent with final statistics. Sender SHOULD use them as immediate signals rather than precise statistics.

7.3. MMF Generation Frequency

Receiver SHOULD generate MMF at the following recommended frequencies:

Scenario	Recommended Frequency	Description
Audio Track (~50 Object/s)	Every 50-100ms	High-frequency Objects require dense feedback
Video Track (~2-30 Object/s)	Every 100-200ms	Low-frequency Objects can reduce feedback frequency
No new Objects arriving	Every 500ms-1s	Heartbeat to prevent sender from misjudging connection state

Table 11

Generation frequency SHOULD NOT exceed once per 50ms (to avoid feedback itself consuming excessive bandwidth). Generation frequency SHOULD NOT be lower than once per 2s (to ensure sender receives timely feedback).

When receiver detects rapid deterioration in delivery quality (e.g., consecutive Object losses), it MAY immediately generate an additional MMF (without waiting for the scheduled cycle) to accelerate sender response.

7.4. Object Entry Selection

When generating MMF, receiver SHOULD follow these Object Entry selection strategies:

* Prioritize covering recent Objects (most recent in time).

* Total number of Object Entries per MMF is RECOMMENDED not to exceed 50.

* For Objects already reported in previous MMF with unchanged status, they MAY be omitted.

* Objects with status NOT_RECEIVED SHOULD be continuously reported (for at least 3 MMF cycles) until status changes or exceeding the report window.

7.5. Exception Handling

* *Object Out-of-Order Arrival:* Receiver MUST record by actual arrival time without reordering. Arrival time reflects actual network behavior; out-of-order itself is a valid network signal.

* *Duplicate Objects:* Receiver SHOULD ignore duplicate arrivals, retaining the first arrival time.

* *Feedback Track Publish Failure:* Receiver SHOULD retry establishing Feedback Track with backoff strategy. Retry interval is RECOMMENDED to use exponential backoff (initial 1s, maximum 30s).

8. Sender Behavior

8.1. Object to QUIC Packet Mapping

If a sender needs to correlate MMF feedback with local transmission events, it MUST maintain the mapping relationship between Objects and QUIC packets. Without this mapping, the sender cannot determine the number of packets, packet loss count, and delivery duration for a single video frame, and the per-Object status reported by MMF cannot be aligned with sender-side statistics.

8.1.1. Object Granularity

In real-time media scenarios, a MoQ Object SHOULD correspond to an independently decodable media unit (a video frame or an audio frame).

This enables per-Object feedback in MMF to directly carry frame-level semantics: Object loss = frame loss, Object expiration = frame expiration. If an Object contains multiple frames or a single frame spans multiple Objects, there will be deviation between the delivery status reported by MMF and the actual media quality.

8.1.2. Packet Not Crossing Object Boundaries

Senders SHOULD avoid merging data from different Objects into the same QUIC packet.

If a packet contains data from two Objects, then loss or delay of that packet cannot be attributed to a single Object, leading to:

* Frame-level packet loss rate statistics distortion (single packet loss affects statistics for both frames)

* Frame-level delivery time cannot be accurately measured (transmission times of two frames overlap)

* Per-Object arrival time at the receiver becomes inaccurate

During implementation, when the QUIC send queue attempts to append new data to an existing packet, it SHOULD check whether both belong to the same Object; if not, it SHOULD create a new packet.

8.1.3. Sender-Side per-Object Transmission Statistics

Senders SHOULD maintain the following transmission statistics for each Object:

Statistic	Description
sent_packets	Number of QUIC packets sent for this Object
lost_packets	Number of QUIC packets lost for this Object
first_sent_time	Transmission time of the first packet for this Object
all_acked_time	Time when all packets for this Object are acknowledged

Table 12

Based on these statistics, the sender can compute:

* *Frame-level bandwidth sample:* object_size / (all_acked_time - first_sent_time)

* *Frame-level packet loss rate:* lost_packets / sent_packets

* *Frame-level delivery duration:* all_acked_time - first_sent_time

These metrics provide the foundation for frame-level BWE and frame-level pacing (Section 8.1.4). Standard CC algorithms (BBR, CUBIC) sample bandwidth at the packet level. Frame-level bandwidth sampling serves as a complementary approach, suitable for real-time video scenarios with large frame size variations (I-frames may be 10x larger than P-frames), helping to reduce noise from single-packet sampling.

Whether a CC algorithm provides BWE output depends on the algorithm type. Model-based algorithms such as BBR and GCC provide bandwidth estimation; pure loss-based algorithms like CUBIC only output cwnd without explicit BWE. When the CC algorithm does not provide BWE, the MoQ layer MAY use frame-level bandwidth sampling as the bandwidth estimation source.

8.1.4. Frame-Level Pacing

Real-time video exhibits significant frame size variations. When using a global fixed pacing rate for transmission, large frames (I-frames) will burst a large number of packets in a short time, while small frames (P-frames) underutilize the sending window.

Senders SHOULD calculate pacing intervals at Object granularity.

The packet sending interval for each Object is RECOMMENDED to be computed as follows:

pkt_send_interval = media_pace_duration / (sent_packets - 1)

Where media_pace_duration is the media duration corresponding to this Object (e.g., 33ms@30fps for video, 20ms@50fps for audio). The first packet of an Object is sent immediately, and subsequent packets are sent at equal intervals of pkt_send_interval.

This approach distributes an Object’s data uniformly across its frame interval. I-frames have denser packet intervals, P-frames have sparser intervals, but neither produces bursts.

Frame-level pacing may conflict with CC’s pacing rate. When they are inconsistent, senders SHOULD use the lower sending rate to prevent frame-level pacing from bypassing CC’s congestion control.

8.2. Application-Layer Consumption

After the sender’s MoQ layer receives MMF, it exposes the following information to the inference scheduler/ABR through application-layer callbacks:

* Delivery quality (loss rate, expiration rate) of the associated media Track

* Per-Object status and timestamps (can be used for detailed analysis)

* Optional media metrics (playout headroom, bandwidth estimation, etc.)

The inference scheduler makes decisions based on the above information (see Section 9 for details).

8.3. Transport-Layer Consumption (Cross-Layer Control)

The MoQ layer synthesizes MMF signals and local CC output, computes control decisions, and issues them to CC algorithms via the cross-layer control interface (Section 3.3). CC algorithms do not directly parse MMF, but instead execute control commands from the MoQ layer. Decision inputs, issuable commands, and CC behavior are described in Section 3.3.

8.4. Example: MoQ Layer Controlling BBR

The core formula of BBR is pacing_rate = bandwidth x pacing_gain.

BBR itself estimates bandwidth through ACK sampling and controls pacing_gain via its state machine. In cross-layer control mode, the MoQ layer takes over control of pacing_gain. BBR remains responsible for bandwidth estimation, while pacing_gain is determined by the MoQ layer based on MMF feedback:

‘‘‘ MoQ Layer: 1. Read BBR’s BWE 2. Read MMF: Objects Lost, Late, PLAYOUT_AHEAD_MS, Delta 3. Compute pacing_gain (comprehensive judgment) 4. Issue pacing_gain --> CC

BBR: 1. Receive pacing_gain 2. pacing_rate = BWE x pacing_gain 3. Send at pacing_rate ‘‘‘

Example logic for MoQ layer computing pacing_gain:

1. *Objects Lost > 0 and BBR local loss = 0* --> Reduce pacing_gain to 1.0 (CC local ACK is normal, but peer is actually losing frames; stop probing upward)

2. *PLAYOUT_AHEAD_MS < 100ms* --> Reduce pacing_gain to 1.0 (Insufficient playout headroom; avoid aggressive probing)

3. *Avg Inter-Arrival Delta consistently positive* --> Reduce pacing_gain to 0.9 (Receiver-side queuing is worsening; proactively reduce speed)

4. *High proportion of Objects Received Late* --> Reduce target_bitrate (Transmission has no packet loss but delay exceeds deadline; reduce per-frame data volume at the source)

5. *None of the above conditions met* --> Do not issue commands; BBR runs according to its own state machine

When the MoQ layer does not issue commands, BBR operates normally according to its own state machine. The MoQ layer overrides pacing_gain only when MMF signals indicate intervention is needed; otherwise, BBR operates fully autonomously.

9. Application Scenarios: Streaming Media and AI Inference

This section describes the usage of MMF in streaming media and AI inference scenarios. Section 9.1 presents a general adaptation framework, and Section 9.2 illustrates with concrete use cases.

9.1. MMF-Driven Rate-Quality Adaptation

The following adaptation rules apply to all MoQ streaming media scenarios:

MMF Field (Section)	Adaptation Action	Effect	Applicable Scenarios
PLAYOUT_AHEAD_MS downward trend (5.5)	Reduce encoding bitrate	Less data per frame	Video/Audio/ Inference
PLAYOUT_AHEAD_MS downward trend (5.5)	Reduce frame rate / Object sending frequency	Reduced transmission volume	Video
Objects Lost > 0 (5.4.5)	Reduce sending rate (with CC)	Match available bandwidth	All
Objects Received Late > 0 (5.4.4)	Reduce quality to meet deadline	Trade quality for timeliness	All
Avg Inter-Arrival Delta increasing (5.4.6)	Preventive quality reduction (before packet loss)	Avoid sudden degradation	All
PLAYOUT_AHEAD_MS recovery (5.5)	Gradually restore bitrate/frame rate	Improve experience	All

Table 13

These adaptations can be executed at the MoQ publishing layer with per-Object granularity and latency < 1 frame cycle, without requiring application-layer media source cooperation. When the application layer is an AI inference pipeline, MMF can also adjust inference pipeline parameters (real-time flush, dynamic chunk_size).

Unidirectional and bidirectional adaptation modes are described in Section 3.4 (Bidirectional MMF).

9.2. Typical Use Cases

The following use cases assume BBR as the CC algorithm (integration method described in Section 8.4) and illustrate the effects of MMF in different scenarios. All cases follow the strategy of prioritizing audio output and progressively reducing quality.

9.2.1. Use Case A: Video Live Streaming ABR Adaptation

* *Scenario:* Sender publishes video live streaming with two Tracks: 1080p (3Mbps) and 720p (1.5Mbps). Receiver is currently subscribed to the 1080p Track.

* *Trigger condition:* Available network bandwidth drops from 4Mbps to 2Mbps.

* *MMF signals:* Proportion of Objects Received Late increases (video frames arrive but exceed deadline), PLAYOUT_AHEAD_MS gradually decreases.

* *CC response:* BBR reduces pacing_rate to match new bandwidth.

* *Application-layer response:* MoQ publishing layer detects persistently high Objects Received Late on the 1080p Track, triggers ABR switching: switches to 720p Track at the next Group boundary (native MoQ Track switching mechanism).

* *Recovery:* After MMF reports Objects Received Late returning to normal and PLAYOUT_AHEAD_MS recovering, sender may attempt to switch back to 1080p.

* *Benefit:* Per-frame integrity and deadline information provided by MMF enables higher ABR decision accuracy than pure CC-driven approaches.

9.2.2. Use Case B: Bandwidth Drop (Audio-Video Mixed)

* *Trigger condition:* Available bandwidth drops sharply.

* *MMF signals:* Objects Lost increases, Avg Inter-Arrival Delta increases, PLAYOUT_AHEAD_MS decreases.

* *CC response:* MoQ layer issues pacing_gain=1.0 based on MMF loss signal, pauses upward probing.

* *Application-layer response:* Reduce Opus bitrate, pause video Track (prioritize audio).

* *Benefit:* Peer-side frame loss signal from MMF and congestion signal from BBR local ACK can cross-validate. Gradually restore bitrate after network recovery.

9.2.3. Use Case C: Multi-Layer Quality Adaptation (AI Inference)

* *Trigger condition:* Persistent congestion.

* *MMF signals:* Objects Lost persistently high, PLAYOUT_AHEAD_MS at low level.

* *CC response:* MoQ layer continuously issues low pacing_gain, pauses upward probing.

* *Application-layer response:* Based on general adaptation from Use Case B, inference pipeline reduces chunk_size, accelerates audio flush, reduces output latency. CC and application layer adjust synchronously.

* *Benefit:* Per-frame level adaptation takes effect within frame cycle; audio remains uninterrupted throughout. During recovery, gradually restore chunk_size and bitrate.

9.2.4. Use Case D: Generation Rate Overload (AI Inference)

* *Trigger condition:* Inference model suddenly generates long response, audio_response traffic spikes.

* *MMF signals:* High proportion of Objects Received Late (frames arrive but exceed playback deadline), Avg Inter-Arrival Delta is large.

* *CC response:* Reduce pacing_rate.

* *Application-layer response:* Reduce encoding bitrate, accelerate flush, guide inference to generate more concise responses (reduce audio duration at the source).

* *Benefit:* Trade quality for timeliness, avoid playback stuttering.

9.2.5. Use Case E: Streaming Input Inference (Bidirectional MMF)

* *Trigger condition:* Uplink network jitter (streaming input inference scenario).

* *MMF signals:* Input MMF reports high proportion of NOT_RECEIVED.

* *Application-layer response:*
  - *Client:* Increase chunk size, reduce uplink bitrate, pause video input.
  - *Server:* Manage KV cache, adjust handling strategy for incomplete input.
* *Benefit:* Bidirectional MMF coordinates quality of both uplink and downlink paths. MMF session activity signals (feedback frequency, PLAYOUT_AHEAD_MS) can be used for KV cache eviction and priority decisions.

10. IANA Considerations

This document defines the following code points for registration:

* *TBD1:* MOQT_MULTIMODAL_FEEDBACK Setup Parameter Type (Section 6.1)
* *Delivery Status Code Registry* (Section 5.3):
  - 0x00: RECEIVED
  - 0x01: RECEIVED_LATE
  - 0x02: NOT_RECEIVED
  - 0x03: PARTIALLY_RECEIVED
* *Optional Metrics Type Registry* (Section 5.5):
  - 0x02: PLAYOUT_AHEAD_MS
  - 0x04: ESTIMATED_BANDWIDTH_KBPS
  - 0x10: PEER_RTT_US
  - 0x12: PEER_LOSS_RATE

11. Acknowledgments

The design of this document references QUIC Extended ACK Receive Timestamps, RTP Transport Congestion Control Feedback, and drafts from the MoQ Working Group (MOQT / MSF / Metrics), as well as real-time multimodal inference systems (LongCat-Flash-Omni, Qwen3-Omni, Voxtral-Realtime), the vLLM-Omni Stage Pipeline framework, and vLLM streaming input mode.

12. Normative References

[MoQTransport]
Nandakumar, S., Vasiliev, V., Swett, I., and A. Frindell, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-17, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-17>.

[quic-receive-ts]
Swett, I. and J. Beshay, "QUIC Extended Acknowledgement for Reporting Packet Receive Timestamps", Work in Progress, Internet-Draft, draft-ietf-quic-receive-ts-01, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-receive-ts-01>.

[QUIC-TRANSPORT]
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>.

[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>.

Authors' Addresses

Minghui Jiang
Alibaba Inc.
Email: shimei.jmh@alibaba-inc.com

Yanmei Liu
Alibaba Inc.
Email: miaoji.lym@alibaba-inc.com

Ronghua Wu
Ant Group.
Email: r.wu@antgroup.com