MOQ
Internet-Draft
Intended status: Informational
Expires: 22 August 2025

A. Pehlivanoglu
K. Bekmez
B. Yumakogullari
Z. Gurel
A. Begen
Ozyegin University

18 February 2025

Synchronized Video-on-Demand (VoD) Viewing with Media over QUIC Transport
draft-pehlivanoglu-moq-shareplay-00

Abstract

This draft presents an approach for synchronized video-on-demand (VoD) viewing with Media over QUIC Transport (MOQT). It extends the current MOQT protocol to enable synchronized VoD functionality. This approach adapts MOQ's push-content architecture to include interactive features such as pause, resume and seek.

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 22 August 2025.

Copyright Notice

Table of Contents

1. Introduction

Media over QUIC Transport (MOQT) is a novel protocol designed for efficient media streaming over the QUIC transport protocol ([RFC9000]). While MOQT has shown promise for live-edge streaming, its current design lacks support for essential VoD functionalities, such as pause, resume and seek, leaving gaps in its applicability for interactive media consumption. This document extends MOQT to enable synchronized VoD playback, introducing mechanisms that allow users to have more control over the media playback.

This document outlines the architectural design, control mechanisms and synchronization logic implemented to achieve these objectives. This document’s innovations include new MOQT tracks for synchronization control, an example media publisher model for serving on-demand video and a Leader-Follower model to demonstrate the potential of MOQT for enhancing synchronized VoD consumption.

1.1. Terms 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.

Client: The party initiating a Transport Session.

Server: The party accepting an incoming Transport Session.

Endpoint: A Client or Server.

Publisher: An endpoint that handles subscriptions by sending requested Objects from the requested track.

Subscriber: An endpoint that subscribes to and receives tracks.

Original Publisher: The initial publisher of a given track.

Relay: An entity that is both a Publisher and a Subscriber, but not the Original Publisher or End Subscriber.

Group: A temporal sequence of objects. A group represents a join point in a track. See ([MoQTransport], Section 2.3).

Object: An object is an addressable unit whose payload is a sequence of bytes. Objects form the base element in the MOQT data model. See ([MoQTransport], Section 2.1).

Track: An encoded bitstream. Tracks contain a sequential series of one or more groups and are the subscribable entity with MOQT. See ([MoQTransport], Section 2.4).

Sync-Track: A dedicated track used to synchronize playback across multiple clients by transmitting PLAY, PAUSE and SEEK commands.

Leader Client: The designated control authority within a synchronized playback session. The Leader client issues playback control messages ([MoQTransport], Section 7) that all subscribers must follow.

Follower Client: A subscriber that follows playback instructions from the Leader client to maintain synchronization.

Video-on-Demand (VoD) Session: A session where pre-recorded media is streamed with user-controlled playback features.

1.2. Notational Conventions

This document uses the conventions detailed in ([RFC9000], Section 1.3) when describing the binary encoding.

As a quick reference, the following list provides a non-normative summary of the parts of RFC9000 field syntax that are used in this specification.

x (..): Indicates that x can be any length, including zero bits. Values in this format always end on a byte boundary.

x (i): Indicates that x holds an integer value using the variable-length encoding as described in ([RFC9000], Section 16).

2. Synchronized Playback Control

The MOQT protocol does not explicitly define dedicated control messages for PLAY, PAUSE or SEEK operations. Instead, these functionalities are achieved through subscription-based messaging.

2.1. Playback Control Mechanisms

Playback is initiated by sending a SUBSCRIBE ([MoQTransport], Section 7.4) control message for the desired media track. The SUBSCRIBE message specifies the track namespace, track name and a filter type (e.g., Latest Group or Absolute Start) to determine the starting point of the media delivery. Pausing playback is accomplished by sending a specific PAUSE message with its corresponding Group ID via Sync-Track. This stops the original

publisher from sending further objects for the specified track. To resume playback, the client issues a PLAY message again via Sync-Track. In the case of staying inactive for an extended period of time, the client will be unsubscribed automatically via MOQT's UNSUBSCRIBE ([MoQTransport], Section 7.6). To start the playback again, the client sends a SUBSCRIBE message.

Seeking to a specific position within a track is supported using the SEEK message via Sync-Track, with the corresponding Group ID. This allows the client to request media starting from a specific group and continuing in a push-content fashion. However, this functionality requires that the original publisher supports the ability to serve media from arbitrary positions within the track, which may not be universally available in all implementations. This is discussed in Section 4 in detail.

This document introduces new Sync-Track messages, which can only be sent by the Leader client. When the Leader client sends these messages, a compatible publisher will respond by playing, pausing or seeking as instructed. Upon receiving a PAUSE command, the original publisher halts broadcasting at the specified Group ID, freezing playback until a PLAY command resumes it from the paused position. A SEEK command directs the original publisher to calculate the frame corresponding to the provided Group ID and restart broadcasting from that frame. This mechanism ensures all subscribers remain synchronized, allowing them to watch the stream simultaneously from the adjusted playback point.

2.1.1. Decoupling Seek from Play/Pause

The SEEK operation functions independently of PLAY and PAUSE commands. A client can issue a SEEK command regardless of the current playback state. This means a client may seek to a different position while paused and later resume playback from the new position. Likewise, if a client is currently playing, a SEEK command will override the current playback position without requiring a PAUSE beforehand.

2.2. Sync-Track Messages

Every single message on the Sync-Track is a MOQT object, which is transmitted using the MOQT streaming architecture. These messages encapsulate playback control commands (PLAY, PAUSE, SEEK) within MOQT objects, ensuring they are processed and relayed consistently across the session. The Sync-Track operates as a track within the MOQT session, where control messages are formatted as follows:

```
Sync-Track Message {
  Message Type (i),
  Message Length (i),
  Message Payload (..),
}
```

Figure 1: Overall Sync-Track message format

Each message type (PLAY, PAUSE, SEEK) is identified by its unique Message Type ID and follows a structured format to ensure integration with the MOQT transport mechanisms. These messages are reliably delivered, maintaining synchronization among all session participants. Since the Sync-Track messages are standard MOQT objects, they leverage the existing MOQT stream handling mechanisms.

ID	Messages
0x1	PLAY (Section 2.2.1)
0x2	PAUSE (Section 2.2.2)
0x3	SEEK (Section 2.2.3)
0x4	STATUS (Section 2.2.4)

Table 1

2.2.1. PLAY

```
PLAY Message {
  Type (i) = 0x1,
  Length (i),
  Timestamp (i),
}
```

Figure 2: Sync-Track PLAY message

2.2.2. PAUSE

```
PAUSE Message {
  Type (i) = 0x2,
  Length (i),
  Timestamp (i),
  Group ID (i),
}
```

Figure 3: Sync-Track PAUSE message

2.2.3. SEEK

```
SEEK Message {
  Type (i) = 0x3,
  Length (i),
  Timestamp (i),
  Group ID (i),
}
```

Figure 4: Sync-Track SEEK message

2.2.4. STATUS

The Playing state is represented by 1, the Paused state is represented by 0 in the PlaybackState parameter.

```
STATUS Message {
  Type (i) = 0x4,
  Length (i),
  Timestamp (i),
  Group ID (i),
  PlaybackState (i),
}
```

Figure 5: Sync-Track STATUS message

2.3. Timestamp for Playback Synchronization

Each Sync-Track control message (PLAY, PAUSE, SEEK) contains a timestamp field, which represents the intended media position when the action is triggered. This timestamp ensures that playback actions (especially SEEK) are synchronized across clients, preventing head-of-line blocking due to out-of-order commands. The original publisher uses this timestamp to adjust the playback position accordingly.

2.4. Grouping of Playback Control Messages

The PLAY and PAUSE commands must be treated as dependent operations and thus belong to the same control subgroup. However, SEEK operates independently and belongs to a separate subgroup. This separation ensures that PLAY/PAUSE messages are processed sequentially within their stream, while the SEEK commands are handled in a separate stream to avoid blocking the playback state transitions.

2.5. Playback Status

To achieve better synchronization, a new status reporting message is introduced. This stream carries periodic updates from the original publisher, indicating the current playback position, current PlaybackState (Playing/Paused), and last acknowledged seek position. Clients can subscribe to this stream to get notified of every playback state change.

2.6. Playback Message Flow

The following message flow illustrates how the playback operations (PLAY, PAUSE and SEEK) are used through the relay servers, thus enabling clients to stay in a synchronized playback.

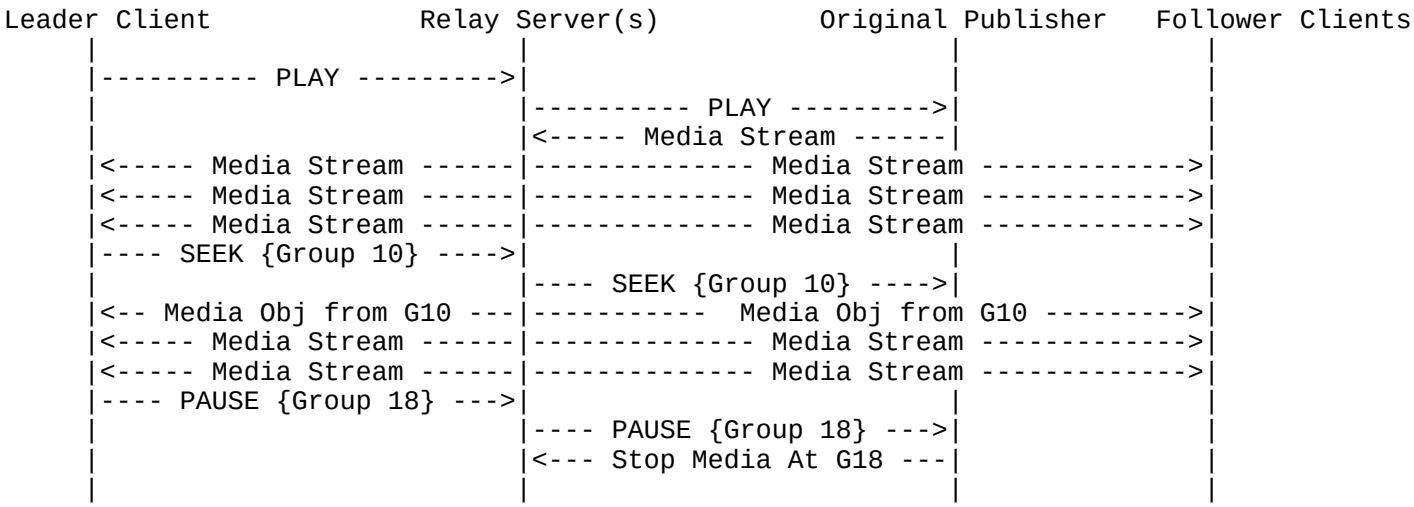

Figure 6: Sync-Track Message Flow and Playback Example Scenario

2.6.1. Message Semantics

These messages MUST be sent exclusively by the Leader client (Client ID 0). Relay servers MUST discard such messages if received from non-Leader clients.

3. Leader-Follower Client Model

The Leader-Follower client model is a subscriber coordination mechanism designed to enforce synchronized media playback across multiple clients. This model ensures deterministic control over playback operations to prevent conflicting actions and maintain synchronization.

3.1. Role Assignment

For simplicity, no Leader election algorithm is implemented in this model. The first subscriber to join a session is automatically designated as the Leader client (Client ID 0). Subsequent subscribers are assigned incremental Follower client IDs (e.g., 1, 2, ...) and inherit the playback state from the Leader. This ensures straightforward session initiation without requiring complex coordination mechanisms.

3.2. Playback Control

Exclusive Control: Only the Leader client may issue playback Sync-Track messages (PLAY, PAUSE, SEEK).

The Sync-Track message requests that are coming from the Follower clients will be forwarded to the Leader Client for review.

State Propagation: Control messages from the Leader client are distributed to all Follower clients via the relay servers, ensuring synchronized execution.

Leader Failure Handling: If the Leader Client disconnects or becomes unresponsive, the lowest-ID active Follower client assumes the Leader role. The new Leader client inherits Client ID 0, but the other Follower IDs do not change.

4. Heartbeat-Based Session Management and Advanced Synchronization

While the Leader-Follower paradigm handles playback control, additional functionality is needed for:

1. Monitoring client synchronization and connectivity.

2. Handling clients that wish to get out of synchronization or rejoin later.

3. Making group-level decisions in the face of network issues.

This section defines a “heartbeat” mechanism and outlines how clients can join, leave and optionally “fall behind” or “skip ahead”.

4.1. Heartbeat Messages

Every client (Leader or Follower) periodically sends a *heartbeat* message to the relay (or or the original publisher, depending on implementation). The relay aggregates these messages to maintain an up-to-date view of each client's playback status and connectivity. A heartbeat message contains at least:

1. *Local Playback Time*: The client's current playback position (e.g., Group ID or time offset (PTS) in the video).

2. *Current Global Time*: The client's global timestamp, allowing the relay (and/or Leader) to account for the delay in message transmission.

4.1.1. New Client Announcement

When a new client joins the session, the *first* heartbeat message includes an additional "New-Client-Join" indicator. This allows the relay (and indirectly the Leader) to:

* Recognize a new participant in the session.

* Potentially notify other participants (if needed).

4.1.2. Heartbeat Timeouts and Disconnections

Each client is expected to send heartbeat messages within a pre-decided interval. If a heartbeat is not received for a duration exceeding that interval (plus some tolerance), the relay assumes the client has become inactive or disconnected. This triggers either:

* *Temporary Removal*: The relay marks the client as "inactive" but does not remove them from the session state immediately.

* *Full Removal*: If no further heartbeat arrives within a longer grace period, the client is marked as "disconnected" and removed from the synchronized group.

4.2. Out of Sync and Re-Syncing

A client may decide to pause or seek independently of the group or otherwise desynchronize its playback. For instance, a user might rewind to re-watch a scene without causing the entire group to follow. Such actions are handled as follows:

1. *Local-Only Rewind/Seek*:

The client can apply a local override of its playback buffer. It continues to send heartbeats but indicates via a flag (e.g., “Out-of-Sync”) that it is not following the group.

2. *Resuming Sync*:

Once the client is ready to rejoin synchronized playback, it sends a heartbeat with a special “Re-Sync Request” flag or re-subscribes to the Sync-Track. The Leader (or relay) then supplies the necessary offset or group position so the client can jump to the current playback point and synchronize with others.

4.3. Handling Network Issues and Group Policy

Clients experiencing connection issues (e.g., low bandwidth or high latency) can degrade the overall group experience if every other client must wait. To mitigate this:

1. *Adaptive Bitrate (ABR) at the Relay*:

The relay can detect from heartbeat messages that a client is consistently behind or losing packets. It may choose to deliver lower-resolution encodings to that specific client (if available) to help it catch up.

2. *Lobby Policy: Wait vs. Continue*:

An initial session-level policy can determine if the group is willing to wait for slow clients or not. If the policy is set to “continue,” the lagging client might partially skip forward (or remain unsynchronized) until network conditions improve. If the policy is set to “wait,” the group collectively applies additional buffering or pause states to allow the slow client to keep up.

4.4. Fetch-Based Architecture

The current design builds on MOQ’s push-based content delivery. The current revision introduces a more scalable, fetch-oriented mechanism for client synchronization. This approach will further refine:

* The structure of heartbeat/status messages.

* How the group membership is signaled and updated.

* Interaction between “push” and “fetch” paradigms for adaptive VoD synchronization.

* More effective use of relays for more scalable systems.

5. Security Considerations

TODO.

6. IANA Considerations

TODO.

7. References

7.1. Normative References

[MoQTransport] Curley, L., Pugin, K., Nandakumar, S., Vasiliev, V., and I. Swett, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-08, 12 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-08>.

[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>.

7.2. Informative References

[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>.

Appendix A. Example Publisher Model

This section defines an example architecture for adapting MOQT to VoD use cases. The model enables efficient retrieval and streaming of archived fragmented MP4 (fMP4) content through a specialized publisher (moq-pub), while maintaining compatibility with live-edge workflows.

A.1. Media Preparation

Encoding: Media is encoded in H.264 (AVC) format.

Containerization: Media is packaged in fragmented MP4 (fMP4) format, where content is divided into discrete "moof" (Movie Fragment) and "mdat" (Media Data) atom pairs.

Initialization Segments: The ftyp (File Type) and moov (Movie Metadata) atoms are prepended to the media file. These provide global metadata (e.g., codec information, track layouts) and are transmitted once per session.

A.2. Publisher (moq-pub) Responsibilities

Storage Integration: Stream archived fMP4 content directly from original publisher's local storage, eliminating the need for real-time transcoding.

Efficient Retrieval of Media: Support arbitrary access to media fragments (moof+mdat pairs) for low-latency seek operations.

Media Codec: moq-pub supports H.264 encapsulated in fMP4.

A.3. Streaming Mechanism

Atomic Units: Media is streamed as sequential "moof+mdat" atom pairs, each representing a MOQT _object_. The moof atom precedes its corresponding mdat atom to ensure decodability.

Group Mapping: Each moof+mdat pair corresponds to a frame in the H.264 stream. A group of frames form a Group of Pictures (GOP). A GOP begins with an intra-coded frame (I-frame), followed by predictively-encoded frames (P/B-frames). The Group ID parameter in MOQT aligns with the GOP index in H.264. This ensures that seeking to a specific Group ID starts playback from the I-frame at the beginning of the requested GOP, enabling correct decoder initialization. The original publisher maintains an internal index mapping Group ID to the byte offset of the corresponding GOP's moof+mdat pair.

A.3.1. Session Initialization

At the start of a streaming session:

1. The original publisher transmits the ftyp and moov atoms as an initialization segment.

2. Subsequent objects (moof+mdat pairs) are transmitted incrementally, adhering to the subscriber’ s playback state.

A.4. Dynamic Playback Control

The original publisher asynchronously monitors a dedicated Sync-Track for control messages (e.g., PLAY, PAUSE, SEEK) issued by the _Leader Client_.

A.4.1. Command Processing

Seek Operations: On receiving a SEEK command:

1. The original publisher identifies the start of the nearest GOP boundary for the requested Group ID, and resumes publishing from the corresponding frame.

2. Streaming resumes from the start of the GOP containing the target I-frame, ensuring all dependent P/B-frames are included for seamless decoding.

3. Subscribers receive the full GOP sequence, eliminating decoder errors caused by missing reference frames. Pause/Resume:

A PAUSE command halts the transmission according to the Group ID specified in the PAUSE message, where the Group ID corresponds to the current group of the playback. A subsequent PLAY command resumes streaming from the next sequential object.

Authors’ Addresses

Ahmet Pehlivanoglu
Ozyegin University
Email: ahmet.pehlivanoglu@ozu.edu.tr

Kerem Bekmez
Ozyegin University
Email: kerem.bekmez@ozu.edu.tr

Basar Yumakogullari
Ozyegin University
Email: basar.yumakogullari@ozu.edu.tr

Zafer Gurel
Ozyegin University
Email: zafer.gurel@ozu.edu.tr

Ali Begen
Ozyegin University
Email: ali.begen@networked.media