moq
Internet-Draft
Intended status: Informational
Expires: 23 January 2026

L. Curley
Discord

22 July 2025

Media over QUIC - Use Cases
draft-lcurley-moq-use-cases-01

Abstract

MoQ is designed to serve live tracks over a CDN to viewers with varying latency and quality targets: the entire spectrum between real-time and VOD. However, it's difficult to understand how to use the transport given the layering and complexity of live media delivery. This document outlines how an application could use MoQ to deliver video, audio, and metadata in a variety of scenarios.

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/kixelated/moq-drafts.

Status of This Memo

Table of Contents

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.

2. Introduction

Media over QUIC is still in the concept phase; a loose collection of ideas and drafts on how to utilize QUIC for live media delivery. It’s difficult to grasp how to utilize the various layers:

* QUIC: A transport layer that provides reliable, ordered, and secure streams.

* moq-lite: A pub/sub layer that provides caching and fanout.

* moq-karp: A proposed media layer on top of moq-lite that deals with encoding and containers.

* Application: Your application that utilizes any of the above layers.

This document briefly overviews how live media works and how you could use MoQ to deliver it.

3. Video

Video encoding involves complex dependencies between frames/slices. The terminology in this section stems from H.264 but is applicable to most modern codecs.

Each frame of video is encoded as one or more slices but to simplify the discussion, we’ll refer to a slice as a frame. There are three types of frames:

* *I-Frame*: A frame that can be decoded independently.

* *P-Frame*: A frame that depends on previous frames.

* *B-Frame*: A frame that depends on previous or future frames.

3.1. Group of Pictures

A simple application can ignore the complexity of P/B frames and focus on I-Frames. This is the optimal approach for many encoding configurations.

Each I-Frame begins a Group of Pictures (GoP). A GoP is a set of frames that MAY depend on each other and MUST NOT depend on other GoPs. Each frame has a decode order (DTS) and a frame MUST NOT depend on frames with a higher DTS.

This perfectly maps to a QUIC stream, as they too are independent and ordered. The easiest way to use moq-lite is to send each GoP as a GROUP with each frame as a FRAME, hence the names.

Each SUBSCRIBE starts at a Group to ensure that it starts with an I-Frame. Each Group is delivered in decode order ensuring that all frames are decodable (no artifacts).

A subscriber can choose the Group Order based on the desired user experience:

* SUBSCRIBE order=DESC: Transmits new Groups first to allow skipping, intended for low-latency live streams.

* SUBSCRIBE order=ASC: Transmits old Groups first to avoid skipping, intended for VOD and reliable live streams.

A publisher or subscriber can skip the remainder of a Group by resetting a Group Stream or by issuing a SUBSCRIBE_UPDATE. A FETCH can be used to recover any partial groups.

3.2. Layers

An advanced application can subdivide a GoP into layers.

The most comprehensive way to do this is with Scalable Video Coding (SVC). There is a base layer and one or more enhancement layers that depend on lower layers. For example, a 4K stream could be broken into 4K, 1080p, and 360p (base) layers. However, SVC has limited support and is complex to encode.

Another approach is to use temporal scalability via something like B-pyramids. The frames within a GoP are sub-divided based on their dependencies, intentionally creating a hierarchy. For example, even frames could be prevented from referencing odd frames, creating a base 30fps layer and an enhancement 60fps layer. This is effectively a custom SVC scheme however it's limited to time and doesn't require special decoder support.

The purpose of these layers is to support degrading the quality of the broadcast. A subscriber could limit bandwidth usage by choose to only receive the base layer or a subset of the enhancements layers. During congestion, the base layer can be prioritized while the enhancement layers can be deprioritized or dropped. However, the cost is a small increase in bitrate (10%) as limiting potential references can only hurt the compression ratio.

When using moq-lite, each layer is delivered as a separate Track. This allows the subscriber to choose which layers to receive and how to prioritize them in SUBSCRIBE. It also enables layers to be prioritized within the layer application, for example Alice's base layer is more important than Bob's enhancement layer.

The application is responsible for determining the relationship between layers, since they're unrelated tracks as MoqTransport is concerned. The application could use a catalog to advertise the layers and how to synchronize them, for example based on the Group Sequence.

3.3. Non-Reference Frames

While not explicitly stated, I believe the complexity in MoqTransport stems almost entirely from a single use-case: the ability to drop individual non-reference frames in the middle of a group.

In theory, transmitting enhancement layers as tracks like mentioned above could introduce head-of-line blocking depending on the encoding. This would occur when enhancement layers are not self-referential, a rare configuration which also hurts the compression ratio. And in practice, there's no discernible user impact given the disproportionate size difference between base and enhancement layers.

The ability to drop individual non-reference frames in the middle of a group is an explicit non-goal for moq-lite.

4. Audio

Unlike video, audio is simple and yet has perhaps more potential for optimization.

4.1. Frames

Audio samples are very small and for the sake of compression, are grouped into a frame. This depends on the codec and the sample rate but each frame is typically 10-50ms of audio.

Audio frames are independent, which means they map nicely to moq-lite Groups. Each audio frame can be transmitted as a GROUP with a single FRAME.

4.2. Groups

Audio FRAMEs can also be combined into periodic GROUPs to reduce overhead at the cost of some introducing head-of-line blocking. This won't increase latency except under significant congestion as each FRAME is still streamed.

For example, an application could then subscribe to video and audio starting at group X for both tracks, instead of trying to maintain a mapping between the two based on timestamp. This is quite common in HLS/DASH as there's no reason to subdivide audio segments at frame boundaries.

4.3. FEC

Real-time audio applications often use Forward Error Correction (FEC) to conceal packet loss. Audio frames are a good candidate for FEC given that they are small and independent.

In an ideal world, FEC would be performed by QUIC based on the properties of the hop. However this is not currently not supported and FEC is left to the application.

In moq-lite, each FEC packet is transmitted as a separate GROUP with a single FRAME. This means using QUIC streams instead of QUIC datagrams to automatically adjust to the viewer's RTT, automatically retransmitting in scenarios where RTT is small. Lost packets will be retransmitted unless a real-time subscriber updates the subscription to skip them. For example, if group 3 and group 5 are used to reconstruct group 4, then the subscriber can update the subscription to start at group 5, causing group 4 to be skipped.

This round trip of feedback to avoid retransmissions is not ideal but keep in mind that Group Order = DESC will be respected, meaning group 4 won't be (re)transmitted if there's more important data to send. This can result in wasted bandwidth versus something like a timeout on the sender, but it does not impact the user experience nor are these unnecessarily retransmittted audio FEC packets large enough to matter.

5. Metadata

There's a number of non-media use cases that can be served by moq-lite.

5.1. Catalog

Originally part of the transport itself, the catalog is a list of all tracks within a broadcast. It's since been delegated to the application and is now just another track with a well-known name.

The proposed MoQ catalog format supports live updates. It does this by encoding a base JSON blob and then applying JSON patches over time. If the number of deltas becomes too large, the producer can start over with a new base JSON blob.

In moq-lite, the base and all deltas are a single GROUP. The base is the first FRAME and all deltas are subsequent FRAMEs. The producer can create a new GROUP to start over, repeating the process.

5.2. Timeline

Another track that is commonly pitched is a timeline track. This records the presentation timestamp of each Group, giving a VOD viewer to seek to a specific time.

The timeline track is a single Group containing a Frame for each timestamp. The live nature of the timeline track is great for DVR applications while being concise enough for VOD. Timed metadata would use a similar approach or perhaps leverage this track.

5.3. Interaction

Another common use-case is to transmit user interactions, such as controller inputs or chat messages. It's up to the application to determine the format and encoding of these messages.

Let's take controller input as an example. The application needs to determine its loss vs latency tolerance, as reordering or dropping inputs will lead to a poor user experience.

* If you don’t want loss, then use a single GROUP with a FRAME per input.

* If you don’t want latency, then use a GROUP per input with a single FRAME.

* If you want a hybrid, then use form of clustering inputs into GROUPs and FRAMEs based on time.

A publisher could monitor the session RTT or stream acknowledgements to get a sense of the latency and create Groups accordingly. However, this only applies to the first hop and won’t be applicable when relays are involved.

6. Latency

One explicit goal of moq-lite is to support multiple latency targets.

This is accomplished by using the same Tracks and Group for all viewers, but slightly changing the behavior based on the subscription. This is driven by the subscriber, allowing them to choose the trade-off between latency and reliability. This may be done on the fly via SUBSCRIBE_UPDATE, for example if a high-latency viewer wishes to join the stage and suddenly needs real-time latency.

The below examples assume one audio and one video track. See the next section for more complicated broadcasts.

6.1. Real-Time

Real-time latency is accomplished by prioritizing the most important media during congestion and skipping the rest.

This is slightly different from other media protocols which instead opt to drop packets. The end result is similar, but prioritization means utilizing all available bandwidth as determined by the congestion controller. A subscriber or publisher can reset groups to avoid wasting bandwidth on old data.

A real-time viewer could issue:

```
SUBSCRIBE track=audio priority=1 order=DESC
SUBSCRIBE track=video priority=0 order=DESC
```

In this example, audio is higher priority than video, and newer groups are higher priority than older groups. Suppose a viewer fell behind after a burst of congestion and has to decide which groups to deliver next. This configuration would result in the transmission order:

```
GROUP track=audio sequence=102
GROUP track=audio sequence=101
GROUP track=audio sequence=100
GROUP track=video sequence=5
GROUP track=video sequence=4
```

The user experience depends on the amount of congestion:

* If there’s no congestion, all audio and video is delivered.

* If there’s moderate congestion, the tail of the old video group is dropped.

* If there’s severe congestion, all video will be late/dropped and some audio groups/frames will be dropped.

6.2. Unreliable Live

Unreliable live is a term I made up. Basically we want low latency, but we don’t need it at all costs and we’re willing to skip some video to achieve it. This is useful for broadcasts where latency is important but so is picture quality.

An unreliable live viewer could issue:

```
SUBSCRIBE track=audio priority=1 order=ASC
SUBSCRIBE track=video priority=0 order=DESC
```

This example is different from the real-time one in that audio is reliable and delivered in order. Of course this is optional and up to the application, as it will result in buffering during significant congestion. If the viewer goes through a tunnel and then comes back online, they won’t miss any audio.

Video will be delivered out of order but the player can maintain a jitter buffer. For example, it could hold onto up to 3s of video frames in memory so it can tolerate an equal amount of congestion. If the buffer fills up, the player can STOP_SENDING any old groups and/or update the subscription to skip them.

6.3. Reliable Live

Reliable live is another term I made up. This is when we have a live stream but primarily care about picture quality. A good example is a sports game where you want to see every frame.

A reliable live viewer could issue:

SUBSCRIBE track=audio priority=0 order=ASC
SUBSCRIBE track=video priority=0 order=ASC

This will deliver both audio and video in order, and with the same priority. The viewer won't miss any content unless the publisher resets a group. However, this can result in buffering during congestion and provides a similar user experience to HLS/DASH.

6.4. VOD / DVR

Video on Demand (VOD) and Digital Video Recorder (DVR) both involve seeking backwards in a live stream. moq-lite can serve this use-case too, don't worry.

A VOD viewer could issue:

SUBSCRIBE track=audio priority=0 order=ASC start=345 end=396
SUBSCRIBE track=video priority=0 order=ASC start=123 end=134

The application is responsible for determining the group sequence numbers based on the desired timestamp. This could be done via a timeline track or out-of-band.

A subscriber will need a specific end or else it will download too much data at once, as old media is transmitted at network speed and not encode speed. It will need to issue an updated SUBSCRIBE to expand the range as playback continues and the buffer depletes. A subscriber could use SUBSCRIBE_UPDATE, however there are race conditions involved.

A DVR player does the same thing but can automatically support joining the live stream. It's perfectly valid to specify a end in the future and it will behave like reliable live viewer once it reaches the live playhead.

Alternatively, a DVR player could prefetch the live playhead by issuing a parallel SUBSCRIBE at a lower priority. This would allow playback to immediately continue after clicking the "Go Live" button, canceling or deprioritizing the VOD subscription.

```
SUBSCRIBE track=video priority=1 order=ASC start=123 end=134
SUBSCRIBE track=video priority=0 order=DESC
```

6.5. Upstream

All of these separate viewers could be watching the same broadcast. How is a relay supposed to fetch the content from upstream?

moq-lite addresses this by providing the publisher's Track Priority and Group Order in the INFO message. This is the intended behavior for the first hop and dictates which viewers are preferred.

For example, suppose the producer chooses:

```
INFO track=audio priority=1 order=DESC
INFO track=video priority=0 order=DESC
```

If Alice is watching a VOD and issues:

```
SUBSCRIBE track=audio priority=0 order=ASC
SUBSCRIBE track=video priority=0 order=ASC
```

If Bob is watching real-time and issues:

```
SUBSCRIBE track=audio priority=1 order=DESC
SUBSCRIBE track=video priority=0 order=DESC
```

For any congestion on the first mile, then the relay will improve Bob's experience by following the producer's preference. However any congestion on the last mile will always use the viewer's preference.

A relay should use the publisher's priority/order only when there's a conflict. If viewers have the same priority/order, then the relay should use the viewer's preference and it can always issue a SUBSCRIBE_UPDATE when this changes.

7. Broadcast

A broadcast is a collection of tracks from a single producer. This usually includes an audio track and/or a video track, but there are reasons to have more than that. This is transparent to moq-lite, as the a higher level application is responsible for any grouping between tracks.

7.1. ABR

Virtually all mass fan-out use-cases rely on Adaptive Bitrate (ABR) streaming. The idea is to encode the same content at multiple bitrates and resolutions, allowing the viewer to choose based on their unique situation.

moq-lite unsurprisingly supports this via multiple Tracks, but relies on the application to determine the relationship between them. This is often done via a catalog track that details each track's name, bitrate, resolution, and codec. This includes how a group in one track corresponds to a group in another track. A common approach is to use the same Group Sequence number for all tracks, or perhaps utilize a timeline track to map between Group Sequences and presentation timestamps.

The viewer may limit the available tracks based on capabilities or preferences. For example, the device may not support the 4K track since it uses AV1, or the screen size may be too small to justify the bandwidth. This is easy enough to support; just ignore these tracks in the catalog.

The primary reason to use ABR is to adapt to changing network conditions. The viewer learns about the estimated bandwidth via the SESSION_UPDATE message, or by measuring network traffic and can then choose the appropriate track based on bitrate.

Transitioning between tracks can be done seamlessly by utilizing prioritization. For example, suppose a viewer is watching the 360p track and wants to switch to 1080p at group 69.

A real-time or unreliable live viewer could issue:

```
SUBSCRIBE_UPDATE track=360p  priority=0 order=DESC end=69
SUBSCRIBE        track=1080p priority=1 order=DESC start=69
```

A reliable live or VOD viewer could issue:

```
SUBSCRIBE_UPDATE track=360p  priority=1 order=ASC end=69
SUBSCRIBE        track=1080p priority=0 order=ASC start=69
```

The difference between them is whether to prioritize the old track or the new track. In both scenarios, the subscription will seamlessly switch at group 69 even if it's seconds in the future. The same behavior can be used to switch down.

7.2. SVC

We touched on SVC before, but it's worth mentioning as an alternative to ABR. I want to see it used more often but I doubt it will be.

Instead of choosing the track based on the bitrate, the viewer subscribes to them all:

```
SUBSCRIBE track=360p  priority=2 order=DESC
SUBSCRIBE track=1080p priority=1 order=DESC
SUBSCRIBE track=4k    priority=0 order=DESC
```

During congestion, the 4k enhancement layer will be deprioritized followed by the 1080p enhancement layer. This is a more efficient use of bandwidth than ABR, but it requires more complex encoding.

8. Conferences

Some applications involve multiple producers, such as a conference calls or a live events. Even though these are separate broadcasts from potentially separate origins, moq-lite can still serve them over the same session.

8.1. Discovery

The first step to joining a conference is to discover the available tracks.

That's where ANNOUNCE comes in. The subscriber indicates interest in any tracks that start with a prefix, such as ["meeting", "1234"]. As new participants join, new tracks are announced, and the subscriber can choose to subscribe to them.

8.2. Participants

Extending the idea that audio is more important than video, we can prioritize tracks regardless of the source. This works because SUBSCRIBE priority is scoped to the session and not the broadcast.

```
SUBSCRIBE track=[alice, audio] priority=3
SUBSCRIBE track=[frank, audio] priority=3
SUBSCRIBE track=[alice, video] priority=1
SUBSCRIBE track=[frank, video] priority=1
```

When Alice starts talking or is focused, we can actually issue a SUBSCRIBE_UPDATE to increase her priority:

```
SUBSCRIBE_UPDATE track=[alice, audio] priority=2
SUBSCRIBE_UPDATE track=[frank, video] priority=0
```

Note that audio is still more important than video, but Alice is now more important than Frank. (poor Frank)

This concept can further be extended to work with SVC or ABR:

```
SUBSCRIBE track=[alice, 360p] priority=4
SUBSCRIBE track=[frank, 360p] priority=3
SUBSCRIBE track=[alice, 720p] priority=2
SUBSCRIBE track=[frank, 720p] priority=1
```

9. Security Considerations

TODO Security

10. IANA Considerations

This document has no IANA actions.

11. Normative References

[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>.

Acknowledgments

TODO acknowledge.

Author’s Address

Luke Curley
Discord
Email: kixelated@gmail.com