Media Over QUIC
Internet-Draft
Intended status: Standards Track
Expires: 4 October 2026

M. Duke
Google

2 April 2026

The Rewind Subscription Filter
draft-duke-moq-subscribe-rewind-02

Abstract

This document proposes a Media Over Quic Transport (MOQT) extension that enables a new Subscription Filter, so that a subscriber can request that a finite number of past groups be delivered with SUBSCRIBE semantics (multiple streams, potentially incomplete) rather than FETCH semantics (single stream, complete, head-of-line-blocking). Service of this request is best-effort by the publisher, and it intended to accelerate joining a track in some use cases.

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://martinduke.github.io/draft-duke-moq-subscribe-rewind/draft-duke-moq-subscribe-rewind.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-duke-moq-subscribe-rewind/.

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/martinduke/draft-duke-moq-subscribe-rewind.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

In MOQT [MOQT], tracks are delivered via atomic Objects that are organized into Groups, which serve as join points, and Subgroups, that imply dependency between Objects and serve as the units to be fed into QUIC or Webtransport streams.

Subscribers can send SUBSCRIBE messages to receive objects that arrive at, or are created by, the publisher in the future. These objects are delivered either one stream per Subgroup, or in QUIC Datagrams, as dictated by the original publisher. The stream mapping allows Objects that are no longer of use to the subscriber to not be sent or retransmitted without blocking later Objects.

Subscribers can also send FETCH messages to retrieve Objects from the past. The requested Object range is delivered on a single stream. A relay might omit certain objects if they are not available in cache, but these are delivered without regard for the dependencies represented by Subgroups. If the entirety of the Object range is not in cache, a relay will have to issue its own FETCH upstream to satisfy the subscriber.

Because the subscriber may not know the live edge at request time, a variant of FETCH known as "Joining FETCH" instructs the publisher to use the current live edge as the end of the Object range. A "Relative Joining FETCH" defines the start of the Object range relative to the live edge. For instance, a Relative Joining FETCH might request two Groups prior to the live edge, which would deliver the two latest complete Groups as well as all Objects in the current Group before the live edge. Joining FETCH uses the same delivery semantics as other FETCH: all Objects are delivered in order on a single stream.

In some use cases, this behavior is not optimal. The subscriber might not need the delivery guarantees associated with FETCH if Objects will arrive too late to be useful. Furthermore, if some of the FETCHed Objects are available in cache, they might have to wait for other, blocking Objects to be delivered from upstream.

This document describes the Subscribe Rewind extension, which specifies a new Subscription Filter type "Rewind", allowing the subscriber to request SUBSCRIBE semantics for some groups before the live edge. The publisher only honors the request if it is able to do so from cache.

2. Conventions and Definitions

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

3. Overview

Each endpoint sends the MAX_REWIND option in its SETUP message. The MAX_REWIND option contains an integer that indicates the maximum number of groups that the peer might request in a Rewind subscription filter. If zero, the peer may only request the current group. If absent, the peer MUST NOT send the Rewind subscription filter. This option is half-duplex; if an endpoint does not send the option, but receives it, it MAY use the Rewind Subscription Filter.

An endpoint might populate the MAX_REWIND option by reporting how many Groups it habitually stores in cache to answer FETCH or service subscribers that are behind on delivery. Other heuristics are also possible, especially if Group IDs do not increment by one.

A subscriber sends a SUBSCRIBE message and can include a Rewind Subscription Filter instead of some other Subscription filter type. The filter contains a single argument. If this argument has a value of zero, it indicates it would like to receive the entire current group; a larger value indicates it would also like to receive the most recent complete groups as well.

If the subscriber wants the Groups even if SUBSCRIBE delivery semantics are not available, it MAY also send a Joining FETCH message, as described below. The object range MAY be larger or smaller than specified in the Rewind filter.

Upon receipt of a Rewind filter, the publisher MAY treat it as a Next Group filter. It will typically do so if the track is not in cache. If it does not do so, it sends a REWIND_GROUPS parameter in the SUBSCRIBE_OK. REWIND_GROUPS is an integer that indicates the number of Groups before the LargestObject parameter that will be served via SUBSCRIBE, which will not be greater than the value in the Rewind filter.

Groups included in the Rewind Groups range will be delivered using SUBSCRIBE semantics: datagrams or Subgroup streams, subject to the delivery timeout and group order specified in the SUBSCRIBE negotiation. For any Joining FETCH(es) associated with this SUBSCRIBE, the Joining Location will be revised to an Object prior to the groups serviced via Rewind. In some cases, this means the Joining FETCH delivers an empty range.

If the Joining FETCH range exceeds the Rewind range, the EndLocation reported in FETCH_OK is the highest Group ID outside the Rewind range, which is expressed as Location {Rewind.Lowest_Group - 1, 0}.

Note that if the original publisher does not publish Groups with monotonically increasing Group IDs, REWIND filters with values greater than zero will have unpredictable results; like FETCH requests that request more than one group, subscribers SHOULD NOT issue these requests.

4. Publisher restrictions

If the SUBSCRIBE message includes the FORWARD parameter with value 0, the publisher MUST NOT send the REWIND_GROUPS parameter in SUBSCRIBE_OK.

The publisher MUST NOT include a Group in a range defined by Rewind Groups unless:

* There are objects in cache for the Group, and

* Some objects in cache have either Datagram forwarding preference, or are known to constitute the beginning of a Subgroup;

* The DELIVERY_TIMEOUT parameters for the SUBSCRIBE indicate the Group can still be sent (the Object was cached within the minimum of the two parameters), and

* The MAX_CACHE_DURATION property for the track indicates the Group can still be sent.

The publisher is not required to verify that it has all objects in the Group to include it in Rewind Groups range. In particular, Groups delivered via SUBSCRIBE might be missing objects and are still eligible for Rewind.

The publisher MAY choose to report fewer groups than what meet these conditions. It might do so because the volume of data implied would consume too many resources, because it knows the current group is about to end, or due to any other policy.

As with any other SUBSCRIBE, if a publisher receives two streams for the same Subgroup from upstream, and cannot account for all object IDs between the end of one and the beginning of another, it MUST NOT deliver them on the same stream. It MAY simply omit the stream with higher object IDs.

A simple way to implement a relay that supports Rewind is to begin with the first full group in active subscription (i.e. where the subscription did not start mid-Group).

However, a relay MAY use data from previous subscriptions to the track, but must take care to avoid incorrectly implying Subgroup continuity, as described above. Furthermore, a relay MAY use FETCH to retrieve missing objects and deliver them via subgroup streams and datagrams, since this will not block objects that are already deliverable from cache.

4.1. Relays with No Existing Upstream Subscription

If a relay does not have an existing upstream subscription for the track, it SHOULD use a Rewind Groups filter with the same or larger value in its upstream SUBSCRIBE, subject to the upstream's MAX_REWIND Setup Option. When it receives a SUBSCRIBE_OK from upstream, it SHOULD forward the REWIND_GROUPS parameter to any Subscriber(s) that sent a Rewind Groups filter. It MAY supplement with additional contiguous Groups in cache.

However, the relay MUST NOT send a REWIND_GROUPS parameter larger than each subscriber's original request.

4.2. Pseudocode

The following pseudocode illustrates this logic:

```
bool HasObjectInGroup(group_id) {
  for (subgroup : cache.group[group_id].subgroups) {
    if (subgroup.HasFirstObject()) {
      return true
    }
  }
  return false
}

void OnRewindFilterNeedUpstreamSubscribe(groups_to_rewind) {
  if (groups_to_rewind > kMyMaxRewind) {
    ProtocolError()
  }
  if (!Upstream.MaxRewind().exists()) {
    return
  }
  SetUpstreamSubscribeFilter(kRewind, min(groups_to_rewind,
                                          Upstream.MaxRewind())
}

void OnRewindFilterHaveSubscription(groups_to_rewind) {
  if (groups_to_rewind > kMyMaxRewind) {
    ProtocolError()
  }
```

```
  while (group = largest_observed_group;
         group >= largest_observed_group - groups_to_rewind;  --group) {
    if (!GroupExists(group) || !HasObjectInGroup(group)) {
      break
    }
  }
  if (group != largest_observed_group) {
    SetRewindGroupsParameter(largest_observed_group - group - 1)
  }
}

// called on SUBSCRIBE_OK at the relay.
void OnRewindGroupsParameterAtRelay(groups_to_rewind, largest_group) {
  // Supplement upstream response with any groups in cache
  for (group = largest_group - groups_to_rewind; group.exists(); --group) {
    if (HasObjectInGroup(group) {
      ++groups_to_rewind;
    }
  }
  for (subscriber : GetSubscribers()) {
    if (subscriber.HasRewindGroupsFilter() {
      SetRewindGroupsParameter(min(groups_to_rewind, subscriber.rewind_groups))
    }
  }
}

// Assemble the queue of objects to send.
void AssembleDataForTransmit(start_group, end_group) {
  for (group = start_group; group <= end_group; ++group) {
     for (datagram : group.datagrams) {
       Send(datagram)
     }
     for (subgroup : group.subgroups) {
       // The subgroup data structure implies that objects are in order. If
       // delivered over two streams, there will be two separate data
       // structures.
       OpenStreamForSubgroup(subgroup)
       for (object : subgroup.objects) {
         Send(object)
       }
     }
  }
}
```

5. Options and Parameters

5.1. Setup Option MAX_REWIND

In addition to the Setup Options in Sec 9.3.1 of [MOQT], the Setup Option MAX_REWIND (0x16) contains an integer that indicates the largest value that can be used in a Rewind Subscription Filter. If it is missing, the peer MUST NOT send a Rewind Subscription Filter.

5.2. Subscription Filter Rewind

In addition to the Subscription Filter Types in Sec 5.1.2. of [MOQT], add filter type Rewind (0x16). The format is as follows:

```
Subscription Filter {
  Filter Type (vi64) = 0x16,
  Rewind Groups (vi64),
}
```

A Rewind Groups of zero means that the subscriber requests SUBSCRIBE semantics from the beginning of the current group. A larger integer value includes that many past Groups in addition to the current Group.

The Rewind Groups field MUST NOT exceed the value in the peer's MAX_REWIND Setup Option and the filter type MUST NOT be sent if the Option was absent. In either, case, the publisher should terminate the session with error PROTOCOL_VIOLATION.

5.3. REWIND_GROUPS Message Parameter

In addition to the MessageParameters in Sec 9.2 of [MOQT], add REWIND_GROUPS (0x16).

It represents the number of groups before the LargestObject that will be delivered via SUBSCRIBE semantics.

If the parameter is sent in response to a Subscription Filter other than Rewind, has a value greater than the Group ID of Largest Location, or exceeeds with error PROTOCOL_VIOLATION.

The publisher MUST truncate the end of any Joining FETCH related to this SUBSCRIBE to end before Object zero of the Group encoded by REWIND_GROUPS. This might result in empty object ranges. The Subscriber MUST close the session with error PROTOCOL_VIOLATION if the ranges overlap.

6. Security Considerations

To the extent this reduces head-of-line-blocking and replaces upstream FETCHes to satisfy Joining FETCH at the relay, it can reduce resource consumption at publishers.

However, SUBSCRIBE semantics consume more QUIC or Webtransport streams. Past Groups might contain a lot of data, and FETCH delivery is contained on a single stream to simplify the flow control of this data. Publishers, who are aware of the content of their cache, SHOULD limit the range encoded by the REWIND_GROUPS parameter when is likely to overwhelm the channel or the subscriber.

7. IANA Considerations

Please add 0x16 (REWIND_GROUPS) to the Message Parameters registry.

There is no Setup Option registry, but if one arises, please add 0x16 (MAX_REWIND) to it.

8. Normative References

[MOQT] 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>.

[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>.

Acknowledgements

Ian Swett designed many components of this proposal. Numerous members of the MOQ Working Group provided comments that refined our thinking.

Author’s Address

Martin Duke
Google
Email: martin.h.duke@gmail.com