The Many Faces of SUBSCRIBE

IETF 120 MOQ WG

Current situation

SUBSCRIBE has four modes: current group, current object, absolute start, and absolute range.

SUBSCRIBE for current object is easy to understand:

- Figure out what the largest delivered object is.
- Send that object (note: unclear if the spec actually requires that).
- Forward all newer objects as they arrive.

SUBSCRIBE with absolute range: flow control

Assume that the subscriber requests (1000, 0) to (2000, 0) of a stream-per-group track. What should the publisher do?

- Should it attempt to open 1000 streams, up to how much the flow control allows, and then open the queued 900+ streams as the old ones are closed?
  - This means streams from other subscriptions are blocked too
- Should it pace the streams to only keep N streams open at a time?
  - Unclear as to when consider previous streams closed.
  - How does it propagate back pressure to backend?
- For object-per-stream, it's unclear what's "N objects from now" is.
  - For (1000, 0), "10 object in future" could be (1000, 10) or (1010, 0).
  - Can't ask backend for "next 10 objects" as a consequence.

SUBSCRIBE with absolute range: current group

Assume that the track is currently at the object (10, 1000), uses stream-per-group.

The subscriber asks for the current group. The relay sees that (10, 0) got evicted from the cache, so it asks the upstream to give it (10, 0)...(10, 1000).

It forwards that range, but by that time, it’s possible that the original publisher is really far ahead, let’s say, (10, 1500). So now the relay has to request (10, 1001)...(10, 1500).

This has to be repeated until things converge.

SUBSCRIBE with absolute range: underdefined status

1. Assume the subscriber requests (1000, 100) to (1000, 9100); the publisher knows that group 1000 only ever had 10 object to begin with. What should it do?
    a. Do not publish anything, send SUBSCRIBE_DONE
    b. Publish 9000 object with “object does not exist” status.
2. Assume stream-per-object, the subscriber requests (1000, 0) to (1000, 1000). The publisher has all even-numbered objects in the cache, but it never saw any of the odd-numbered ones. What should it do?
    a. Publish the available objects and keep track of the 500 objects it still owes the subscriber until those arrive.
    b. Send “object does not exist” for the missing ones.
    c. Resubscribe for that range to the upstream (this doesn’t solve the problem).

Discussion pause slice

Do the described problems make sense?

Assumption: common use cases

Two most common use cases:

- **Live**. Subscribe to new objects, potentially asking for some objects immediately before the live head.
- **VOD**. Fetch a range of objects from the past.

Let’s focus on the first one.

Previous attempt: SUBCRIBE and FETCH

(proposal from Denver interim)

Split SUBSCRIBE into two basic operation:

1. Subscribe, for receiving things in the future.
2. Fetch, for receiving things in the past.

This would come with an ATOMIC_SUBSCRIBE_AND_FETCH message that would work as following:

- Establish the largest sequence at the moment of the message.
- Subscribe to new
- Fetch the old with respect to the largest sequence in question.

Problems and peculiarities of SUBSCRIBE_AND_FETCH

Problem: this doesn’t actually solve the “past vs future” problem fully, because:

- Due to reordering in stream-per-object, the relay might receive new objects that are before “largest received” sequence.
- It is possible that the original publisher is still generating objects for a previous group while the new one is open.

Peculiarity: since SUBSCRIBE and FETCH are split at the current group, this might mean that the same group is bound to multiple streams.

Fix attempt: redefining Subscribe

Previous definition: Subscribe means delivering objects “in future”.

New definition: Subscribe means:

- At the original publisher: deliver only objects that are newly generated.
- At the relay: deliver only object that just arrived via a Subscribe upstream.

Fix attempt: Fetch?

Desired properties of Fetch:

- Supports arbitrary ranges
- Backpressure (due to arbitrary ranges)
- Authoritative resolution: if an object is not in the relay's cache, it has to request it from the upstream.

All are useful properties, but are not particularly useful for the common use case of "I am subscribing to a track and want a few groups from the past".

Proposed new verb: Backfill

Backfill: send the most recently received N groups if they are available in cache.

Things not in cache -> things do not arrive.

Things expire from cache -> things do not arrive.

Advantages:

- This is effectively what applications want (“subscribe as if I subscribed N groups ago”).
- Limited N means similar backpressure properties to regular SUBSCRIBE.
- Really easy to implement (everything is in memory, no going back upstream).
- Since all groups are recent, timeouts and priorities are meaningful.

Specific proposal

Add FETCH with an absolute range.

Alter SUBSCRIBE:

- Remove variants with absolute start offsets.
- Add a “most recent N groups”.
- Define “current group” and “most recent N groups” in terms of backfill semantics.