Proposal For MoQT SUBSCRIBE

Cullen, Jana, Ian
March 18, 2024
Brisbane

Agenda

1. lay out some issues with existing draft
2. explain the proposal
3. explain why we came to this proposal
4. discuss if this is good idea or how to change it

Some existing issues

Existing issues

What exactly should SUBSCRIBE send?

- Unclear what objects should be sent first #396
- How many times does an Object get sent if there are overlapping subscriptions? #269
- Unsure if/when we should use Track Alias and Subscribe ID #350
- How does sendOrder work for VOD? #326
- What is the largest object if the relay hasn't forwarded Objects #377

Existing issues

SUBSCRIBE is unclear/illegal

- A number of start and end values are illegal
- A number of start and end values are confusing #312 or undefined #353
- A subscription can have no objects in it #296
- A relay can’t optimize for typical use cases with such broad functionality

The proposed design

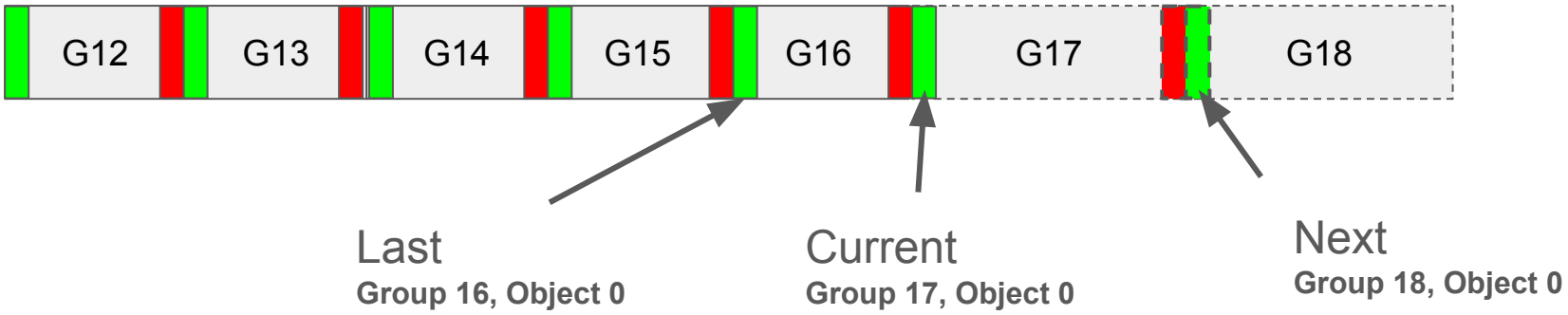
Subscribe

Can select one of the following join points in a Track:

Current: the largest received group, object 0

Next: waits until the beginning of a new group larger than the current

Last: the largest group for which the relay has received the last object



Subscribe (contd)

Creates **“state in the relay”** that tells the relay to forward objects in the track.

Can have an **optional number of groups till stop group**. This defines the stop group as increment to the group where the subscribed started. Start and Stop are treated as “Range Filter” for forwarding objects.

- A subscribe does not end when this stop group is reached; it simply does not forward any objects with a group greater than the stop group.
- Subscriptions end via an UNSUBSCRIBE / SUBSCRIBE_DONE.

Fetch

Retrieves **“absolute range”** of objects specified by start/end group/object.

- No relative requests
- No future data requests

A relay makes an upstream query for any objects not in its cache.

Relays processing multiple downstream requests SHOULD do the logic to make appropriate non-overlapping requests.

> Don’t get hung up on the name. Of course we could overload the subscribe to do this as well. More on this later.

Fetch (contd)

Fetch is a **“StateFul”** request, finds out about **“non-available objects”** in that range.

The only info that a client gets about unavailable objects is that neither the relay cache nor the upstream providers all the way to the publisher can provide a copy of the object.

Does not differentiate between never produced, not currently available, etc.

Delivery order is not guaranteed,

> If you need some part of the range first, ask for it in its own range Fetch request. A relay can send what it has in its cache while it is requesting upstream data.

If a client asks for the same object in two different Fetch requests, it gets one copy for each Fetch request. Perhaps the client lost it. The client knows why it is doing this.

> Can think of this being similar to HTTP GET.

Relay Caching

When relays cache, they SHOULD cache at least the last and current group for each track.

Why this design??

Principle

The MoQT protocol has two fairly distinct parts:

- the forwarding fan out,
- and the cache

Subscribe sets up state in the relay that tells the relay it should forward objects it receives plus a few it may already have (in case of “last”).

Fetch tells the relay to do whatever it needs to do to get a copy of the requested data.

Why “no” relative fetch

We run into hard corner cases when two receivers fetch partially overlapping relatives sets of the past.

- Hard to reason about what will happen.
- Hard for relay to know what it should ask for, especially when combined with simultaneous requests.
- Hard to debug because the “right answer” is vague at best.
- Implementation complexity → Scalability implications.

Subscribe ID or Track Alias

Fetch gets duplicate objects, so subscribe ID per transaction makes sense.

One Subscribe per Track, so no duplicates. Track alias makes sense.

(note subscribe id would change name to fetch id)

Proposal:

Use “track alias” in data from a Subscribe, and

Use “fetch id” (aka Subscribe ID) from Fetch.

Why FETCH & SUBSCRIBE vs just SUBSCRIBE

If we combine Fetch and Subscribe into one request method, we end up with needing to mark many combinations of parameters as invalid which will:

- complicates the reasoning about what happens in all the edge cases
- makes it semantically less clear to developers what they do
- complicate testing

Why nothing earlier than “last” for Subscribe

“last” allows subscriber to get a full group right away, typically from the cache.

Matches application playout buffer size.

> Applications that want to get the “last 5 seconds” when they start, are very likely to have set the group size to a similar length of time. This is not coincidence: it is because that is the optimal thing to do if the application has a playout buffer of this size.

Using Fetch to get earlier than last is not a significant performance penalty.

> Applications that want to go further back than one group can Subscribe, see the current group, and then doing a Fetch for past data they want.
>
> This extra RTT won’t typically impact performance because a group can be immediately delivered by the cache when going back more than one group.

Is this conceptually the right direction to put into the draft?