MOQT PRs and Issues 4/27

#1608 Make Subgroup ID identical to first Object Id in the Subgroup

Solves a real problem:

“Did this Subgroup start with the first Object in the Subgroup”

Rationalizes same group datagram vs subgroup prioritization tiebreaker

→ Subgroup IDs all represent an Object ID

Con:

Possible API change

Do applications need to assign meaning to subgroup IDs?

They were created to stitch Subgroups back together on disconnect/etc

Required Request ID

Receiver state is unbounded because REQUEST_UPDATE has a Request ID but doesn’t consume bidi stream

Besides Joining FETCH, are there remaining ordering use cases?

- I think so? Ordering REQUEST_UPDATE between streams?
- Note: UNSUBSCRIBE is now a QUIC level FIN/RST and is *not* orderable
  - This was billed as a feature of bidi → no HOL blocking cancellation

Remove Required Request ID Completely?

- Use a different solution for Joining FETCH
  - #1604 Explores putting Joining FETCH on SUBSCRIBE stream
    - Can no longer have independent parameters (auth, group order, subscriber priority)
    - Fate sharing questions (if one fails, kill the other)?
    - REQUEST_UPDATE/OK/ERROR ambiguity
  - Latest SWITCH proposal triggers a publisher bidi stream for FETCH if needed
- Non-starter if we need cross-stream REQUEST_UPDATE or other ordering

#1613 Add MAX_REQUEST_UPDATES setup option and TOO_MANY_REQUEST_UPDATES error

- Limits the number of outstanding REQUEST_UDPATEs per stream
- Bounds the receiver state for RRID to
  max_concurrent_streams x (1 + max_request_updates)
- Bonus: relatively simple backpressure for REQUEST_UPDATE floods (#1053)
- Cons:
  - Is max_concurrent_streams available from common transports?
  - Tracking is somewhat subtle

Victor’s Proposal #1519

Idea: move the “block on request ID” logic from the logic layer to the parser layer.

A message `WAIT_FOR x` causes the receiver to stop reading control stream until x is triggered, or timeout occurs.

A message `UNBLOCK x` causes all `WAIT_FOR x` messages to unblock.

Messages can occur on any control stream.

“We need some text about state tracking.”: WAIT_FOR have to time out due to stream resets and DoS concerns. The receiver also has to keep those around for reordering

#1605 Split DELIVERY_TIMEOUT into two types of timeout

As a motivation, consider the following two scenarios:

- No-SVC video: there is only one subgroup per group in the track; the subgroup is non-trivially long (two seconds or more).
- Subgroup-per-layer SVC: there is one subgroup per every SVC layer; for example, if there are three layers, there would be three subgroups, all lasting for the entire duration of the group.

Current timeout semantics (if one object times out, time out entire group) work well for enhancement layer of the subgroup-per-layer scenario. They don’t work well for the base layer: if it is timed out, there is nothing else to transmit.

#1605 Split DELIVERY_TIMEOUT into two types of timeout

Proposal:

- DELIVERY_TIMEOUT_OBJECT works the same way as it does today (PR clarifies semantics)
- DELIVERY_TIMEOUT_SUBGROUP: once the subgroup is finished (FIN queued), start a timer that will reset the stream.

Why FIN? If the original publisher has not sent a FIN, it may send objects that depend on previous objects -> it is worth to keep trying to deliver the subgroup.

Why support both? They are useful in different situations.

Was it a mistake to require negotiation for unknown params?

Use Case: GraphQL subscriptions (seems in the MOQT pub/sub wheelhouse)

How it works in HTTP:

```
POST /graphql
Content-Length: X

<Query Body>
```

How can you map this to MOQT?

Problem 1 - Query Body

The only subscriber input that goes to the original publisher is Full Track Name, limited to 4096 bytes

Query body *could* be a track name tuple, but might be too long

Also, your logs will be gross

Can we put query in a parameter?

→ Only if the track name is unique per response stream (e.g. `/graphql/<hash(query)>` or `/graphql/<unique_id>`)

→ Either needs to be a standard parameter (REQUEST_METADATA?) or each hop needs to negotiate it on

→ Are we sure 64k in control messages is enough for anybody?

Problem 2 - Intermediaries

In HTTP, terminating proxies (edge, origin load balancer) often add headers before forwarding upstream:

- Client IP address
- Extracted SN from Client certificate
- Via Header
- …

MOQT has no way to do this for subscriber sent requests, unless it negotiates hop-by-hop

Publishers can use Track Properties – is this our intent?

Strawman

Revert Parameter TV to TLV with even/odd (draft-16)

Use mandatory to understand range like we do for Track Properties

    Receiver MUST fail if it receives an unknown param in mandatory range

If forwarding, MUST include unknown

MUST NOT change track content for the same Track Name

Be Kind - REWIND

1) There’s a consensus call on list about REWIND specifically – please respond

2) Going to walk through the Joining FETCH Dissent issues

#861 Are Groups really join points anymore / SUBSCRIBE start on group boundaries

Yes, Groups are Join Points

See NextGroup, Joining FETCH, NGR

The issue is asking about removing Start Object from Absolute Range Filters

Not sure there’s a strong reason to remove at this point

Recommendation: Close without action

#1039 Simplifying joining at the latest available join point

Joining at the latest available point can be as simple as a library wants it to be

Several libraries expose a “Join” API that translates to SUBSCRIBE+Joining FETCH

There are two wire proposals right now

1) CurrentGroupFill
2) LargestAvailableGroup

It’s unclear if either can achieve consensus rapidly, or who is willing to do the work to convince the group.

Recommendation: No synchronous working group time until proponents build consensus on list / asynchronously. Close without action unless consensus is reached on a proposal by London.

#1313 Joining FETCH as a separate control message creates edge cases and feature gaps

This is about merging SUBSCRIBE+Joining FETCH into a single message

- #1602 covers this and #1604 is an exploration of it, but it seems fraught

Lack of FETCH_UPDATE – but now there's REQUEST_UDPATE

Issue also talks about moving SUBSCRIBE objects after largest object onto the FETCH stream, but that can also be handled via priority

Recommendation: Close as duplicate?

#1358 Subscribing to the start of the current Group could be optimized

Talks about HoL blocking if an object is missing from cache that might show up via subscription

FILL_TIMEOUT=0 mostly addresses this

Only gap is to add a Current Group *Filter* to catch late arriving objects and forward using subscribe

Recommendation: Land #1362

#1368 Can a publisher 'lie' about what Largest Object is? And does Largest Object need to exist?

This was REWIND - comment on consensus call?

There’s nothing stopping you from lying - though it’s an interop problem that some relays do it automatically and others don’t.

Recommendation: Wait for REWIND call. If no consensus there, close with no action.

#1391 Unclear how to use Joining FETCH with New Group Request

The current dance is suboptimal - send SUBSCRIBE/TRACK_STATUS, then decide (FETCH, NGR, NextGroup)

The main implementer of NGR seems ok with this

Recommendation: Close with no action