

Joining FETCH Dissent

Framing Questions - Discussion

1. Should there be a single dedicated message in the protocol, or is a dedicated API sufficient?
2. To make joining faster, can we avoid significant HoLB from lower-priority subgroups or cache misses while retaining desirable properties of FETCH during join, like back-pressure, getting all requested objects and detecting gaps in the ID space?
3. How problematic is splitting a subgroup in the current group across FETCH and SUBSCRIBE streams?
4. Is it acceptable that the Subscriber makes the decision for when to use NGR vs Joining Fetch?

Single Message vs Single API + Composite Messages

Single Message

- 1) Demonstrates clear intent (I want to join the track)
- 2) Eliminates some error cases (subscribe failed before join or id nonexistent)

Composite Messages

- 1) Less specification text and code
- 2) Combinable
 - a) joining fetch sg=0 now, join sg=1 later
 - b) rejoin after pause
 - c) Maybe this works with a single message if we spell it right?

HoLB

The main use cases are:

- 1) I want to join as soon as possible
- 2) I want to build a buffer as fast as possible to begin playback
 - a) Descending fetch is interesting, but the use-case might be more subtle?
 - b) I want to start 5s back and I want a 5s buffer but I will start playing with a 1s buffer and if G - 5 is not in cache I'd rather skip it and start a G - 4

Problems:

- 1) Missing groups/objects -> block all delivery to go upstream
 - a) Upstream might be too busy to respond
 - b) Could be a result of reordering and would arrive soon via SUBSCRIBE if you had an earlier filter?
- 2) Delivery of cached Groups/Objects can be blocked on upstream Fetch

Splitting Subgroups in the current Group

With 1 subgroup / group, joining FETCH the current group results in delivering it in at least 2 QUIC streams

- 1) It's nice if QUIC does the ordering and reassembly
BUT
- 2) In-order delivery is a lie told by a buffering receiver – can be easily implemented on top of MOQT

Subscriber vs Relay driven NGR

What should a subscriber to do join a known DYNAMIC_GROUPS track as quickly as possible?

1. Send plain SUBSCRIBE with eg: LARGEST_GROUP
2. Check LARGEST_OBJECT in SUBSCRIBE_OK. How far is the relay into the group?
 - a. Near the beginning - Send Joining FETCH
 - b. In the middle - Send REQUEST_UPDATE+NGR
 - c. Near the end - Send REQUEST_UPDATE Start=G+1

This costs 1-RTT. If the subscriber can tell the relay the boundaries between beginning, middle and end - could be faster.

Do we care?

Proposals

1. Minimal diff from -16
2. Single Message - SUBFETCH
3. Allow SUBSCRIBE to Largest-N Group
- 4.

Minimal Change

1. Keep separate Messages
2. Land #1335 - allow joining fetch for PUBLISH and after Forward 0 → 1
3. Land at least the ability to filter FETCH by subgroup ID (included in 1401 now)
4. Add a new FETCH parameter “MISS_BEHAVIOR={ Fill, Skip }”
5. Allow Joining Fetch to join SUBSCRIBEs that are not Largest Object

Leaves questions 1, 3, and 4 unchanged

Provides options for subscribers to reduce HoLB if they care:

SUB + JFETCH(G - 4, sg=0, Asc, Skip)

→ No HoLB, will get all the cached objects for sg=0

If something is missing (rare) can be re-FETCHed with Fill.

Can JFETCH sg=1 at a later time or lower pri

Approach #1: One Message

Same delivery semantics but one control message:

Objects on a Fetch response stream would have the semantics that FETCH does today.

Objects on unidirectional subgroup streams would have the semantics SUBSCRIBE does today.

Filters would still allow fetching Objects only in the past and subscribing only to Objects in the future, so no change in functionality.

Example PR [#1377](#) replaces SUBSCRIBE and FETCH with 1 Message

Approach #2: Allow SUBSCRIBE to Largest-N Group

Add a 'Go N groups back' option to SUBSCRIBE

Get identical Objects as though you had started the Subscription earlier

Keeps Fetch, removes Joining Fetch

If relay already has a Subscription, send cached Object data immediately

If not, forward the SUBSCRIBE upstream

Publishers can say 'TOO_FAR_BACK', provide an alternate starting Object

- Subscribers set how far back to limit buffering
- But you shouldn't be starting farther back than your jitter buffer anyway

PR #?????: Similar to [draft-07](#), before Joining Fetch

Approach #3: Subscription Flow Control ([#869](#))

With flow control, Objects published in the past could be delivered via Subscription

Removes the need for Joining FETCH, and possibly for FETCH entirely

Also solves the existing problem that Relays might want upstream flow control, even for SUBSCRIBE.

Example PR [#1404](#) adds MAX_SEND_LOCATION

Approach #3: Subscription Flow Control ([#869](#))

Example Use Case: A relay would like to isolate upstream subscriptions from one another

Disabled by default, enabled via Params

- Limit unidirectional streams per subscription
- Limit total data sent, similar to QUIC's MAX_DATA