Switching between streams

Gwendal Simon - Synamedia
Will Law - Akamai

Feb 2025

Motivation

Enabling Adaptive Bit-Rate (ABR) in MoQ

- Switch
  - from a subscription to track A
  - to a subscription to track B
- Maintaining the agreed subscription settings
- Minimizing switch delay (ideally switching at next group boundary)
  - Even if subscriber and/or relay lags several groups behind live due to accumulated congestion
- Avoiding throughput increase during the switch
  - Duplicates, i.e. the same Objects delivered from track A and track B
  - Concurrent subscriptions over the same wire
- Robust over a number of real-world conditions, including
  - High RTT between Subscriber and Relay and/or between Relay and Publisher
  - Short-duration groups, especially when group duration < RTT

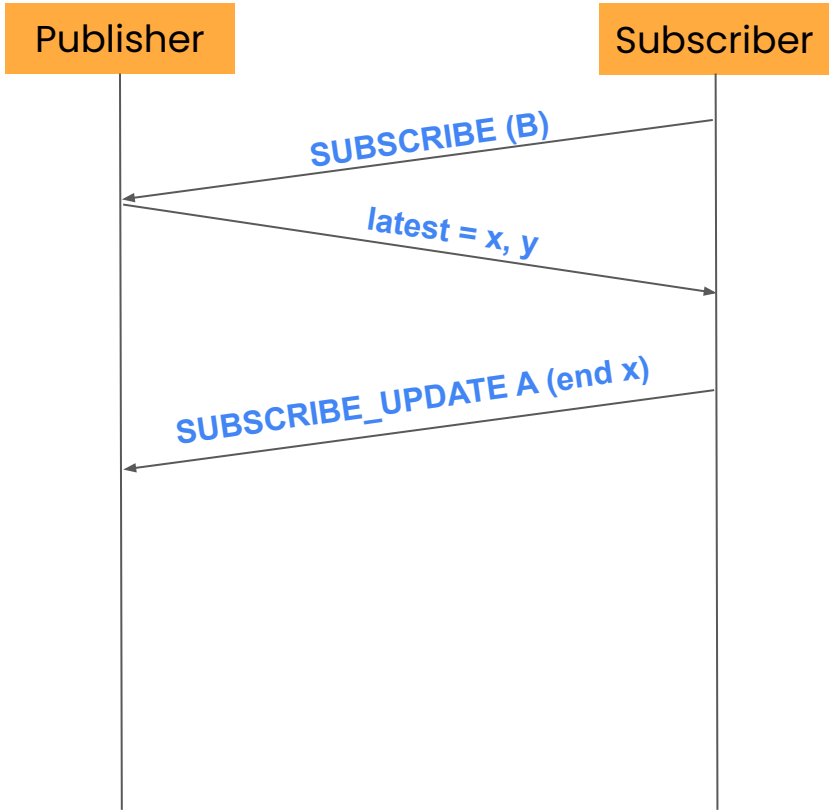
How would we switch using the existing APIs (+ #652) ?
OPTION 1: SUBSCRIBE latest

Switching from playing track A to track B

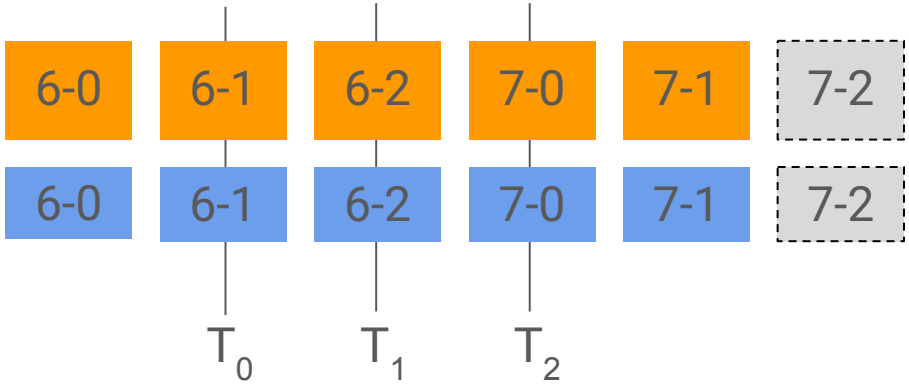
PR#652 ⇒ no risk of invalid SUBSCRIBE

However

- Some objects in group X are duplicated
- Two concurrent flows
  - The new SUBSCRIBE B
  - The end of the SUBSCRIBE A
- Late switch for lagging Subscribers



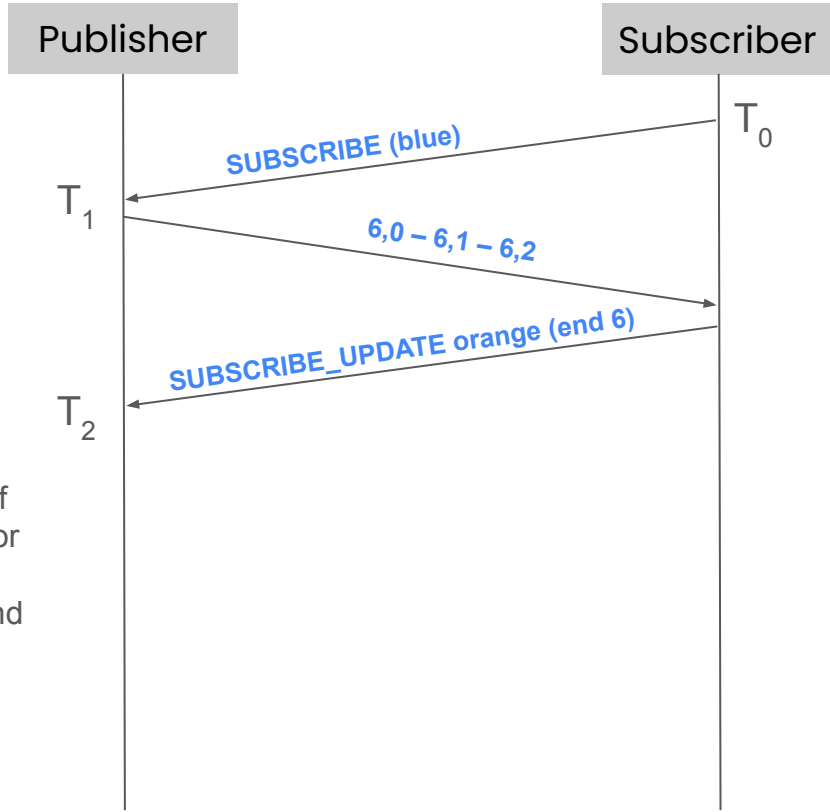
Can priority fix object duplication and concurrent flows?



Problem #1: Object duplication 6-0, 6-1, 6-2, and 7-0 in flight.

- Set a LOW priority for BLUE ⇒ Orange objects arrive ahead of blue. In case of down-switch, we download the higher bitrate for longer, which increases rebuffer risk.
- Set a HIGH priority for BLUE ⇒ Orange segments arrive behind blue. Waste of time since we cannot play Blue until the next group boundary 7-0.

Problem #2: Concurrent flows ⇒ increase of network resources, especially painful in congested conditions and down-switch

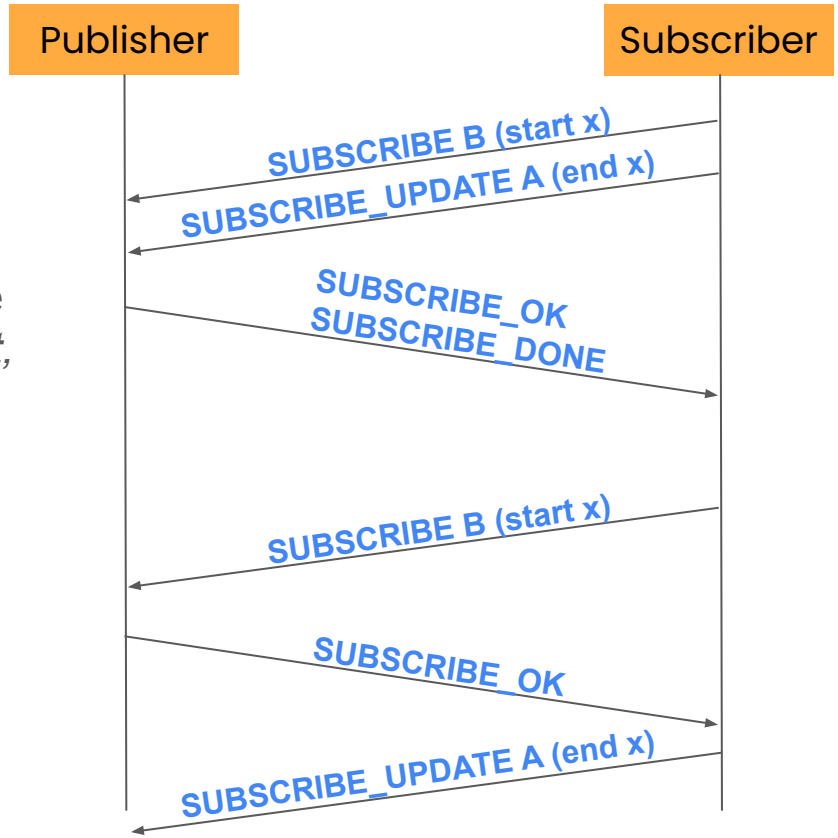


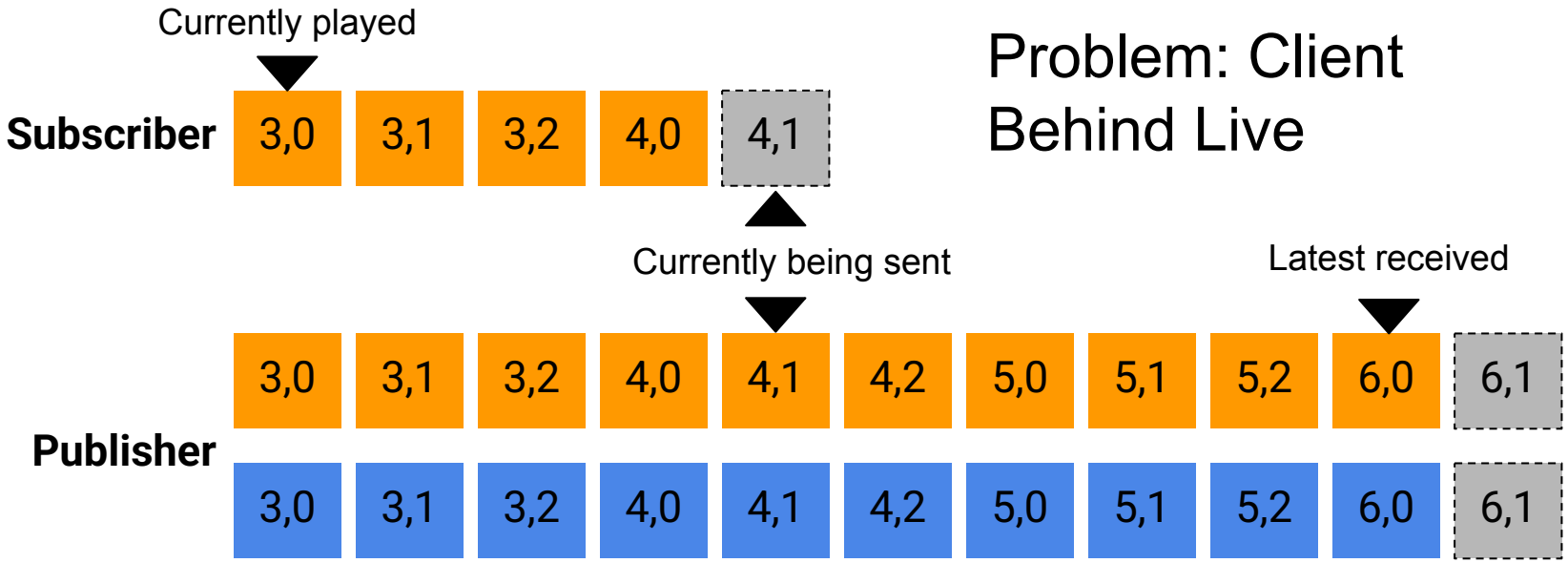
How would we switch using the existing APIs (+ #652) ?
OPTION 2: SUBSCRIBE absolute

Switching from playing track A to track B

Two variants

- Client sends both commands at the same time
  - *Main problem: The client must guess that, by the time, the relay receives the Subscribe request, **x** is either the current group or in the future.*
- Client waits for SUBSCRIBE_OK then sends SUBSCRIBE_UPDATE
  - *Main problem: duplicate objects, concurrent flows, late switch*





SUBSCRIBE B (start = 5) ⇒ Publisher sends (6,0)

A. **SUBSCRIBE_UPDATE A (end=5)** ⇒ Publisher continues (4,1) – (4,2) ⇒ **Gap group 5**

B. Wait for **SUBSCRIBE_OK B**, then **SUBSCRIBE_UPDATE A (end=6)** ⇒ **Concurrent flows** from (4,1) to (5.2) + from (6,0)

Our goal ⇒ switching at (5,0) [even (4,0) is possible], no gap, and no double concurrent SUBSCRIBE on the wire

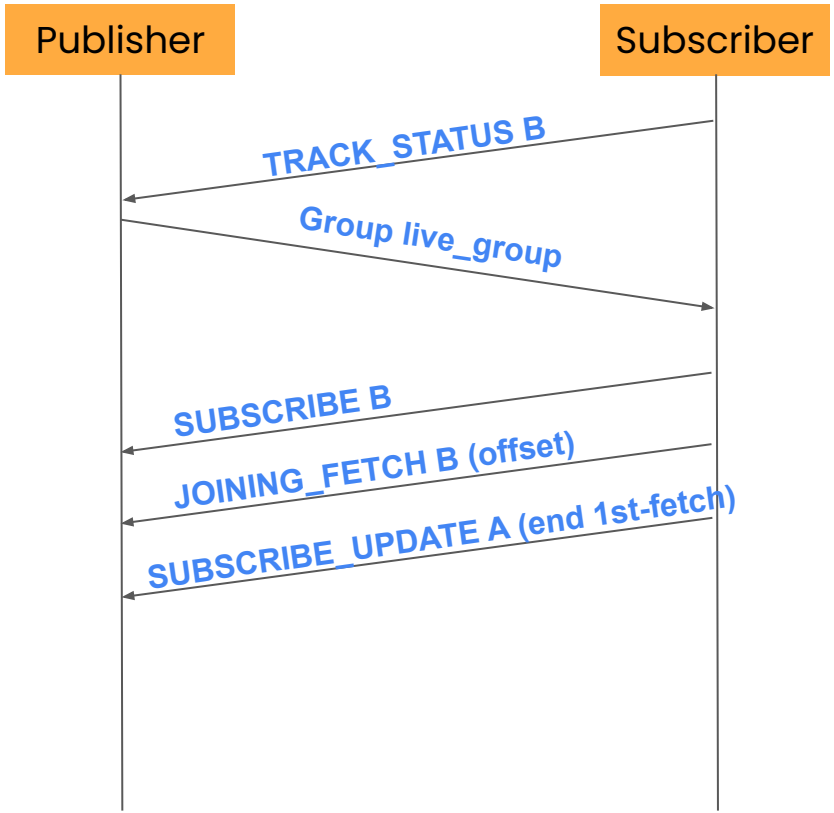
How would we switch using the existing APIs ?
OPTION 3: JOINING FETCH

Switching from playing track A to track B

1. TRACK_STATUS B to know the latest group
2. Compute difference between live_group and next_group = offset
3. SUBSCRIBE B *(low-priority)*
4. JOINING_FETCH B with offset (optional)
5. Wait for first B Object received.
6. SUBSCRIBE_UPDATE A end (first B object)
7. SUBSCRIBE_UPDATE B (*high priority*)

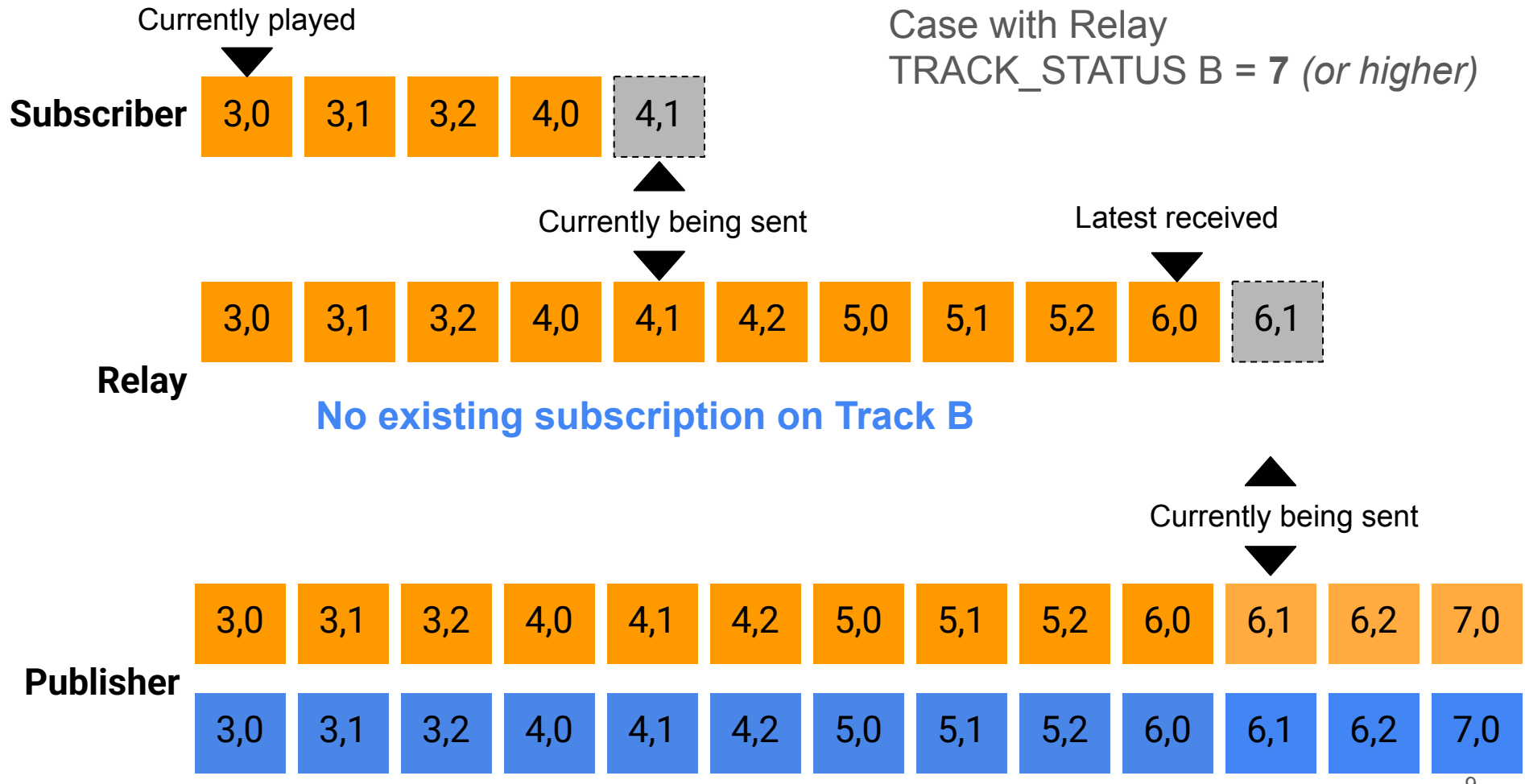
However

- 1 x RTT for Track status update
- Offset calculation in long RTT / short group



Case with Relay having track B active. Flexible join point for client.

TRACK_STATUS B gives (6,0)
Choice of switch target = *4* (it could be 5) ⇒ offset : 6 - *4* = 2
SUBSCRIBE B ⇒ Publisher sends (6,0)...
JOINING_FETCH B ascending ⇒ Publisher sends (*4*,0) - (5,2)
SUBSCRIBE_UPDATE A end=*4*


Case with Relay
TRACK_STATUS B = 7 (or higher)
Currently played
Subscriber
3,0
3,1
3,2
4,0
4,1
Currently being sent
Latest received
3,0
3,1
3,2
4,0
4,1
4,2
5,0
5,1
5,2
6,0
6,1
Relay
No existing subscription on Track B
Currently being sent
3,0
3,1
3,2
4,0
4,1
4,2
5,0
5,1
5,2
6,0
6,1
6,2
7,0
Publisher
3,0
3,1
3,2
4,0
4,1
4,2
5,0
5,1
5,2
6,0
6,1
6,2
7,0

Our Suggestions

Either use existing APIs

- Pros: no additional MOQT message, pure client-based switch
- Cons: complexity at client, no perfect solution, depend on configurations
- Remark: Adding an Absolute_Start option to JOINING_FETCH would simplify the switch and remove the 1xRTT delay in discovering offset via Track_Status.

Or define SWITCH to automate the above processes on the relay

- Pros: Clean SWITCH, simple at client, fast, no gap, no duplication
- Cons: another MOQT message, relay complication

SWITCH API proposal

The goal of this API is to avoid race-conditions and duplicate in-flight data when transitioning between tracks carrying identical content at different bitrates.

SWITCH (oldSubscriptionID, newSubscriptionID, trackAlias, newTrackName, newAuthInfo)

oldSubscriptionID: the ID of the current SUBSCRIPTION.

newSubscriptionID: the ID of the new SUBSCRIPTION on the new track

trackAlias: a session specific identifier for the new track

newTrackName: the track name you wish to switch to.

newAuthInfo: an optional string defining the auth parameter to be used for the new track. If not supplied, then any existing auth parameter for the initial subscription is reapplied.

Upon completion of the switch the relay will send either

SUBSCRIBE_OK (newSubscriptionID) + **SUBSCRIPTION_DONE** (oldSubscriptionID), or

SUBSCRIBE_ERROR (newSubscriptionID)

SWITCH API assumptions

1. The switch MUST be preceded by a SUBSCRIPTION on the same connection.
2. The switch occurs at Object 0 at the **next** mutual Group boundary. There are some possible optimizations in which an earlier group boundary may be used.
3. The GroupIDs are time-synchronized between tracks.
4. The track namespace remains consistent between the two tracks.
5. The subscription priority remains consistent.
6. The Filter Types, Delivery Timeout, and Max Cache duration are preserved.

Flow Diagram – SWITCH at Relay

- START
  - Receive SWITCH message from Subscriber
  - *[Is Relay already active in Track B?]*
    - *YES*:
      - Determine transition strategy:
        - Complete Group Switch → Switch at **next boundary**
        - Immediate Switch → Switch at **current group start**
        - Partial Cache → Switch at **earliest cached object**
      - SEND `SUBSCRIBE_OK(newSubscriptionID)` to Subscriber
      - Once transition is complete:
        - SEND `SUBSCRIBE_DONE(oldSubscriptionID)` to Subscriber
    - *NO*:
      - SEND `SUBSCRIBE(newSubscriptionID)` to Publisher
      - Wait for PUBLISHER response
      - *[Did Publisher confirm SUBSCRIBE_OK?]*
        - *YES*:
          - Determine transition approach:
            - Is fast switch needed?
              - *YES* → SEND `FETCH(newSubscriptionID, missing objects)` to Publisher
                - WAIT FOR `FETCH` response
                - Switch at **first fetched object**
              - *NO* → Switch at **first received object**
          - SEND `SUBSCRIBE_OK(newSubscriptionID)` to Subscriber
          - Once transition is complete:
            - SEND `SUBSCRIBE_DONE(oldSubscriptionID)` to Subscriber
        - *NO*:
          - SEND `SUBSCRIBE_ERROR(newSubscriptionID)` to Subscriber
  - END