June 2026

802.1CBec FRER Vector Recovery – Combined Proposal – Details

David McCall (Intel – david.mccall@intel.com)

Lisa Maile (TU/e Eindhoven University of Technology – l.t.maile@tue.nl)

Christophe Mangin (Mitsubishi Electric – c.mangin@fr.merce.mee.com)

Max Turner (Ethernovia – max.turner@ethernovia.com)

Balázs Varga (Ericsson – balazs.a.varga@ericsson.com)

Stephan Keher (Belden – stephan.kehrer.committees@gmail.com)

Venkat Anruthi (Broadcom – venkat.arunarthi@broadcom.com)

References

[1] Lisa Maile, David McCall, Christophe Mangin, Max Turner, Balazs Varga, “P802.1CBec, FRER Analysis + Solutions”, contribution to 802.1 TSN TG, May 2026

Content

- Current vs. Proposed
  - Overview
- Expected Behaviour
  - Details
- Further Work

Introduction

- Previous contributions have identified a number of issues with the current **FRER Vector Recovery Algorithms** behaviour.
- [1] summarises discussions between various individuals covering the issues, goals and potential solutions. It provides context for a combined proposal to alter **both** the **Generation** and **Recovery** algorithms, and a summary of the proposal.
- This contribution describes the proposal in more detail, highlighting the differences between it and the current version.
  - It also calls out the reasoning behind some aspects of the proposal; some limitations; plus, areas where configuration is necessary and the factors that might affect configuration choices.

Terminology & Levels of Detail

- The current **FRER Sequence Generation and Vector Recovery algorithms** will frequently be referred to in this document just as "**Current**".
- The proposed algorithms will similarly be referred to as "**Proposed**" or "**Proposal**"
- This document aims to describe the intended behaviour of the algorithms but does not provide pseudo-code of the sort found in IEEE 802.1CB
  - If group consensus is achieved, the next step will be to develop potential specification text, including pseudo-code.

Notes

- The Overview section skips a few definitions which are, hopefully, somewhat intuitive.
  - See Expected Behaviour for full definitions
  - Some definitions also aim to fix some muddy definitions in the current document
    - Example: current doc refers to **BEGIN** event, which triggers SeqGeneration**Reset** routine, but the initialisation state (**INIT**?) is not described.
- Where **SmashCase** or **camelCase** is used, it refers to existing defined terms from IEEE 802.1CB or to new, similar proposed terms.
  - We've tried to use this consistently…but please don't worry about the occasional anomaly; this is not a specification document.

Comparison: Current vs Proposed

The basics of the proposed algorithms

Main Goals of Proposal

- Minimise generation of erroneous counts (e.g. LostPackets)
- Minimise the chance of frames being erroneously dropped
- Enable processing of an “old” stream (i.e. frames arriving via the Slow path) following a reset at the Generation algorithm (with frames from the “new” stream arriving via the Fast path)
- Better handling of pauses in the stream vs. lost packets (i.e. lost on both fast and slow paths simultaneously)
- Enable frequent resets / initialisations of the Generation algorithm (that are likely to be more common with vPLC deployments)
  - This behaviour is risky without the improvements listed above

Sequence Generation Algorithm

Current

- Single Sequence Number Space
  - Continuous loop
- Sequence Number Space Length = 65,536
  - GenSeqSpace = 65,536 = 2^{16}, i.e. 16bit value
- On INIT & RESET, sets GenSeqNum = 0

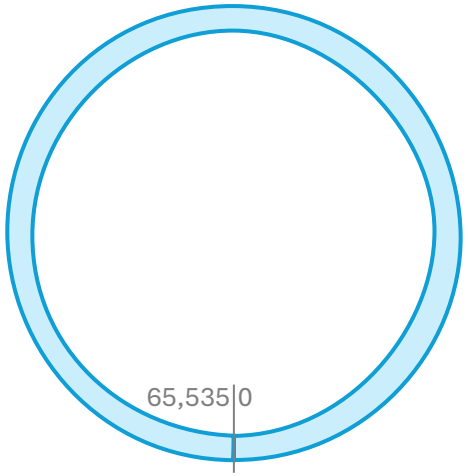
Proposed

- Two Sequence Number Spaces
  - Regular (REG, ResetSpace = 0); continuous loop
  - Reset (RESET, ResetSpace = 1); linear
  - Reset space feeds into Regular
- REG Sequence Number Space Length = 65,536
- RESET Sequence Number Space Length…

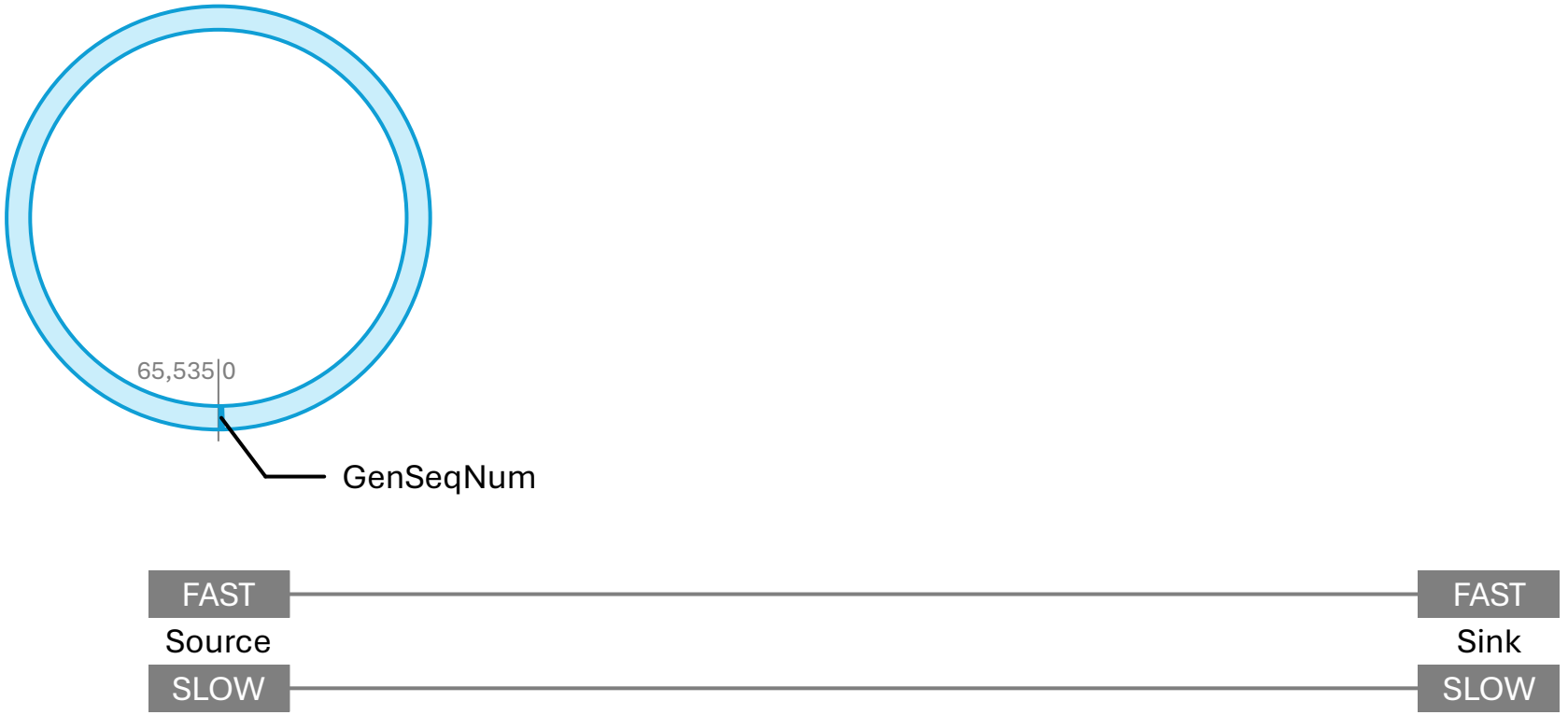
  $resetSeqSpace = ValidWindowLength + (2 \times seqRecoveryHistoryLength)$
  - i.e. resetStart = 65,536 – resetSeqSpace
- On INIT & RESET, sets GenSeqNum = resetStart
  - Starts in Reset space, i.e. ResetSpace= 1
  - First few frames also carry ResetFlag = 1
    - After resetFlagLength packets, ResetFlag= 0

Current – Sequence Number Space

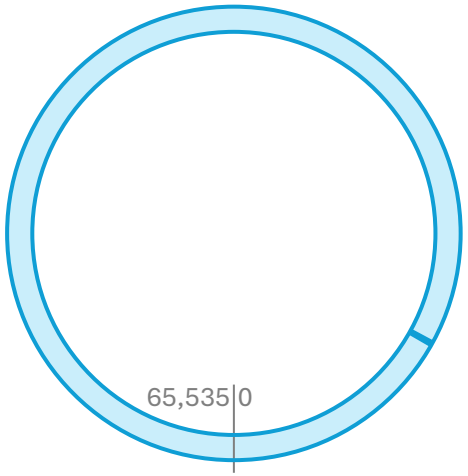
Current – Sequence Number Space



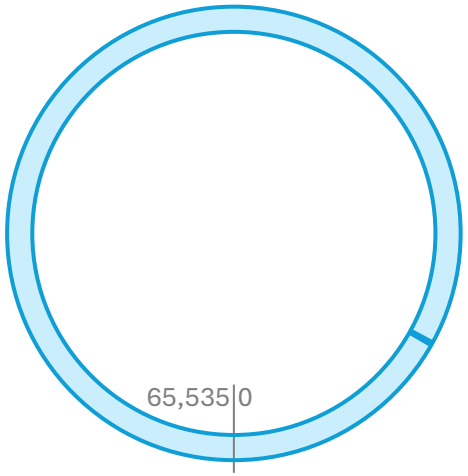
Current Generation Algorithm



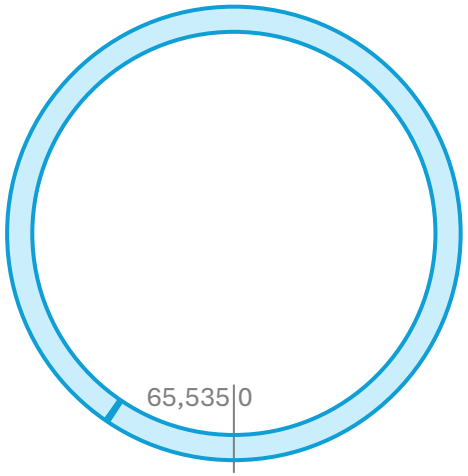
Current Generation Algorithm



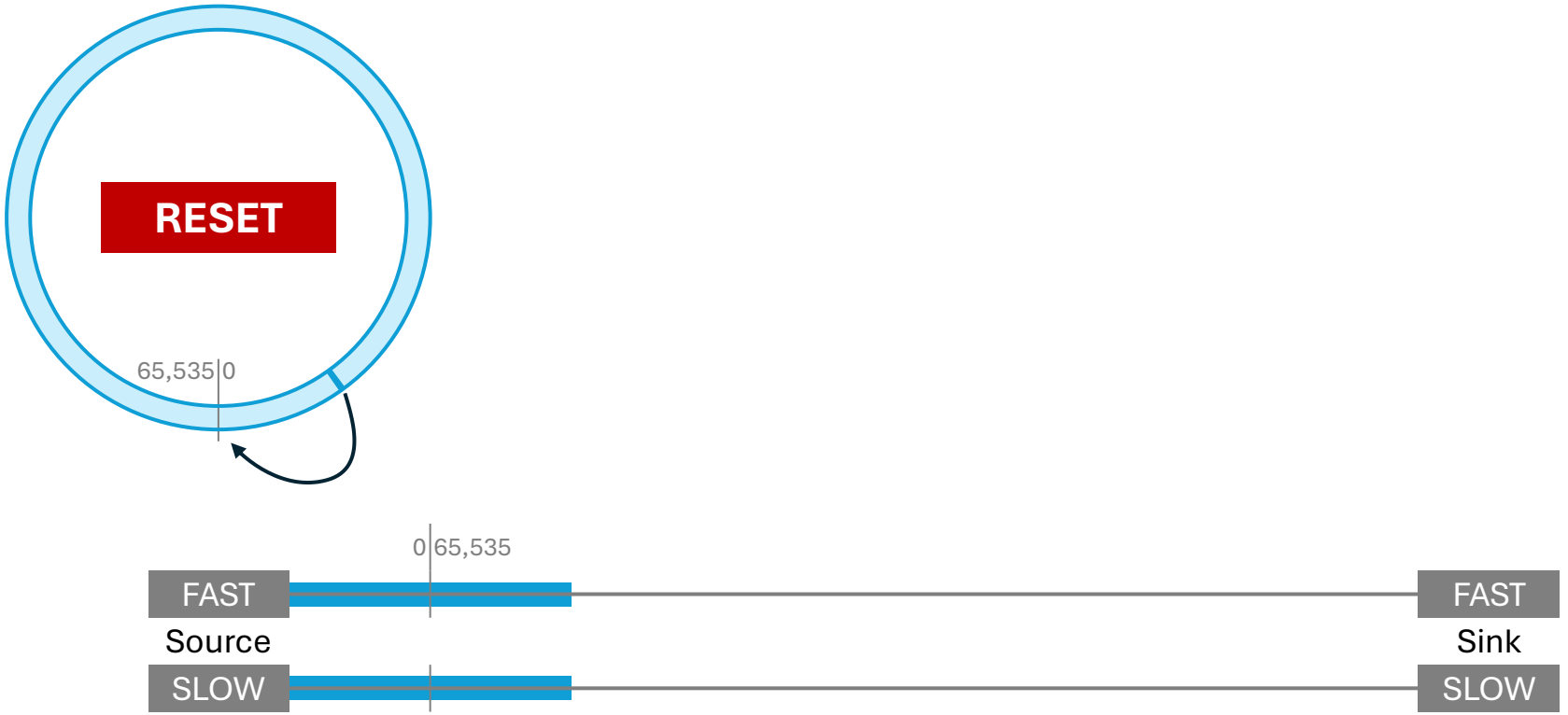
Current Generation Algorithm



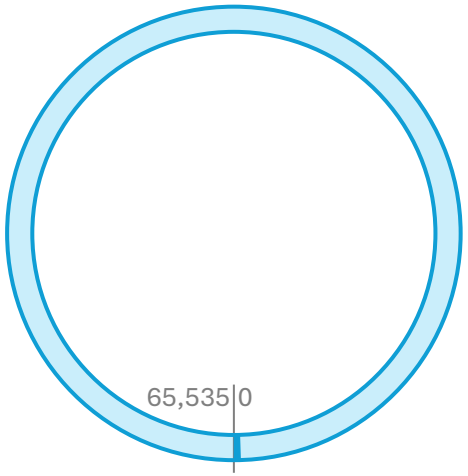
Current Generation Algorithm



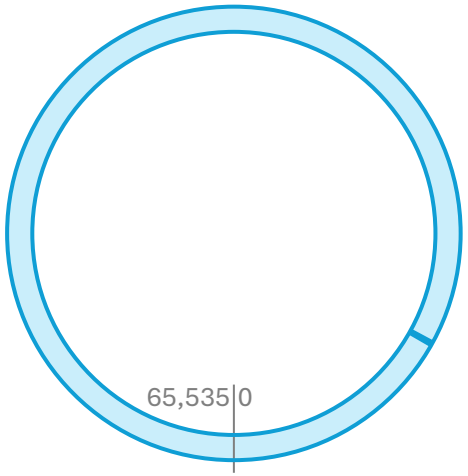
Current Generation Algorithm



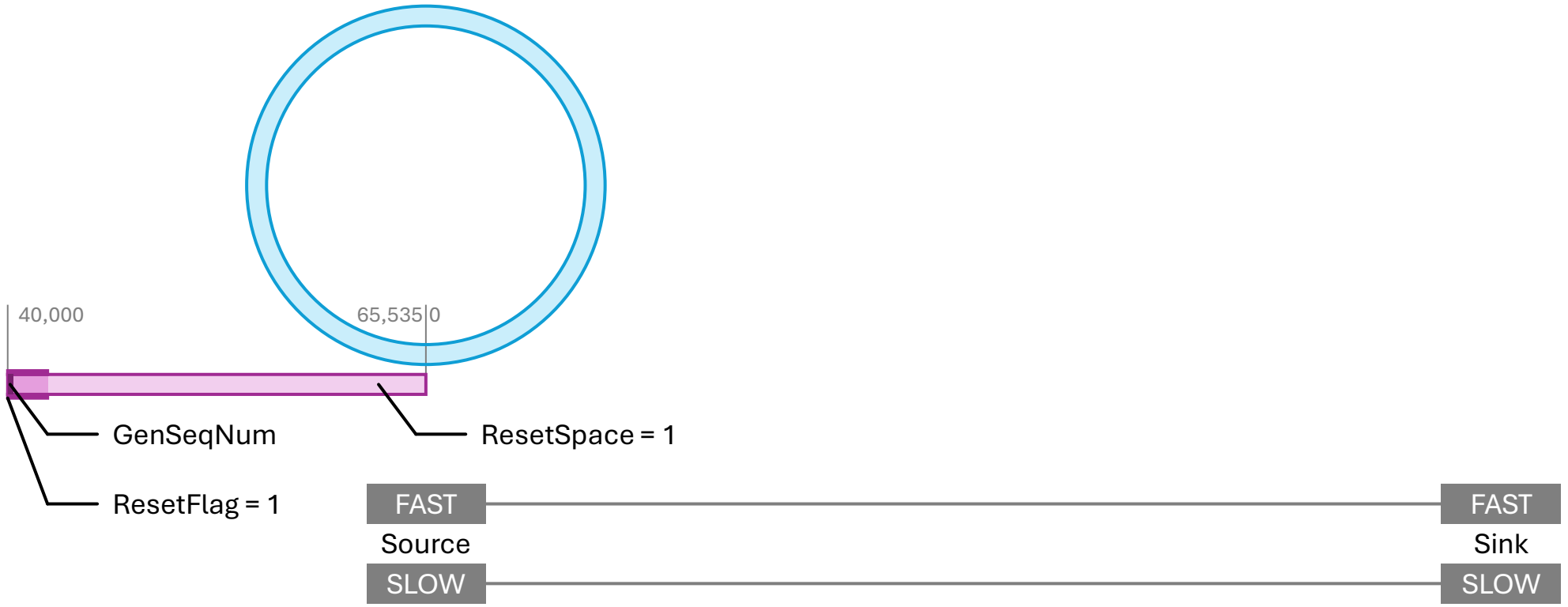
Current Generation Algorithm



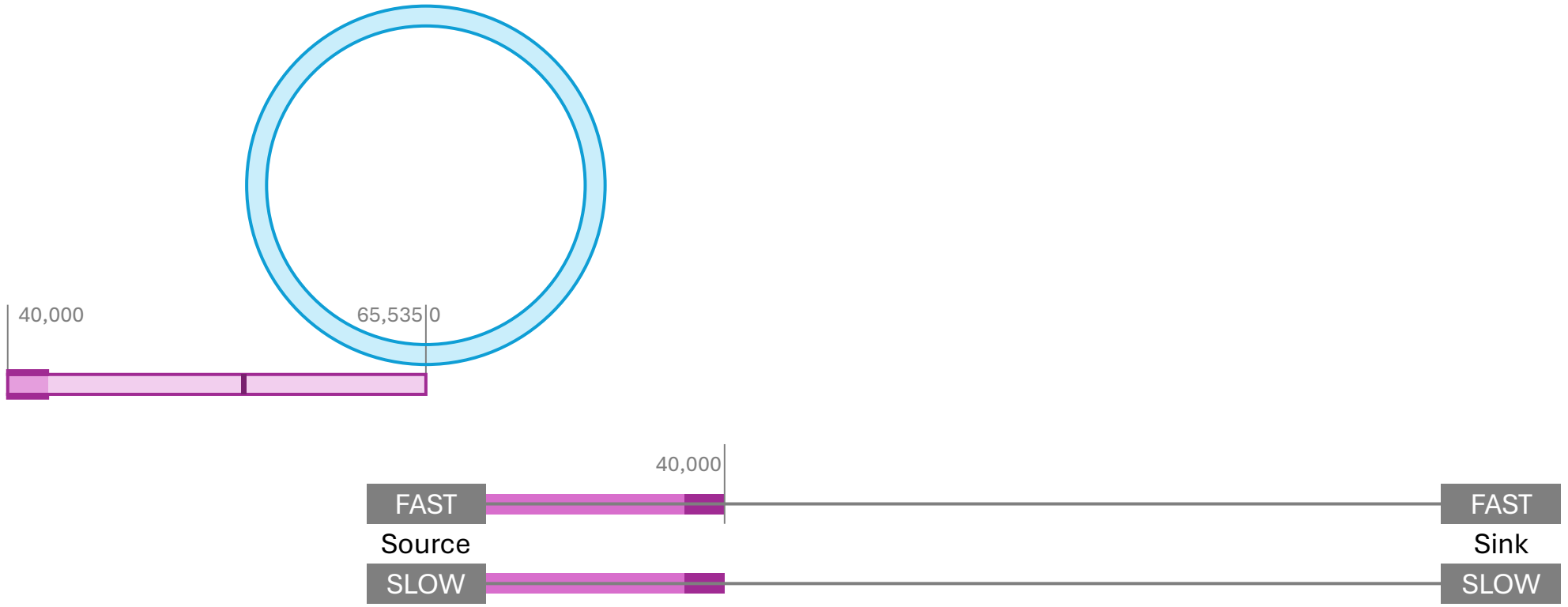
Current Generation Algorithm



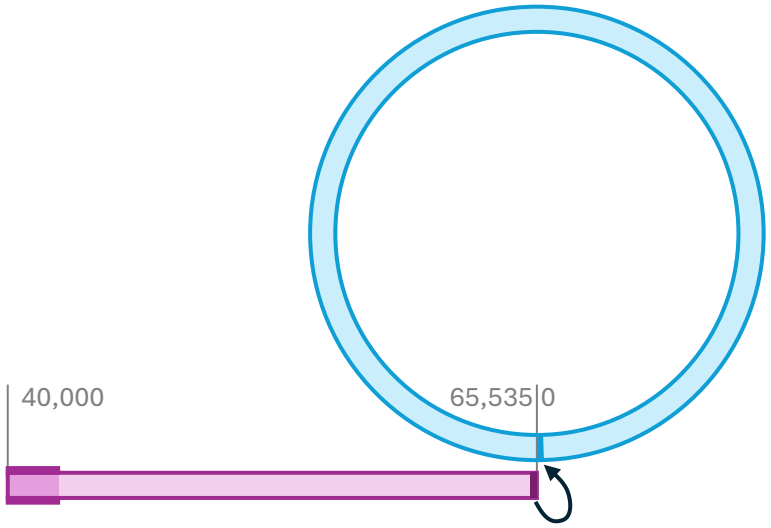
Proposed Generation Algorithm



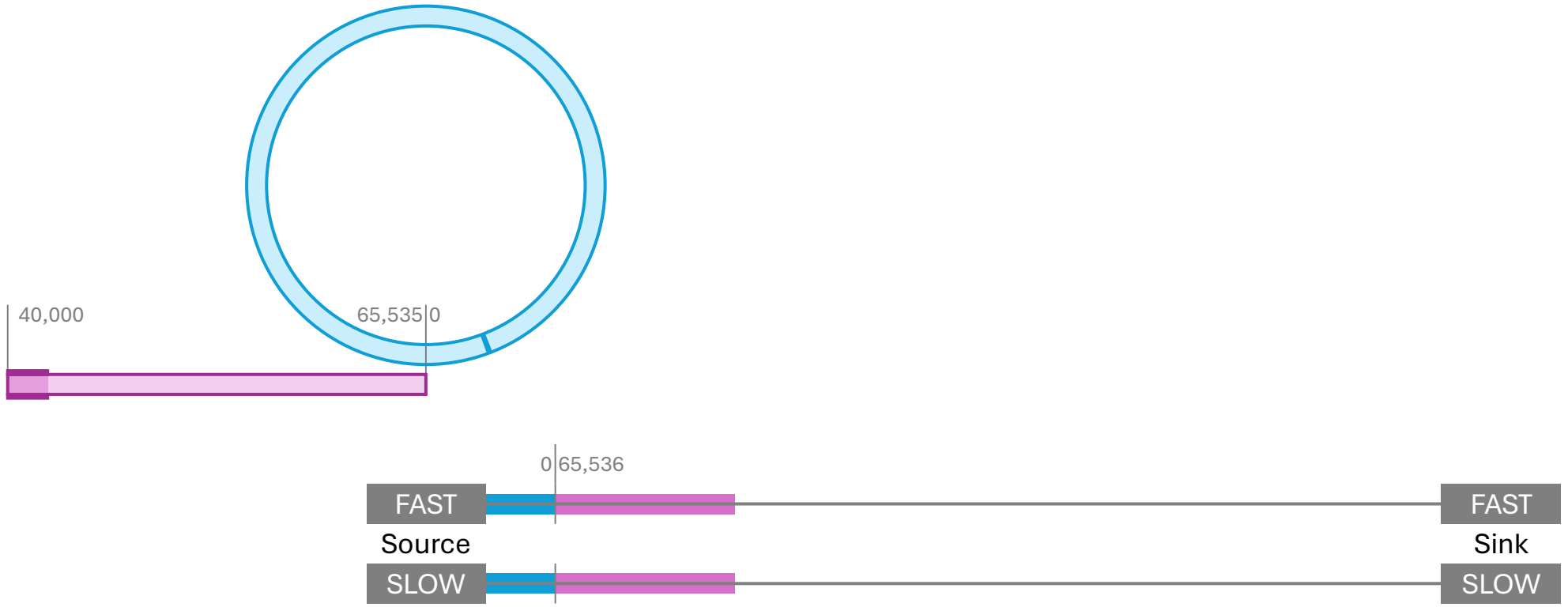
Proposed Generation Algorithm



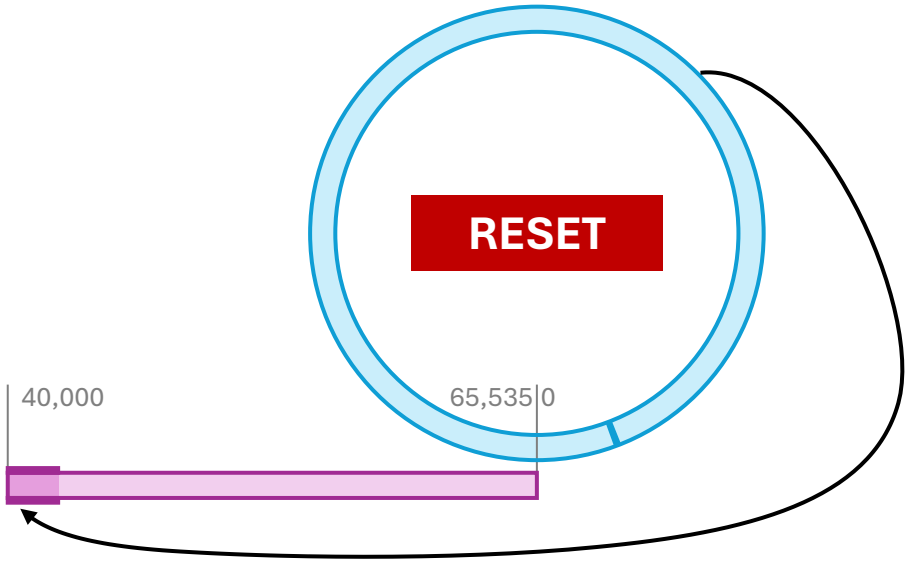
Proposed Generation Algorithm



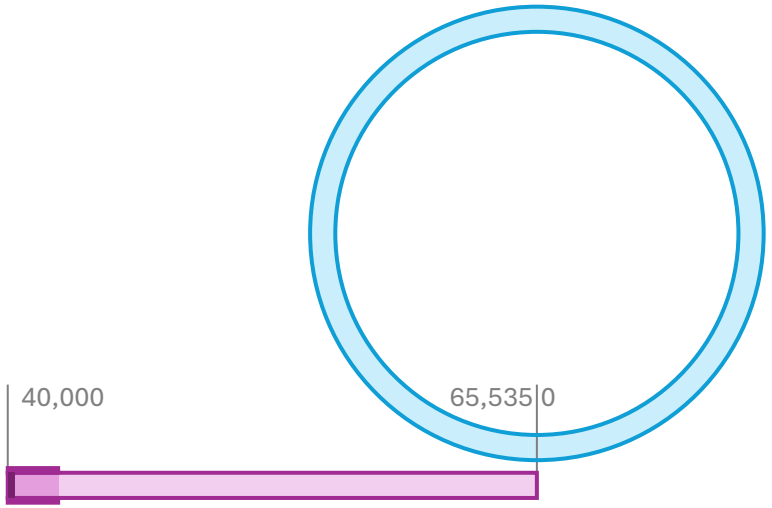
Proposed Generation Algorithm



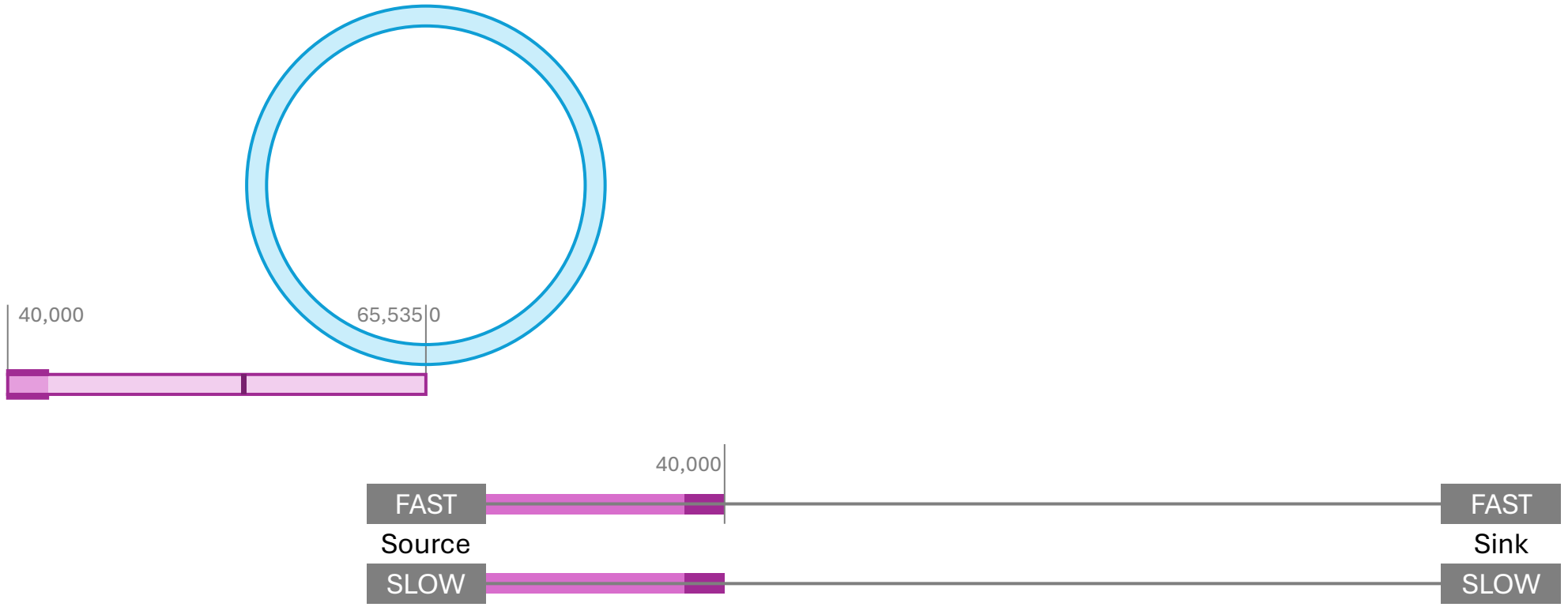
Proposed Generation Algorithm



Proposed Generation Algorithm



Proposed Generation Algorithm



Sequence Vector Recovery Algorithm

Current

- Same behaviour for INIT, RESET & TIMEOUT
  - INIT isn't explicitly defined. BEGIN is defined and triggers RESET behaviour
  - TIMEOUT also triggers RESET behaviour
- Behaviour is…
  - TakeAny = TRUE
  - SequenceHistory = All 0s (expecting frames)
- Valid Window is same length as SequenceHistory

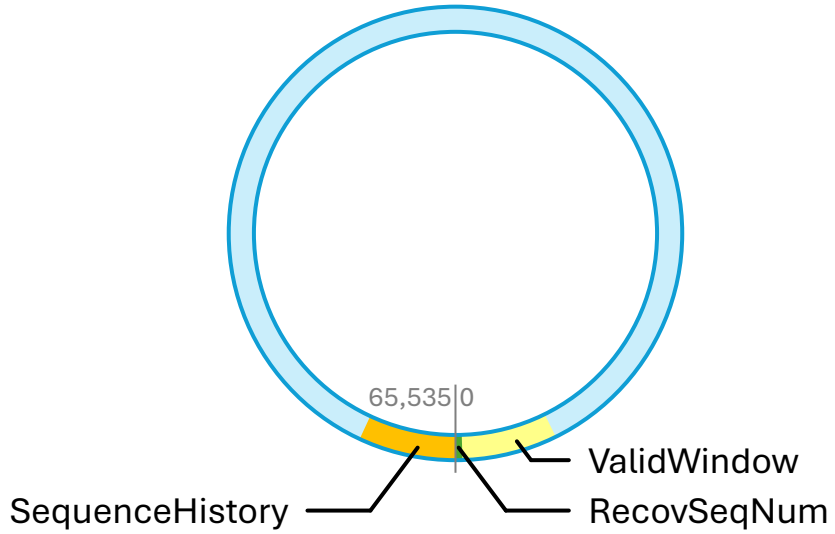
Proposed

- Different behaviour for INIT, RESET and TIMEOUT
  - Uses ResetFlag & ResetSpace to identify an INIT / RESET at Sequence Generator & differentiate from TIMEOUT
- Uses ResetSpace & ResetFlag to identify "new" sequence from Generation algorithm
- Uses ResetSpace to enable processing of "old" sequence in parallel with "new" sequence
- Processes Sequence History on TIMEOUT
- After TIMEOUT, does not assume SequenceHistory will be filled (does not expect frames) but handles slow-path packets gracefully
- Valid Window can be arbitrarily longer than Sequence History (within limits)
  - Allows longer interval before TIMEOUT

Current Vector Recovery Algorithm

At RESET / TIMEOUT (INIT is undefined, but BEGIN = RESET)

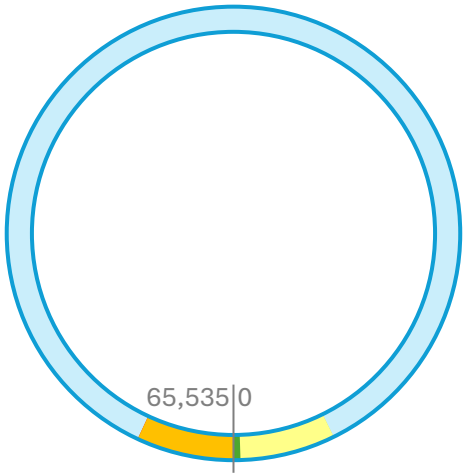
- RecovSeqNum = 65,535
- TakeAny = TRUE; expected first SeqNum is 0
- SequenceHistory length = frerSeqRcvyHistoryLength
  starts at RecovSeqNum (always 1, because it's the highest value valid packet received)
- ValidWindow (not a defined term in current algorithm) is same length as SequenceHistory



Current Vector Recovery Algorithm

Sequence History

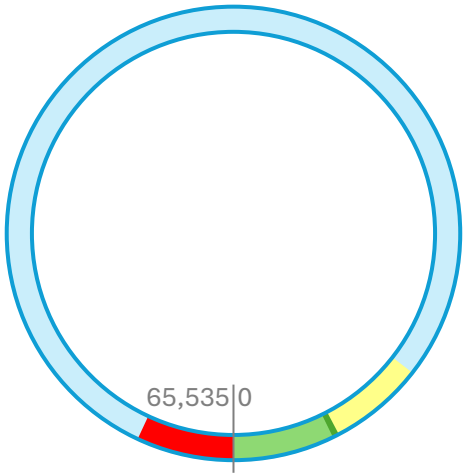
- After RESET, all values are 0, i.e. frames are expected
- Once valid frame is received, set to 1
- If “0” falls out the back of the SequenceHistory (i.e. no valid frame received) it is counted as a LostPacket



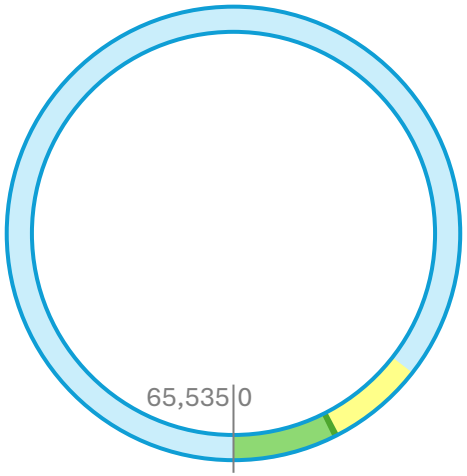
Current Vector Recovery Algorithm

This is one of the issues with the current algorithm

- After every RESET or TIMEOUT the algorithm is likely to generate erroneous LostPacket counts (up to frerSequenceHistoryLength)



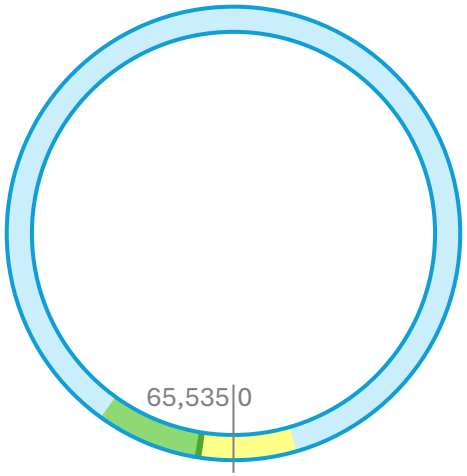
Current Vector Recovery Algorithm

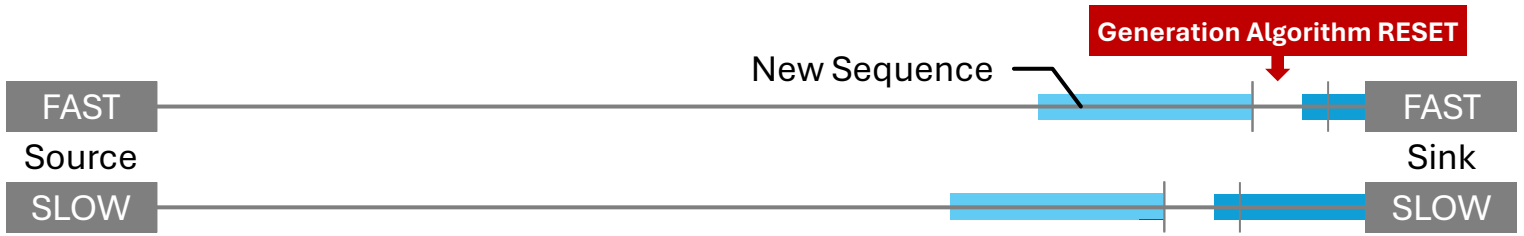


Current Vector Recovery Algorithm

Generation Algorithm Resets when the “old” sequence number is close to zero (in either direction), i.e. when the Recovery Algorithm Sequence History and Valid Window overlap with the “new” sequence, are problematic.

- May result in erroneously dropped frames



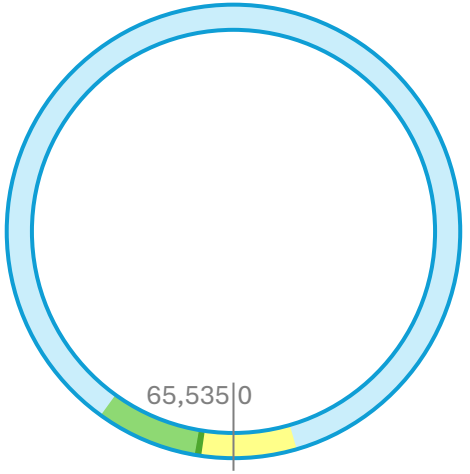


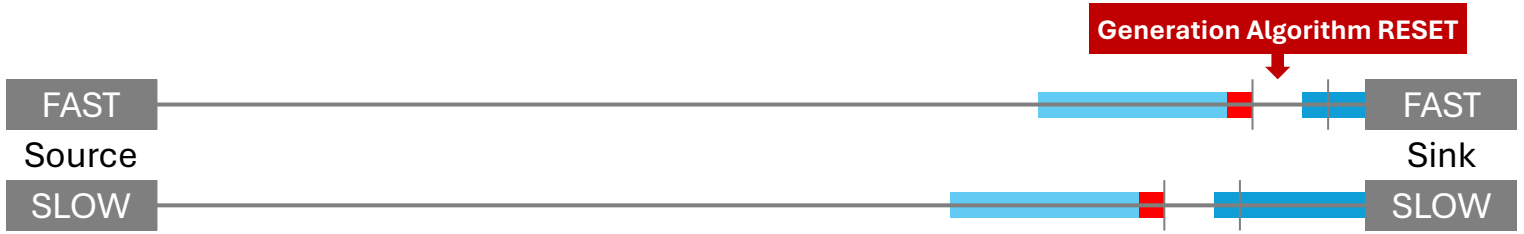
Current Vector Recovery Algorithm

In this example, the frames immediately following the RESET will be dropped, because the Recovery Algorithm thinks it has already seen them.

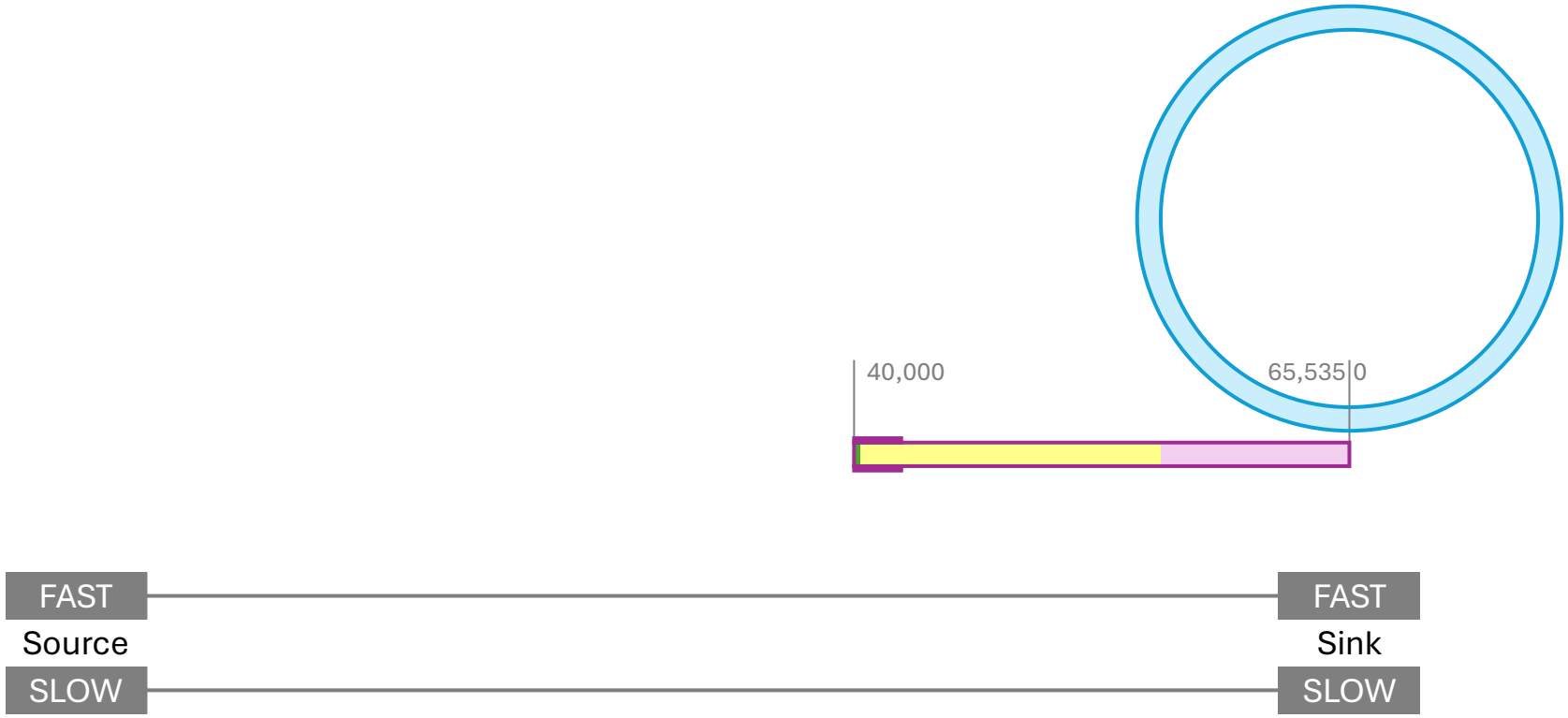
This still happens if the “new” sequence is delayed and doesn’t overlap with the “old” sequence

- Only if there is a TIMEOUT or RESET of the Vector Recovery Algorithm will the initial frames in the “new” sequence not be dropped (but there will be erroneous LostPackets)



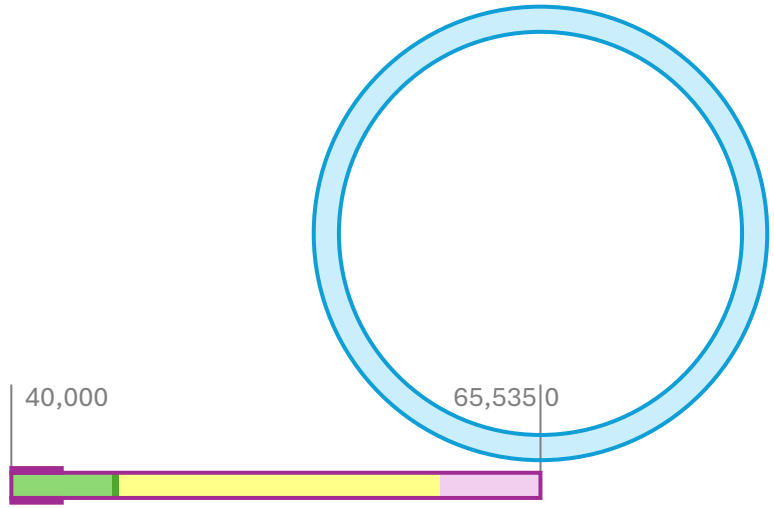


Proposed Vector Recovery Algorithm

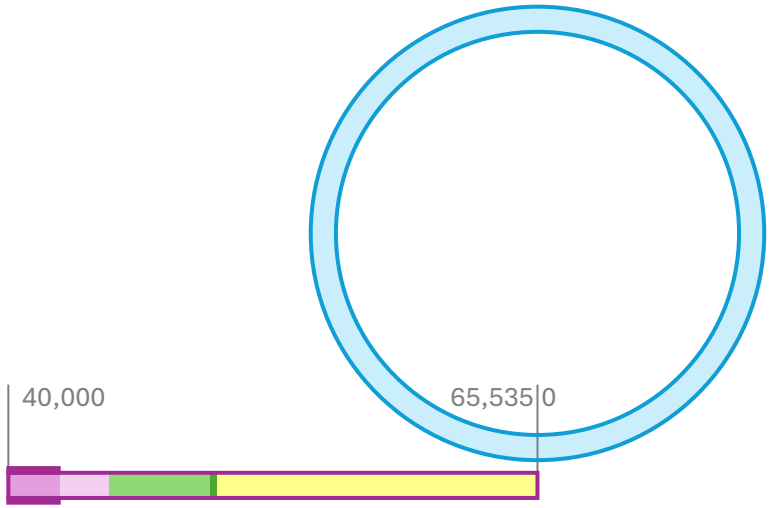


Proposed Vector Recovery Algorithm

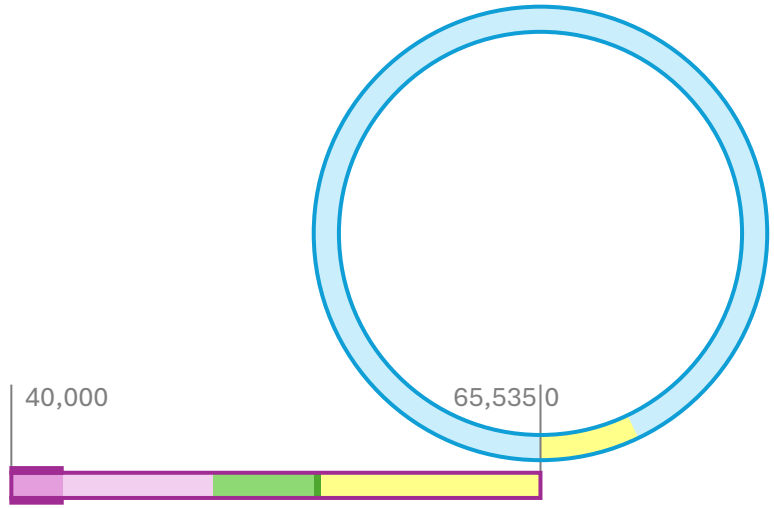
No LostPackets are recorded, because the algorithm knows there are no packets prior to the start of the ResetSpace.



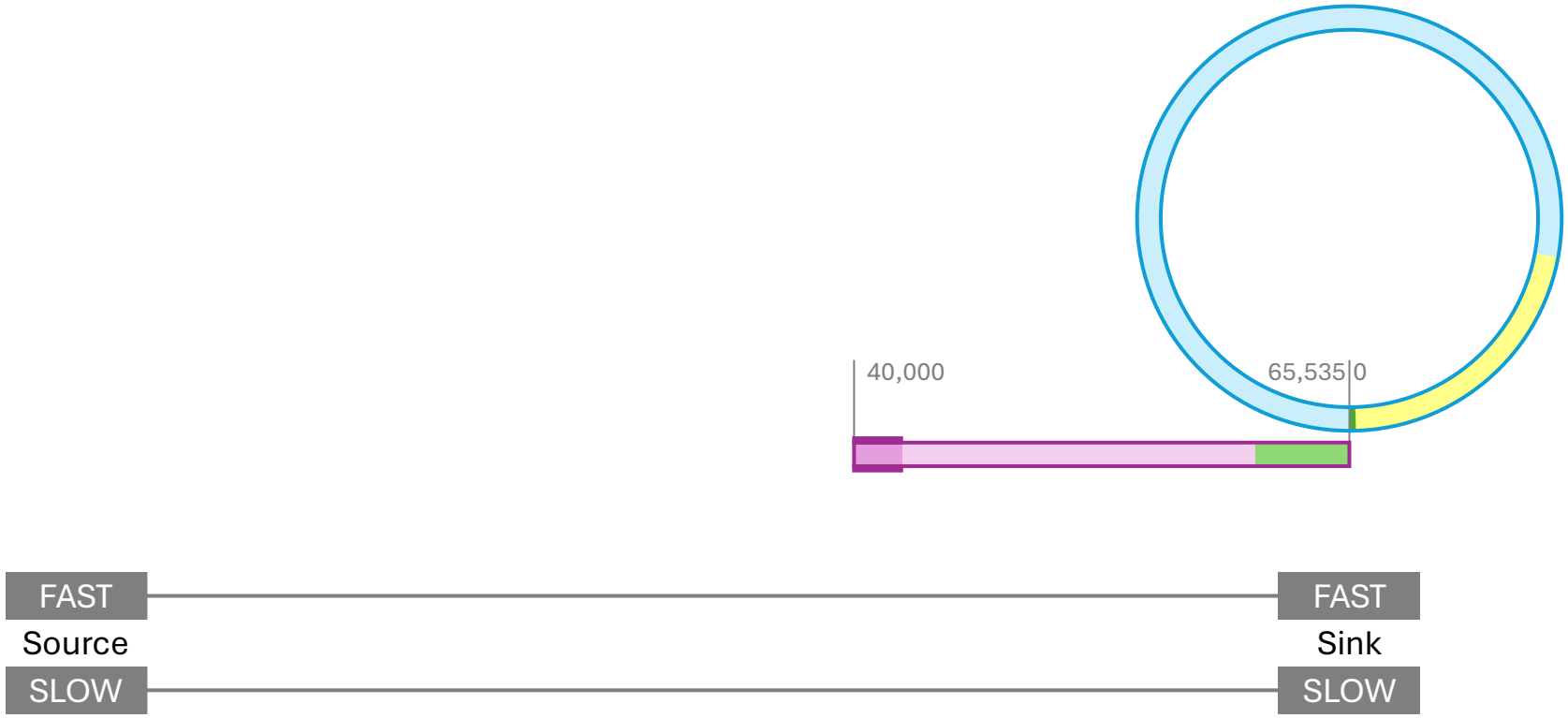
Proposed Vector Recovery Algorithm



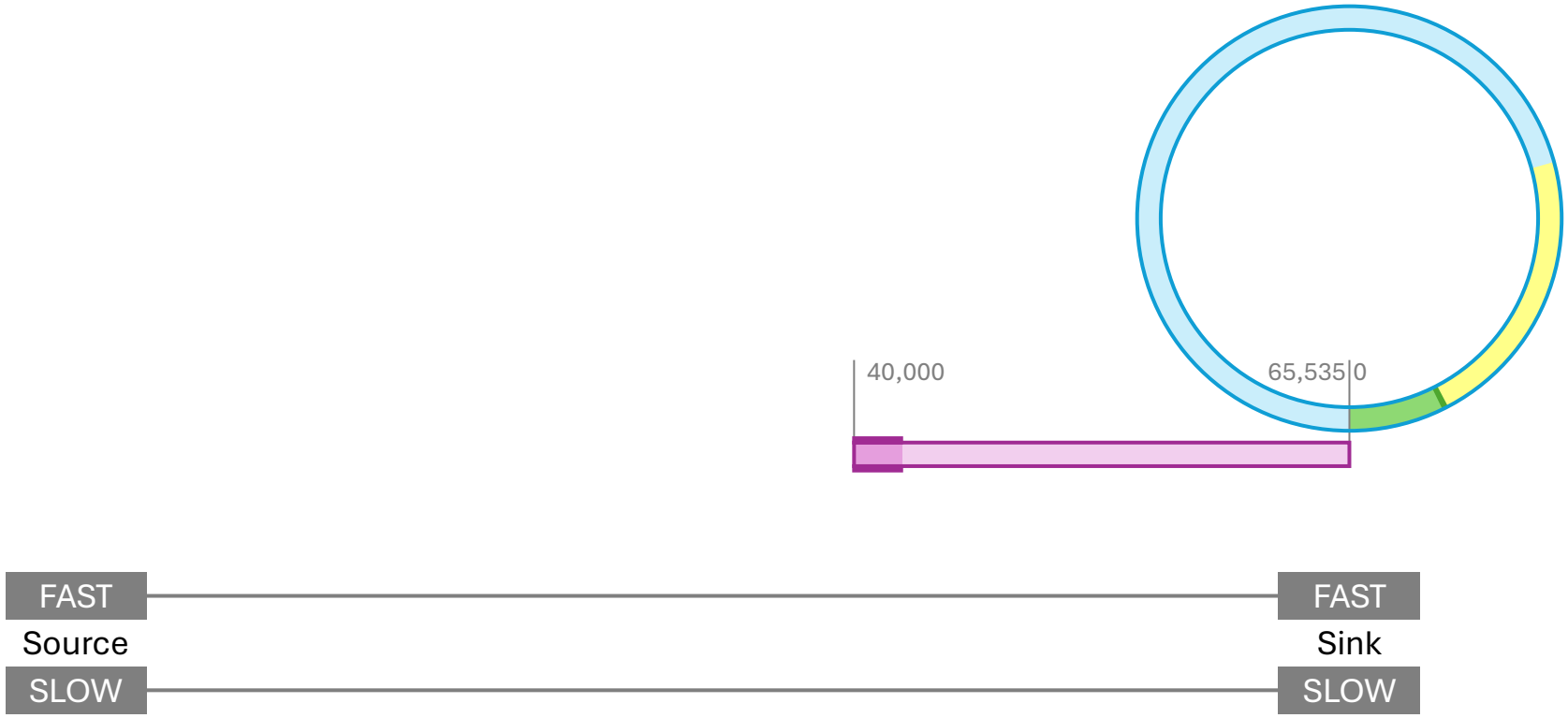
Proposed Vector Recovery Algorithm



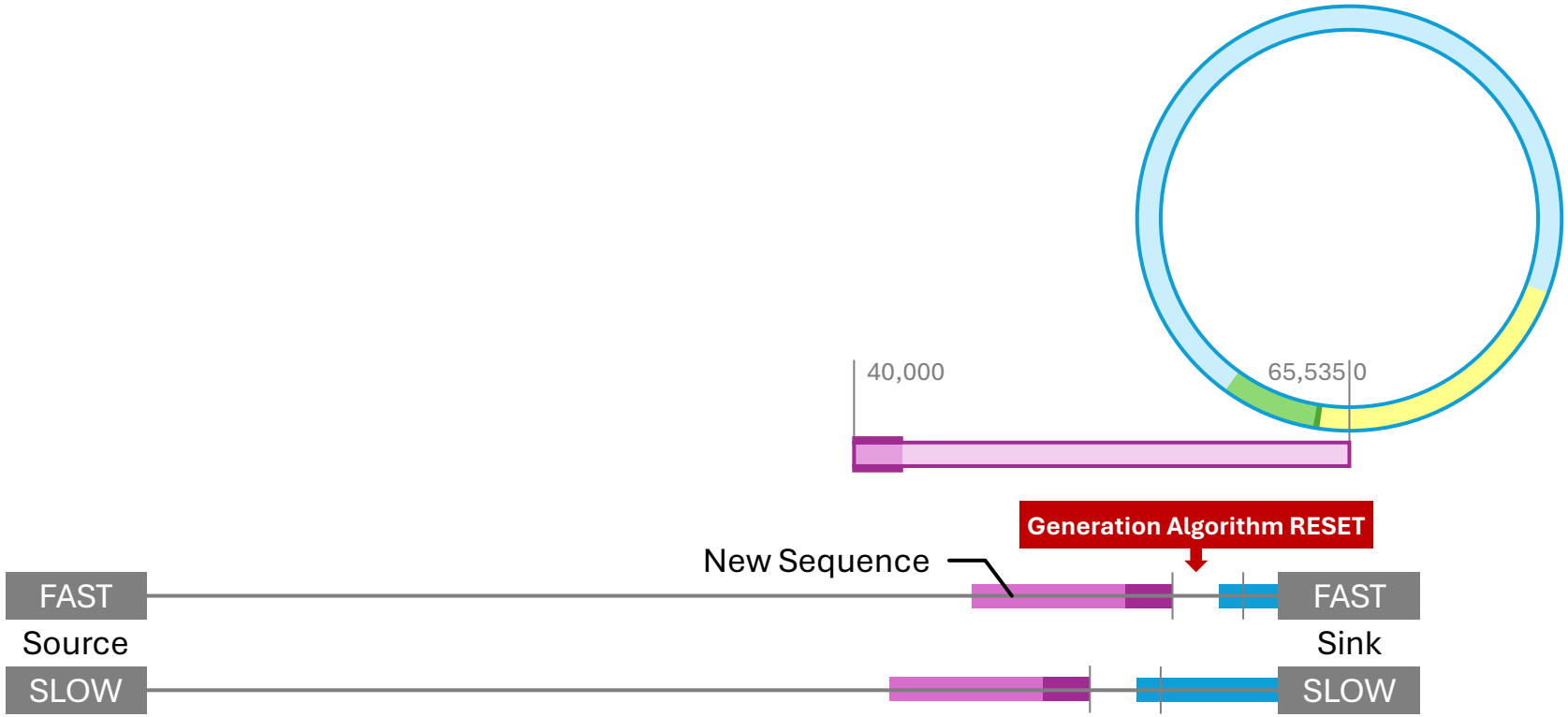
Proposed Vector Recovery Algorithm



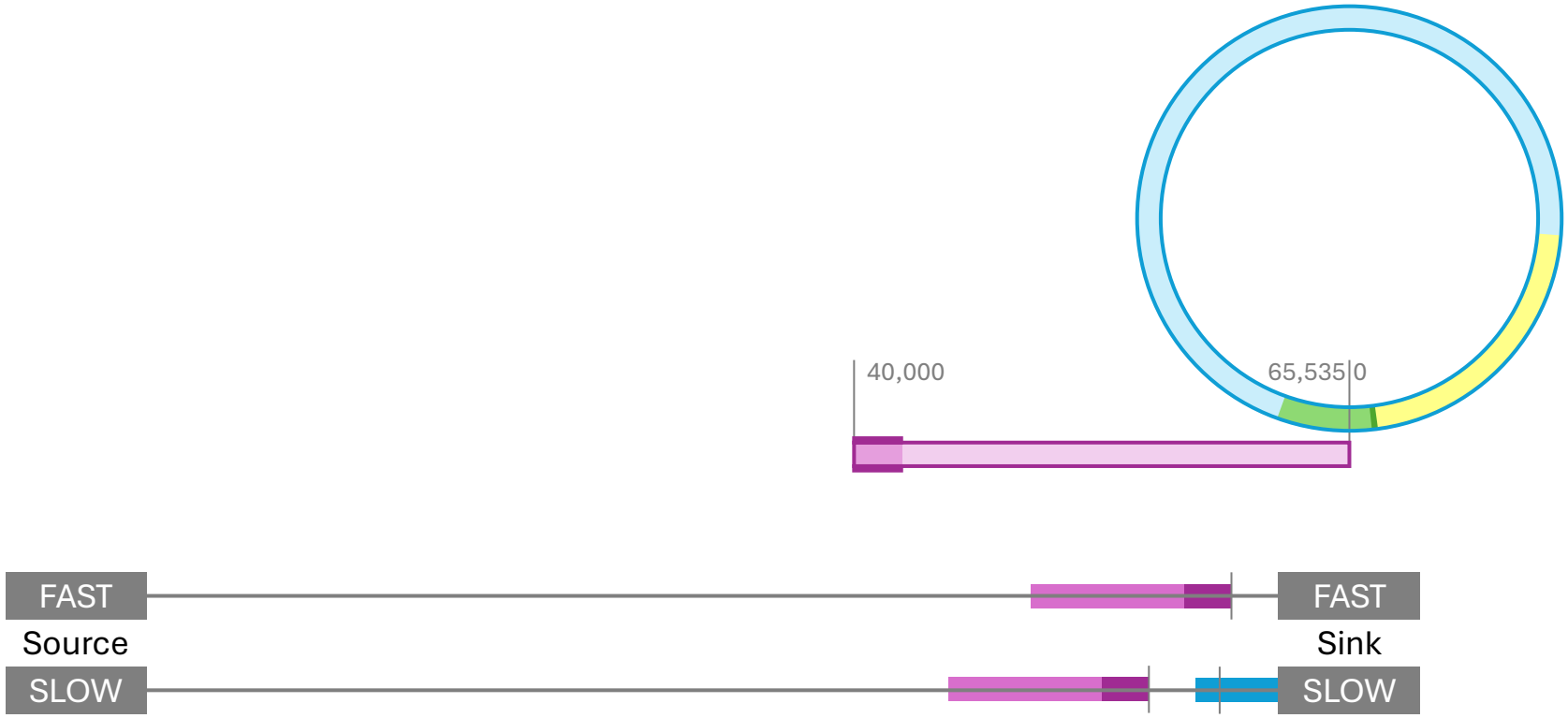
Proposed Vector Recovery Algorithm



Proposed Vector Recovery Algorithm



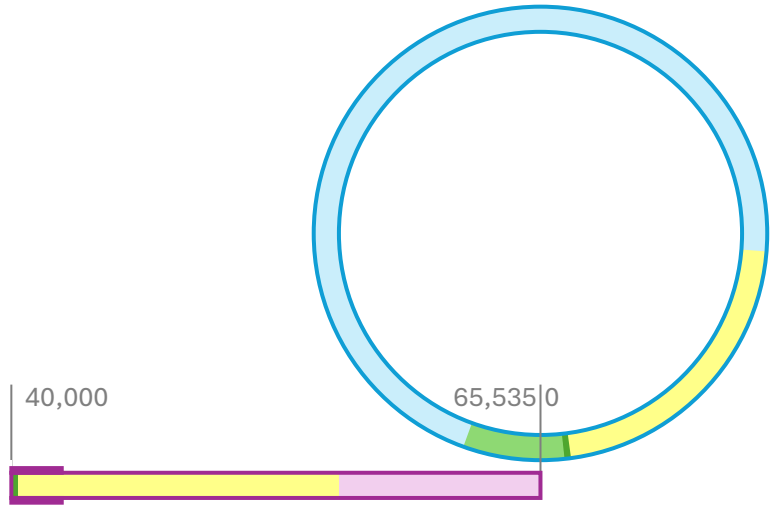
Proposed Vector Recovery Algorithm



Proposed Vector Recovery Algorithm

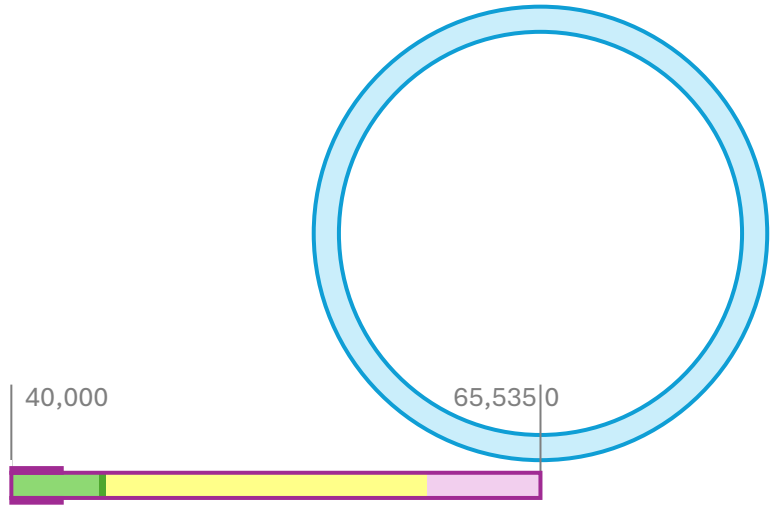
"Old" sequence can continue to be processed in REG sequence space while "new" sequence is processed in RESET sequence space.

- This is optional. The proposed algorithm enables this but does not require it.



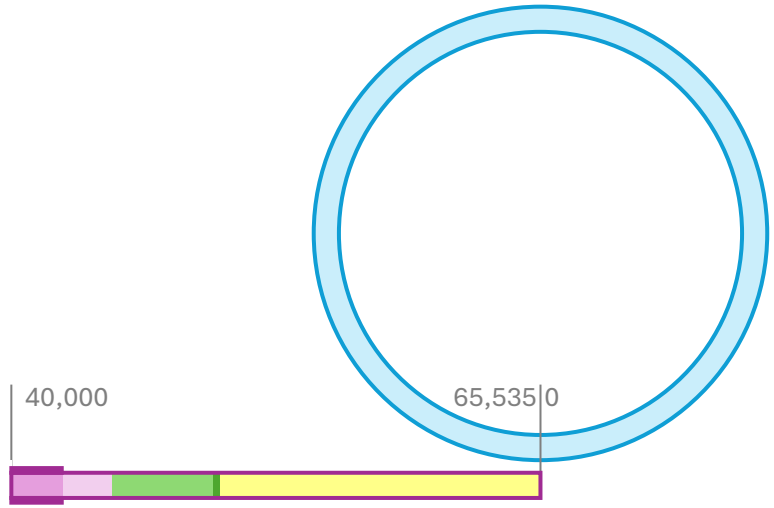
Proposed Vector Recovery Algorithm

"Old" sequence **can** cease being processed when "new" sequence starts to arrive on the slow path (e.g. algorithm sees duplicate frames) or after a timeout (e.g. maximum difference between fastest frame on fast path and slowest frame on slow path.



Proposed Vector Recovery Algorithm

"Old" sequence **must** cease being processed before "new" sequence's Valid Window enters REG sequence space to avoid processing two sequences in REG sequence space at the same time (which may overlap).

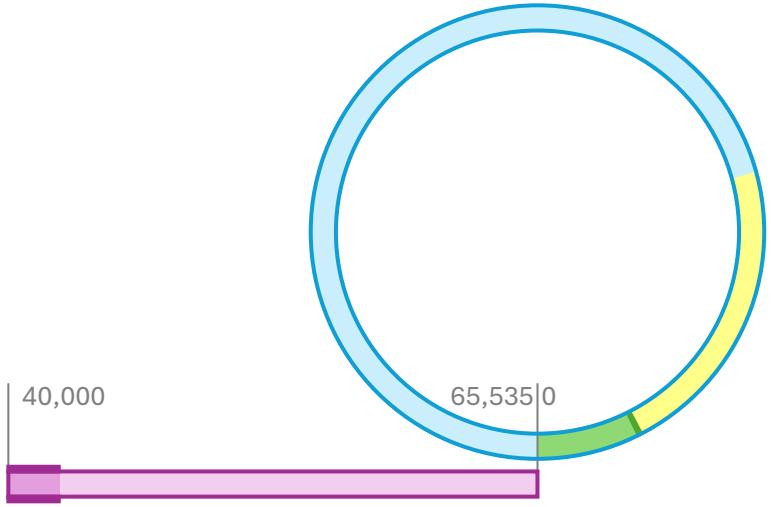


Proposed Recovery Algorithm – TIMEOUT

- On TIMEOUT
  - Process SequenceHistory
    - Any expected frames become Lost Packets
    - This ensures that Lost Packets are counted well before the next frame is received, which could be well into the future (unbounded)
  - TakeAny = TRUE
- On reception of next frame
  - Move RecovSeqNum to received frame (any frame; in REG or RESET space)
  - Do not expect earlier frames to be received (i.e. if moved out of SequenceHistory, they won't be counted as Lost Packets), but…
  - If earlier frames are received (within SequenceHistory limit) then pass them and expect any frames between the earlier frame and the GenSeqNum to be received.
  - Note: this is the same TakeAny behaviour as when TakeAny applies after INIT or RESET

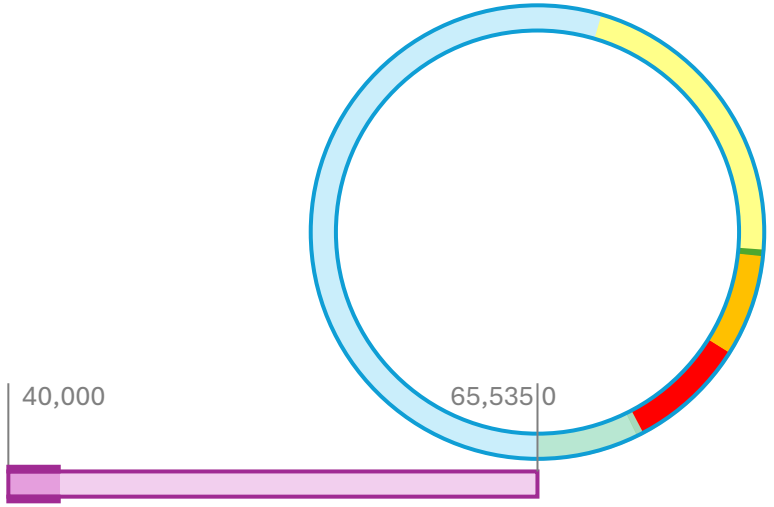
Extended Valid Window

- Extending the Valid Window means the TIMEOUT interval can be longer
  - TIMEOUT should be set to trigger when it's otherwise possible for a valid frame to be dropped because it falls outside the Valid Window



Extended Valid Window

- Extending the Valid Window means the TIMEOUT interval can be longer
  - TIMEOUT should be set to trigger when it's otherwise possible for a valid frame to be dropped because it falls outside the Valid Window

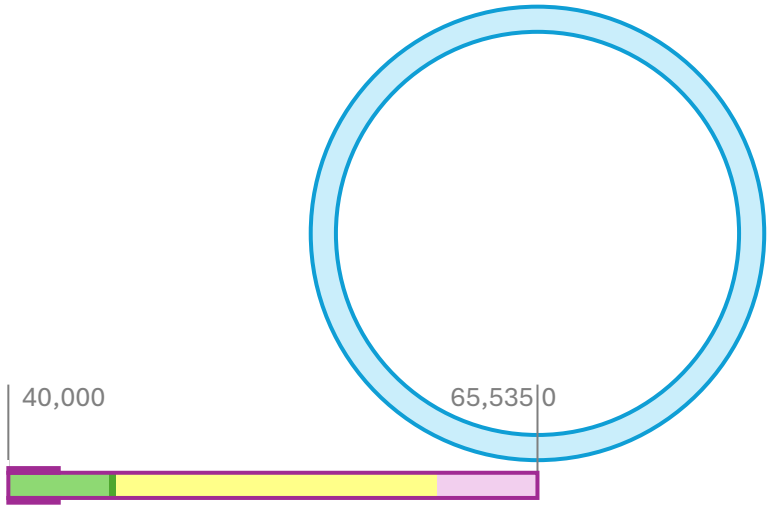


- If the SequenceHistory moves far enough ahead that it no longer overlaps with the pre-move SequenceHistory, the intervening frames are counted as Lost Packets
  - SequenceHistory is set to all 0, i.e. frames are expected and will also be counted as Lost Packets if the don't arrive

Expected Behaviour

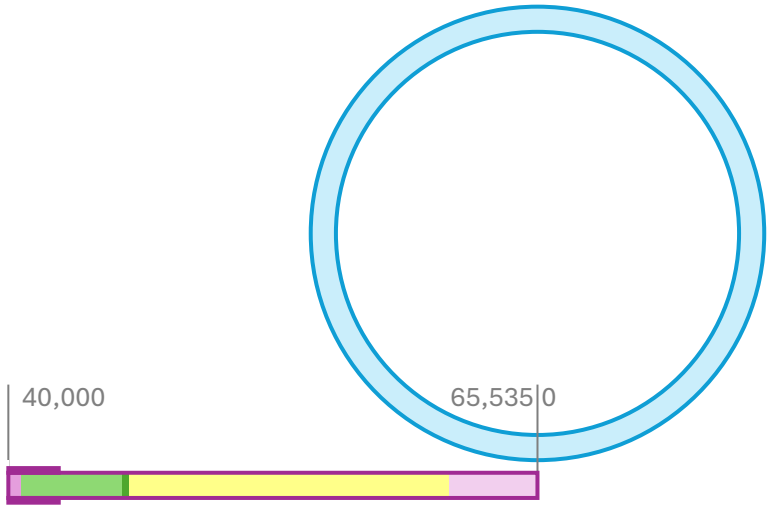
Reset Behaviour

- If there is a 2nd reset while Sequence History still includes the entire ResetFlag section, the algorithm can not recognise it and will not reset
  - This places a hard limit on how frequently the Generation algorithm can reset



Reset Behaviour

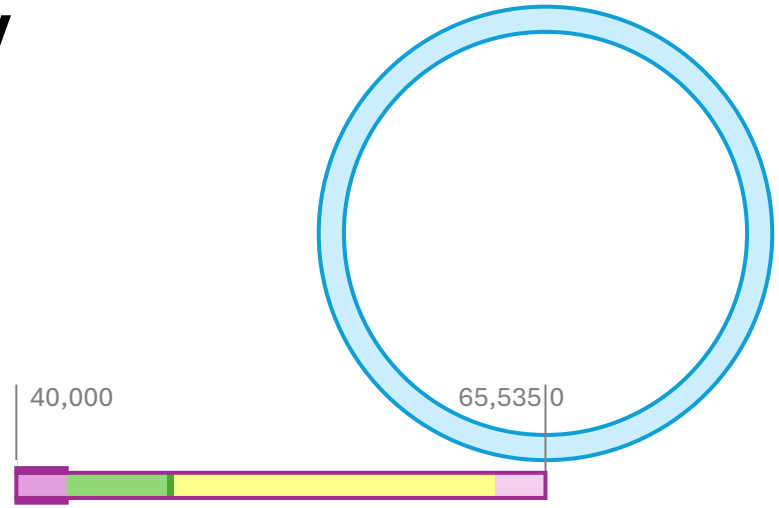
- Once the SequenceHistory is clear of the **start** of the Reset Sequence Space, it is theoretically possible to recognise a frame with the ResetFlag and a sufficiently low value as a “new” sequence and reset



- But that involves continuous calculation of what sequence numbers trigger a Reset vs. those that don’t. It is therefore not proposed.

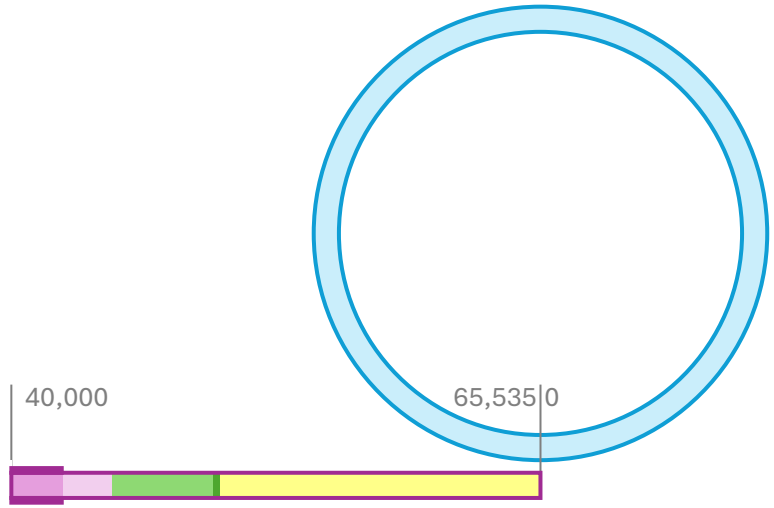
Reset Behaviour

- Once the SequenceHistory is **completely** clear of the ResetFlag section, any frame with the ResetFlag could be expected to trigger a reset
- But that involves the Recovery algorithm knowing how large the ResetFlag section actually is (vs. knowing just the size of the SequenceHistory. So, this is not proposed either.



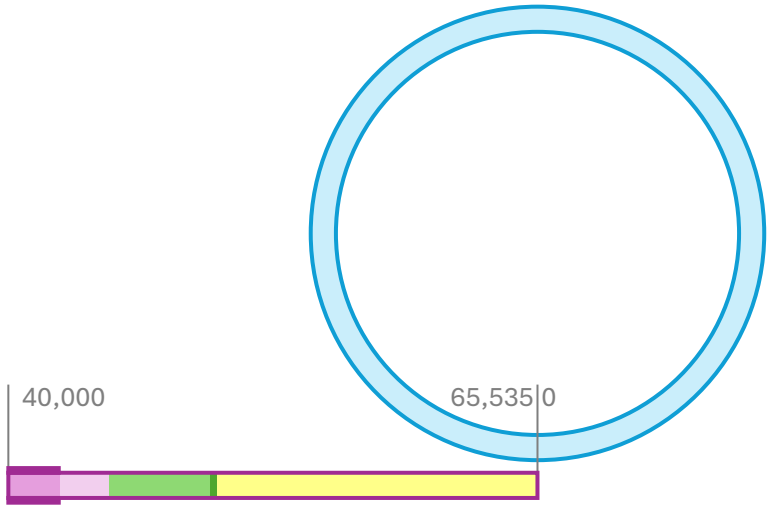
Reset Behaviour

- The proposal is that a Reset is only triggered on reception of frame with the ResetFlag **once the Valid Window reaches the end of the Reset sequence space**.
- This implies that the Reset sequence space should be of length:
  1 x Valid Window + 2 x SequenceHistory
- Also, that ResetFlag section < SequenceHistory length
- Which implies a maximum limit of Valid Windows and SequenceHistory such that the above is < GenSeqSpace



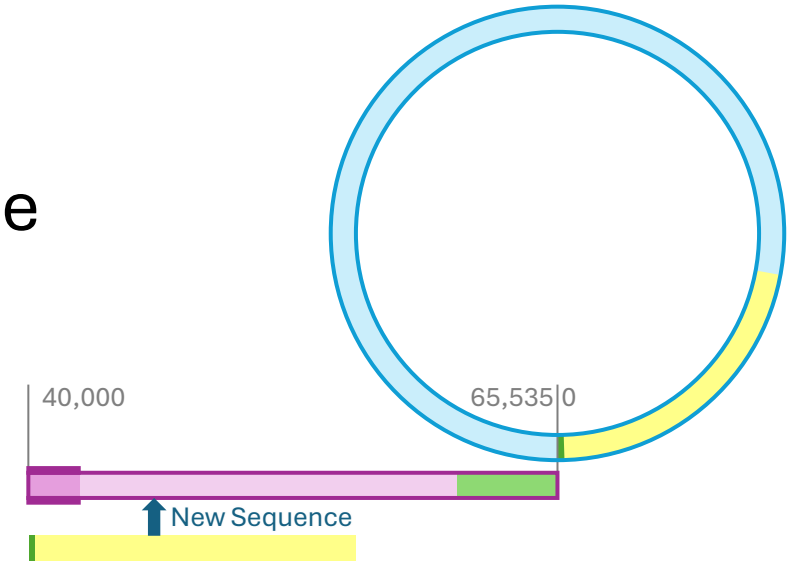
Parallel Processing of “old” and “new” sequences following Gen RESET

- Reminder: this is an optional capability that is merely enabled by the proposal
- If the “old” Valid Window and/or SequenceHistory is mostly still in RESET sequence space, processing the “new” sequence based on ResetSpace flag is impossible



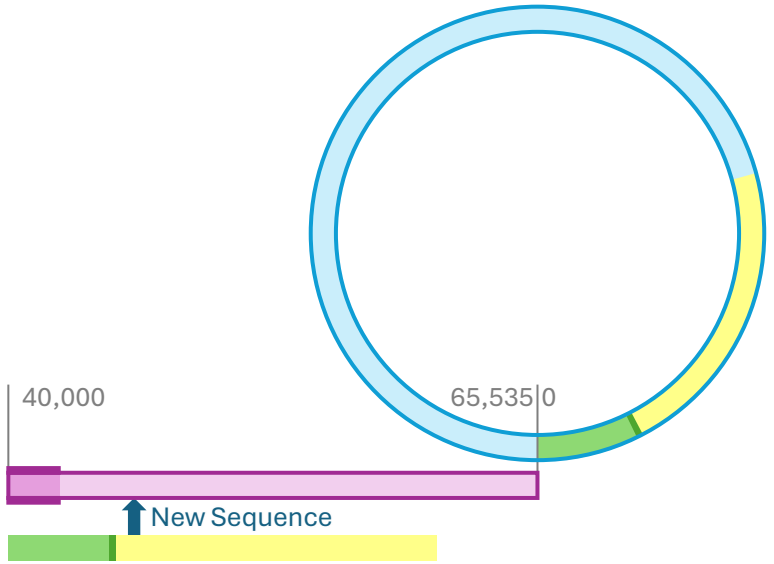
Parallel Processing of “old” and “new” sequences following Gen RESET

- If processing of the “old” sequence has progressed to the point where only the SequenceHistory is in the Reset sequence space, processing the “new” sequence is theoretically possible
  - But, it would require careful tracking of the division in Reset sequence space
  - Depending on when packets arrive, could start to overlap again as the “new” stream is processed, breaking the separation.
  - Therefore: not expected



Parallel Processing of “old” and “new” sequences following Gen RESET

- It is expected that parallel processing of “old” and “new” sequences will only be possible if the “old” sequence is entirely in the REG sequence space
  - i.e. all SequenceHistory and Valid Window
  - ResetSpace can be used to differentiate



Further Work

Further Work

- Document any non-obvious choices and their rationales
- Double check backwards compatibility and effect of proposed Generation algorithm on Match Sequence Recovery algorithm
- Agree on which elements of the proposal should be optional vs. mandatory in a future spec version, e.g.:
  - Generation algorithm use of ResetSpace
  - Recovery algorithm TIMEOUT vs RESET behaviour
  - Recovery algorithm use of ResetSpace
- If the group agrees to proceed, generate pseudocode that implements the proposed algorithms

Thank you!